SPRING

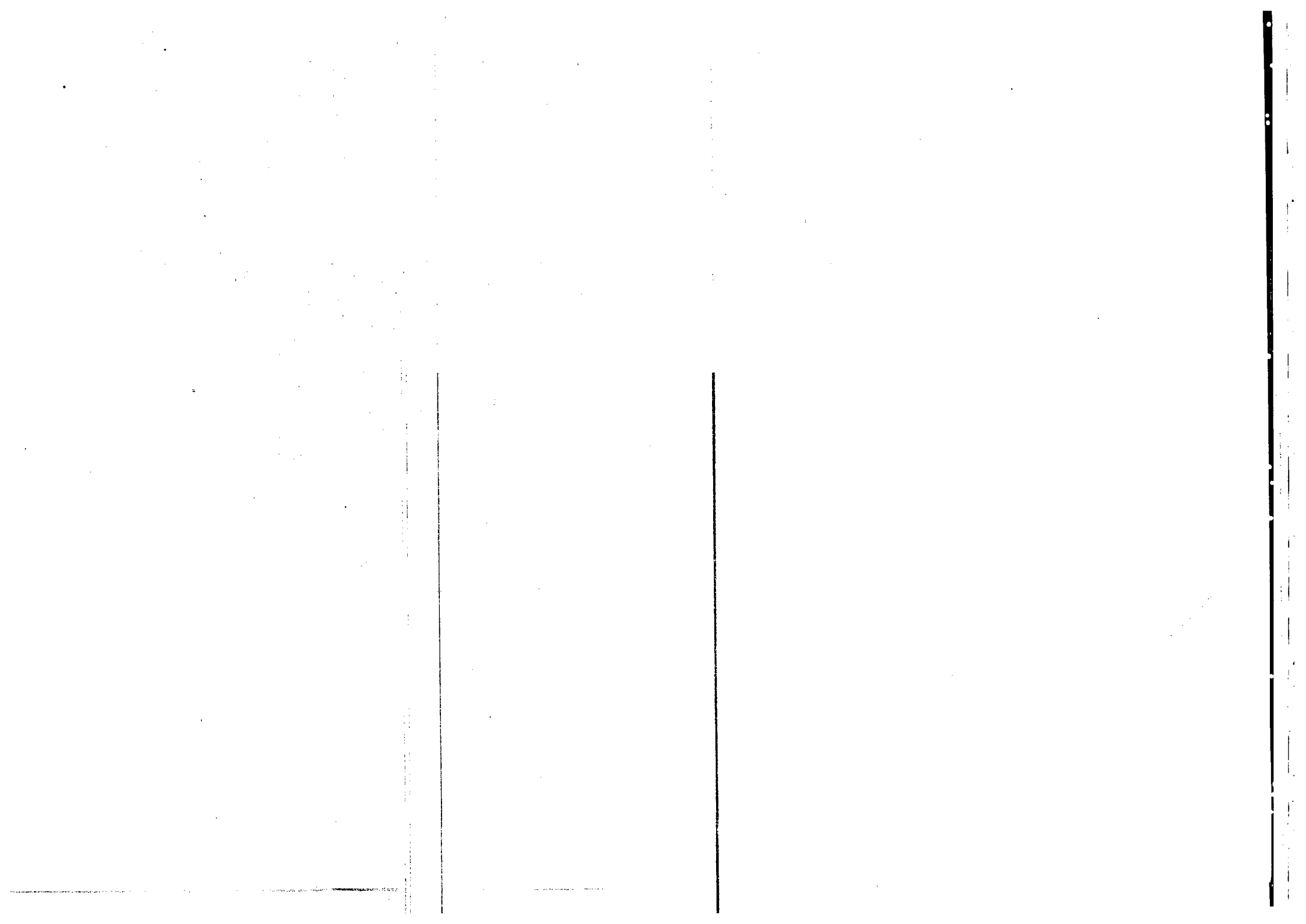
MATERIAL

DURGA TECHNOLOGIES

MANO ENTERPRISES

Shop No.3, Behind Mytrivanam, Syamala Plaza,
MITHRIVANAM,AMEERPET,HYD.

CONTACT : 040-65501230,9912991230.



SPRING FRAMEWORK

Spring is grate framework for development of Enterprise grade applications. Spring is a light-weight framework for the development of enterprise-ready applications. Spring can be used to configure declarative transaction management, remote access to your logic using RMI or web services, mailing facilities and various options in persisting your data to a database. Spring framework can be used in modular fashion, it allows to use in parts and leave the other components which is not required by the application.

Features of Spring 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.
- **AOP Framework:** Spring is best AOP framework
- **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..

Spring Architecture

Spring is well-organized architecture consisting of seven modules. Modules in the Spring framework are:

1. **Spring AOP**
   One of the key components of Spring is the *AOP framework*. AOP is used in Spring:
   - 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 Spring's transaction abstraction.
   - To allow users to implement custom aspects, complementing their use of OOP with AOP
2. **Spring ORM**
   The *ORM* package is related to the database access. It provides integration layers for popular object-relational mapping APIs, including JDO, Hibernate and iBatis.
3. **Spring Web**
   The Spring Web module is part of Spring's web application development stack, which includes Spring MVC.
4. **Spring DAO**
   The DAO (Data Access Object) support in Spring is primarily for standardizing the data access work using the technologies like JDBC, Hibernate or JDO.
5. **Spring Context**
   This package builds on the beans package to add support for message sources and for the Observer design pattern, and the ability for application objects to obtain resources using a consistent API.

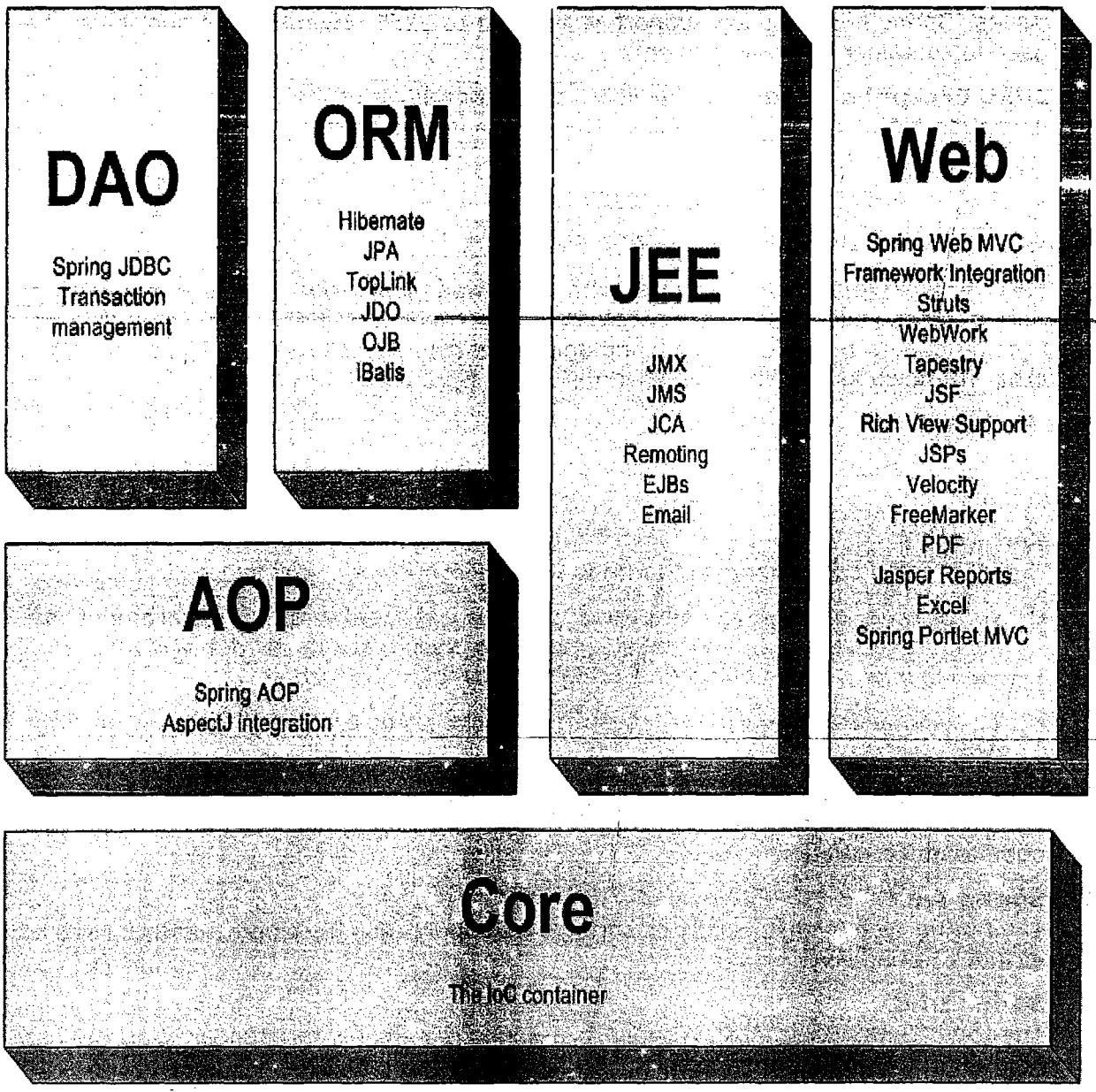
6. **Spring Web MVC**
   This is the Module which provides the MVC implementations for the web applications.

7. **Spring Core**
   The *Core* package is the most import component of the Spring Framework.
   This component provides the Dependency Injection features. The BeanFactory provides a factory pattern which separates the dependencies like initialization, creation and access of the objects from your actual program logic.

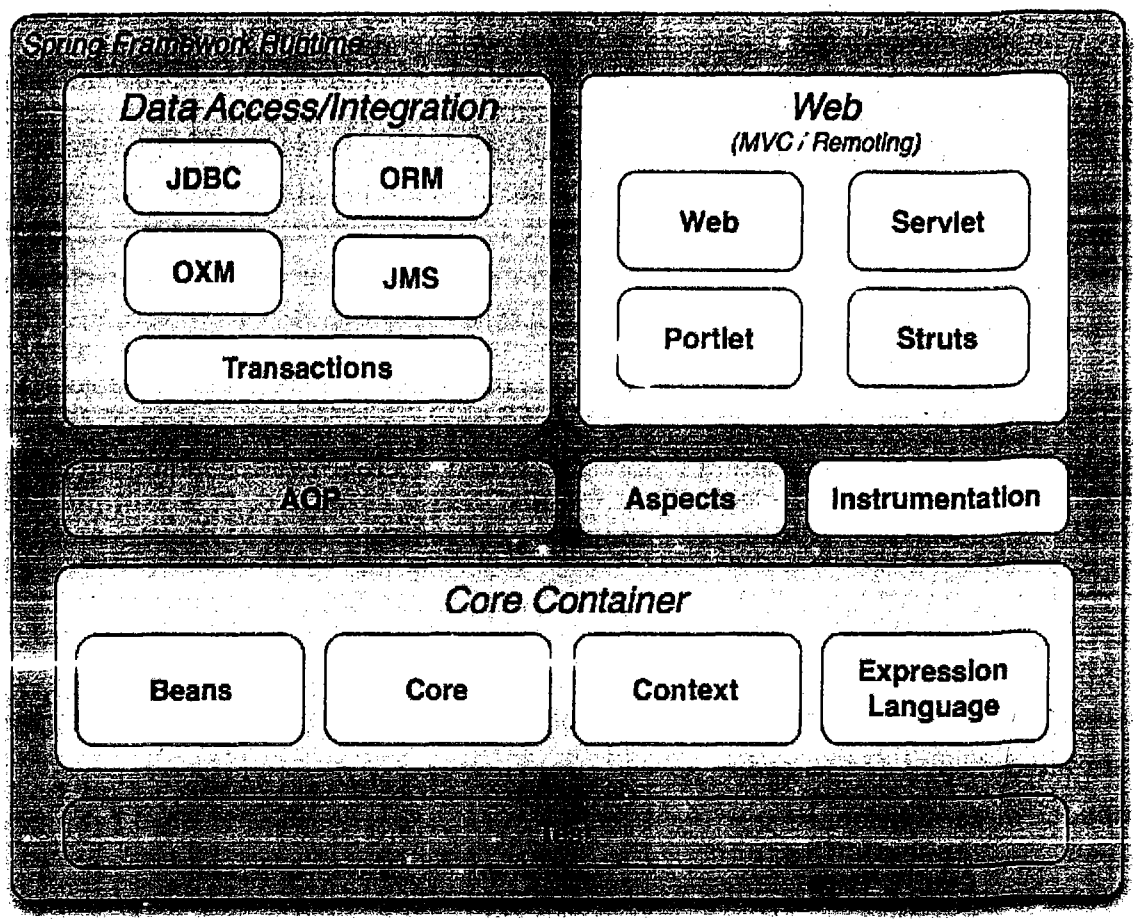
The following diagram represents the Spring Framework Architecture

Spring1.x Framework Architecture

Figure 1. The seven modules of the Spring framework



Spring 2. X Architecture



Spring 3.x Architecture

Spring is an open-source application framework, introduced and developed in 2004. The main ideas were suggested by an experienced J2EE architect, **Rod Johnson**.

He had earlier, written a book titled '**J2EE Develoment without using EJB**' and had introduced the concept of Light-weight container. Primarily, his argument is that while EJB has its merits, it is not always necessary and suitable in all applications.

Just as Hibernate attacks CMP as primitive ORM technology, Spring attacks EJB as unduly complicated and not susceptible to unit-testing. Instead of EJB, Spring suggests that we make use of ordinary Java beans, with some slight modifications, to get all the supposed advantages of EJB environment. Thus, Spring is posed as an alternative to EJB essentially. However, as a concession to the existing EJB investments, Spring is designed to operate with EJB if required.

Much of the philosophy and approach of Spring framework, however, predates , the latest EJB vesrion (ie) EJB-3, about to arrive shortly. It is said that EJB-3 has absorbed a number of new ideas suggested by Spring and some more, to answer the criticisms. There is a debate going on in the Java community about Spring and EJB-3. Spring is not a Persistence technolgy but a framework which allows plug in of other such technologies. But **EJB-3** is primarily focussed on Persistence Technology and has now incorporated Hibernate, the best ORM todate.Talks are going on to incorpotrate another equally nice ORM Technology known as JDO, which provides for Object Database also. Moreoveer, EJB-3 's Attribute-Orientaed Meta tags, help in vastly reducing the size of XML lines. Some have complained that Spring is still too dependent on XML files. In this tutorial, any reference to EJB, is only to EJB-2.My aim in presenting this tutorial is not to advocate the choice of either Spring or EJB-3, a topic still being devbated by experts but to share my perception with our readers

In spring, we can make use of plain Java Beans to achieve things that were previously possible with EJB only. The main aim of Spring is to simplify the J2EE development and testing.

EJB has been around since 1988 and is an established standard and specification in Enterprise world. Though there have been numerous Open-source Java technologies recently, no other technology has claimed to be superior to EJB, in its total features.

Rod Johnson, in his book, critices a number of features in EJB. Significantly, he has also spoken approvingly of some features of EJB and has recommended its use in some special circumstances.

EJB is a standard with wide Enterprise-level Industry support. It has been deployed in thousands of successful applications around the world. The EJB specification is a fixed target and so tool-developers are able to develop wizards, which can make ejb development quick and easy. There are many vendors for EJB application server like IBM (Web-Sphere), BEA (Weblogic), Oracle (JDeveloper) etc.

To quote from another book on Spring ("Spring in Action" - Craig Walls-Manning press) " **EJB is complex ",** as the author put it nicely**, "not for just being complex'. It is complex because it attempts to provide solutions for complex problems".** EJB is developed mainly for the remote transaction and distributed objects. But a number of enterprise projects do not have this level of complexity and still use EJBs and even the simple application becomes complex. In such cases Spring claims to be an alternative.

Since Spring comes with rich support to enterprise level services, it claims to be an alternative to EJB It is worthwhile to begin with a comparison of **EJB-2** and Spring, in some main features.

The main advantages of EJB are :

a) Transaction Management
b) Declarative Transaction support
c) Persistence (CMP & BMP)
d) Declarative Security
e) Distributed Computing (Container managed RPC)

Spring does not attempt to do everything by itself but supports the best of breed technologies for each of these requirements.

For example, instead of **CMP & BMP**, it supports several persistence technologies like **JDO, Hibernate and OJB**. These ORM toools are far more capable than the implementation in CMP.To simplify JDBC coding, there are tools like **iBatis** and Spring supports iBatis also.

Spring makes use of **Acegi**, an open-source Security framework and provides declarative security through spring configuration file or class **metadata** while in EJB declarative security is configured through deployment descriptor.

Spring provides proxying for **RMI (special remoting technologies like Burlap) JAX-RPC & web-service** while EJB provides container-managed remote method calls.

Spring can offer declarative transaction like EJB. But spring provides declarative rollback behavior also, for methods and exceptions

Thus, while EJB is monolithic and attempts to do many things, some tasks fairly well and some others not so well, Spring uses ordinary Java beans only and through special techniques provides many of the functionalities of EJB, by integrating with a number of open-source technologies.

Thereby, it gives the following advantages over **EJB2.**

a) testing is easier. We do not need to start the EJB container , for testing.

b) As Spring is based on the standard JavaBeans naming convention, programmers find it easy to work with.

c) It makes use of AOP(Aspect-Oriented Programming) which is a very recent and useful paradigm., in addition to classic OOP and preserves the purity of OOP.

d) It is flexible.

Spring's goal is to be an entire Application Framework. Other popular framewoks like **Struts, Tapestry, JSF** etc., are very good web tier frameworks but when we use these framework, we have to provide additional framework to deal with enterprise tier that integrates well with these framework. Spring tries to alleviate this problem by providing a comprehensive framework, which includes

a core bean container,
an MVC framework,
an AOP integration framework,
a JDBC integration framework and
an EJB integration framework.

It also provides integration modules for O/R mapping tools like Hibernate and JDO. Thus spring framework can be thought of as a layered architecture consisting of seven well defined modules.

The function of each component is as follows:

1. **Core Container**: The core container provides the fundamental functionality of Spring. It's primary component is the **'BeanFactory'**, an implementation of the Factory pattern. The BeanFactory applies the IOC pattern to separate an application's configuration and dependency specification from the actual application code.
2. **Spring Context/Application Context:** The Spring context is a configuration file that provides context information to the Spring framework . The Spring context supplies enterprise services such as **JNDI access, EJB integration, e-mail, internalization, validation, and scheduling functionality.**
3. **Spring AOP:(Aspect-Oriented)** The Spring AOP module integrates aspect-oriented programming functionality directly into the Spring framework, through its configuration management feature. As a result we 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 we can incorporate **declarative transaction management** into our applications without relying on EJB components. The Spring AOP module also introduces **metadata** programming to Spring. Using this we can add **annotation** to the source code that instructs Spring on where and how to apply aspects.
4. **Spring DAO:** The Spring's **JDBC and DAO** abstraction layer offers a meaningful exception hierarchy for managing the databaase connection, exception handling and error messages thrown by different database vendors. The exception hierarchy simplifies error handling and greatly reduces the amount of code that we need to write, such as opening and closing connections. This module also provide transaction management services for objects in a spring application.
5. **Spring ORM:** The Spring framework can be integrated to several ORM frameworks to provide Object Relational tool, including **JDO, Hibernate, OJB and iBatis SQL Maps.**
6. **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, JSF and webworks.** The Web module also eases the tasks of handling multipart requests and binding request parameters to domain objects.
7. **Spring MVC Framework:** The 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 and the generation of PDF and Excel Files.** I would venture to suggest that Spring will win sure acceptance among j2ee developers, very soon because of its ready-made adapters for various hot web-tier and presentation technologies.!

For example, there is a great variety of technologies in the web-tier like MVC PATTERN, STRUTS, JSF, WEB-WORK, JSP, TAPESTRY,FREEMARKER etc. Developers are now puzzled and confused about the relative merits and demerits of all these. Once they choose a technology and start implementing and later want to change over to another technology, it is very difficult. But, as Spring offers modules for aall the above technologies, it is most often simply changing the configuraion file. With this approach, it is even possible for a development team to try and test a given task in all the above forms and see the effect and performance before deciding the choice. Spring offers its own version of MVC architecture. It also offers adapters for Struts.

In the MVC adapter, it offers the following View choices.

JSP is default view template. **'InternalResouceViewResolver'** can be used for this purpose. Spring can be integrated with other template solutions like **Velocity, FreeMarker, tiles** etc., Also Spring can be used to produce dynamic binary **Excel** spreadsheet, **PDF** documents etc.,

To configure the velocity engine **'VelocityConfigurer'** bean is declared in spring configuration. The view resolver is configured by **'VelocityViewResolver'** bean.

To configure the **FreeMarker** engine **'FreeMarkerConfigurer'** bean is declared in spring configuration. The view resolver is configured by **'FreeMarkerViewResolver'** bean.

'TilesConfigurer' can used to used to load Tiles configuration file for rendering Tiles view. **'AbstractExcelView'** is used to generate Excel SpreadSheet as views. **'AbstactPdfView'** supports the creation of PDF as views. **'buildPdfDocument()'** is used to create PDF decument. Similarly we have **'buildExcelDocument()'** to create the excel document.

For delegation purpose, Spring provides **'DelegatingRequestProcessor'** and to use the tiles **'DelegatingTilesRequestProcessor'** is used. **'SpringTapestryEngine'** is used for integrating Tapestry to Spring. **'FacesSpringVariableResolver'** is used to resolve spring-managed beans in JSF.

Next we shall see the main concepts of Spring, **Inversion of Control (IoC) and Aspect Oriented Programming**. Spring is based on dependency injection type of IoC . We don't directly connect our components and services together **in code** but describe which services are needed by which components in a **configuration file**. A container is responsible for hooking it up. This concept is similar to **'Declarative Management'**. IOC is a broad concept. the two main types are

1. **Dependency Lookup:**

   The container provides callbacks to components and a lookup context. The managed objects are responsible for their other lookups. This is the **EJB Approach.** The Inversion of Control is limited to the Container involved callback methods that the code can use to obtain resources. Here we need to use JNDI to look up other EJBs and resources. Because of this reason EJB is not branded as 'IOC framework'. There are some problems in this implementation. The class **needs a application server environment** as it is dependent on JNDI and it is hard to test as we need to provide a dummy JNDI contest for testing purpose.

2. **Dependency Injection:**

   In this application objects is not responsible for looking up resources they depend on. Instead IoC container configures the object externalizing resource lookup from application code into the container. That is, **dependencies are injected into objects.** Thus lookups are completely removed from application objects and it can be used **outside the container also.** In this method, the objects can be populated via **Setter Injection** (Java-Beans properties) or **Constructor Injection** (constructor arguments). Each method has its own advantage and disadvantage.

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' by using the <property> subelement of <bean> tag 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.**

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 creatinf 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>
```

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" />
```

Aspect-Oriented Programming

Aspect Oriented programming is a new

programming technique that promotes separation of concerns within the system. System are composed of several components each responsible for a specific piece of functionality. Whatever may be the core function of the program, the system service like logging, transaction management, security etc., must be included in the program. These system services are commonly refered to as **'cross-cutting concerns'** as they tend to cut across multiple components in a system**. AOP** makes it possible to modularize and separate these services and then apply them declaratively to the components and we can focus on our own specific concerns. In spring, aspects are **wired** into objects in the spring XML file in the same way as JavaBean. This process is known as **'Weaving'**.

The container is at the core of Spring Container. In manages the life cycle and configuration of application objects. We can configure how each of our beans should be created either to create a single instance of bean or produce a new instance every time and how they should be associated with each other. Spring should not, however, be confused with traditionally heavyweight EJB containers, which are often large. The Spring actually comes with two distinct containers: Bean Factories-defined by **"org.springframework. beans.factory.BeanFactory"** are the simplest containers, providing support for dependency injection. Application contexts - defined by **"org.springframework.context.Application Context"** provides application framework services.

BEAN FACTORY:

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");

APPLICATION CONTEXT:

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.

   Spring is lightweight in terms of both size and overhead. The entire Spring framework can be distributed in a single JAR file that weighs just over **1.7 MB.** And the processing overhead required by Spring is negligible. Also Spring is nonintrusive:(ie) objects in a Spring-enabled application typically have no dependencies on Spring-specific classes.

   There are other lightweight containers like **HiveMind, Avalon, PicoContainer** etc., **Avalon** was one of the first IoC containers. Avalon mainly provides interface-dependent IoC. so, we much change our code in order to use a different container. This couples your code to a particular framework which is an undesirable feature.

   PicoContainer is a minimal (very small size nearly 50k) lightweight container that provides IoC in the form of constructor and setter injection. By using PicoContainer we can only assemble components programmatically through PicoContainer's API. But it allows only one instance of any particular type to be present in the registry. Also PicoContainer is only a container. It does not offer various special features as spring like integration.

   HiveMind is relatively new IoC container. Like PicoContainer, it focuses on wiring with support for both constructor and setter injections. It also allows us to define our configuration in an XML file. Like PicoContainer, HiveMind is only a container. It does not offer integration with other technology. Thus Spring makes it possible to configure and compose complex applications from simpler components. In Spring, application objects are composed declaratively, typically in an XML file. Spring also provides much infrastructure functionality

like transaction management, persistence framework integration, etc., leaving the development of application logic to us.

With this inroduction, we shall see a simple example in Spring in next article.

The **Spring Framework** comes in the form of ZIP file with the necessary jars, examples etc., The Spring framework can be downloaded from **http://www.springframework.org.** There will be two zip files one with dependencies and other without. The spring framework with dependencies is larger and includes all dependent libraries. Download **Spring framework 1.2 without dependency** and unzip on the hard disk as spring12

Inside the folder spring12 we can find 7 folders. The name and the contents of all the folders are given below,

1. **dist:** It contains various Spring distribution jar files.
2. **docs:** It contains general documentation and API javadocs.
3. **mock:** It contains mock JNDI contexts and a set of servlet API mock objects.
4. **samples:** It contains demo applications and skeletons.
5. **src:** It contains the Java source files for the framework.
6. **test:** It contains the Java source files for Spring's test suite.
7. **tiger:** It contains JDK1.5 examples and test suite.

Inside the "dist" directory, we can find all the jar files necessary for compiling and executing the program. The jar files and its contents are listed below:

1. **spring.jar :** It contains the entire Spring framework including everything in the other JAR files also.
2. **spring-core.jar** : It contains the core Spring container and its utilities.
3. **spring-beans.jar :** It contains the bean Spring container and JavaBeans support utilities.
4. **spring-aop.jar :** It contains Spring's AOP framework, source-level metadata support, AOP Alliance interfaces etc.,
5. **spring-context.jar :** It contains application context, validation framework, JNDI, templating support and scheduling.
6. **spring-dao.jar :** It contains DAO support and transaction infrastructure.
7. **spring-jdbc.jar :** It contains the JDBC support.
8. **spring-support.jar :** It contains JMX support, JCA support, scheduling support, mail support and caching support.
9. **spring-web.jar :** It contains the web application context, multipart resolver, Struts support, JSF support and web utilities.
10. **spring-webmvc.jar :** It contains the framework servlets, web MVC framework, web controllers and web views.
11. **spring-remoting.jar :**It contains remoting support, EJB support and JMS support.
12. **spring-orm.jar :** It contains iBATIS SQL Maps support, Apache OJB support, TopLink support and JDO support.
13. **spring-hibernate.jar :** It contains Hibernate 2.1 support, Hibernate 3.x support.
14. **spring-mock.jar :** It contains JNDI mocks, Servlet API mocks and JUnit support.

Now we will run a greeter example in Spring.

f:\>md springdemo

f:\>cd springdemo

As the entire Spring Framework is included in spring.jar. We use that to run our examples. Copy **spring.jar** from spring12\dist folder to the working folder. Also copy **commons-logging.jar** from tomcat41\server\lib to the working directory.

f:\springdemo>set path=c:\windows\command;**g:\jdk1.5\bin**
(* Set path for jdk1.5 or jdk1.4.2 only.)
f:\springdemo>set classpath=f:**springdemo;**
f:\springdemo**spring.jar;**

f:\springdemo**commons-logging.jar**

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.

Edit
1. hello.java
2. helloimpl.java
3. hello.xml
4. helloclient.java

----------f:\springdemo\hello.java----------

```
public interface hello
{
  public String sayhello(String a);
}
```

----------f:\springdemo\helloimpl.java----------

```
public class helloimpl implements hello
{
    private String greeting;
    public helloimpl()
    {
    }
    public helloimpl(String a)
    {
    greeting=a;
    }
    public String sayhello(String s)
    {
        return greeting+s;
    }
    public void setGreeting(String a)
    {
```

```
        greeting=a;
    }
}
```

----------f:\springdemo\hello.xml----------

```
 <?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="hello"
      class="helloimpl">
  <property name="greeting">
     <value>Good Morning!...</value>
  </property>
  </bean>
</beans>
```

----------f:\springdemo\helloclient.java----------

```
import java.io.*;
import org.springframework.beans.factory.*;
import org.springframework.beans.factory.xml.*;
import org.springframework.core.io.*;
public class helloclient
{
  public static void main(String args[])  throws Exception
  {
    try
    {
      System.out.println("please Wait.");
      Resource    res = new  ClassPathResource("hello.xml");
      BeanFactory  factory = new  XmlBeanFactory(res);
      hello bean1 = (hello)factory.getBean("hello");
     String s = bean1.sayhello(args[0]);
     System.out.println(s);
    }
    catch(Exception e1)
        { System.out.println(""+e1); }
  }
}
```

To run:

```
f:\springdemo>javac hello.java
```

```
f:\springdemo>javac helloimpl.java
f:\springdemo>javac helloclient.java
f:\springdemo>java helloclient "sam"
```

We will get the output as follows:

```
please wait..
Aug 5, 2002 2:10:56 AM org.springframework.
beans.factory.xml.XmlBeanDefinitionReader
loadBeanDefinitions

INFO: Loading XML bean definitions from
class path resource [hello.xml]
Aug 5, 2002 2:10:57 AM org.springframework.
beans.factory.support.
AbstractBeanFactorygetBean

INFO: Creating shared instance of singleton
bean 'hello'
Good Morning!...sam
```

Thus we have run our first Spring program. Here helloimpl implements the hello interface. Although it is not necessary to hide the implementation behind an interface, it is recommended as a way to seperate implementation from interface. helloimpl has a single property greeting. This property can be set in two different ways: by property's setter method or by the constructor. The XML file hello.xml declares the instance of helloimpl.java in the spring container and configures its property greeting with the value 'Good Morning!...'

The root of the hello.xml file is the <beans> element, which is the root element of any Spring configuration file. The <bean> element is used to tell the Spring container about the class and how it should be configured. Here, the id attribute takes the interface name and the class attribute specifies the bean's fully qualified class name.

Within the <bean> element, the <property> element is used to set the property, in this case the greeting property. By using <property>, we're telling the Spring container to call setGreeting() while setting the property. The value of the greeting is defined within the <value> element. Here we have given **'Good Morning!...'** as the value.

The container instantiates the 'helloimpl' based on the XML definition as,

```
helloimpl hello = new helloimpl();
helloimpl.setGreeting("Good Morning!...");
```

Similarly, greeting property may be set through the single argument constructor of 'helloimpl' as,

```
<bean id="hello" class="helloimpl">
<constructor-arg>
<value>Good Morning!...</value>
</constructor-arg>
</bean>
```

Now the container instantiates the 'helloimpl' based on the XML definition as,

```
helloimpl hello = new
helloimpl("Good Morning...");
```

In the client program, **'BeanFactory'** class is used which is the Spring container. Then the 'getBean()' method is called to get the reference to the 'hello'. With this reference, sayhello() method is called.

We can also use a frame client. The code is very much similar to console client except the GUI code.

----------f:\springdemo\helloframe.java----------

```
import java.io.*;
import java.awt.*;
import org.springframework.beans.factory.*;
import org.springframework.beans.factory.xml.*;
import org.springframework.core.io.*;
public class helloframe  extends Frame
{
TextField    text1;
TextArea     area1;
Label        label1;
Button       button1;
  public static void main(String args[])
  {
    helloframe   app = new helloframe();
    app.setSize(700,500);
    app.setVisible(true);
  }
  helloframe()
  {
    setLayout(new FlowLayout());
    setBackground(Color.green);
    label1=new Label("Type Name: ");
    text1=new TextField(25);
    area1=new TextArea(10,50);
    button1=new Button("Exit");
    button1.setBackground(Color.red);
    add(label1);
    add(text1);
    add(area1);
    add(button1);
  }
  public boolean action (Event e,Object c)
  {
    if(e.target==text1)
    {
      try
      {
```

```
        area1.append("Please Wait..\n");
        Resource res = new ClassPathResource("hello.xml");
        BeanFactory factory = new XmlBeanFactory(res);
        hello bean1 = (hello)factory.getBean("hello");
        String s = bean1.sayhello(text1.getText());
        area1.append(s);
      }
      catch(Exception e1)
            {area1.append(""+e1);}
    }
    if(e.target==button1)
    {
      System.exit(0);
    }
    return true;
  }
```

Then compile and run the frame client program. We will a textbox, a exit button and a text area. Type a name (say 'tom') in text area and press enter. **'Good Morning!... tom'** will appear in the text area

Next we shall see how to run this program as a servlet. Consider Tomcat-5 is installed in g drive.

First copy g:\spring12\spring.jar to g:\tomcat5\common\lib and start tomcat server.
Then set classpath as shown below and edit the servletclient.java.

f:\springdemo>set classpath=f:**springdemo;**

f:\springdemo**spring.jar;**

f:\springdemo**commons-logging.jar;**

g:\tomcat5\common\lib\servlet-api.jar

----------f:\springdemo\servletclient.java ----------

```
import java.io.*;
import org.springframework.beans.factory.*;
import org.springframework.beans.factory.xml.*;
import org.springframework.core.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class servletclient extends HttpServlet
{
  public void doPost(HttpServletRequest req,HttpServletResponse resp)
      throws ServletException,IOException

  {
    resp.setContentType("text/html");
    PrintWriter out =resp.getWriter();
```

```
    String a = req.getParameter("text1");
    try
    {
      System.out.println("Please wait.");

      Resource res = new ClassPathResource("hello.xml");
      System.out.println("Resource ok");

      BeanFactory factory = new XmlBeanFactory(res);
      System.out.println("BeanFactory ok");

      hello bean1 = (hello)factory.getBean("hello");

      String s = bean1.sayhello(a);
      out.println(s);
    }
    catch(Exception e1)
         {System.out.println(""+e1);}
  }
}
```

----------f:\springdemo\servletclient.htm----------

```
<html>
<body>
<form    method=post
       action="http://localhost:8080/
                servlet/servletclient">
  <input type=text name="text1">
  <input type=submit>
</form>
</body>
</html>
```

Then compile the servlet and copy all the class files ie., hello.class, helloimpl.class, servletclient.class and the xml file hello.xml to g:\tomcat5\webapps\root\web-inf\classes. Copy html file servletclient.htm to g:\tomcat5\webapps\root. Add entry to web.xml file.
Restart Tomcat server and open browser and type url as http://localhost:8080/servletclient.htm. We will get a text box and a button. Type a name (say 'tom') and click the 'submit' button.

Good Morning!... tom will appear

In the next article, we shall see how to contact the database from Spring.

IOC Container

The org.springframework.beans and org.springframework.context packages provide the basis for the Spring Framework's IoC container. The BeanFactory interface provides an advanced configuration mechanism capable of managing objects of any nature. The ApplicationContext interface builds on top of the BeanFactory (it is a sub-interface) and adds other functionality such as easier integration with Spring's AOP features, message resource handling (for use in internationalization), event propagation, and application-layer specific contexts such as the WebApplicationContext for use in web applications.

In short, the BeanFactory provides the configuration framework and basic functionality, while the ApplicationContext adds more enterprise-centric functionality to it. The ApplicationContext is a complete superset of the BeanFactory, and any description of BeanFactory capabilities and behavior is to be considered to apply to the ApplicationContext as well.

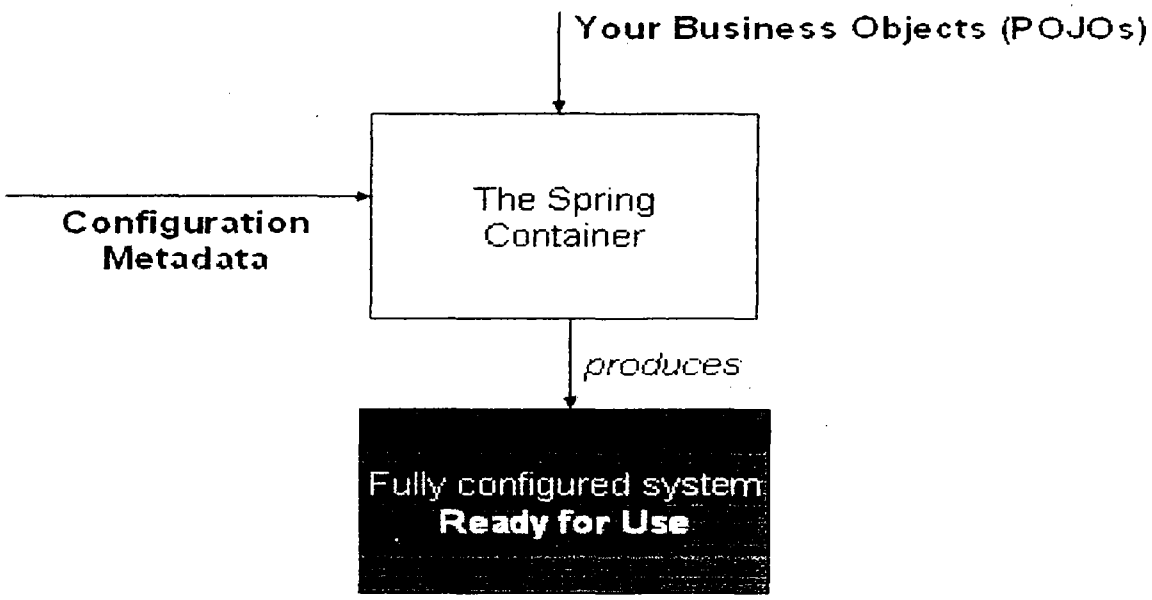
In Spring, those objects that form the backbone of your application and that are managed by the Spring IoC container are referred to as beans. A bean is simply an object that is instantiated, assembled and otherwise managed by a Spring IoC container; other than that, there is nothing special about a bean (it is in all other respects one of probably many objects in your application). These beans, and the dependencies between them, are reflected in the configuration metadata used by a container.

The org.springframework.beans.factory.BeanFactory is the actual representation of the Spring IoC container that is responsible for containing and otherwise managing the aforementioned beans. The BeanFactory interface is the central IoC container interface in Spring. Its responsibilities include instantiating or sourcing application objects, configuring such objects, and assembling the dependencies between these objects.

There are a number of implementations of the BeanFactory interface that come supplied straight out-of-the-box with Spring. The most commonly used BeanFactory implementation is the XmlBeanFactory class. This implementation allows you to express the objects that compose your application, and the doubtless rich interdependencies between such objects, in terms of XML. The XmlBeanFactory takes this XML configuration metadata and uses it to create a fully configured system or application.

The Spring IoC container consumes some form of configuration metadata; this configuration metadata is nothing more than how you (as an application developer) inform the Spring container as to how to "instantiate, configure, and assemble [the objects in your application]". This configuration metadata is typically supplied in a simple and intuitive XML format. When using XML-based configuration metadata, you write bean definitions for those beans that you want the Spring IoC container to manage, and then let the container do its stuff.

XML-based metadata is by far the most commonly used form of configuration metadata. It is **not** however the only form of configuration metadata that is allowed. Spring configuration consists of at least one bean definition that the container must manage, but typically there will be more than one bean definition. When using XML-based configuration metadata, these beans are configured as <bean/> elements inside a top-level <beans/> element. These bean definitions correspond to the actual objects that make up your application.



The Spring IoC container

Example of the basic structure of XML-based configuration metadata.

```
<?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-2.5.xsd">

        <bean id = "..."  class = "...">
                <!-- collaborators and configuration for this bean go here -->
        </bean>

        <!-- more bean definitions go here -->
</beans>
```

Beans

A Spring IoC container manages one or more beans. These beans are created using the configuration metadata that has been supplied to the container (typically in the form of XML <bean/> definitions). Within the container itself, these bean definitions are represented as BeanDefinition objects, which contain (among other information) the following metadata:

- a package-qualified class name: typically this is the actual implementation class of the bean being defined.

- bean behavioral configuration elements, which state how the bean should behave in the container (scope, lifecycle callbacks, and so forth).
- references to other beans which are needed for the bean to do its work; these references are also called collaborators or dependencies.
- other configuration settings to set in the newly created object. An example would be the number of connections to use in a bean that manages a connection pool, or the size limit of the pool.

Naming beans - Every bean has one or more ids (also called identifiers, or names; these terms refer to the same thing). These ids must be unique within the container the bean is hosted in. A bean will almost always have only one id, but if a bean has more than one id, the extra ones can essentially be considered aliases. When using XML-based configuration metadata, you use the 'id' or 'name' attributes to specify the bean identifier(s). The 'id' attribute allows you to specify exactly one id, and as it is a real XML element ID attribute, the XML parser is able to do some extra validation when other elements reference the id; as such, it is the preferred way to specify a bean id. However, the XML specification does limit the characters which are legal in XML IDs. If you have a need to use one of these special XML characters, or want to introduce other aliases to the bean, you may also or instead specify one or more bean ids, separated by a comma (,), semicolon (;), or whitespace in the 'name' attribute.

Aliasing beans - In a bean definition itself, you may supply more than one name for the bean, by using a combination of up to one name specified via the id attribute, and any number of other names via the name attribute. All these names can be considered equivalent aliases to the same bean, and are useful for some situations, such as allowing each component used in an application to refer to a common dependency using a bean name that is specific to that component itself.

Consider the case where component A defines a DataSource bean called componentA-dataSource, in its XML fragment. Component B would however like to refer to the DataSource as componentB-dataSource in its XML fragment. And the main application, MyApp, defines its own XML fragment and assembles the final application context from all three fragments, and would like to refer to the DataSource as myApp-dataSource. This scenario can be easily handled by adding to the MyApp XML fragment the following standalone aliases:

```
<alias name="componentA-dataSource" alias="componentB-dataSource"/>

<alias name="componentA-dataSource" alias="myApp-dataSource" />
```

Instantiating beans - A bean definition essentially is a recipe for creating one or more objects. The container looks at the recipe for a named bean when asked, and uses the configuration metadata encapsulated by that bean definition to create (or acquire) an actual object.

- Using Constructor

```
<bean id="exampleBean" class="examples.ExampleBean"/>

<bean name="anotherExample" class="examples.ExampleBeanTwo"/>
```

- Using Static Factory method

```
<bean id="exampleBean" class="examples.ExampleBean" factory-method="createInstance"/>
```

- Using Instance Factory method

```
<!-- the factory bean, which contains a method called createInstance() -->

<bean id="serviceLocator" class=" DefaultServiceLocator" />


<!-- the bean to be created via the factory bean -->
```

```
<bean id="exampleBean"
      factory-bean="serviceLocator"
      factory-method="createInstance"/>
```

- Using Container

```
Resource res = new FileSystemResource("beans.xml");
BeanFactory factory = new XmlBeanFactory(res);
```

Dependencies

The basic principle behind **Dependency Injection** (DI) is that objects define their dependencies (that is to say the other objects they work with) only through constructor arguments, arguments to a factory method, or properties which are set on the object instance after it has been constructed or returned from a factory method. Then, it is the job of the container to actually inject those dependencies when it creates the bean.

This is fundamentally the inverse, hence the name **Inversion of Control** (IoC), of the bean itself being in control of instantiating or locating its dependencies on its own using direct construction of classes, or something like the Service Locator pattern. It becomes evident upon usage that code gets much cleaner when the DI principle is applied, and reaching a higher grade of decoupling is much easier when objects do not look up their dependencies. DI exists in two major variants, namely Constructor Injection and Setter Injection.

Constructor Injection

```
package x.y;
public class Foo {
    public Foo(Bar bar, Baz baz) {
        // ...
    }
}
```

```
<beans>
    <bean name="foo" class="x.y.Foo">
        <constructor-arg>
            <bean class="x.y.Bar"/>
        </constructor-arg>
        <constructor-arg>
            <bean class="x.y.Baz"/>
        </constructor-arg>
    </bean>
</beans>
```

```
package examples;
public class ExampleBean
{
        private int years;
        private String ultimateAnswer;

        public ExampleBean(int years, St ing ultimateAnswer) {
                this.years = years;
                this.ultimateAnswer = ultimateAnswer;
        }
}
```

Constructor Argument Type Matching

```
<bean id="exampleBean" class="examples.ExampleBean">
        <constructor-arg type="int" value="7500000"/>
        <constructor-arg type="java.lang.String" value="42"/>
</bean>
```

Constructor Argument Index

```
<bean id="exampleBean" class="examples.ExampleBean">
        <constructor-arg index="0" value="7500000"/>
        <constructor-arg index="1" value="42"/>
</bean>
```

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.

How Injection works?

- The BeanFactory is created and initialized with a configuration which describes all the beans. (Most Spring users use a BeanFactory or ApplicationContext implementation that supports XML format configuration files.)

- Each bean has dependencies expressed in the form of properties, constructor arguments, or arguments to the static-factory method when that is used instead of a normal constructor. These dependencies will be provided to the bean, when the bean is actually created.
- Each property or constructor argument is either an actual definition of the value to set, or a reference to another bean in the container.
- Each property or constructor argument which is a value must be able to be converted from whatever format it was specified in, to the actual type of that property or constructor

argument. By default Spring can convert a value supplied in string format to all built-in types, such as int, long, String, boolean, etc.

The Spring container validates the configuration of each bean as the container is created, including the validation those properties which are bean references are actually referring to valid beans. However, the bean properties themselves are not set until the bean is actually created. For those beans that are singleton-scoped and set to be pre-instantiated (such as singleton beans in an ApplicationContext), creation happens at the time that the container is created, but otherwise this is only when the bean is requested. When a bean actually has to be created, this will potentially cause a graph of other beans to be created, as its dependencies and its dependencies' dependencies (and so on) are created and assigned.

Inner beans

A <bean/> element inside the <property/> or <constructor-arg/> elements is used to define a so-called inner bean. An inner bean definition does not need to have any id or name defined, and it is best not to even specify any id or name value because the id or name value simply will be ignored by the container.

```
<bean id="outer" class="...">
    <!-- instead of using a reference to a target bean, simply define the target bean inline -->
    <property name="target">
        <bean class="com.example.Person"> <!-- this is the inner bean -->
            <property name="name" value="Fiona Apple"/>
            <property name="age" value="25"/>
        </bean>
    </property>
</bean>
```

Note that in the specific case of inner beans, the 'scope' flag and any 'id' or 'name' attribute are effectively ignored. Inner beans are always anonymous and they are always scoped as prototypes. Please also note that it is not possible to inject inner beans into collaborating beans other than the enclosing bean.

Compound property names

Compound or nested property names are perfectly legal when setting bean properties, as long as all components of the path except the final property name are not null.

```
<bean id="foo" class="foo.Bar">
    <property name="x.y.z" value="123" />
</bean>
```

The foo bean has a x property which has a y property, which has a z property, and that final z property is being set to the value 123. In order for this to work, the x property of foo, and the y property of x must **not** be null after the bean is constructed, or a NullPointerException will be thrown.

depends-on

For most situations, the fact that a bean is a dependency of another is expressed by the fact that one bean is set as a property of another. This is typically accomplished with the <ref/> element in XML-based configuration metadata. For the relatively infrequent situations where dependencies between beans are less direct (for example, when a static initializer in a class needs to be triggered, such as database driver registration), the 'depends-on' attribute may be used to

explicitly force one or more beans to be initialized before the bean using this element is initialized. Find below an example of using the 'depends-on' attribute to express a dependency on a single bean.

```
<bean id="beanOne" class="ExampleBean" depends-on="manager"/>

<bean id="manager" class="ManagerBean" />
```

If you need to express a dependency on multiple beans, you can supply a list of bean names as the value of the 'depends-on' attribute, with commas, whitespace and semicolons all valid delimiters, like so:

```
<bean id="beanOne" class="ExampleBean" depends-on="manager,accountDao">
    <property name="manager" ref="manager" />
</bean>

<bean id="manager" class="ManagerBean" />

<bean id="accountDao" class="x.y.jdbc.JdbcAccountDao" />
```

The 'depends-on' attribute and property is used not only to specify an initialization time dependency, but also to specify the corresponding destroy time dependency (in the case of singleton beans only). Dependent beans that are defined in the 'depends-on' attribute will be destroyed first prior to the relevant bean itself being destroyed. This thus allows you to control shutdown order too.

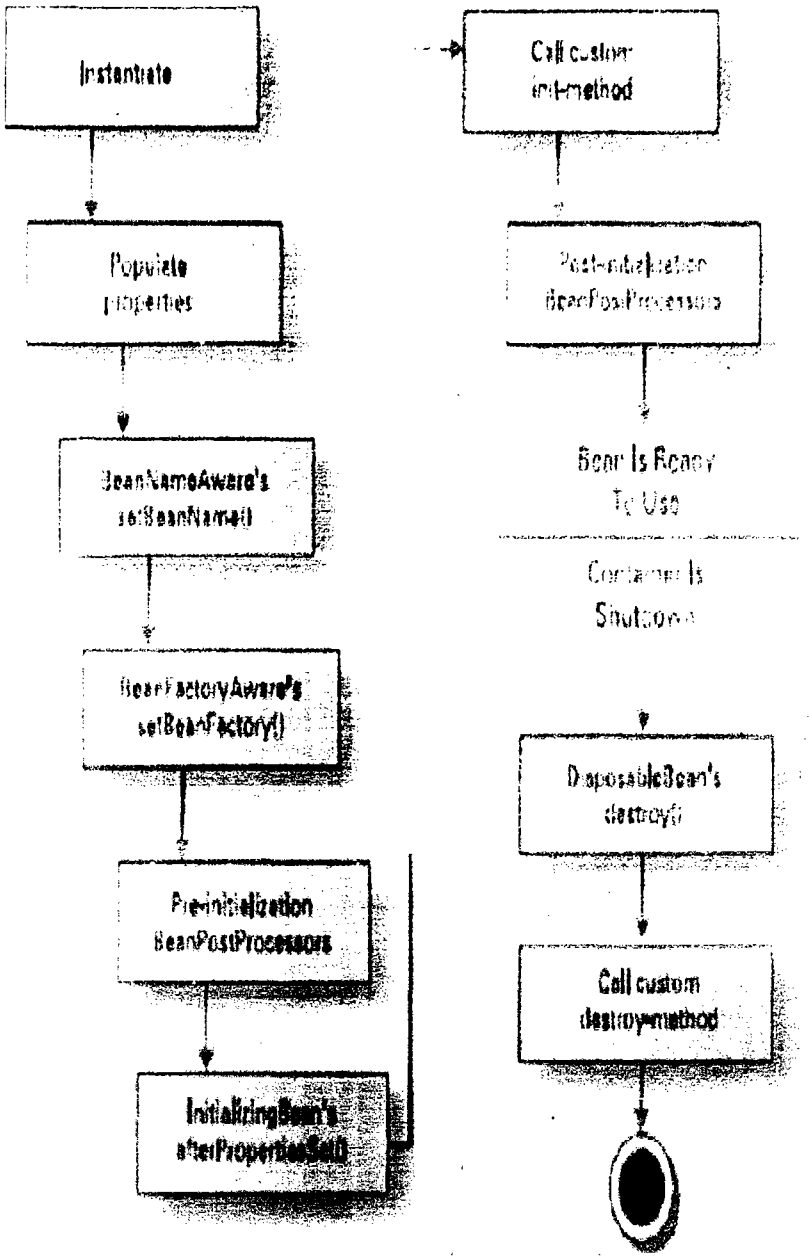


Figure 2.1 The life cycle of a bean within a Spring bean factory container

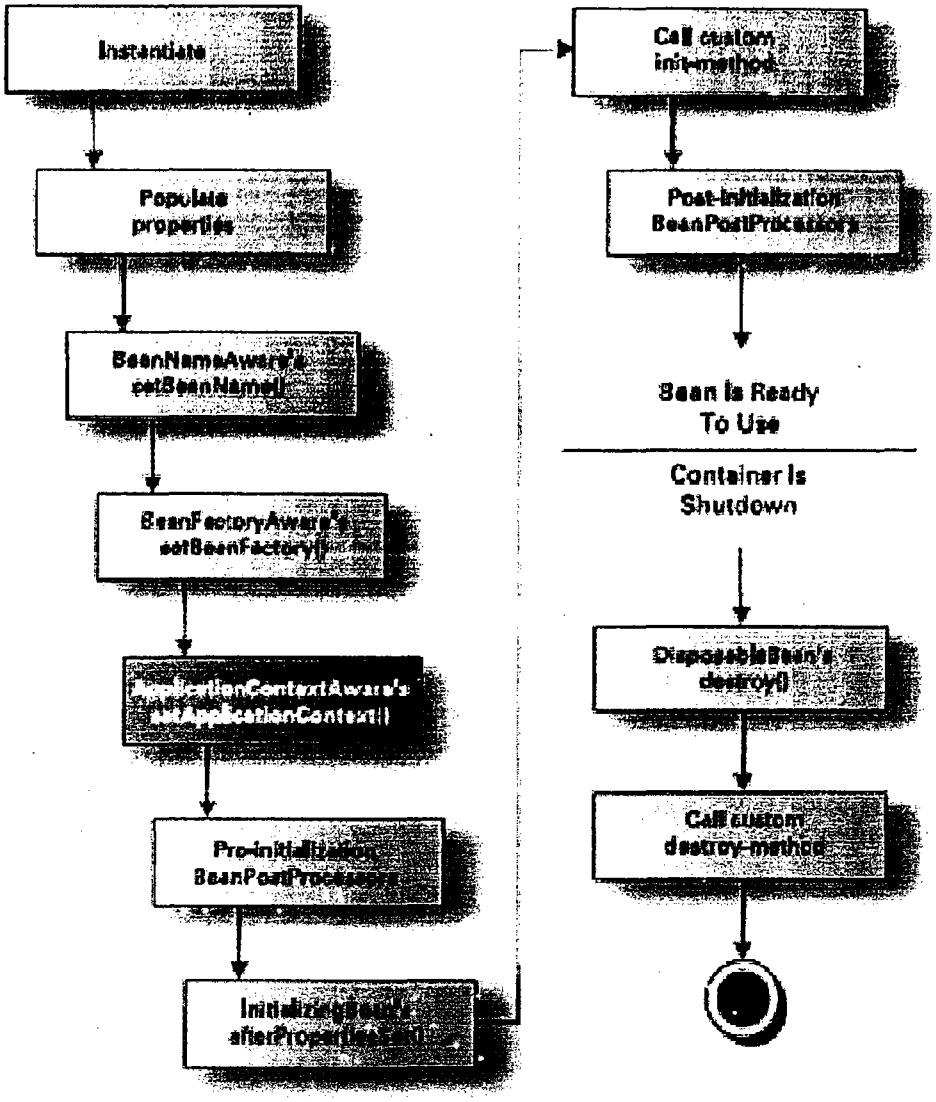


Figure 2.2 The life cycle of a bean in a Spring application context

```
1  ================================================================================
2  Monolithic Solution
3  ================================================================================
4  public class Calculate {
5
6      private void showResult(String result) {
7          System.out.println(result);
8      }
9
10     private long operate(long op1, long op2) {
11         return op1 + op2;
12     }
13
14     private String getOpsName() {
15         return " plus ";
16     }
17
18     public void execute(String [] args) {
19         long op1 = Long.parseLong(args[0]);
20         long op2 = Long.parseLong(args[1]);
21         showResult("The result of " + op1 +
22                 getOpsName() + op2 + " is "
23                 + operate(op1, op2) + "!");
24
25     }
26
27     public static void main(String[] args) {
28         Calculate calc = new Calculate();
29         calc.execute(args);
30     }
31 }
32
33
34 ================================================================================
35 Modularized Solution
36 ================================================================================
37 public interface Operation {
38
39     long operate(long op1, long op2);
40
41     String getOpsName();
42 }
43 ====================================
44 public class OpAdd implements Operation {
45
46     public OpAdd() {}
47
48     public String getOpsName() {
49         return " plus ";
50     }
51
52     public long operate(long op1, long op2) {
53         return op1 + op2;
54     }
55
56 }
57 ====================================
58 public class OpMultiply implements Operation {
59
60     public OpMultiply() {}
61
```

```
62        public String getOpsName() {
63            return " times ";
64        }
65
66        public long operate(long op1, long op2) {
67            return op1 * op2;
68        }
69    }
70    ====================================
71    public interface ResultWriter {
72
73        void showResult(String result) ;
74
75    }
76    ====================================
77    public class ScreenWriter implements ResultWriter{
78
79        public ScreenWriter() {}
80
81        public void showResult(String result) {
82            System.out.println(result);
83        }
84    }
85    ====================================
86    import java.io.BufferedWriter;
87    import java.io.File;
88    import java.io.FileWriter;
89    import java.io.PrintWriter;
90
91    public class DataFileWriter implements ResultWriter {
92
93        public DataFileWriter() {}
94
95        public void showResult(String result) {
96            File file = new File("/output.txt");
97            try {
98                PrintWriter fwriter = new PrintWriter(
99                    new BufferedWriter(new FileWriter(file)));
100               fwriter.println(result);
101               fwriter.close();
102
103           }
104           catch (Exception ex) {
105               ex.printStackTrace();
106           }
107       }
108   }
109   ====================================
110   public class CalculateScreen {
111
112       private Operation ops = new OpAdd();
113       private ResultWriter wtr = new ScreenWriter();
114
115       public static void main(String[] args) {
116           CalculateScreen calc = new CalculateScreen();
117           calc.execute(args);
118       }
119
120       public void execute(String [] args)  {
121           long op1 = Long.parseLong(args[0]);
122           long op2 = Long.parseLong(args[1]);
```

```
123             wtr.showResult("The result of " + op1 +
124                     ops.getOpsName() + op2 + " is "
125                     + ops.operate(op1, op2) + "!");
126
127         }
128
129 }
130 ====================================
131 public class CalculateMultFile {
132
133         private Operation ops = new OpMultiply();
134         private ResultWriter wtr = new DataFileWriter();
135
136         public static void main(String[] args) {
137             CalculateMultFile calc = new CalculateMultFile();
138             calc.execute(args);
139         }
140
141         public void execute(String [] args)  {
142             long op1 = Long.parseLong(args[0]);
143             long op2 = Long.parseLong(args[1]);
144             wtr.showResult("The result of " + op1 +
145                     ops.getOpsName() + op2 + " is "
146                     + ops.operate(op1, op2) + "!");
147
148         }
149 }
150 ================================================================================
151 Spring Solution
152 ================================================================================
153 public interface Operation {
154
155         long operate(long op1, long op2);
156
157         String getOpsName();
158 }
159 ====================================
160 public class OpAdd implements Operation {
161
162         public OpAdd() {}
163
164         public String getOpsName() {
165             return " plus ";
166         }
167
168         public long operate(long op1, long op2) {
169             return op1 + op2;
170         }
171 }
172 ====================================
173 public class OpMultiply implements Operation {
174
175         public OpMultiply() {}
176
177         public String getOpsName() {
178             return " times ";
179         }
180
181         public long operate(long op1, long op2) {
182             return op1 * op2;
183         }
```

```
184 }
185 ====================================
186 public interface ResultWriter {
187
188       void showResult(String result) ;
189
190 }
191 ====================================
192 public class ScreenWriter implements ResultWriter{
193
194       public ScreenWriter() {}
195
196       public void showResult(String result) {
197             System.out.println(result);
198       }
199 }
200 ====================================
201 import java.io.*;
202
203 public class DataFileWriter implements ResultWriter {
204
205       public DataFileWriter() {}
206
207       public void showResult(String result) {
208
209             File file = new File("/output.txt");
210
211             try
212             {
213                   PrintWriter fwriter = new PrintWriter(
214                         new BufferedWriter(new FileWriter(file)));
215                   fwriter.println(result);
216                   fwriter.close();
217             }
218             catch (Exception ex) {
219                   ex.printStackTrace();
220             }
221       }
222 }
223 ====================================
224 public class CalculateSpring
225 {
226       private Operation ops;
227       private ResultWriter wtr;
228
229       public void setOps(Operation ops) {
230             this.ops = ops;
231       }
232
233       public void setWriter(ResultWriter writer) {
234             wtr = writer;
235       }
236
237       public static void main(String[] args) {
238             ApplicationContext context =
239                         new ClassPathXmlApplicationContext(
240                                     "beans.xml");
241
242             BeanFactory factory = (BeanFactory) context;
243             CalculateSpring calc =
244                   (CalculateSpring) factory.getBean("opsbean");
```

```
245
246             calc.execute(args);
247         }
248
249         public void execute(String [] args)  {
250
251             long op1 = Long.parseLong(args[0]);
252             long op2 = Long.parseLong(args[1]);
253             wtr.showResult("The result of " + op1 +
254                     ops.getOpsName() + op2 + " is "
255                     + ops.operate(op1, op2) + "!");
256
257         }
258 }
259 ================================================================================
260 beans.xml
261 ================================================================================
262 <?xml version="1.0" encoding="UTF-8"?>
263
264 <beans xmlns="http://www.springframework.org/schema/beans"
265         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
266         xsi:schemaLocation="http://www.springframework.org/schema/beans
267             http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
268
269         <bean id = "screen" class = "ScreenWriter" />
270         <bean id = "multiply" class = "OpMultiply" />
271         <bean id = "add" class = "OpAdd" />
272
273         <bean id = "opsbean" class = "CalculateSpring">
274             <property name = "ops" ref = "add" />
275             <property name = "writer" ref = "screen"/>
276         </bean>
277
278 </beans>
279 ================================================================================
```

```
1  ================================================================================
2  App 1)>>>>>>>>>> Spring core module application that demonstrates setter injection<<<<<<<<<
3  ================================================================================
4  ----------------------------------Demo.java-------------------------------
5  public interface Demo
6  {
7      public String generateWishMsg(String name);
8  }
9  ----------------------------------DemoBean.java-------------------------
10 import java.util.*;
11
12 public class DemoBean implements Demo
13 {
14       private String msg;
15
16       public void setMsg(String msg)
17       {
18            this.msg=msg;
19       }
20       public String generateWishMsg(String name)
21       {
22         Calendar cl=Calendar.getInstance();
23         int h=cl.get(Calendar.HOUR_OF_DAY);
24
25         if(h<=12)
26            return msg+"   good morning    :"+name;
27         else if(h<=16)
28            return msg+" Good afternoon   :"+name;
29         else if(h<=20)
30            return msg+"   Good Evening    :"+name;
31         else
32            return msg+"   Good Night    :"+name;
33       }
34 }
35 ----------------------------------DemoCfg.xml-----------------------------------
36 <!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
37              "http://www.springframework.org/dtd/spring-beans-2.0.dtd">
38 <beans>
39   <bean id="db" class="DemoBean">
40        <property name="msg">
41                <value>hai</value>
42        </property>
43    </bean>
44 </beans>
45 ----------------------------------------DemoClient.java-----------------------------
46 import org.springframework.core.io.FileSystemResource;
47 import org.springframework.beans.factory.xml.XmlBeanFactory;
48
49
50 public class DemoClient
51 {
52       public static void main(String args[])
53       {
54            FileSystemResource res=new FileSystemResource("DemoCfg.xml");
55            XmlBeanFactory factory=new XmlBeanFactory(res);
56            Object obj=factory.getBean("db");
57            DemoInterface bobjref=(DemoInterface)obj;
58            System.out.println(bobjref.generateWishMsg("Nataraz"));
59       }
60 }
61 Note:- jar files in classpath:
```

```
  1.spring.jar
  2.commons-logging.jar

==========================================================================
App 2) >>>>>>>Spring core module application that demonstrates constructor injection<<<<<<
==========================================================================
----------------------------Demo.java-------------------------------------

public interface Demo
{
  public String generateWishMsg(String name);
}
-----------------------------------DemoBean.java----------------------------
import java.util.*;

public class DemoBean implements Demo
{
      private String msg;

      public DemoBean(String msg)
      {
            System.out.println("Demobean:1-paramconstructor");
            this.msg=msg;
      }
      public String generateWishMsg(String name)
      {
         Calendar cl=Calendar.getInstance();
         int h=cl.get(Calendar.HOUR_OF_DAY);

        if(h<=12)
            return msg+"  good morning   :"+name;
        else if(h<=16)
            return msg+" Good afternoon  :"+name;
        else if(h<=20)
            return msg+"  Good Evening   :"+name;
        else
            return msg+"  Good Night    :"+name;
      }
}

-------------------------------------------DemoCfg.xml-----------------------------------

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
              "http://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>
  <bean id="db" class="DemoBean">
      <constructor-arg value="raja"/>
  </bean>
</beans>
-------------------------------------------DemoClient.java-------------------------------
import org.springframework.core.io.FileSystemResource;
import org.springframework.beans.factory.xml.XmlBeanFactory;


public class DemoClient
{
      public static void main(String args[])
      {
            FileSystemResource res=new FileSystemResource("DemoCfg.xml");
            XmlBeanFactory factory=new XmlBeanFactory(res);
```

```
            Object obj=factory.getBean("db");
            DemoInterface bobjref=(DemoInterface)obj;
            System.out.println(bobjref.generateWishMsg("rani"));
        }
}
Note:- jar files in classpath:
   1.spring.jar
   2.commons-logging.jar

=============================================================================
App 3) Spring core application on injecting one spring bean class object  to another spring bean
class property(Using <ref> tag)
=============================================================================
----------------------------------------Demo java----------------------------------------
public interface Demo
{
      public String sayHello();
}
----------------------------------------DemoBean.java----------------------------------------
import java.util.*;

public class DemoBean implements Demo
{
      String name;
      int age;
      Date d;
      TestBean tb;

      public void setName(String name)
      {
            System.out.println("DemoBean:setName()");
            this.name=name;
      }
      public void setAge(int age)
      {
            System.out.println("DemoBean:setAge()");
            this.age=age;
      }
      public void setD(Date d)
      {
            System.out.println("DemoBean:setD()");
            this.d=d;
      }
      public void setTb(TestBean tb)
      {
            System.out.println("DemoBean:setTb()");
            this.tb=tb;
      }
      public String sayHello()
      {
            return "GoodMorning"+"name="+name+"age="+age+"d="+d+"tb="+tb;
      }
}
----------------------------------------DemoCfg.xml----------------------------------------
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
            "http://www.springframework.org/dtd/spring-beans-2.0.dtd">

<beans>
    <bean id="t1" class="TestBean">
          <property name="msg"><value>Hello</value></property>
    </bean>
```

```
183
184     <bean id="dt" class="java.util.Date">
185         <property name="year"><value>110</value></property>
186         <property name="month"><value>12</value></property>
187         <property name="date"><value>20</value></property>
188     </bean>
189
190     <bean id="db" class="DemoBean">
191         <property name="name"><value>Raja</value></property>
192         <property name="age"><value>30</value></property>
193         <property name="d"><ref bean="dt"/></property>
194         <property name="tb"><ref bean="t1"/></property>
195     </bean>
196 </beans>
197 ---------------------------------TestBean.java-------------------------------
198 public class TestBean
199 {
200     String msg;
201
202     public void setMsg(String msg)
203     {
204         System.out.println("TestBean:setMsg(-)");
205         this.msg=msg;
206     }
207     public String toString()
208     {
209         return "TestBean.msg="+msg;
210     }
211 }
212 ---------------------------------DemoClient.java-----------------------
213 import org.springframework.beans.factory.xml.XmlBeanFactory;
214 import org.springframework.core.io.FileSystemResource;
215
216 public class DemoClient
217 {
218     public static void main(String args[])
219     {
220         FileSystemResource res=new FileSystemResource("DemoCfg.xml");
221         XmlBeanFactory factory=new XmlBeanFactory(res);
222         Object obj=factory.getBean("db");
223         Demo bobjref=(Demo)obj;
224         System.out.println(bobjref.sayHello());
225     }
226 }
227 Note:- jar files in classpath:
228    1.spring.jar
229    2.commons-logging.jar
230
231 ============================================================================
232 App 4) >>>>>>>>>>>>>>.Spring application on injecting DataSource objects<<<<<<<<<<<<<<
233 ============================================================================
234 ---------------------------------Select.java-------------------------------
235
236 public interface Select {
237
238 public String fetchName(int eno);
239 public int fetchSalary(int eno);
240 }
241 ---------------------------------SelectBean.java-----------------------
242 import java.sql.Connection;
243 import java.sql.ResultSet;
```

```
import java.sql.Statement;
import javax.sql.DataSource;

public class SelectBean implements Select {

//Bean Property
DataSource ds;
public void setDs(DataSource ds)
{
      this.ds=ds;
      System.out.println("SelectBean:setDs()");
}

public String fetchName(int eno) {
      // TODO Auto-generated method stub
try
 {
      Connection con=ds.getConnection();
      Statement st=con.createStatement();
      ResultSet rs=st.executeQuery("select ename from emp where empno="+eno);
      String name=null;
      if(rs.next())
      {
            name=rs.getString(1);
      }
      rs.close();
      st.close();
      con.close();
      return name;
 }
catch(Exception e)
{
      return null;
 }
}


public int fetchSalary(int eno) {
// TODO Auto-generated method stub
try
{
      Connection con=ds.getConnection();
      Statement st=con.createStatement();
      ResultSet rs=st.executeQuery("select sal from emp where empno="+eno);
      int bsal=0;
      if(rs.next())
      bsal=rs.getInt(1);
      rs.close();
      st.close();
      con.close();
      return bsal;
      }
 catch(Exception e)
 {
  return 0;
 }
}//business method
}//class
----------SelectCfg.xml----------
<?xml version="1.0" encoding="UTF-8"?>
<beans
```

```
        xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:p="http://www.springframework.org/schema/p"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-2.5.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:orcl"/>
        <property name="username" value="scott"/>
        <property name="password" value="tiger"/>
    </bean>

     <bean id="sb" class="SelectBean">
        <property name="ds" ref="drds"/>
    </bean>

</beans>
----------------------------------SelectClient.java----------------------------
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.FileSystemResource;

public class SelectClient {
public static void main(String args[])
{

    FileSystemResource res=new FileSystemResource("DemoCfg.xml");
    XmlBeanFactory factory=new XmlBeanFactory(res);

    Object obj=ctx.getBean("sb");
    Select bobjref=(Select)obj;

    System.out.println("7900 emp name is"+bobjref.fetchName(7900));
    System.out.println("7900 emp salary is"+bobjref.fetchSalary(7900));

  }
}
Note: jar files in classpath:
    1.spring.jar
    2.commons-logging.jar
    3.ojdbc.jar
=========================================================================
App 5)>>>>>>>>>>>>>>>>setter Injection on different types of properties<<<<<<<<<<<<<<<<<
=========================================================================
-----------------------------Demo.java-----------------------------

public interface DemoInterface {
        public String sayHello();

}
-----------------------------DemoBean.java-----------------------------
public class DemoBean implements Demo {

        String name;
        int age;
        int marks[];
        String subjects[];

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

```
366
367       public void setAge(int age) {
368            this.age = age;
369       }
370
371       public void setMarks(int[] marks) {
372            this.marks = marks;
373       }
374
375       public void setSubjects(String[] subjects) {
376            this.subjects = subjects;
377       }
378
379
380       public String sayHello() {
381       System.out.println("name="+name);
382       System.out.println("age="+age);
383       System.out.println("marks=");
384       for(int i=0;i<marks.length;++i)
385       {
386       System.out.println(marks[i]+".....");
387
388       }
389       System.out.println("subjects=");
390       for(int i=0;i<subjects.length;++i)
391       {
392            System.out.println(subjects[i]+".....");
393       }
394
395       return "GOOD MORNING";
396    }
397
398  }
399  ---------------------------------DemoCfg.xml----------------------
400  <?xml version="1.0" encoding="UTF-8"?>
401  <beans
402       xmlns="http://www.springframework.org/schema/beans"
403       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
404       xmlns:p="http://www.springframework.org/schema/p"
405       xsi:schemaLocation="http://www.springframework.org/schema/beans
406       http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">
407
408    <bean id="db" class="DemoBean">
409      <property name="name" value="raja"></property>
410      <property name="age" value="22"></property>
411      <property name="marks">
412        <list>
413         <value>90</value>
414         <value>95</value>
415         <value>98</value>
416        </list>
417      </property>
418      <property name="subjects">
419        <list>
420          <value>JAVA</value>
421          <value>J2EE</value>
422         <value>.NET</value>
423        </list>
424       </property>
425     </bean>
426
```

```
</beans>
----------DemoClient.java----------
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;

public class DemoClient {
public static void main(String args[])
{
      ClassPathResource res= new ClassPathResource("DemoCfg.xml");
      XmlBeanFactory factory=new XmlBeanFactory(res);
      Demo bref=(Demo)factory.getBean("db");
      System.out.println("Result is:"+bref.sayHello());

  }
}
==========
App 6)>>>>>>Spring application performs setter injection  on all types of properties<<<<<<<<<
==========
----------Demo.java----------

public interface Demo {
  public String sayHello();

}
----------DemoBean.java----------
import java.util.List;
import java.util.Map;
import java.util.Properties;
import java.util.Set;


public class DemoBean implements Demo{
 String name;
 int marks[];
 List names;
 Set phones;
 Map capitals;
 Properties faculties;


public void setName(String name) {
      this.name = name;
}
public void setMarks(int[] marks) {
      this.marks = marks;
}
public void setNames(List names) {
      this.names = names;
}
public void setPhones(Set phones) {
      this.phones = phones;
}
public void setCapitals(Map capitals) {
      this.capitals = capitals;
}
public void setFaculties(Properties faculties) {
      this.faculties = faculties;
}

//implementation of business method
public String sayHello()
```

```
{
        System.out.println("name="+name);

        System.out.println("marks=");
        for(int i=0;i<marks.length;i++)
          System.out.println(marks[i]+".....");

        System.out.println("names="+names.toString());
        System.out.println("phones="+phones.toString());
        System.out.println("Capitals="+capitals.toString());
        System.out.println("Faculties="+faculties.toString());

        return "Good morning";
 }
}
----------DemoCfg.xml----------

<?xml version="1.0" encoding="UTF-8"?>
<beans
        xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:p="http://www.springframework.org/schema/p"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">
            db  <bean id="dt" class="java.util.Date"/>
<bean id="dt" class="DemoBean">
 <property name="name" value="raja"/>
 <property name="marks">
   <list>
     <value>80</value>
     <value>85</value>
     <value>90</value>
   </list>
 </property>

   <property name="names">
     <list>
       <value>Raja</value>
       <value>Rani</value>
       <value>Ram</value>
       <ref bean="dt"/>(512)
     </list>
   </property>
   <property name="phones">
     <set>
       <value>9999999999</value>
       <value>9393939393</value>
       <ref bean="dt"/> (512)
    </set>
  </property>
   <property name="capitals">
     <map>
       <entry>
         <key><value>India</value></key>
         <value>New delhi</value>
       </entry>

       <entry>
        <key><value>Japan</value></key>
        <value>Tokyo</value>
       </entry>
```

```
549          <entry>
550            <key><ref bean="dt"/></key>
551            <ref bean="dt"/> (512)
552          </entry>
553         </map>
554    </property>
555
556    <property name="faculties">
557    <props>
558      <prop key="Nataraz">Jee Faculty</prop>
559      <prop key="raja">Spring Faculty</prop>
560      <prop key="raz">Struts Faculty</prop>
561    </props>
562   </bean>
563  </beans>
564  ----------DemoClient.java----------
565  import org.springframework.beans.factory.xml.XmlBeanFactory;
566  import org.springframework.core.io.ClassPathResource;
567
568  public class DemoClient {
569   public static void main(String args[])
570    {
571        ClassPathResource res= new ClassPathResource("DemoCfg.xml");
572        XmlBeanFactory factory=new XmlBeanFactory(res);
573        Demo bref=(Demo)factory.getBean("db");
574        System.out.println("Result is:"+bref.sayHello());
575
576     }
577  }
578
579  ==========
580  App 7)>>>>>.Spring application performs construcor injection  on all types of properties<<<<<<
581  ==========
582  ----------Demo.java----------
583
584  public interface Demo {
585        public String sayHello();
586
587  }
588  ----------DemoBean.java----------
589  import java.util.List;
590  import java.util.Map;
591  import java.util.Properties;
592  import java.util.Set;
593
594
595  public class DemoBean implements Demo{
596        int marks[];
597        Date d;
598        List fruits;
599        Set phones;
600        Map capitals;
601        Properties players;
602
603     public DemoBean(int[] marks, Date d, List fruits, Set phones, Map capitals,
604             Properties players) {
605        super();
606        this.marks = marks;
607        this.d = d;
608        this.fruits = fruits;
609        this.phones = phones;
```

```
610         this.capitals = capitals;
611         this.players = players;
612         }
613
614
615     //implementation of business method
616     public String sayHello()
617     {
618
619           System.out.println("marks=");
620           for(int i=0;i<marks.length;i++)
621           System.out.println(marks[i]+".....");
622
623           System.out.println("d="+d);
624           System.out.println("Fruits="+fruits);
625           System.out.println("Phones="+phones);
626           System.out.println("Capitals="+capitals);
627           System.out.println("Players="+players);
628
629           return "Good morning";
630      }
631     }
632     ----------DemoCfg.xml----------
633     <?xml version="1.0" encoding="UTF-8"?>
634     <beans
635           xmlns="http://www.springframework.org/schema/beans"
636           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
637           xmlns:p="http://www.springframework.org/schema/p"
638           xsi:schemaLocation="http://www.springframework.org/schema/beans
639            http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">
640
641     <bean id="dt" class="java.util.Date"/>
642     <bean id="db" class="DemoBean">
643
644      <constructor-arg>
645         <list>
646            <value>90</value>
647             <value>95</value>
648              <value>100</value>
649         </list>
650      </constructor-arg>
651
652       <constructor-arg><ref bean="dt"/></constructor-arg>
653
654       <constructor-arg>
655           <list>
656              <value>Apple</value>
657              <value>Grapes</value>
658           </list>
659      </constructor-arg>
660
661      <constructor-arg>
662         <set>
663          <value>9999999999</value>
664          <value>9393939393</value>
665        </set>
666      </constructor-arg>
667
668      <constructor-arg>
669         <map>
670           <entry>
```

```
671           <key><value>AP</value></key>
672           <value>Hyderabad</value>
673         </entry>
674
675          <entry>
676           <key><value>TN</value></key>
677            <value>Chennai</value>
678          </entry>
679        </map>
680   </constructor-arg>
681
682    <constructor-arg>
683        <props>
684          <prop key="Sachin">Cricket</prop>
685          <prop key="Rana">Shooting</prop>
686        </props>
687    </constructor-arg>
688
689   </bean>
690  </beans>
691  ----------DemoClient.java----------
692  import org.springframework.beans.factory.xml.XmlBeanFactory;
693  import org.springframework.core.io.ClassPathResource;
694
695  public class DemoClient {
696  public static void main(String args[])
697  {
698        ClassPathResource res= new ClassPathResource("DemoCfg.xml");
699
700        XmlBeanFactory factory=new XmlBeanFactory(res);
701
702        Demo bref=(Demo)factory.getBean("db");
703        System.out.println("Result is:"+bref.sayHello());
704
705    }
706  }
707  ========================================================================
708  App 8)>>>>>>>>>>>>>>>>Spring Application using properties file<<<<<<<<<<<<<<<<<<<
709  ========================================================================
710  ----------myfile.properties----------
711  #jdbc properties
712  my.driver=oracle.jdbc.driver.OracleDriver
713  my.url=jdbc:oracle:thin:@localhost:1521:orcl
714  my.dbuser=scott
715  my.dbpwd=tiger
716  ----------SelectCfg.xml----------
717  <?xml version="1.0" encoding="UTF-8"?>
718  <beans
719        xmlns="http://www.springframework.org/schema/beans"
720        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
721        xmlns:p="http://www.springframework.org/schema/p"
722        xsi:schemaLocation="http://www.springframework.org/schema/beans
723        http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">
724
725
726   <bean id="propconfig"
      class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
727       <property name="location" value="myfile.properties"/>
728   </bean>
729
730  <bean id="drds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
```

```
731     <property name="driverClassName" value="${my.driver}"/>
732     <property name="url" value="${my.url}"/>
733     <property name="username" value="${my.dbuser}"/>
734     <property name="password" value="${my.dbpwd}"/>
735  </bean>
736
737   <bean id="sb" class="SelectBean">
738      <property name="ds" ref="drds"/>
739  </bean>
740
741  </beans>
742  ------------------Select.java----------
743  public interface Select {
744      public String fetchName(int eno);
745      public int fetchSalary(int eno);
746  }
747  -------------------------SelectBean.java-------------------
748  import java.sql.Connection;
749  import java.sql.ResultSet;
750  import java.sql.Statement;
751  import javax.sql.DataSource;
752
753
754  public class SelectBean implements Select {
755
756  //Bean Property
757  DataSource ds;
758  public void setDs(DataSource ds)
759  {
760        this.ds=ds;
761        System.out.println("SelectBean:setDs()");
762  }
763
764  public String fetchName(int eno) {
765        try
766        {
767        Connection con=ds.getConnection();
768        Statement st=con.createStatement();
769        ResultSet rs=st.executeQuery("select ename from emp where empno="+eno);
770        String name=null;
771        if(rs.next())
772        {
773              name=rs.getString(1);
774        }
775        rs.close();
776        st.close();
777        con.close();
778        return name;
779       }
780        catch(Exception e)
781        {
782              return null;
783        }
784  }
785
786
787  public int fetchSalary(int eno) {
788   try
789    {
790        Connection con=ds.getConnection();
791        Statement st=con.createStatement();
```

```
792         ResultSet rs=st.executeQuery("select sal from emp where empno="+eno);
793         int bsal=0;
794         if(rs.next())
795          bsal=rs.getInt(1);
796          rs.close();
797          st.close();
798          con.close();
799         return bsal;
800    }
801    catch(Exception e)
802    {
803        return 0;
804    }
805  }//business method
806  }//class
807  ----------------------------------SelectClient.java-------------------------
808  import org.springframework.context.support.FileSystemXmlApplicationContext;
809
810  public class SelectClient {
811   public static void main(String args[])
812   {
813     FileSystemXmlApplicationContext ctx=new FileSystemXmlApplicationContext("SelectCfg.xml");
814
815      Object obj=ctx.getBean("sb");
816      Select bobjref=(Select)obj;
817
818      System.out.println("7900 emp name is"+bobjref.fetchName(7900));
819      System.out.println("7900 emp salary is"+bobjref.fetchSalary(7900));
820
821     }
822  }
823
824  ================================================================================
825  App 9)>>>>>>>>>>>>>>>>>Application on I18N(Internationalization) plain<<<<<<<<<<<<<<<<<
826  ================================================================================
827  ----------App.properties----------
828  #(BaseFile-->English)
829  str1 = delete
830  str2 = save
831  str3 = stop
832  str4 = cancel
833
834  ----------App.fr_CA.properties----------
835  #for french language
836  str1 = EFFACER
837  str2 = SAUF
838  str3 = ARRÊT
839  str4 = ANNULER
840
841  ----------App_de_DE.properties----------
842  #for german language
843  str1 = LÖSCHEN
844  str2 = DENN
845  str3 = ZUG
846  str4 = ABSAGEN
847  ----------App_hi_IN.properties----------
848  #for hindhi language
849  str1=\u0928\u0940\u0916\u093e\u0932\u094a
850  str2=\u0936\u0941\u0930\u0936\u094d\u0915\u093f\u0924\u094d
851  str3=\u0930\u0941\u0916\u094a
852  str4=\u0930\u0926\u094d\u0926\u094d
```

```
----------I18nApp.java----------
//I18nApp.java
// internationalization of an application (the captions on buttons will come
//based on the locale specific properties file that is activated)
import java.awt.*;
import javax.swing.*;
import java.util.*;

public class I18nApp {
 public static void main(String args[]){

  // create Locale obj based on given language,country codes
  Locale l= new Locale(args[0],args[1]);

// picks up properties file based on the Locale obj data
ResourceBundle r= ResourceBundle.getBundle("App",l);

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

// create btns by getting labels from activated properties file..
JButton b1 = new JButton(r.getString("str1"));
JButton b2 = new JButton(r.getString("str2"));
JButton b3 = new JButton(r.getString("str3"));
JButton b4 = new JButton(r.getString("str4"));

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

jf.pack();
jf.setVisible(true);
  }//main
}//class

//>javac I18nApp.java
//>java I18nApp fr CA    (App_fr_CA.properties file will be activated)
//>java I18nApp hi IN    (App_hi_IN.properties file will be activated)
//>java I18nApp x y      (App.properties file will be activated(base file)
==========================================================================
App 10)>>>>>>>>>>>>>>>>>>>>Application On I18N using spring<<<<<<<<<<<<<<<<<<<<
==========================================================================
----------DemoCfg.xml----------
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
 <bean id="messageSource"
 class="org.springframework.context.support.ResourceBundleMessageSource">
  <property name="basenames">
    <list>
        <value>App</value>
        <value>App_de_DE</value>
        <value>App_fr_CA</value>
        <value>App_hi_IN</value>
      </list>
  </property>
 </bean>

</beans>
```

```
912  ----------DemoClient.java----------
913  //DemoClient.java
914  import org.springframework.context.support.*;
915  import java.util.*;
916  import javax.swing.*;
917  import java.awt.*;
918
919  public class DemoClient {
920        public static void main(String[] args)throws Exception {
921        //Locale Object
922        Locale l=new Locale(args[0],args[1]);
923       //Activate ApplicationContext container
924          ClassPathXmlApplicationContext ctx=new
           ClassPathXmlApplicationContext("DemoCfg.xml");
925        // get Msgs from proeprties file
926        String msg1=ctx.getMessage("str1",null,"default msg1",l);
927        String msg2=ctx.getMessage("str2",null,"default msg2",l);
928        String msg3=ctx.getMessage("str3",null,"default msg3",l);
929        String msg4=ctx.getMessage("str4",null,"default msg4",l);
930          //Develope Swing Frame
931        JFrame jf = new JFrame();
932        Container cp = jf.getContentPane();
933
934        // create btns by getting labels from activated properties file..
935        JButton b1 = new JButton(msg1);
936        JButton b2 = new JButton(msg2);
937        JButton b3 = new JButton(msg3);
938        JButton b4 = new JButton(msg4);
939
940        cp.setLayout(new FlowLayout());
941        cp.add(b1);
942        cp.add(b2);
943        cp.add(b3);
944        cp.add(b4);
945
946        jf.pack();
947        jf.show();
948        }//main
949  }//class
950  //>javac DemoClient.java
951  //>java DemoClient fr FR        (uses App_fr_FR.properties)
952  //>java DemoClient de DE        (uses App_de_DE.properties)
953  //>java DemoClient x y          (uses App.properties)
954  ================================================================
955  App 11)>>>>>>>>>>>>>>>>>Application on Interface Injection<<<<<<<<<<<<<<<<
956  ================================================================
957  >>>>>>>>>>>>>>>Example App on Interface injection>>>>>>>>>>>>>>>
958  ----------Demo.java----------
959  public interface Demo
960  {
961    public String sayHello();
962  }
963  ----------DemoCfg.xml----------
964  <!-- DemoCfg.xml (Spring cfg file)  -->
965  <!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
966                "http://www.springframework.org/dtd/spring-beans-2.0.dtd">
967
968   <beans>
969    <bean id="db"  class="DemoBean" scope="prototype">
970        <property name="msg"><value>hello</value></property>
971     </bean>
```

```
  <bean  id="dt" class="java.util.Date"/>
  <bean  id="dt1" class="java.util.Date"/>
</beans>
----------DemoBean.java----------
// DemoBean.java (Spring Bean -non pojo class)

import org.springframework.context.*;
import org.springframework.beans.factory.*;

public class  DemoBean implements Demo,BeanNameAware,ApplicationContextAware
{
   String msg;
   String bname;
   ApplicationContext ctx;

    //setter injection
    public void setMsg(String msg)
    {
       this.msg=msg;
     }

    // to support interface injection (method of BeanNameAware (I))
      public void setBeanName(String name)
      {
       bname=name;
      }

      // to support Interface injection (method of ApplicationContextAware(i))
       public void setApplicationContext(ApplicationContext ctx)
      {
       this.ctx=ctx;
       }

   public String sayHello()
    {
     System.out.println("logical name current bean is"+bname);
     System.out.println("no.of bean managed by uderlying
     cotainer"+ctx.getBeanDefinitionCount());
     System.out.println("current bean is singleton Bean ?"+ctx.isSingleton(bname));
     System.out.println("current bean is prototype Bean ?"+ctx.isPrototype(bname));

    return "Good Morning";
   }//method
}//class
----------DemoClient.java----------
// DemoClient.java (java app as Local Client App )
import org.springframework.core.io.*; //( for FileSystemResource(c) )
import org.springframework.beans.factory.xml.*; //(for XmlBeanFactory (c))
import org.springframework.context.support.*; //(for FileSystemXmlApplicationContext (c))

import java.util.*;

public class DemoClient
{
public static void main(String args[])throws Exception
{
   FileSystemXmlApplicationContext ctx=new
   FileSystemXmlApplicationContext("DemoCfg.xml");
   System.out.println("ApplicationContext container is activated");

   Demo bobj=(Demo)ctx.getBean("db");
```

```
1031      System.out.println("Result is"+bobj.sayHello());
1032
1033   }//main
1034  }//class
1035  ==========================================================================
1036  App 12)>>>>>>>>>>>>>>>>Spring Mini-Project based on Spring Core Module<<<<<<<<<<<<<<<
1037  ==========================================================================
1038  ------------------------------------Main.html-------------------------------------
1039  <frameset rows="30%,*">
1040      <frame src="Search.jsp">
1041      <frame name="resultframe">
1042  </frameset>
1043  ------------------------------------Search.jsp-------------------------------------
1044   <body bgcolor="pink" >
1045
1046    <form action="controller" target="resultframe">
1047     <b>Select job</b>
1048       <select name="job">
1049            <option value="clerk">CLERKS</option>
1050            <option value="analyst">ANALYSTS</option>
1051            <option value="salesman">SALESMEN</option>
1052           <option value="manager">MANAGERS</option>
1053
1054       </select>
1055     <input type="submit" value="search">
1056    </form>
1057  </body>
1058  <%System.out.println("Serach.jsp"); %>
1059  ------------------------------------------------web.xml-----------------------------------------------
1060  <?xml version="1.0" encoding="UTF-8"?>
1061  <web-app version="2.4"
1062         xmlns="http://java.sun.com/xml/ns/j2ee"
1063         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1064         xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
1065         http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
1066    <servlet>
1067     <servlet-name>MainServlet</servlet-name>
1068     <servlet-class>MainServlet</servlet-class>
1069    </servlet>
1070
1071    <servlet-mapping>
1072     <servlet-name>MainServlet</servlet-name>
1073     <url-pattern>/controller</url-pattern>
1074    </servlet-mapping>
1075
1076  </web-app>
1077  ------------------------------------MainServlet.java-------------------------------------
1078  import java.io.IOException;
1079  import java.util.ArrayList;
1080  import javax.servlet.*;
1081  import javax.servlet.http.*;
1082
1083  import org.springframework.beans.factory.xml.*;
1084  import org.springframework.core.io.*;
1085
1086  public class MainServlet extends HttpServlet {
1087
1088  Model mod1=null;
1089
1090
1091  public void init()
```

```
{
        System.out.println("Init Method");
        ClasspathResource res=new ClassPathResource("modelconfig.xml");
        XmlBeanFactory factory=new XmlBeanFactory(res);
        mod1=(Model)factory.getBean("mdb");
}//init

public void destroy() {
        System.out.println("destroy()");
        mod1=null;
}

public void doGet(HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException {
        System.out.println("doGet() of MainServlet");
         String job=request.getParameter("job").trim();
        ArrayList al=mod1.search(job);
        request.setAttribute("result",al);

        RequestDispatcher rd=request.getRequestDispatcher("Result.jsp");
         if(rd!=null)
          rd.forward(request,response);
 }

public void doPost(HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException {
        System.out.println("doPost() of MainServlet");
        doGet(request,response);
 }//doPost

}//MainServlet
---------- Spring Environment----------
==========
----------Model.java----------
import java.util.*;
public interface Model {
        public ArrayList search(String desg);
}
----------ModelBean.java----------
import java.util.*;
import javax.sql.*;
import java.sql.*;
import p1.EmpBean;

public class ModelBean implements Model{
        public DataSource ds;

public void setDs(DataSource ds) {
        this.ds = ds;
}
public ArrayList search(String desg)
{
        System.out.println("search() of ModelBean class");
        ArrayList al=new ArrayList();

        try {
         Connection con=ds.getConnection();
         PreparedStatement ps=con.prepareStatement("select * from spring_employee where
         desg=?");
         ps.setString(1,desg);

```

```
1152        ResultSet rs=ps.executeQuery();
1153
1154          while(rs.next())
1155          {
1156             EmpBean eb=new EmpBean();
1157             eb.setId(rs.getInt(1));
1158             eb.setName(rs.getString(2));
1159             eb.setDesg(rs.getString(3));
1160             eb.setBsal(rs.getFloat(4));
1161             al.add(eb);
1162           }
1163         }
1164         catch (SQLException e) {
1165             e.printStackTrace();
1166        }
1167
1168        return al;
1169
1170    }
1171  }
1172  ----------modelconfig.xml----------
1173  <?xml version="1.0" encoding="UTF-8"?>
1174  <!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
     "http://www.springframework.org/dtd/spring-beans.dtd">
1175
1176  <beans>
1177   <bean id="ds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
1178     <property
     name="driverClassName"><value>oracle.jdbc.driver.OracleDriver</value></property>
1179     <property name="url"><value>jdbc:oracle:thin:@localhost:1521:orcl</value></property>
1180     <property name="username"><value>scott</value></property>
1181     <property name="password"><value>tiger</value></property>
1182   </bean>
1183    <bean id="mdb" class="ModelBean" autowire="byName"/>
1184   </beans>
1185   ----------EmpBean.java----------
1186   package p1;
1187
1188   public class EmpBean {
1189
1190        int id;
1191        String name,desg;
1192        float bsal;
1193
1194        public EmpBean()
1195        {
1196          System.out.println("EmpBean Constructor");
1197        }
1198
1199        public float getBsal() {
1200             return bsal;
1201        }
1202        public void setBsal(float bsal) {
1203          this.bsal = bsal;
1204        }
1205        public String getDesg() {
1206             return desg;
1207        }
1208        public void setDesg(String desg) {
1209             this.desg = desg;
1210        }
```

```
1211        public int getId() {
1212              return id;
1213        }
1214        public void setId(int id) {
1215              this.id = id;
1216        }
1217        public String getName() {
1218              return name;
1219        }
1220        public void setName(String name) {
1221              this.name = name;
1222        }
1223
1224  }
1225  ================================End of spring Environment=======================
1226  ------------------------------------------------Result.jsp-------------------------
1227  <%@page import="java.util.*,p1.EmpBean"%>
1228  <table>
1229  <% ArrayList list=(ArrayList)request.getAttribute("result");
1230    if(list!=null)
1231    { %>
1232     <tr>
1233       <th>ID</th>
1234       <th>Name</th>
1235       <th>Desg</th>
1236       <th>Salary</th>
1237     </tr>
1238     <%for(int i=0;i<list.size();++i)
1239      {
1240      EmpBean eb=(EmpBean)list.get(i); %>
1241      <tr>
1242         <td><%=eb.getId()%></td>
1243         <td><%=eb.getName()%></td>
1244         <td><%=eb.getDesg()%></td>
1245         <td><%=eb.getBsal()%></td>
1246      </tr>
1247      <% }       %>
1248      </table>
1249     <% } %>
1250
1251     <%System.out.println("Result.jsp"); %>
1252
1253
1254  ======================End of Application============================
1255
1256  Note:jar files required in WEB-INF\lib folder & classpath---------->spring.jar,commons-logging.jar,
     ojdbc14.jar
1257  ------------------------------------------------------------------------------------------------------
1258
1259  ========================================================================
1260  Spring core module with annotations:
1261  ----------------------------------------
1262  ================================================
1263  App 13)Spring 2.5 Applicaiton on Annotations (IOC)
1264  ================================================
1265  ------------Test.java-------------
1266  // Test.java (Spring Interface)
1267  public interface Test
1268  {
1269        public String sayHello();
1270  }
```

```
----------TestBean.java----------
//TestBean.java
import org.springframework.stereotype.*;
import org.springframework.beans.factory.annotation.*;
import java.util.*;

@Service
public class TestBean implements  Test
{
  // Bean property
   Date d;

 @Autowired  //to perform auto wiring(by type)  on bean property
 public void setD(Date d)
{
      this.d=d;
 }

 public String sayHello()
 {
      return "good morning  "+d.toString();
 }
}
----------DemoCfg.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:context="http://www.springframework.org/schema/context"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop-2.5.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-2.5.xsd">

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

  <bean id="tb" class="TestBean"/>
 <bean id="dt" class="java.util.Date"/>

</beans>
---------- ---------TestClient.java----------
// TestClient.java
import org.springframework.context.support.*;
public class  TestClient
{
  public static void main(String[] args)
  {
      FileSystemXmlApplicationContext ctx=new
      FileSystemXmlApplicationContext("DemoCfg.xml");
      Test beanobj=(Test)ctx.getBean("tb");
      System.out.println("result is "+beanobj.sayHello());

  }//main
}//class
==========================================================================
App 14)>>>>>>>>>>>>>>>>Spring 3.x Application on Annotations(IOC)<<<<<<<<<<<<<<<<<<
==========================================================================
----------TestInter.java----------
package p1;
```

```
1331
1332  public interface TestInter
1333  {
1334        public String sayHello();
1335  }
1336  ----------TestBean.java----------
1337  package p1;
1338   import java.util.Date;
1339  import java.util.List;
1340  import javax.annotation.Resource;
1341  import javax.annotation.Resources;
1342  import org.springframework.beans.factory.annotation.Value;
1343  import org.springframework.stereotype.Component;
1344  import org.springframework.stereotype.Service;
1345
1346  @Component("tb")
1347  public class TestBean implements TestInter
1348  {
1349  //Bean Property
1350  UserBean u1;
1351
1352  @Value("10")
1353  int no;
1354
1355  Date d1;
1356
1357  @Value("#{T(java.util.Arrays).asList('India','Bharat','Hindustan')}")
1358  String nicknames[];
1359
1360  @Value("#{T(java.util.Arrays).asList('Red','Blue','Green')}")
1361  List colors;
1362
1363
1364  @Resource(name="dfb")
1365  public void setD1(Date d)
1366  {
1367        d1=d;
1368  }
1369
1370  @Resource(name="ub")
1371  public void setU1(UserBean u1)
1372  {
1373        this.u1=u1;
1374  }
1375
1376
1377  public String sayHello()
1378  {
1379   System.out.println("no"+no);
1380   System.out.println("d1"+d1.toString());
1381   System.out.println("nicknames");
1382
1383  for(int i=0;i<nicknames.length;i++)
1384    System.out.println(nicknames[i]);
1385
1386   System.out.println("Colors="+colors.toString());
1387   System.out.println("u1="+u1);
1388   return "Good Morning";
1389  }
1390  }
1391  ----------UserBean.java----------
```

```
package p1;

import org.springframework.stereotype.*;

@Service("ub")
public class UserBean
{
  String msg="Hello World";

   public String toString()
   {
       return "DemoBean.msg"+msg;
   }
}
----------DateFactoryBean.java----------
package p1;

import org.springframework.beans.factory.FactoryBean;
import org.springframework.stereotype.Component;

@Component("dfb")
public class DateFactoryBean implements FactoryBean
{
 public Object getObject() throws Exception {
       return new java.util.Date();
  }
  public Class getObjectType() {
       return java.util.Date.class;
   }
  public boolean isSingleton() {
       return false;
  }

}
----------TestClient.java----------
package p1;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.context.support.*;

public class TestClient
{
 public static void main(String s[])
  {
       AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext();
       ctx.scan("p1");  //p1 is the package name
       ctx.refresh();
       TestInter test=(TestInter)ctx.getBean("tb");
       System.out.println("result is"+test.sayHello());

  }
}
==========================================================================
>>>>>>>>>>>>Example Applications on Spring Annotations(IOC)<<<<<<<<<<<<<<
==========================================================
App 15) >>>>>>>>>>>>>on constructor Injection<<<<<<<<<<<<<<<<<<<<<<<
==========================================================
----------Demo.java----------
public interface Demo
{
public String sayHello();
```

```
}
----------DemoBean.java----------
import javax.inject.Inject;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;
@Component("db")
public class DemoBean implements Demo
{  int age;
   String name;
   float avg;

 @Inject
public DemoBean(@Value("30")int age, @Value("Raja")String name, @Value("50.67")float avg) {
  this.age = age;
  this.name = name;
  this.avg = avg;
} 
 public String sayHello()
 {
 return "Welcome to Spring age= "+age+" name= "+name+" avg= "+avg;
 }
}
----------DemoClient.java----------
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class DemoClient {
public static void main(String s[]) throws Exception {
  ApplicationContext ctx = new AnnotationConfigApplicationContext(DemoBean.class);
      //ask spring container to give springbean class object
      Demo d1=(Demo)ctx.getBean("db");
      // calls B.method of SpringBean class
      String result = d1.sayHello();
      System.out.println(result);
  }
}
-----------------------------------------------------
=====================================================================
App 16) >>>>>>>>Reading values from properties file(using annotations)<<<<<<<<<<<
=====================================================================
App.properties
----------------
jdbc.driver=oracle.jdbc.driver.OracleDriver
jdbc.url=jdbc:oracle:thin:@localhost:1521:orcl
db.user=scott
db.pwd=tiger
----------select.java----------
package p1;
public interface Select
{
  public void readDBDetails();
}
----------SelectBean.java----------
package p1;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.ImportResource;
import org.springframework.stereotype.Component;

@ImportResource("classpath:/SpringCfg.xml")
@Component("sb")
```

```
1514 public class SelectBean implements Select
1515 {   private @Value("${db.user}")
1516     String uname;
1517
1518     private @Value("${db.pwd}")
1519     String pwd;
1520
1521    @Override
1522    public void readDBDetails() {
1523       System.out.println("userName="+uname);
1524      System.out.println("Password="+pwd);
1525    }
1526 }
1527 ----------Springcfg.xml----------
1528 <?xml version="1.0" encoding="UTF-8"?>
1529 <beans xmlns="http://www.springframework.org/schema/beans"
1530    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1531    xmlns:context="http://www.springframework.org/schema/context"
1532    xsi:schemaLocation="http://cxf.apache.org/core http://cxf.apache.org/schemas/core.xsd
1533  http://www.springframework.org/schema/beans
1534  http://www.springframework.org/schema/beans/spring-beans.xsd
1535  http://www.springframework.org/schema/context
1536  http://www.springframework.org/schema/context/spring-context.xsd">
1537
1538    <context:annotation-config/>
1539    <context:component-scan base-package="p1"/>
1540    <context:property-placeholder location="./App.properties"/>
1541    <!-- <context:property-placeholder location="classpath:/App.properties"/> -->
1542 </beans>
1543 ----------SelectClient.java----------
1544 package p1;
1545 import org.springframework.context.support.ClassPathXmlApplicationContext;
1546
1547 public class SelectClient
1548 {
1549   public static void main(String[] args)
1550   {
1551       System.out.println("main():SelectClient");
1552       ClassPathXmlApplicationContext ctx=new
          ClassPathXmlApplicationContext("SpringCfg.xml");
1553       Select bobj=(Select)ctx.getBean("sb");
1554       bobj.readDBDetails();
1555   }
1556 }
1557 ------------------------------------------------------------------------
1558 ========================================================================
1559 App 17) >>>>>>>custom init-method and custom-destroy methods cfg(annotations)<<<<<<<
1560 ========================================================================
1561 ----------Demo.java----------
1562 package p1;
1563 public interface Demo
1564 {
1565 public String sayHello();
1566 }
1567 ----------DemoBean.java----------
1568 package p1;
1569 import javax.annotation.PostConstruct;
1570 import javax.annotation.PreDestroy;
1571 import org.springframework.beans.factory.annotation.Value;
1572 import org.springframework.stereotype.Component;
1573
```

```
@Component("db")
public class DemoBean implements Demo {
@Value("durga")
String msg;

@Value("10")
int age;

DemoBean()
{
 System.out.println("DemoBean():constructor");
}

@PostConstruct
public void myInit()
{
System.out.println("Init-Method");
if(age<0){
    age=18;
  System.out.println("Age cannot be -ve,it is set to 18");
  }
}

@PreDestroy
public void myDestroy()
{
   System.out.println("Destroy-method");
   age=0;
}

  public String sayHello() {
      return "Hello :" + msg+"\n Your Age :"+age;
  }

}
----------------------DemoCfg.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-2.5.xsd
      http://www.springframework.org/schema/context
      http://www.springframework.org/schema/context/spring-context-2.5.xsd">
       <context:component-scan base-package="p1" />
</beans>
-----------------DemoClient.java----------------
package p1;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class DemoClient {
  public static void main(String s[]) throws Exception {

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

// ask spring container to give springbean class object
Demo d1 = (Demo) ctx.getBean("db");

// calls B.method of SpringBean class
  String result = d1.sayHello();
  System.out.println(result);

  ctx.close();
```

```
1635    }
1636  }
1637  ============================================================
1638  App 18) >>>>>>>>>>>>>>>>>>>on Factory Bean(annotations)<<<<<<<<<<<<<<<<<<<<
1639  ============================================================
1640  ----------TestBean.java-----------
1641  package p1;
1642  import java.text.SimpleDateFormat;
1643  import java.util.Date;
1644  import org.springframework.beans.factory.FactoryBean;
1645  import org.springframework.beans.factory.annotation.Value;
1646  import org.springframework.context.annotation.Scope;
1647  import org.springframework.stereotype.Component;
1648
1649  @Component("tb")
1650  @Scope("prototype")
1651  public class TestBean implements FactoryBean
1652  {
1653     @Value("21/12/2012")
1654     private String dt;
1655
1656     public Object getObject()
1657     {
1658        Date d=null;
1659        try{
1660        System.out.println(" getObject():TestBean");
1661        SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
1662        d = sdf.parse(dt);
1663        }catch(Exception e)
1664        {
1665         System.out.println("Exception while Parsing Dt"+e);
1666        }
1667        return d;
1668     }
1669     public Class getObjectType()
1670     {
1671        System.out.println(" getObjectType():TestBean");
1672        return java.util.Date.class;
1673     }
1674     public boolean isSingleton()
1675     {
1676        System.out.println("isSingleton():TestBean");
1677        return false;
1678     }
1679  }
1680  ----------Demo.java----------
1681  package p1;
1682
1683  public interface Demo
1684  {
1685     public String sayHello();
1686  }
1687  ----------DemoBean.java----------
1688  package p1;
1689  import java.util.Date;
1690  import javax.annotation.Resource;
1691  import org.springframework.stereotype.Component;
1692
1693  @Component("db")
1694  public class DemoBean implements Demo
1695  {
```

```
1696    @Resource(name="tb")
1697    Date  dt;
1698
1699    public String sayHello()
1700    {
1701       return "The time is"+dt;
1702    }
1703 }
1704 ----------DemoCfg.xml----------
1705 <?xml version="1.0" encoding="UTF-8"?>
1706 <beans xmlns="http://www.springframework.org/schema/beans"
1707       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1708       xmlns:context="http://www.springframework.org/schema/context"
1709       xsi:schemaLocation="http://www.springframework.org/schema/beans
1710       http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
1711       http://www.springframework.org/schema/context
1712       http://www.springframework.org/schema/context/spring-context-2.5.xsd">
1713  <context:component-scan base-package="p1" />
1714 </beans>
1715 ----------DemoClient.java----------
1716 package p1;
1717 import org.springframework.context.support.ClassPathXmlApplicationContext;
1718 public class DemoClient
1719 {
1720    public static void main(String[] args)
1721    {
1722       System.out.println("Hello");
1723       ClassPathXmlApplicationContext ctx=new
          ClassPathXmlApplicationContext("DemoCfg.xml");
1724       Demo bobj=(Demo)ctx.getBean("db");
1725       System.out.println(bobj.sayHello());
1726    }
1727 }
1728 ==========
1729 PLAIN JNDI
1730 ==========
1731 ================================================================================
1732 App 19 )>>>>>>>>>>.Java application to create InitialContext object representing connectivity
       with weblogic registry s/w<<<<<<<<<<<<
1733 ================================================================================
1734 ----------JndiConnTest.java----------
1735 import javax.naming.*;
1736 import java.util.*;
1737
1738 public class JndiConnTest
1739 {
1740
1741 public static void main(String args[])throws Exception
1742   {
1743      // prepare jndi properties(web logic)
1744      Hashtable ht=new Hashtable();
1745      ht.put(Context.INITIAL_CONTEXT_FACTORY,"weblogic.jndi.WLInitialContextFactory");
1746      ht.put(Context.PROVIDER_URL,"t3://localhost:7001");
1747
1748      // create InitialContext object
1749      InitialContext ic=new InitialContext(ht);
1750
1751      if(ic!=null)
1752      System.out.println("connection established");
1753      else
1754      System.out.println("connection not established");
```

```
1755
1756        NamingEnumeration ne=ic.list("");
1757        while(ne.hasMore())
1758        {
1759             NameClassPair ncp=(NameClassPair)ne.next();
1760             System.out.println(ncp.getName()+"<---------->"+ncp.getClassName());
1761        }
1762
1763        }//main
1764   }//class
1765   ==========================================================================
1766   App 20)>>>>>>>>>>>>>>>>>>>Java application to perform "List" operations<<<<<<<<<<<<<<
1767   ==========================================================================
1768   ----------------------------JndiListTest.java--------------------------
1769   import javax.naming.*;
1770   import java.util.*;
1771
1772   public class  JndiListTest
1773   {
1774   public static void main(String[] args) throws Exception
1775   {
1776   //prepare Jndi properties
1777    Hashtable ht=new Hashtable();
1778    ht.put(Context.INITIAL_CONTEXT_FACTORY,"weblogic.jndi.WLInitialContextFactory");
1779    ht.put(Context.PROVIDER_URL,"t3://localhost:7001");
1780
1781   /* //prepare Jndi properties (JBoss)
1782     Hashtable ht=new Hashtable();
1783     ht.put(Context.INITIAL_CONTEXT_FACTORY,"org.jnp.interfaces.NamingContextFactory");
1784     ht.put(Context.PROVIDER_URL,"jnp://localhost:1099");*/
1785
1786   /* //prepare Jndi properties (GlassFish)
1787    Hashtable ht=new Hashtable();
1788
       ht.put(Context.INITIAL_CONTEXT_FACTORY,"com.sun.enterprise.naming.SerialInitContextFacto
       ry");
1789    ht.put(Context.PROVIDER_URL,"iiop://localhost:4849");*/
1790
1791    InitialContext ic=new InitialContext();
1792    // ic represents connectivity with Naming/Directory registry S/w
1793
1794    //bind operation
1795     ic.bind("today",new Date());
1796     ic.bind("banana",new String("yellow"));
1797
1798    //listing
1799     System.out.println("---->Listing (after binding)");
1800    NamingEnumeration e=ic.list("");
1801
1802    while(e.hasMore())
1803    {
1804     NameClassPair np=(NameClassPair)e.next(); // each binding is accessed
1805    System.out.println(np.getName()+"-----> "+np.getClassName());
1806    }
1807   /*  //lookup
1808    System.out.println("---->lookup");
1809
1810    Object obj=ic.lookup("today");
1811    Date d1=(Date)obj;
1812    System.out.println("date is "+d1.toString());
1813
```

```
1814    // rebind
1815     System.out.println("---->rebind");
1816     ic.rebind("banana",new StringBuffer("green"));
1817
1818    System.out.println("Listing (after rebind) ");
1819    NamingEnumeration e1=ic.list("");
1820
1821   while(e1.hasMore())
1822   {
1823    NameClassPair np=(NameClassPair)e1.next(); // each binding is accessed
1824    System.out.println(np.getName()+"-----> "+np.getClassName());
1825   } */
1826
1827    /*  System.out.println("---->unbind operation");
1828   //unbind operation
1829     ic.unbind("today");
1830
1831     //listing
1832    System.out.println("\n\n\nListing---------");
1833     NamingEnumeration  e2=ic.list("");
1834    while(e2.hasMore())
1835    {
1836      NameClassPair np=(NameClassPair)e2.next();
1837      System.out.println(np.getName()+" ---> "+np.getClassName());
1838    }*/
1839
1840     }//main
1841   }//class
1842
1843   ==========================================================================
1844   App 21)Java application to use the weblogic JDBC connection pool in standalone environment
1845   ==========================================================================
1846   ----------------------------------JndiDsPoolTest.java-----------------------------------
1847   import javax.naming.*;
1848   import java.util.*;
1849   import java.sql.*;
1850   import javax.sql.*;
1851
1852   public class JndiDsPoolTest
1853   {
1854   public static void main(String args[])
1855   {
1856   //prepare Jndi properties
1857   Hashtable ht=new Hashtable();
1858
1859    ht.put("java.naming.factory.initial","weblogic.jndi.WLInitialContextFactory");
1860    ht.put("java.naming.provider.url","t3://localhost:7575");
1861
1862     //create Initial Context
1863   InitialContext ic=new InitialContext(ht);
1864
1865   DataSource ds=(DataSource)ic.lookup("DsJndi");
1866
1867   //write JDBC persistence logic
1868   Statement st=con.createStatement();
1869
1870   ResultSet rs=st.executeQuery("select * from student");
1871
1872   while(rs.next())
1873   {
1874    System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
```

```
}//while

//close jdbc connections
 rs.close();
 st.close();
 con.close();

}//main
}//class

Note: Add weblogic.jar file to the classpath
========== ==========
SPRING JNDI
==========
================================================================================
App 22) >>>>>>>>>>>>>>>>>>>spring based JNDI operations<<<<<<<<<<<<<<<<<<<<<<
================================================================================
----------DemoCfg.xml----------
<?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="jt" class="org.springframework.jndi.JndiTemplate">
    <constructor-arg>
      <props>
        <prop key="java.naming.factory.initial">weblogic.jndi.WLInitialContextFactory</prop>
        <prop key="PROVIDER_URL">t3://localhost:7001</prop>
      </props>
    </constructor-arg>
  </bean>
  <bean id="test" class="JndiTemplateTest">
     <property name="template"><ref bean="jt"/></property>
  </bean>
</beans>

----------JndiTemplateTest.java----------
//DemoClient.java
import org.springframework.context.support.*;
import org.springframework.jndi.*;
import java.util.*;

public class  JndiTemplateTest
{
 static JndiTemplate template;

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

public static void main(String[] args) throws Exception
{
   FileSystemXmlApplicationContext ctx=new FileSystemXmlApplicationContext("DemoCfg.xml");

   System.out.println("bind().....");
   template.bind("today",new Date());
   template.bind("apple",new String("it is red"));

 System.out.println("lookup operation");
 Date d1=(Date)template.lookup("today");
 System.out.println("(today)lookup value is"+d1.toString());
```

```
1936
1937  template.unbind("today");
1938
1939  /*System.out.println("lookup operation");
1940  d1=(Date)template.lookup("today");
1941  System.out.println("(today)lookup value is"+d1.toString());*/
1942
1943  template.rebind("apple",new String("it is green"));
1944
1945  System.out.println("lookup operation");
1946  String s1=(String)template.lookup("apple");
1947  System.out.println("(apple)lookup value is"+s1.toString());
1948  }
1949  }
1950  ==========================================================================
1951  App 23)>>>>>>>>>>>>>>>>>>>working with JNDI call back interfaces<<<<<<<<<<<<<<<<<<<
1952  ==========================================================================
1953  ---------------------------DemoCfg.xml----------------------------
1954  <?xml version="1.0" encoding="UTF-8"?>
1955  <!DOCTYPE beans PUBLIC  "-//SPRING//DTD BEAN//EN"
1956   "http://www.springframework.org/dtd/spring-beans.dtd">
1957  <beans>
1958    <bean id="jt" class="org.springframework.jndi.JndiTemplate">
1959      <constructor-arg>
1960        <props>
1961         <prop key="java.naming.factory.initial">weblogic.jndi.WLInitialContextFactory</prop>
1962         <prop key="PROVIDER_URL">t3://localhost:7001</prop>
1963        </props>
1964      </constructor-arg>
1965    </bean>
1966    <bean id="test" class="JndiTemplateTest">
1967      <property name="template"><ref bean="jt"/></property>
1968    </bean>
1969  </beans>
1970  ------------------------------JnditemplateTest.java---------------------------
1971  import org.springframework.context.support.*;
1972  import org.springframework.jndi.*;
1973  import java.util.*;
1974
1975  public class  JndiTemplateTest
1976  {
1977   static JndiTemplate template;
1978
1979   public void setTemplate(JndiTemplate template)
1980   {
1981          this.template=template;
1982   }
1983   public static void main(String args[])throws Exception
1984  {
1985     FileSystemXmlApplicationContext ctx=new
       FileSystemXmlApplicationContext("DemoCfg.xml");
1986  }
1987
1988  //implement list operation.....logic
1989    template.execute(new JndiCallBack)
1990   {
1991     public Object doInContext(Context ctx) throws NamingException
1992    {
1993         //Use plain JNDI API to implement list operation logic
1994        NamingEnumeration ne=ctx.list("");
1995
```

```
1996        while(ne.hasMore())
1997         {
1998            NameClassPair np=(NameClassPair)ne.next();
1999            System.out.println(np.getName()+"-------->"+np.getClassName());
2000         }//while
2001        return null;
2002       }//doInContext(-)
2003     }//inner class
2004    );//execute(-)
2005  }//main
2006 }//class
2007 ==========================================================================
2008 App 24)>>>>>>>>>>>>>>>>>>>>DAO based Spring App <<<<<<<<<<<<<<<<<<<<<<
2009 ==========================================================================
2010 ----------------------------------Select java-----------------------------------------
2011 import java.util.*;
2012
2013 public interface Select
2014 {
2015       public int getSal(int no);
2016       public boolean updateSalary(int no,int newsal);
2017       public boolean fireEmp(int no);
2018       public List getAllEmpDetails()throws Exception;
2019       public Map getEmpDetails(int no)throws Exception;
2020       public boolean registerEmp(int no,String name,String desg,int sal);
2021 }
2022 ---------------------------------------SelectBean.java---------------------------------------
2023 //SelectBean.java (SpringBean class--- B.logic)
2024
2025 import  java.util.*;
2026
2027 public class SelectBean implements Select
2028 {
2029 EmpDAO  dao;
2030
2031 public void setDao(EmpDAO dao)
2032 {
2033       this.dao=dao;
2034 }
2035
2036 public int getSal(int no)
2037   {
2038       int sal=0;
2039
2040       if(no>0)
2041       sal=dao.fetchSalary(no);
2042       return sal;
2043 }
2044
2045 public boolean updateSalary(int no,int newsal)
2046 {
2047       // b.logic
2048       int sal=dao.fetchSalary(no);
2049
2050       float nsal=sal+(sal*(newsal/100.0f));
2051       System.out.println(nsal);
2052
2053       newsal=Math.round(nsal);
2054       System.out.println(newsal);
2055
2056      // persistance logic
```

```
2057        int res=dao.modifySalary(no,newsal);
2058
2059        if(res==0)
2060         return false;
2061        else
2062         return true;
2063    }
2064
2065
2066    public boolean fireEmp(int no)
2067    {
2068        //persistance logic
2069        int res=dao.deleteRecord(no);
2070
2071        if(res==0)
2072         return false;
2073        else
2074         return true;
2075    }
2076
2077    public List getAllEmpDetails()throws Exception
2078    {
2079     List l=dao.getAllRecords();
2080
2081        if(l!=null)
2082         return l;
2083        else
2084          throw new Exception("no records found");
2085    }
2086
2087    public Map getEmpDetails(int no)throws Exception
2088    {
2089        Map m=dao.getEmpRecord(no);
2090
2091        if(m!=null)
2092         return m;
2093        else
2094        throw new Exception("no record found");
2095    }
2096
2097    public boolean registerEmp(int no,String name,String desg,int sal)
2098    {
2099
2100     int res=dao.insertRecord(no,name,desg,sal);
2101
2102      if(res==0)
2103          return false;
2104      else
2105          return true;
2106
2107     }//method
2108    }//class
2109    ----------SelectCfg.xml----------
2110    ?xml version="1.0" encoding="UTF-8"?>
2111    <!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
       "http://www.springframework.org/dtd/spring-beans.dtd">
2112
2113    <beans >
2114     <bean id="bds" class="org.apache.tomcat.dbcp.dbcp.BasicDataSource"
       destroy-method="close">
2115      <property name="driverClassName"><value>oracle.jdbc.driver.OracleDriver</value></property>
```

```
    <property name="url"><value>jdbc:oracle:thin:@localhost:1521:orcl</value></property>
    <property name="username"><value>scott</value></property>
    <property name="password"><value>tiger</value></property>
    <property name="initialSize"><value>5</value></property>
   </bean>

   <bean id="template" class="org.springframework.jdbc.core.JdbcTemplate">
       <property name="dataSource"><ref bean="bds"/></property>
    </bean>

    <bean id="mydao" class="EmpDAO">
         <property name="jt"><ref bean="template"/></property>
    </bean>

    <bean id="sb" class="SelectBean">
      <property name="dao"><ref bean="mydao"/></property>
    </bean>

  </beans>
----------------------------------------EmpDAO.java-----------------------------
import org.springframework.jdbc.core.*;
import java.util.*;

public class EmpDAO
{
JdbcTemplate jt;

public void setJt(JdbcTemplate jt)
{
      this.jt=jt;
}

public int insertRecord(int no,String name,String desg,int sal)
{

 // uses PreparedStatement obj internally
 int res=jt.update("insert into Emp (empno,ename,job,sal) values(?,?,?,?)",
                          new Object[]{new Integer(no),name,desg,new Integer(sal)});

   /*uses PrepredStatement
    int res=jt.update("insert into Emp (empno,ename,job,sal) values(?,?,?,?)",
                              new Object[]{new Integer(no),name,desg,new Integer(sal)},
                              new
                              int[]{Types.INTGER,Types.VARCHAR,Types.VARCHAR,Types.INTGER});*/

      /* uses SimpleStatement object
      int res=jt.update("insert into Emp(empno,ename,job,sal) values(no,name,desg,sal)"); */

      return res;
 }

 public int fetchSalary(int eno)
 {
  // uses SimpleStatement object
   int sal=jt.queryForInt("select sal from emp where empno="+eno);

      /*  uses PreparedStatement obj
       int sal=jt.queryForInt("select sal from emp where empno="+eno,new Object[]{}) */

       return sal;
 }
```

```
2176
2177   public int  modifySalary(int eno,int newsal)
2178   {
2179     //uses PreparedStaement object
2180     int res=jt.update("update Emp set  sal=? where empno=?",
2181                 new Object[]{new Integer(newsal),
2182                 new Integer(eno)});
2183         return res;
2184   }
2185
2186
2187    public int deleteRecord(int eno)
2188   {
2189         int res=jt.update("delete from Emp where empno=?",new Object[]{new Integer(eno)});
2190         return res;
2191   }
2192
2193
2194  public Map getEmpRecord(int eno)
2195  {
2196  // uses PrepareStatement object
2197   Map m=jt.queryForMap("select  *  from emp where empno=?",new Object[]{new Integer(eno)});
2198   return m;
2199   }
2200
2201  public List getAllRecords()
2202  {
2203   // uses PrepareStatement object
2204         List l=jt.queryForList("select  *  from emp",new Object[]{});
2205         return l;
2206   }
2207
2208  }//class
2209  ----------------------------------------SelectClient.java----------------------------------
2210  import org.springframework.context.support.*;
2211  import java.util.*;
2212
2213  public class SelectClient {
2214   public static void main(String[] args)throws Exception {
2215
2216    System.out.println("main method");
2217    FileSystemXmlApplicationContext ctx=new FileSystemXmlApplicationContext("selectCfg.xml");
2218
2219     Select bobj=(Select)ctx.getBean("sb");
2220
2221     // Get Salary (Select)
2222         System.out.println("Salary is"+bobj.getSal(7499));
2223
2224     // Update Salary (Update)
2225         System.out.println("Salary Updated ?"+bobj.updateSalary(7499,10));
2226
2227     // Delete Employee (Delete)
2228         System.out.println("Record Deleted ?"+bobj.fireEmp(7876));
2229
2230    // displays one emp record(Select)
2231          Map emprecord=bobj.getEmpDetails(7499);
2232          System.out.println("Emp Details are"+emprecord.get("EMPNO")+
2233                                       emprecord.get("ENAME")+
2234                                       emprecord.get("JOB")+
2235                                       emprecord.get("SAL"));
2236
```

```
2237    // display all emp records (Select)
2238       List records=bobj.getAllEmpDetails();
2239
2240    System.out.println("All Emp details are");
2241    for(int i=0;i<records.size();++i)
2242     {
2243         Map record=(Map)records.get(i);
2244         System.out.println("Emp Details are"+record.get("EMPNO")+
2245                                    record.get("ENAME")+
2246                                    record.get("JOB")+
2247                                    record.get("SAL"));
2248     }//for
2249
2250   // Inserting Employee (Insert)
2251     System.out.println("Employee Inserted ?"+bobj.registerEmp(678,"raja","CLERK",6789));*/
2252  }//main
2253
2254  }//class
2255
2256  ------------------------------------------------------------------------------------------------------
2257  ======================================================================
2258  App 25)>>>>>>>>>>>>>>>>>>>>(SimpleJdbcTemplateTest.java)<<<<<<<<<<<<<<<<<<
2259  ======================================================================
2260  import org.springframework.jdbc.core.JdbcTemplate;
2261  import org.springframework.jdbc.datasource.DriverManagerDataSource;
2262
2263  public class SimpleJdbcTemplateTest {
2264
2265    public static void main(String args[]) {
2266
2267        DriverManagerDataSource dataSource = new DriverManagerDataSource();
2268
2269        dataSource.setDriverClassName("oracle.jdbc.driver.OracleDriver");
2270        dataSource.setUrl("jdbc:oracle:thin:@localhost:1521:orcl");
2271        dataSource.setUsername("scott");
2272        dataSource.setPassword("tiger");
2273
2274        JdbcTemplate jt = new JdbcTemplate(dataSource);
2275
2276
2277        jt.execute("INSERT INTO STUDENT VALUES(900, 'raja1', 'HYD')");
2278        jt.execute("INSERT INTO STUDENT VALUES(901, 'raja2', 'HYD')");
2279        jt.execute("INSERT INTO STUDENT VALUES(902, 'raja3', 'HYD')");
2280
2281        int count = jt.queryForInt("SELECT COUNT(*) FROM STUDENT");
2282        System.out.println("Count : " + count);
2283
2284        jt.execute("DELETE FROM STUDENT WHERE SNO >= 900");
2285
2286        count = jt.queryForInt("SELECT COUNT(*) FROM STUDENT");
2287        System.out.println("Count : " + count);
2288    }
2289  }
2290  ------------------------------------------------------------------------------------------------------
2291  ======================================================================
2292  App 26)>>>>>>>>>>>>>>>>>(JdbcTemplateSelectTest.java)<<<<<<<<<<<<<<<<<<<<<
2293  ======================================================================
2294  import java.sql.Connection;
2295  import java.sql.PreparedStatement;
2296  import java.sql.ResultSet;
2297  import java.sql.SQLException;
```

```
2298
2299  import java.util.ArrayList;
2300  import java.util.List;
2301
2302  import javax.sql.DataSource;
2303
2304
2305  import org.springframework.context.ApplicationContext;
2306  import org.springframework.context.support.ClassPathXmlApplicationContext;
2307
2308  import org.springframework.jdbc.core.JdbcTemplate;
2309  import org.springframework.jdbc.core.PreparedStatementCreator;
2310  import org.springframework.jdbc.core.PreparedStatementSetter;
2311  import org.springframework.jdbc.core.ResultSetExtractor;
2312  import org.springframework.jdbc.core.RowMapper;
2313  import org.springframework.jdbc.support.rowset.SqlRowSet;
2314  import org.springframework.jdbc.datasource.DriverManagerDataSource;
2315
2316  public class JdbcTemplateSelectTest {
2317
2318  public static void main(String args[]) {
2319
2320   DriverManagerDataSource dataSource = new DriverManagerDataSource();
2321   dataSource.setDriverClassName("oracle.jdbc.driver.OracleDriver");
2322   dataSource.setUrl("jdbc:oracle:thin:@localhost:1521:orcl");
2323   dataSource.setUsername("scott");
2324   dataSource.setPassword("tiger");
2325   JdbcTemplate template = new JdbcTemplate(dataSource);
2326
2327  doSelectOperations(template);
2328   }
2329
2330   private static void doSelectOperations(JdbcTemplate template) {
2331
2332  // queryForInt()
2333  int count1 = template.queryForInt("SELECT COUNT(*) FROM EMP");
2334  System.out.println("Total count : " + count1);
2335
2336  // queryForInt() with Parameter
2337       Object[] parameters1 = new Object[] {"MANAGER"};
2338       int count2 = template.queryForInt(
2339        "SELECT COUNT(*) FROM EMP WHERE JOB LIKE ?",parameters1);
2340       System.out.println("Total count : " + count2);
2341
2342       // queryForObject()
2343       Object[] parameters2 = new Object[] {new Integer(7844)};
2344       Object o = template.queryForObject(
2345            "SELECT ENAME FROM EMP WHERE EMPNO = ?",
2346            parameters2,
2347            String.class);
2348       System.out.println("Name of 7844 employee is : " + o);
2349
2350
2351       // Querying for a single domain object.
2352       Employee emp =
2353            (Employee) template.queryForObject(
2354              "SELECT ENAME, JOB, SAL FROM EMP WHERE EMPNO = ?",
2355               new Object[] {new Integer(7844)},
2356               new RowMapper() {
2357                  public Object mapRow(ResultSet rs, int rowNum) throws SQLException {
2358
```

```
2359                    Employee e = new Employee();
2360
2361                    e.setNo(7844);
2362                    e.setName(rs.getString("ENAME"));
2363                    e.setDesignation(rs.getString(2));
2364                    e.setSalary(rs.getDouble(3));
2365
2366                    return e;
2367                }//mapRow
2368            });
2369
2370    System.out.println("Details of 7844 employee : " + emp);
2371
2372    // PreparedStatementCreator
2373    final String sql = "SELECT ENAME FROM EMP WHERE DEPTNO = ?";
2374    Object results = template.query(
2375    new PreparedStatementCreator() {
2376        public PreparedStatement createPreparedStatement(Connection con)throws
          SQLException {
2377          PreparedStatement ps = con.prepareStatement(sql);
2378           ps.setInt(1, 10);
2379           return ps;
2380            }
2381          },
2382
2383    new ResultSetExtractor() {
2384
2385    public Object extractData(ResultSet rs)throws SQLException {
2386     List names = new ArrayList();
2387     while(rs.next()){
2388         names.add(rs.getString("ENAME"));
2389     }
2390      return names;
2391    }extractData
2392  });
2393
2394  System.out.println("Employees working in Dept 10 : " + ((ArrayList)results));
2395
2396   // PreparedStatementSetter
2397    String sql1 = "SELECT ENAME FROM EMP WHERE JOB LIKE ?";
2398
2399    Object results1 = template.query(sql1,
2400
2401      new PreparedStatementSetter() {
2402          public void setValues(PreparedStatement ps)throws SQLException {
2403             ps.setString(1, "MANAGER");
2404           }setValues
2405          },
2406
2407      new ResultSetExtractor() {
2408          List names = new ArrayList();
2409          public Object extractData(ResultSet rs)throws SQLException {
2410               while(rs.next()){
2411                  names.add(rs.getString("ENAME"));
2412               }
2413
2414               return names;
2415           }//extractData
2416          }
2417    );
2418
```

```
2419        System.out.println("All Manager Names : " + ((ArrayList)results1));
2420
2421        // Example of RowSet supported by Spring JDBC.
2422        SqlRowSet rowSet = template.queryForRowSet(
2423                    "SELECT EMPNO, ENAME, JOB, SAL FROM EMP");
2424        rowSet.last();
2425
2426        while(! rowSet.isBeforeFirst()) {
2427            int empno = rowSet.getInt(1);
2428            String ename = rowSet.getString(2);
2429            String job = rowSet.getString(3);
2430            double salary = rowSet.getDouble(4);
2431
2432            System.out.println(empno + "\t" + ename + "\t" +
2433                        job + "\t" + salary);
2434
2435            rowSet.previous();
2436        }
2437    } // main()
2438 } // class
2439 ----------Employee.java----------
2440 public class Employee {
2441
2442        private int no;
2443        private String name;
2444        private String designation;
2445        private double salary;
2446
2447        public Employee() {
2448            super();
2449        }
2450
2451        public int getNo() {
2452            return no;
2453        }
2454
2455        public void setNo(int no) {
2456            this.no = no;
2457        }
2458
2459        public String getName() {
2460            return name;
2461        }
2462
2463        public void setName(String name) {
2464            this.name = name;
2465        }
2466
2467        public String getDesignation() {
2468            return designation;
2469        }
2470
2471        public void setDesignation(String designation) {
2472            this.designation = designation;
2473        }
2474
2475        public double getSalary() {
2476            return salary;
2477        }
2478
2479        public void setSalary(double salary) {
```

```
2480            this.salary = salary;
2481        }
2482
2483        public String toString() {
2484            return this.no + "\t"
2485                + this.name + "\t"
2486                + this.designation + "\t"
2487                + this.salary;
2488        }
2489    }
2490    ------------------------------------------------------------------------------------------------------------
2491    ============================================================================
2492    App 27) >>>>>>>>>>>>>>>>>> NamedParameterTest.java<<<<<<<<<<<<<<<<<<<<<<<<
2493    ============================================================================
2494    import org.springframework.jdbc.core.namedparam.MapSqlParameterSource;
2495    import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;
2496    import org.springframework.jdbc.core.namedparam.SqlParameterSource;
2497    import org.springframework.jdbc.datasource.SingleConnectionDataSource;
2498
2499    public class NamedParameterTest {
2500        static final String DRIVER_CLASS = "oracle.jdbc.driver.OracleDriver";
2501        static final String URL = "jdbc:oracle:thin:@localhost:1521:orcl";
2502        static final String USER_NAME = "scott";
2503        static final String PASSWORD = "tiger";
2504
2505        public static void main(String args[]) {
2506        SingleConnectionDataSource ds=new SingleConnectionDataSource();
2507        ds.setDriverClassName(DRIVER_CLASS);
2508        ds.setUrl(URL);
2509        ds.setUsername(USER_NAME);
2510        ds.setPassword(PASSWORD);
2511
2512        NamedParameterJdbcTemplate template =
2513                new NamedParameterJdbcTemplate(ds);
2514
2515        int id = 100;
2516        String QUERY =
2517            "SELECT COUNT(*) FROM STUDENT WHERE sno >= :ID";
2518        SqlParameterSource namedParameters =
2519                new MapSqlParameterSource("ID",new Integer(id));
2520
2521        int count = template.queryForInt(QUERY, namedParameters);
2522        System.out.println("No. of students : " + count);
2523        }
2524    }
2525    ============================================================================
2526    App 28)>>>>>>>>>>>>>Application on springDAO using JdbcTemplate object<<<<<<<<<<<<<
2527    ============================================================================
2528    ------------------------------selectcfg.xml-----------------------------
2529    <?xml version="1.0" encoding="UTF-8"?>
2530    <!DOCTYPE beans PUBLIC  "-//SPRING//DTD BEAN//EN"
2531    "http://www.springframework.org/dtd/spring-beans.dtd">
2532    <beans>
2533      <bean  id="d1" class="org.springframework.jdbc.datasource.DriverManagerDataSource" >
2534        <property  name="driverClassName"
          ><value>oracle.jdbc.driver.OracleDriver</value></property>
2535        <property  name="url" ><value>jdbc:oracle:thin:@localhost:1521:orcl</value></property>
2536        <property  name="username" ><value>scott</value></property>
2537        <property  name="password" ><value>tiger</value></property>
2538      </bean>
2539    <bean id="m1" class="MyDAO">
```

```
    <property name="ds"><ref bean="d1"/></property>
</bean>
</beans>

----------------------------------MyDAO.java----------------------------
import javax.sql.*;
import org.springframework.core.io.*;
import org.springframework.beans.factory.*;
import org.springframework.beans.factory.xml.*;
import org.springframework.jdbc.core.*;

class  MyDAO
{
      static Resource res=null;
      static BeanFactory factory=null;
      static
      {
      res=new ClassPathResource("selectcfg.xml");
       factory=new XmlBeanFactory(res);
        factory.getBean("m1");
      }

   private static DataSource ds;

      public void setDs(DataSource  ds)
      {
       this.ds=ds;
      }

      public static JdbcTemplate getTemplate()
      {
       JdbcTemplate jt=new JdbcTemplate(ds);
       return jt;
      }
}
----------------------------TestApp.java--------------------------
import org.springframework.jdbc.core.*;
class TestApp
{
      public static void main(String[] args)
      {
      JdbcTemplate jt=MyDAO.getTemplate();
      int res=jt.queryForInt("select count(*) from student");
      System.out.println("count is"+res);
      }
}
-------------------------------------------------------------------------
=========================================================
App 29)>>>>>>>>> (Calling PL/SQL Procedure using Spring Dao )<<<<<<<<<<<<<<<<<<
=========================================================
Stored Procedure in ORACLE.
=============================================================================
CREATE OR REPLACE PROCEDURE GET_EMP_DATA
      (NAME IN VARCHAR,
       NO OUT NUMBER,
       DESG OUT VARCHAR,
       SALARY OUT NUMBER)
AS

BEGIN
     SELECT EMPNO, JOB, SAL
```

```
2601            INTO NO, DESG, SALARY
2602            FROM EMP
2603            WHERE ENAME = NAME;
2604    END;
2605    /
2606    ------------------------------SimpleJdbcCallTest.java------------------------------
2607    import java.sql.Types;
2608    import java.util.HashMap;
2609    import java.util.Map;
2610
2611    import javax.sql.DataSource;
2612
2613    import org.springframework.jdbc.core.SqlOutParameter;
2614    import org.springframework.jdbc.core.SqlParameter;
2615    import org.springframework.jdbc.object.StoredProcedure;
2616    import org.springframework.jdbc.datasource.*;
2617
2618    public class SimpleJdbcCallTest {
2619
2620        public static void main(String args[]) {
2621
2622        DriverManagerDataSource dataSource = new DriverManagerDataSource();
2623          dataSource.setDriverClassName("oracle.jdbc.driver.OracleDriver");
2624          dataSource.setUrl("jdbc:oracle:thin:@localhost:1521:orcl");
2625          dataSource.setUsername("scott");
2626          dataSource.setPassword("tiger");
2627
2628          Employee emp = (Employee)getEmployeeDetailsWithStoredProcedure(dataSource,
              "ADAMS");
2629          System.out.println("ADAMS Details : " + emp);
2630       }
2631
2632     private static Object getEmployeeDetailsWithStoredProcedure
2633                                 (DataSource ds, String ename) {
2634
2635          MyStoredProcedure sp = new MyStoredProcedure(ds);
2636          //call procedure
2637          Map results = sp.execute(ename);
2638          //set outparmeter values to emp object
2639          Employee emp = new Employee();
2640          emp.setNo((Integer)results.get("NO"));
2641          emp.setName(ename);
2642          emp.setDesignation((String)results.get("DESG"));
2643          emp.setSalary((Double)results.get("SALARY"));
2644
2645          return emp;
2646     }
2647
2648     private static class MyStoredProcedure extends StoredProcedure {
2649
2650          public MyStoredProcedure(DataSource ds) {
2651
2652               super(ds, "GET_EMP_DATA");
2653               this.setFunction(false);
2654               SqlParameter[] params = {
2655                    new SqlParameter("NAME", Types.VARCHAR),
2656                    new SqlOutParameter("NO", Types.INTEGER),
2657                    new SqlOutParameter("DESG", Types.VARCHAR),
2658                    new SqlOutParameter("SALARY", Types.DOUBLE)
2659                              };
2660               this.setParameters(params);
```

```
2661            compile();
2662
2663        }
2664
2665        public Map execute(String name) {
2666            HashMap map = new HashMap();
2667            map.put("NAME", name);
2668            return super.execute(map);
2669        }
2670
2671    }//inner class
2672 }//outer class
2673
2674 public class Employee {
2675
2676        private Integer no;
2677        private String name;
2678        private String designation;
2679        private Double salary;
2680
2681
2682        public void setNo(Integer no) {
2683            this.no = no;
2684        }
2685
2686        public void setName(String name) {
2687            this.name = name;
2688        }
2689
2690
2691        public void setDesignation(String designation) {
2692            this.designation = designation;
2693        }
2694
2695
2696        public void setSalary(Double salary) {
2697            this.salary = salary;
2698        }
2699
2700        public String toString() {
2701            return this.no + "\t"
2702                + this.name + "\t"
2703                + this.designation + "\t"
2704                + this.salary;
2705        }
2706 }
2707 ------------------------------------------------------------
2708 ============================Spring DAO mini project============================
2709 App 30)>>>>>>>>>>>>>>>>>>>>>>implementing DAO, MVC2 architectures<<<<<<<<<<<<<<<
2710 ================================================================================
2711 ----------------------------ModelCfg.xml----------------------
2712 <?xml version="1.0" encoding="UTF-8"?>
2713 <!DOCTYPE beans PUBLIC  "-//SPRING//DTD BEAN//EN"
2714 "http://www.springframework.org/dtd/spring-beans.dtd">
2715 <beans>
2715     <bean id="jofb" class="org.springframework.jndi.JndiObjectFactoryBean">
2717         <property name="jndiName"><value>MyDsJndi</value></property>
2718       </bean>
2719       <bean id="template" class="org.springframework.jdbc.core.JdbcTemplate">
2720         <property name="dataSource"><ref bean="jofb"/></property>
2721       </bean>
```

```
2722        <bean id="dao" class="MyDao">
2723            <property name="jt"><ref bean="template"/></property>
2724        </bean>
2725        <bean id="mb" class="ModelBean">
2726            <property name="dao"><ref bean="dao"/></property>
2727        </bean>
2728    </beans>
2729    -------------------------MyDao.java-------------------------------
2730    import org.springframework.jdbc.core.*;
2731    import java.util.*;
2732    public class  MyDao
2733    {
2734        JdbcTemplate jt;
2735
2736      public void setJt(JdbcTemplate jt)
2737      {
2738        this.jt=jt; //setter injection
2739      }
2740
2741     public List getEmpDetails(String condition)
2742      {
2743
2744      // persistance logic
2745        List l=jt.queryForList("select empno,ename,job,sal from emp where job in "+condition+" order
          by job");
2746        return l;
2747       }
2748
2749    }
2750    ------------------------------Model.java----------------------
2751    import java.util.*;
2752
2753    public interface Model
2754    {
2755        public List search(String []jobs);
2756    }
2757    ----------------------ModelBean.java--------------------
2758    //ModelBean.java
2759    import java.util.*;
2760
2761    public class  ModelBean implements Model
2762    {
2763        MyDao dao;
2764
2765        public  void  setDao(MyDao dao)
2766        {
2767            this.dao=dao; //setter injection
2768        }
2769
2770        public List search(String[] jobs)
2771        {
2772            // b.logic
2773        StringBuffer sb=new StringBuffer("(");
2774
2775        for (int i=0;i<jobs.length;++i )
2776        {
2777            if(i==(jobs.length-1))
2778                sb.append("'"+jobs[i]+"'");
2779            else
2780                sb.append("'"+jobs[i]+"',");
2781        }//for
```

```
2782
2783        sb.append(")");
2784        String cond=sb.toString();
2785
2786        // use dao's persistance logic
2787         List l=dao.getEmpDetails(cond);
2788
2789        return l;
2790    }//method
2791 }//class
2792 ------------------Main.html-----------------------
2793
2794 <frameset rows="30%,*">
2795     <frame src="Search.jsp" name="f1"/>
2796         <frame  name="f2"/>
2797  </frameset>
2798  -------------------------result.jsp-----------------
2799 <%@ page import="java.util.*;" %>
2800
2801 <% List l=(List) request.getAttribute("result");
2802   if(l!=null)
2803   {  %>
2804    <table border=1>
2805      <tr>
2806      <th>EMPNO</th>
2807      <th>ENAME</th>
2808      <th>SALARY</th>
2809      <th>JOB</th>
2810    </tr>
2811    <%
2812      for(int i=0;i<l.size();++i)
2813      {
2814        Map m=(Map)l.get(i);  %>
2815          <tr>
2816            <td><%=m.get("EMPNO") %></td>
2817            <td><%=m.get("ENAME") %></td>
2818           <td><%=m.get("SAL") %></td>
2819           <td><%=m.get("JOB") %></td>
2820         </tr>
2821         <%} %>
2822    </table>
2823    <%}
2824   else
2825   { %>
2826         <b>No employees are found </b>
2827   <% } %>
2828 ------------------------------------------Search.jsp----------------------
2829
2830 <form target="f2" action="controller" method="get">
2831     <b>select job(s)</b>
2832     <select name="job" multiple>
2833          <option value='CLERK'>clerks</option>
2834          <option value='MANAGER'>managers</option>
2835          <option value='ANALYST'>analyts</option>
2836          <option value='SALESMAN'>salesmen</option>
2837    </select>
2838      <input type="submit" value="search"/>
2839  </form>
2840 ---------- ----------MainServlet.java--------------------
2841 import javax.servlet.*;
2842 import javax.servlet.http.*;
```

```
2843  import java.io.*;
2844  import java.util.*;
2845
2846  import org.springframework.context.support.*;
2847
2848  public class MainServlet extends HttpServlet
2849  {
2850  Model beanobj=null;
2851
2852  public void init()
2853  {
2854  // activate spring container
2855   ClassPathXmlApplicationContext ctx=new ClassPathXmlApplicationContext("ModelCfg.xml");
2856   beanobj=(Model)ctx.getBean("mb");
2857   }//init
2858
2859  public void doGet(HttpServletRequest req,HttpServletResponse res)throws
      ServletException,IOException
2860  {
2861  // read form data
2862   String desg[]=req.getParameterValues("job");
2863
2864        // call B.method
2865        List l=beanobj.search(desg);
2866
2867        // keep result in request attribute
2868        req.setAttribute("result",l);
2869
2870        // forward the control to result.jsp page
2871        RequestDispatcher rd=req.getRequestDispatcher("result.jsp");
2872        rd.forward(req,res);
2873    }//doGet(-,-)
2874
2875  public void doPost(HttpServletRequest req,HttpServletResponse res)throws
      ServletException,IOException
2876  {
2877        doGet(req,res);
2878   }//method
2879
2880  public void destroy()
2881   {
2882        beanobj=null;
2883   }
2884
2885  }//class
2886
2887
2888  ================================Spring ORM================================
2889  App 31)>>>>>>>>>>>>>>>>>>>>>>>>>Spring with hibernate<<<<<<<<<<<<<<<<<<<<<<<<
2890  ==========================================================================
2891  ----------------------demointer.java----------------------
2892  import java.util.*;
2893  public interface   demointer
2894  {
2895     public Iterator getData();
2896  }
2897  ----------------demoimpl.java----------------
2898  import java.util.*;
2899  import org.springframework.orm.hibernate3.*;
2900
2901  public class demoimpl implements demointer
```

```
{
private HibernateTemplate ht;

 public void setHt(HibernateTemplate ht)
 {
  this.ht=ht;
 }

 public Iterator getData()
 {
  List l=ht.find("from User");
    Iterator it=l.iterator();
   return it;
 }
}
----------SpringHB.xml----------
<?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="myds"
    class="org.springframework.jdbc.datasource.DriverManagerDataSource">
  <property name="driverClassName">
    <value>oracle.jdbc.driver.OracleDriver</value>
  </property>
  <property name="url">
    <value>jdbc:oracle:thin:@localhost:1521:orcl</value>
  </property>
  <property name="username">
    <value>scott</value>
  </property>
  <property name="password">
    <value>tiger</value>
  </property>
 </bean>

<bean id="mySessionFactory"
class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
 <property name="dataSource" ref="myds"/>
 <property name="mappingResources">
   <list>
     <value>User.hbm.xml</value>
   </list>
</property>
<property name="hibernateProperties">
  <props>
    <prop key="hibernate.dialect">org.hibernate.dialect.Oracle9Dialect</prop>
    <prop key="show_sql">true</prop>
  </props>
</property>
</bean>

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

 <bean  id="d1"  class="demoimpl">
   <property name="ht"> <ref bean="template"/> </property>
 </bean>

</beans>
```

```
----------selectservlet.java----------
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;

import org.springframework.beans.factory.*;
import org.springframework.context.*;
import org.springframework.context.support.*;

public class  selectservlet  extends HttpServlet
{
 public void service(HttpServletRequest request,HttpServletResponse response)
  throws ServletException,IOException
  {   PrintWriter out=response.getWriter();
       System.out.println("In service method of Servlet");

  ApplicationContext ctx=new ClassPathXmlApplicationContext("SpringHB.xml");

  demointer d1=(demointer)ctx.getBean("d1");
  Iterator i1=d1.getData();

 out.println("<body bgcolor=#ffffcc text=red>");
 out.println("<h1><center>all users</h1><hr><br><h3>");
 out.println("<table width=80% border=2>");
  while(i1.hasNext())
  {
      User u1=(User)i1.next();
      out.println("<tr><td>"+u1.getUid()+" <td>"+u1.getUname()+" <td>"+u1.getRole()+"</tr>");
  }
  out.println("</table>");
 }//service(-,-)
}//class
----------web.xml----------
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
   http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
   version="2.4">
  <servlet>
     <servlet-name>select</servlet-name>
     <servlet-class>selectservlet</servlet-class>
   </servlet>
   <servlet-mapping>
     <servlet-name>select</servlet-name>
     <url-pattern>/selectaction</url-pattern>
   </servlet-mapping>

</web-app>
----------User.java----------
public class User
{
private int uid;
private String uname,role;

 public void setUid(int n){uid=n;}
 public int getUid(){ return uid; }
 public void setUname(String s){ uname=s; }
 public String getUname(){ return uname; }
 public void setRole(String r){ role=r; }
```

```
3022  public String getRole(){  return role; }
3023 }
3024
3025 /*
3026   create table users
3027   ( userid number(5) primary key,
3028     uname varchar2(20),
3029    role varchar2(20)
3030    );
3031 */
3032 --------------------User.hbm.xml-------------------
3033 <?xml version="1.0"?>
3034 <!DOCTYPE hibernate-mapping PUBLIC
3035 "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
3036 "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
3037 <hibernate-mapping>
3038
3039   <class    name="User"   table="users" >
3040       <id   name="uid"  column="userid"  />
3041       <property   name="uname" />
3042       <property   name="role" />
3043   </class>
3044
3045 </hibernate-mapping>
3046 ================================================================================
3047 App 32)>>>>>>>>>>>>>>>>>>Spring with hibernate using annotations<<<<<<<<<<<<<<<<<<<
3048 ================================================================================
3049 ---------------------DemoInter.java-------------------------
3050 package p1;
3051 import java.util.*;
3052 public interface   DemoInter
3053 {
3054     public Iterator getData()throws Exception;
3055 }
3056
3057 ----------------------------------DemoImpl.java--------------------------
3058 package p1;
3059 import org.hibernate.*;
3060 import java.util.*;
3061 import org.springframework.stereotype.*;
3062 import org.springframework.beans.factory.annotation.Autowired;
3063
3064 @Component("d1")
3065 public class DemoImpl implements DemoInter
3066 {
3067 @Autowired
3068  private SessionFactory  sesfact;
3069
3070    public void setSesfact(SessionFactory f1)
3071     {
3072    sesfact=f1;
3073   }
3074
3075    public Iterator getData()throws Exception
3076    {
3077         Session ses=sesfact.openSession();
3078         Query query=ses.createQuery("from  User");
3079     Iterator i1=query.iterate();
3080     return i1;
3081    }
3082 }
```

```
3083
3084  ---------------------------------User.java----------------------------
3085  package p1;
3086  import javax.persistence.*;
3087
3088  @Entity
3089  @Table (name= "users")
3090  public class User
3091  {
3092    @Id
3093    @GeneratedValue
3094    @Column(name = "userid")
3095    private int uid;
3096
3097    @Column(name= "uname")
3098    private String uname;
3099
3100    @Column (name= "role")
3101     private String role;
3102
3103     public void setUid(int n)
3104     {
3105       uid=n;
3106     }
3107
3108     public int getUid(){ return uid; }
3109     public void setUname(String s){ uname=s; }
3110     public String getUname(){ return uname; }
3111     public void setRole(String r){ role=r; }
3112     public String getRole(){  return role; }
3113  }
3114  /*
3115  DBTable
3116  -----------
3117    create table users
3118    ( userid number(5) primary key,
3119      uname varchar2(20),
3120     role varchar2(20)
3121     );  */
3122
3123  ---------------------------------------SpringHB.xml---------------------------------
3124
3125  <beans xmlns="http://www.springframework.org/schema/beans"
3126        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3127        xmlns:context="http://www.springframework.org/schema/context"
3128        xsi:schemaLocation="http://www.springframework.org/schema/beans
3129        http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
3130        http://www.springframework.org/schema/context
3131        http://www.springframework.org/schema/context/spring-context-2.5.xsd">
3132
3133  <context:component-scan base-package="p1" />
3134
3135    <bean id="myds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
3136     <property name="driverClassName"><value>oracle.jdbc.driver.OracleDriver</value>
       </property>
3137     <property name="url"> <value>jdbc:oracle:thin:@localhost:1521:orcl</value> </property>
3138     <property name="username"> <value>scott</value> </property>
3139     <property name="password"> <value>tiger</value>
3140     </property>
3141    </bean>
3142
```

```
<bean id="sesfact" class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
   <property name="dataSource" ref="myds"/>
      <!-- use this to avoid configuring mapping file-->
    <property name="packagesToScan" value="p1"></property>
 </bean>

</beans>

----------SelectServlet.java----------

package p1;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import org.hibernate.*;
import java.util.*;
import org.springframework.beans.factory.*;
import org.springframework.context.*;
import org.springframework.context.support.*;
public class  SelectServlet  extends HttpServlet
{
 public void service(HttpServletRequest request,HttpServletResponse response)
  throws ServletException,IOException
  {

 PrintWriter out=response.getWriter();
  System.out.println("in service method of Serviet");
  try{
    ApplicationContext ctx=new ClassPathXmlApplicationContext("SpringHB.xml");
    demointer d1=(demointer)ctx.getBean("d1");
    Iterator i1=d1.getData();

   out.println("<body bgcolor=#ffffcc text=red>");
  out.println("<h1><center>all users</h1><hr><br><h3>");
 out.println("<table width=80% border=2>");
  while(i1.hasNext())
  {
      User u1=(User)i1.next();
      out.println("<tr><td>"+u1.getUid()+" <td>"+u1.getUname()+" <td>"+u1.getRole()+"</tr>");
  }
  out.println("</table>");
 }
  catch(Exception e)
  {
      out.println(e);
      e.printStackTrace();
  }
 }//service(-,-)
}//class
----------web.xml----------

<web-app>
  <servlet>
      <servlet-name>select</servlet-name>
      <servlet-class>p1.SelectServlet</servlet-class>
   </servlet>

   <servlet-mapping>
      <servlet-name>select</servlet-name>
      <url-pattern>/selectaction</url-pattern>
```

```
3204     </servlet-mapping>
3205
3206  </web-app>
3207  ----------------------------------------------------------
3208
3209  jars in lib
3210  -----------
3211  antlr-2.7.7.jar
3212  commons-logging.jar
3213  dom4j-1.6.1.jar
3214  hibernate-commons-annotations-4.0.1.Final.jar
3215  hibernate-core-4.1.8.Final.jar
3216  hibernate-ehcache-4.1.8.Final.jar
3217  hibernate-jpa-2.0-api-1.0.1.Final.jar
3218  javassist-3.15.0-GA.jar
3219  javax.persistence.jar
3220  jboss-logging-3.1.0.GA.jar
3221  jboss-transaction-api_1.1_spec-1.0.0.Final.jar
3222  ojdbc14.jar
3223  org.springframework.asm-3.1.1.RELEASE.jar
3224  org.springframework.aspects-3.1.1.RELEASE.jar
3225  org.springframework.beans-3.1.1.RELEASE.jar
3226  org.springframework.context.support-3.1.1.RELEASE.jar
3227  org.springframework.context-3.1.1.RELEASE.jar
3228  org.springframework.core-3.1.1.RELEASE.jar
3229  org.springframework.expression-3.1.1.RELEASE.jar
3230  org.springframework.jdbc-3.1.1.RELEASE.jar
3231  org.springframework.orm-3.1.1.RELEASE.jar
3232  org.springframework.transaction-3.1.1.RELEASE.jar
3233
3234  ================================Plain Java Mail=================================
3235  App 33) >>>>>>>>>>>>>>>>>>>>>>>>.programs on java mail API<<<<<<<<<<<<<<<<<<<<
3236  ================================================================================
3237  ----------------------------------------SendMail.java--------------------
3238  import javax.mail.*;
3239  import javax.mail.internet.*;
3240  import java.util.*;
3241
3242  public class SendMail
3243  {
3244  public static void main(String []args) throws Exception
3245     {
3246        //Creating Properties Objct
3247        Properties p=new Properties();
3248
3249        //adding protocol,mailserver address & the port number of the mailserver
3250        p.put("mail.transport.protocol","smtp");
3251        p.put("mail.smtp.host","localhost");
3252        p.put("mail.smtp.port","25");
3253
3254        //Creating the Session Object
3255        Session session=Session.getInstance(p);
3256
3257        //Creating and configuring the Message object
3258        Message msg=new MimeMessage(session);
3259
3260        msg.setFrom(new InternetAddress("ramesh"));
3261        InternetAddress address[]={new InternetAddress("raja")};
3262        msg.setRecipients(Message.RecipientType.TO,address);
3263        msg.setSentDate(new Date());
3264        msg.setSubject("open it to know it");
```

```
3265        msg.setText("Hello ! this is the first e-mail from ramesh using Java Mail API on wednesday ");
3266        //Sending the mail.
3267        Transport.send(msg);
3268        System.out.println("mail has been delivered");
3269     }//main
3270  }//class
3271
3272  ================================ReceiveMail.java================================
3273  /*
3274    Java Application to receive an E-mail using JAVA Mail API.
3275    ReceiveMail.java
3276  */
3277  import javax.mail.*;
3278  import javax.mail.internet.*;
3279  import java.util.*;
3280  public class ReceiveMail
3281  {
3282    public static void main(String []args) throws Exception
3283     {
3284        Properties p=new Properties();
3285        p.put("mail.transport.protocol","pop3");
3286        p.put("mail.pop.host","localhost");
3287        p.put("mail.pop.port","25");
3288
3289       Session session=Session.getInstance(p);
3290       Store store=session.getStore("pop3");
3291
3292       store.connect("localhost","raja","raja");
3293
3294        Folder folder=store.getDefaultFolder();
3295       Folder myinbox=folder.getFolder("INBOX");
3296        myinbox.open(Folder.READ_ONLY);
3297
3298        System.out.println("No. of messages in the inbox:"+myinbox.getMessageCount());
3299
3300        Message message=myinbox.getMessage(1);
3301        message.writeTo(System.out);
3302        myinbox.close(false);
3303        store.close();
3304     }
3305  }
3306  ============================DeleteMail.java================================
3307  /*
3308    Java Application to Delete  an E-mail using JAVA Mail API.
3309  */
3310  import javax.mail.*;
3311  import javax.mail.internet.*;
3312  import java.util.*;
3313  public class DeleteMail
3314  {
3315    public static void main(String []args) throws Exception
3316     {
3317        Properties properties=new Properties();
3318        properties.put("mail.trasport.protocol","pop");
3319        properties.put("mail.pop.port","110");
3320        properties.put("mail.host","localhost");
3321
3322        Session session=Session.getInstance(properties);
3323
3324        Store store=session.getStore("pop3");
3325
```

```
3326        store.connect("localhost","raja","raja");
3327
3328        Folder folder=store.getDefaultFolder();
3329        Folder myinbox=folder.getFolder("INBOX");
3330        myinbox.open(Folder.READ_WRITE);
3331
3332        System.out.println("No. of messages in the inbox:"+myinbox.getMessageCount());
3333        Message message=myinbox.getMessage(1);
3334        message.setFlag(Flags.Flag.DELETED,true);
3335        myinbox.close(true);
3336
3337         System.out.println("No. of messages in the inbox:"+myinbox.getMessageCount());
3338
3339        store.close();
3340    }
3341 }
3342
3343 ------------------------------------------------------------------------------------
3344 ====================================================================
3345 App 35)>>>>>>>>>>>>>>>>Application on Spring Mail<<<<<<<<<<<<<<<<<<<<<<<<<
3346 ====================================================================
3347 ----------------------------Student.java------------
3348 public interface Student {
3349    public void sendEmailMsg();
3350 }
3351 ------------------------------StudentImpl.java----------------------------------
3352 import org.springframework.mail.MailException;
3353 import org.springframework.mail.MailSender;
3354 import org.springframework.mail.SimpleMailMessage;
3355
3356 public class  StudentImpl implements Student
3357 {
3358  private MailSender mailSender;
3359  private SimpleMailMessage message;
3360
3361    public void setMailSender(MailSender mailSender) {
3362        this.mailSender = mailSender;
3363    }
3364
3365    public void setMessage(SimpleMailMessage message) {
3366        this.message = message;
3367    }
3368    public void sendEmailMsg()
3369    {
3370      try{
3371         mailSender.send(message);
3372       }
3373     catch(MailException me) {
3374        me.printStackTrace();
3375       }
3376    }//sendEmailMsg
3377
3378 }//class
3379 ===================================mailcfg.xml=======================
3380 <?xml version="1.0" encoding="UTF-8"?>
3381 <!DOCTYPE beans PUBLIC  "-//SPRING//DTD BEAN//EN"
3382 "http://www.springframework.org/dtd/spring-beans.dtd">
3383 <beans>
3384 <bean id="ms" class="org.springframework.mail.javamail.JavaMailSenderImpl">
3385   <property name="host"><value>localhost</value></property>
3386    <property name="port"><value>25</value></property>
```

```
    <property name="protocol"><value>smtp</value></property>
</bean>

<bean id="msg" class="org.springframework.mail.SimpleMailMessage">
  <property name="from"><value>ramesh</value></property>
  <property name="subject"><value>open it to know it</value></property>
  <property name="to"><value>nataraj</value></property>
  <property name="text"><value>first mail from spring based java mail</value></property>
</bean>

<bean id="sti" class="StudentImpl">
  <property name="mailSender"><ref bean="ms"/></property>
  <property name="message"><ref bean="msg"/></property>
</bean>
</beans>
==================================MailClient.java============================
import org.springframework.beans.factory.*;
import org.springframework.context.*;
import org.springframework.context.support.*;
public class  MailClient
{
 public static void main(String args[])
   {
ApplicationContext  ctx=new FileSystemXmlApplicationContext("mailcfg.xml");
 Student bobj=(Student)ctx.getBean("sti");
  bobj.sendEmailMsg();
System.out.println("mail has been delivered");
  }
}
-----------------------------------------
App 37) >>>>>>>>>Spring Mail with Attachments <<<<<<<<<<<<<<<<
================================================
----------------------------Student.java----------------------
public interface Student {
   public void sendEmailMsg();
}
-------------------------StudentImpl.java----------------------
import org.springframework.mail.*;
import org.springframework.mail.javamail.*;
import javax.mail.*;
import javax.mail.internet.*;
import org.springframework.core.io.*;

public class  StudentImpl implements Student
{
 private JavaMailSenderImpl mailSender;

  public void setMailSender(JavaMailSenderImpl mailSender) {
      this.mailSender = mailSender;
  }

 public void sendEmailMsg()
 {
   try{
    mailSender.send(new MimeMessagePreparator() {

     public void prepare(MimeMessage mimeMessage) throws MessagingException {

       MimeMessageHelper message=new MimeMessageHelper(mimeMessage, true, "UTF-8");
       message.setFrom("advani");
       message.setTo("kalam");
```

```
3448        message.setSubject("my subject");
3449        message.setText("hello This mail Content/body");
3450        message.addAttachment("abc.txt", new ClassPathResource("abc.txt"));
3451        }
3452     });
3453     }//try
3454    catch(MailException me) {
3455      me.printStackTrace();
3456    }
3457   }//sendEmailMsg
3458  }//class
3459  ------------------------------------ ---mailcfg.xml----------------------------------
3460  <?xml version="1.0" encoding="UTF-8"?>
3461  <!DOCTYPE beans PUBLIC  "-//SPRING//DTD BEAN//EN"
3462  "http://www.springframework.org/dtd/spring-beans.dtd">
3463  <beans>
3464    <bean id="ms" class="org.springframework.mail.javamail.JavaMailSenderImpl">
3465      <property name="host"><value>localhost</value></property>
3466      <property name="port"><value>25</value></property>
3467      <property name="protocol"><value>smtp</value></property>
3468    </bean>
3469
3470     <bean id="sti" class="StudentImpl">
3471      <property name="mailSender"><ref bean="ms"/></property>
3472     </bean>
3473  </beans>
3474  ------------------------------------MailClient.java----------------------------
3475  import org.springframework.beans.factory.*;
3476  import org.springframework.context.*;
3477  import org.springframework.context.support.*;
3478  public class   MailClient
3479  {
3480     public static void main(String args[])
3481      {
3482        ApplicationContext  ctx=new FileSystemXmlApplicationContext("mailcfg.xml");
3483        Student bobj=(Student)ctx.getBean("sti");
3484        bobj.sendEmailMsg();
3485        System.out.println("mail has been delivered");
3486      }
3487  }
3488  ========================================
3489  App 36)Prgs on plain java mail api (with Gmail Env.....)
3490  ========================================
3491  ------------------------------SendMail.java-----------------------------------------------------
3492  //Mail.java
3493  import java.util.*;
3494  import javax.mail.*;
3495  import javax.mail.internet.*;
3496
3497  public class  SendMail
3498   {
3499  public SendMail() {
3500  // mail properties outgoing server (gmail.com)
3501        Properties props = new Properties();
3502        props.put("mail.smtp.host", "smtp.gmail.com");
3503        props.put("mail.smtp.port", "465");
3504        props.put("mail.smtp.auth", "true");
3505        props.put("mail.smtp.socketFactory.class", "javax.net.ssl.SSLSocketFactory");
3506
3507        try {
3508        // create Session obj
```

```
3509        Authenticator auth = new SMTPAuthenticator():
3510
3511        Session session = Session.getInstance(props,auth);
3512
3513        //prepare mail msg
3514        MimeMessage msg = new MimeMessage(session);
3515        //set header values
3516        msg.setSubject("open to it know it");
3517        msg.setFrom(new InternetAddress("nataraz@gmail.com"));
3518        msg.addRecipient(Message.RecipientType.TO, new
            InternetAddress("testemail123@gmail.com"));
3519        //msg text
3520        msg.setText("mail from durga...");
3521
3522        Transport.send(msg);
3523        }//try
3524         catch (Exception ex) {
3525       ex.printStackTrace();
3526         }//catch
3527  }//constructor
3528
3529  private class SMTPAuthenticator extends javax.mail.Authenticator {
3530   public PasswordAuthentication getPasswordAuthentication() {
3531    return new PasswordAuthentication("testmail123@gmail.com","javaj2ee");
3532   }//method
3533  }//inner class
3534  public static void main(String[] args){
3535      SendMail mail=new SendMail();
3536       System.out.println("mail has been delivered");
3537   }//main
3538  }// class
3539  ----------GetMail.java----------
3540  import java.util.*;
3541  import javax.mail.*;
3542
3543  public class GetMail {
3544
3545  String host="pop.gmail.com";
3546        String emailfrom="testmail123@gmail.com";
3547        String password="javaj2ee";
3548        String port="995";
3549
3550   GetMail()
3551   {
3552    Properties props = new Properties();
3553
3554        props.put("mail.pop3.host", host);
3555        props.put("mail.pop3.port", port);
3556        props.put("mail.pop3.auth", "true");
3557        props.put("mail.pop3.socketFactory.class", "javax.net.ssl.SSLSocketFactory");
3558
3559     try {
3560        Authenticator auth = new SMTPAuthenticator();
3561        Session session = Session.getInstance(props, auth);
3562
3563         Store store = session.getStore("pop3s");
3564
3565        store.connect(host,emailfrom,password);
3566
3567         Folder folder = store.getFolder("inbox");
3568
```

```
3569        folder.open(Folder.READ_ONLY);
3570        Message[] msg = folder.getMessages();
3571
3572        System.out.println((msg.length+1)+" message found");
3573        for (int i = 1; i <= msg.length; i++) {
3574             Message mess=folder.getMessage(i);
3575             mess.writeTo(System.out);
3576        }//for
3577      }//try
3578     catch(Exception e)
3579      {e.printStackTrace();}
3580     }//constructor
3581
3582     private class SMTPAuthenticator extends javax.mail.Authenticator {
3583      public PasswordAuthentication getPasswordAuthentication() {
3584          return new PasswordAuthentication(emailfrom, password);
3585      }
3586    } //private class
3587
3588    public static void main(String args[])
3589     {
3590      new GetMail();
3591     }//main
3592    }//class
3593    ----------------------------------------------DeleteMail.java-----------------------------------
3594    import java.util.*;
3595    import javax.mail.*;
3596
3597    public class DeleteMail {
3598       String host="pop.gmail.com";
3599       String emailfrom="testmail123@gmail.com";
3600       String password="javaj2ee";
3601       String port="995";
3602
3603      DeleteMail()
3604      {
3605       Properties props = new Properties();
3606
3607       props.put("mail.pop3.host", host);
3608       props.put("mail.pop3.port", port);
3609       props.put("mail.pop3.auth", "true");
3610       props.put("mail.pop3.socketFactory.class", "javax.net.ssl.SSLSocketFactory");
3611
3612       try {
3613          Authenticator auth = new SMTPAuthenticator();
3614          Session session = Session.getInstance(props, auth);
3615
3616          Store store = session.getStore("pop3s");
3617
3618          store.connect(host,emailfrom,password);
3619
3620          Folder folder = store.getFolder("inbox");
3621
3622          folder.open(Folder.READ_WRITE);
3623          Message[] msg = folder.getMessages();
3624
3625          msg[0].setFlag(Flags.Flag.DELETED,true); //mark the msg for deletion
3626          folder.close(true); //close inbox with flag true
3627          store.close();
3628          }//try
3629          catch(Exception e)
```

```
        {e.printStackTrace();}
   }//constructor
    private class SMTPAuthenticator extends javax.mail.Authenticator {
     public PasswordAuthentication getPasswordAuthentication() {
         return new PasswordAuthentication(emailfrom, password);
     }//method
  } //private class

  public static void main(String args[])
   {
    new DeleteMail();
   }
  }
  ==================================================
  ================================================================================
  App 38) >>>>>>>>>>Sending email message using spring mail to gmail account<<<<<<<<<<
  ================================================================================
  ----------------------------Student.java-----------------
  public interface Student {
     public void sendEmailMsg();
  }
  ----------------------------StudentImpl.java----------------
  import org.springframework.mail.MailException;
  import org.springframework.mail.MailSender;
  import org.springframework.mail.SimpleMailMessage;

  public class  StudentImpl implements Student
  {
    private MailSender mailSender;
    private SimpleMailMessage message;

     public void setMailSender(MailSender mailSender) {
        this.mailSender = mailSender;
     }

     public void setMessage(SimpleMailMessage message) {
        this.message = message;
     }
   public void sendEmailMsg()
  {
     try{
        mailSender.send(message);
      }
     catch(MailException me) {
        me.printStackTrace();
      }
   }//sendEmailMsg
  }//class
  ----------------------------------mailcfg.xml-----------------------------------
  <?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="ms" class="org.springframework.mail.javamail.JavaMailSender">
      <property name="host" value="smtp.gmail.com"/>
      <property name="username" value="testmail123@gmail.com"/>
       <property name="password" value="javaj2ee"/>
       <property name="javaMailProperties">
          <props>
            <prop key="mail.transport.protocol">smtp</prop>
            <prop key="mail.smtp.auth">true</prop>
```

```
3691          <prop key="mail.smtp.SocketFactory.class">javax.net.ssl.SSLSocketFactory</prop>
3692        </props>
3693    </bean>
3694
3695  <bean id="msg" class="org.springframework.mail.SimpleMailMessage">
3696    <property name="from"><value>testmail123@gmail.com</value></property>
3697    <property name="subject"><value>open it to know it</value></property>
3698    <property name="to"><value>nataraz@gmail.com</value></property>
3699    <property name="text"><value>first mail from spring based java mail</value></property>
3700  </bean>
3701
3702  <bean id="sti" class="StudentImpl">
3703    <property name="mailSender"><ref bean="ms"/></property>
3704    <property name="message"><ref bean="msg"/></property>
3705  </bean>
3706  </beans>
3707  ----------MailClient.java----------
3708  import org.springframework.beans.factory.*;
3709  import org.springframework.context.*;
3710  import org.springframework.context.support.*;
3711  public class   MailClient
3712  {
3713  public static void main(String args[])
3714   {
3715   ApplicationContext  ctx=new FileSystemXmlApplicationContext("mailcfg.xml");
3716   BeanFactory  factory=(BeanFactory)ctx;
3717   Student beanref=(Student)factory.getBean("sti");
3718    beanref.sendEmailMsg();
3719   System.out.println("mail has been delivered");
3720    }
3721  }
3722  ----------
3723  ==========Plain RMI==========
3724  App 39) >>>>>>>>>>>>>>>>>>>>Plain RMI application<<<<<<<<<<<<<<<<<<
3725  ============================================================
3726  ----------Fact.java----------
3727  import java.rmi.*;
3728
3729  public interface Fact extends Remote
3730  {
3731   //declaration of business methods
3732   public long findFactorial(int val)throws RemoteException;
3733  }
3734  ----------ServerApp.java----------
3735  import java.rmi.*;
3736  import java.rmi.server.*;
3737
3738  class FactImpl extends UnicastRemoteObject implements Fact
3739  {
3740  public FactImpl() throws RemoteException
3741  {
3742        super();
3743        System.out.println("FactImpl:0-param constructor");
3744   }
3745
3746  //implement business method
3747  public long findFactorial(int val)throws RemoteException
3748   {
3749        System.out.println("FactImpl:findFactorial(-)");
3750        long res=1;
3751        for(int i=1;i<=val;++i)
```

```
3752        {
3753              res=res*i;
3754        }//for
3755        return res;
3756    }//findFactorial(-)
3757  }//class
3758
3759  public class ServerApp
3760  {
3761  public static void main(String args[])throws Exception
3762  {
3763    FactImpl bobj=new FactImpl();
3764
3765  //bind and register business object reference with RMIregistry
3766    Naming.bind("rmi://localhost:1099/india",bobj);
3767
3768        System.out.println("server application is ready");
3769
3770   }//main
3771  }//class
3772  ----------------------------------ClientApp.java--------------------
3773  //ClientApp.java
3774
3775  import java.rmi.*;
3776
3777  public class ClientApp
3778  {
3779  public static void main(String args[])throws Exception
3780   {
3781        //Gather business object reference from RMI registry
3782        Object obj=Naming.lookup("rmi://localhost:1099/india");
3783
3784        //type casting
3785        Fact bobj=(Fact)obj;
3786
3787        //call business methods
3788        System.out.println("Result is:"+bobj.findFactorial(5));
3789   }//main
3790  }//class
3791  ------------------------------
3792  ============================================================
3793  App 40)>>>>>>>>>>>Developing  spring rmi application<<<<<<<<<
3794  ============================================================
3795  ----------------------------------------first.java-----------------------------------
3796  public interface firstinter
3797  {
3798   public String wish(String uname);
3799  }
3800  ---------------------------------firstimpl.java-------------------------------
3801  public class firstimpl  implements  firstinter
3802  {
3803    public String wish(String uname)
3804    {
3805      return "Good Morning "+uname;
3806    }
3807  }
3808  ----------------------------------------first.xml-------------------------
3809  <?xml version="1.0" encoding="UTF-8"?>
3810  <!DOCTYPE beans PUBLIC  "-//SPRING//DTD BEAN//EN"
3811  "http://www.springframework.org/dtd/spring-beans.dtd">
3812  <beans>
```

```
3813   <bean id="f1"  class="firstimpl" />
3814    <bean id="pfb" class="org.springframework.remoting.rmi.RmiServiceExporter" >
3815     <property name="service" ref="f1"/>
3816     <property name="serviceName" value="india"/>
3817     <property name="serviceInterface" value="firstinter"/>
3818  </bean>
3819 </beans>
3820 ----------------------------------firstserver.java----------------------------------
3821 import org.springframework.context.*;
3822 import org.springframework.context.support.*;
3823 public class firstserver
3824 {
3825   public static void main(String args[])
3826   {
3827       ApplicationContext ctx=new FileSystemXmlApplicationContext("first.xml");
3828       System.out.println("server is started..");
3829   }
3830 }
3831 ------------------------------end of server prg-------------------------
3832 ==============================clientinter.java=====================
3833 public interface clientinter
3834 {
3835    public firstinter getInter();
3836 }
3837 ============================clientimpl.java=======================
3838 public class clientimpl implements clientinter
3839 {
3840   private firstinter firstin;
3841   public void setFirstin(firstinter f1)
3842   {
3843    firstin=f1;
3844   }
3845   public firstinter getInter()
3846    {
3847   return firstin;
3848   }
3849 }
3850 ==============================firstclient.xml=====================
3851 <?xml version="1.0" encoding="UTF-8"?>
3852 <!DOCTYPE beans PUBLIC  "-//SPRING//DTD BEAN//EN"
3853 "http://www.springframework.org/dtd/spring-beans.dtd">
3854 <beans>
3855  <bean id="pfb" class="org.springframework.remoting.rmi.RmiProxyFactoryBean" >
3856   <property name="serviceUrl" value="rmi://localhost:1099/india"/>
3857    <property name="serviceInterface" value="firstinter"/>
3858  </bean>
3859
3860  <bean id="c1"  class="clientimpl" >
3861    <property name="firstin" ref="pfb"/>
3862   </bean>
3863 </beans>
3864 ------------------------------firstclient.java-------------------------------
3865 import org.springframework.beans.factory.*;
3866 import org.springframework.context.*;
3867 import org.springframework.context.support.*;
3868 import java.rmi.*;
3869 public class firstclient
3870 {
3871   public static void main(String args[]) throws RemoteException
3872   {
3873   ApplicationContext ctx=new FileSystemXmlApplicationContext("firstclient.xml");
```

```
3874
3875     clientinter i1=(clientinter)ctx.getBean("c1");
3876
3877     firstinter f1=i1.getInter();
3878
3879      System.out.println(f1.wish("raj"));
3880     }
3881  }
3882  ----------------------------------------
3883  ==========================================================================
3884  App 42)>>>Spring Remoting Application Using Spring DAO in Businness Logic Developement<<<
3885  ==========================================================================
3886    ------ServerSide---------
3887  ----------------------------------Salary.java--------------------------------------
3888  public interface Salary
3889  {
3890   public int  getSalary(String uname);
3891  }
3892  ----------------------------SalaryImpl.java---------------------------
3893  import javax.sql.*;
3894  import org.springframework.jdbc.core.*;
3895
3896  public class SalaryImpl  implements  Salary
3897  {
3898    JdbcTemplate jt;
3899    public void setJt(JdbcTemplate jt)
3900    {
3901        this.jt=jt;
3902    }
3903
3904
3905   public int getSalary(String uname)
3906   {
3907    int sal=jt.queryForInt("select  SAL from emp where ename='"+uname+"'");
3908    return sal;
3909   }
3910  }
3911  ----------------------------------------salary.xml---------------------------------------
3912  <!DOCTYPE beans PUBLIC  "-//SPRING//DTD BEAN//EN"
3913  "http://www.springframework.org/dtd/spring-beans.dtd">
3914  <beans>
3915   <bean   id="d1" class="org.springframework.jdbc.datasource.DriverManagerDataSource" >
3916     <property  name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
3917     <property  name="url" value="jdbc:oracle:thin:@localhost:1521:orcl"/>
3918     <property  name="username" value="scott"/>
3919     <property  name="password" value="tiger/>
3920    </bean>
3921
3922    <bean   id="template" class="org.springframework.jdbc.core.JdbcTemplate" >
3923       <constructor-arg> <ref bean="d1"/></constructor-arg>
3924     </bean>
3925
3926     <bean  id="s1"   class="SalaryImpl" >
3927        <property name="jt"><ref bean="template"/></property>
3928      </bean>
3929
3930     <bean  id="pfb" class="org.springframework.remoting.rmi.RmiServiceExporter" >
3931      <property  name="service" ><ref bean="s1" /></property>
3932      <property  name="serviceName" ><value>india</value></property>
3933      <property name="serviceInterface"> <value>Salary</value></property>
3934     </bean>
```

```
  </beans>
  ----------ServerApp.java----------
  import org.springframework.context.*;
  import org.springframework.context.support.*;
   public class SalaryServerClient
   {
    public static void main(String args[])
     {
     ApplicationContext ctx=new FileSystemXmlApplicationContext("salary.xml");
     System.out.println("server is started..");
    }
}
 ==========
  Client code
 ==========
 ----------salaryclient.xml----------
  <?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="pfb" class="org.springframework.remoting.rmi.RmiProxyFactoryBean" >
    <property  name="serviceUrl" value="rmi://localhost:1099/india"/>
    <property name="serviceInterface" value="Salary"/>
  </bean>
  <bean  id="c1"   class="SalaryClient" >
   <property  name="sal" ref="pfb"/>
  </bean>
</beans>
----------SalaryClient.java----------
import org.springframework.beans.factory.*;
import org.springframework.context.*;
import org.springframework.context.support.*;
public class  SalaryClient
{
 static Salary sal=null;
 public void setSal(Salary sal)
 {
  this.sal=sal;
  }

   public static void main(String args[])
   {
       ApplicationContext ctx=new FileSystemXmlApplicationContext("salaryclient.xml");
       ctx.getBean("c1");
      System.out.println(sal.getSalary("SMITH"));
    }
}
----------
==========
App 43)>>>>>>>>>>>>>>>>>>>Http Invoker  application<<<<<<<<<<<<<<<<<<<<<<<
==========
  ----------serverside----------
==========httpinter.java==========
public interface   httpinter
{
  public String getWeek();
  public String getMonth();
}
==========httpimpl.java==========
import java.util.*;
public class   httpimpl implements httpinter
```

```
{
  public String getWeek()
  {
   String weeks[]=new String[]{"","Sunday","Monday","Tuesday","Wednesday",
                    "Thursday","Friday","Saturday"};
     Calendar  c1=Calendar.getInstance();
     int w=c1.get(Calendar.DAY_OF_WEEK);

    return   "current week :"+weeks[w];
  }
  public String getMonth()
  {
     String months[]=new String[]{"January","February","March","April","May","June","July",
                  "August","September","October","November","December"};
     Calendar  c1=Calendar.getInstance();
     int  m=c1.get(Calendar.MONTH);
     return   "current month :"+months[m];
   }
}
==================================remoting-servlet.xml==================
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC  "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>
 <bean  id="h1"   class="httpimpl"/>

 <bean   name="/DemoService"
 class="org.springframework.remoting.httpinvoker.HttpInvokerServiceExporter" >
  <property name="service"><ref  bean="h1"/></property>
   <property  name="serviceInterface"><value>httpinter</value></property>
 </bean>

 <bean id="surl"   class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="mappings">
     <props>
      <prop key="my.do">/DemoService</prop>
     </props>
    </property>
 </bean>

</beans>
==============================================web.xml==================
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
   http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
   version="2.4">
 <servlet>
  <servlet-name>remoting</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
 </servlet>
 <servlet-mapping>
  <servlet-name>remoting</servlet-name>
  <url-pattern>*.do</url-pattern>
 </servlet-mapping>
</web-app>
-----------------------------------------
========
clientapp
```

```
========
----------clientinter.java----------
public interface  clientinter
{
   public httpinter  getInter();
}
----------clientimpl.java----------
public class clientimpl implements  clientinter
{ private httpinter  httpin;

 public void setHttpin(httpinter f1)
 {
     httpin=f1;
 }
  public httpinter  getInter()
  {
     return httpin;
  }
}
----------Demo.xml----------
<?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="pfb"
 class="org.springframework.remoting.httpinvoker.HttpInvokerProxyFactoryBean" >
  <property name="serviceUrl" value="http://localhost:2020/HttpServerApp/my.do"/>
  <property  name="serviceInterface" value="httpinter"/>
 </bean>
 <bean  id="c1"  class="clientimpl" >
  <property  name="httpin" ref bean="pfb"/>
 </bean>
</beans>
----------DemoClient.java----------
import org.springframework.beans.factory.*;
import org.springframework.context.*;
import org.springframework.context.support.*;
public class  DemoClient
{
  public static void main(String args[])
   {
   ApplicationContext  ctx=new FileSystemXmlApplicationContext("Demo.xml");
   clientinter c1=(clientinter)ctx.getBean("c1");
   httpinter h1=c1.getInter();
   System.out.println(h1.getWeek());
   System.out.println(h1.getMonth());
   }
}
----------
==========
App 44)Plain JMS
==========
----------MyQueueSender.java----------
import javax.jms.*;
import javax.naming.*;
import java.util.Properties;

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

```
4116  Properties prop=new Properties();
4117
      prop.put(Context.INITIAL_CONTEXT_FACTORY,"com.sun.enterprise.naming.SerialInitContextFac
      tory");
4118  prop.put(Context.PROVIDER_URL,"iiop://localhost:4848");
4119
4120
4121   InitialContext ic=new InitialContext(prop);
4122   QueueConnectionFactory qcf=(QueueConnectionFactory)ic.lookup("QConFactJndi");
4123   QueueConnection qconn=qcf.createQueueConnection();
4124   QueueSession qsession=qconn.createQueueSession(false,Session.AUTO_ACKNOWLEDGE);
4125
4126   Queue queue=(Queue)ic.lookup("QueueJndi");
4127   QueueSender qsender=qsession.createSender(queue);
4128
4129       TextMessage message;
4130       String text;
4131
4132    int k=1;
4133     for(int i=1;i<=4;i++)
4134     {
4135      if(i==1)
4136        text=" Hot mail";
4137      else if(i==2)
4138         text="Amazon";
4139      else if(i==3)
4140         text="JGuru";
4141      else
4142         text="Bazee";
4143      for(int j=1;j<=5;j++)
4144        {
4145         message =qsession.createTextMessage();
4146         message.setStringProperty("Name",text);
4147         message.setText("This is behaviour"+k+"----------"+text);
4148        System.out.println("sending"+k+"\t"+text);
4149        k++;
4150      }//for
4151    }//for
4152
4153     qsession.close();
4154    qconn.close();
4155     }
4156  }
4157  ----------MyQueueReceiver.java----------
4158  import javax.jms.*;
4159  import javax.naming.*;
4160  import java.util.Properties;
4161
4162
4163  public class MyQueueReceiver implements MessageListener
4164  {
4165   public static void main(String args[])throws Exception
4166   {
4167      Properties prop=new Properties();
4168
      prop.put(Context.INITIAL_CONTEXT_FACTORY,"com.sun.enterprise.naming.SerialInitContextF
      actory");
4169     prop.put(Context.PROVIDER_URL,"iiop://localhost:4848");
4170
4171
4172        InitialContext ic=new InitialContext(prop);
```

```
4173        QueueConnectionFactory qcf=(QueueConnectionFactory)ic.lookup("QConFactJndi");
4174        QueueConnection qconn=qcf.createQueueConnection();
4175        QueueSession qsession=qconn.createQueueSession(false,Session.AUTO_ACKNOWLEDGE);
4176
4177        Queue queue=(Queue)ic.lookup("QueueJndi");
4178        String selector="Name IN('Bazee','Hotmail')";
4179        QueueReceiver qreceiver=qsession.createReceiver(queue,selector);
4180        MyQueueReceiver mq=new MyQueueReceiver();
4181        qreceiver.setMessageListener(mq);
4182
4183        qconn.start();
4184   }//main
4185   public void onMessage(Message msg)
4186   {
4187     try
4188     {
4189       TextMessage tmsg=(TextMessage)msg;
4190       System.out.println("Received Message is"+tmsg.getText());
4191      }//try
4192     catch(Exception e)
4193      {
4194        e.printStackTrace();
4195     }
4196   }
4197   }
4198   ----------TopicDemo.java----------
4199   import javax.jms.*;
4200   import javax.naming.*;
4201   import java.util.Properties;
4202
4203   public class TopicDemo
4204   {
4205    public static void main(String args[])throws Exception
4206    {
4207     Properties prop=new Properties();
4208
       prop.put(Context.INITIAL_CONTEXT_FACTORY,"com.sun.enterprise.naming.SerialInitContextFa
       ctory");
4209     prop.put(Context.PROVIDER_URL,"iiop://localhost:4848");
4210     Context ctx=new InitialContext(prop);
4211     TopicDemo tt=new TopicDemo();
4212     tt.doPub(ctx);
4213     ctx.close();
4214    }
4215    public void doPub(Context ctx)throws JMSException,NamingException
4216    {
4217        Topic topic=(Topic)ctx.lookup("TopicJndi");
4218
4219        TopicConnectionFactory tcf=(TopicConnectionFactory)ctx.lookup("TConFactJndi");
4220        TopicConnection tc=tcf.createTopicConnection();
4221
4222         TopicSession ts=tc.createTopicSession(false,TopicSession.AUTO_ACKNOWLEDGE);
4223         TopicPublisher tp=ts.createPublisher(topic);
4224
4225         tc.start();
4226
4227         TextMessage msg;
4228         msg=ts.createTextMessage("This is from my  home");
4229         System.out.println("Publishing msg on the Destination(Topic)");
4230
4231
```

```
        tp.publish(msg);
        tc.close();

    }
}
----------SubsciberDemo.java----------
import javax.jms.*;
import javax.naming.*;
import java.util.Properties;

public class SubscriberDemo
{
  public static void main(String args[])throws NamingException,JMSException
  {
        Properties prop=new Properties();

        prop.put(Context.INITIAL_CONTEXT_FACTORY,"com.sun.enterprise.naming.SerialInitContex
        tFactory");
        prop.put(Context.PROVIDER_URL,"iiop://localhost:4848");
        Context ctx=new InitialContext(prop);

        SubscriberDemo tt=new SubscriberDemo();
        tt.doSub(ctx);
        ctx.close();
  }//main
  public void doSub(Context ctx)throws JMSException,NamingException
  {
        Topic topic=(Topic)ctx.lookup("TopicJndi");

        TopicConnectionFactory tcf=(TopicConnectionFactory)ctx.lookup("TConFactJndi");
        TopicConnection tc=tcf.createTopicConnection();

        TopicSession ts=tc.createTopicSession(false,TopicSession.AUTO_ACKNOWLEDGE);
        TopicSubscriber tsub=ts.createSubscriber(topic);

        tc.start();


        TextMessage msg;
        msg=(TextMessage)tsub.receive();


        System.out.println("Received Message is:"+msg.getText());
        tc.close();
  }//doSub(-)
}//class
----------
==========Spring JMS==========
App 45) >>>>>>>>>>>>>>>On queue<<<<<<<<<<<<<<<<<<<<<<<<
==========
SenderApp
----------MessageSender.java----------
import javax.jms.*;
import org.springframework.jms.core.*;
public class MessageSender {

    private JmsTemplate jt;

    public MessageSender() {}

    public void setJt(JmsTemplate jt)
```

```
4291    {
4292       this.jt = jt;
4293    }
4294
4295    public void sendMessage()
4296    {
4297      MessageCreator creator = new MessageCreator()
4298      {
4299      public Message createMessage(Session session)
4300       {
4301        TextMessage message = null;
4302        try
4303         {
4304          message = session.createTextMessage();
4305          message.setText("Durga soft");
4306          message.setStringProperty("text", "Hello World");
4307        }
4308        catch (JMSException e)
4309        {
4310          e.printStackTrace();
4311        }
4312       return message;
4313     }
4314    };
4315    jt.send(creator);
4316   }
4317  }
4318  ----------spring.xml----------
4319  <?xml version="1.0" encoding="UTF-8"?>
4320  <!DOCTYPE beans PUBLIC  "-//SPRING//DTD BEAN//EN"
4321  "http://www.springframework.org/dtd/spring-beans.dtd">
4322  <beans>
4323   <bean id="cf"  class="org.springframework.jndi.JndiObjectFactoryBean">
4324    <property name="jndiEnvironment">
4325     <props>
4326       <prop
        key="java.naming.factory.initial">com.sun.enterprise.naming.SerialInitContextFactory</prop>
4327       <prop key="java.naming.provider.url">iiop://localhost:4343</prop>
4328     </props>
4329   </property>
4330   <property name="jndiName" value="ConFactJndi"/>
4331  </bean>
4332
4333   <bean id="q"  class="org.springframework.jndi.JndiObjectFactoryBean">
4334    <property name="jndiEnvironment">
4335      <props>
4336         <prop
          key="java.naming.factory.initial">com.sun.enterprise.naming.SerialInitContextFactory</pro
          p>
4337        <prop key="java.naming.provider.url">iiop://localhost:4343</prop>
4338     </props>
4339    </property>
4340    <property name="jndiName" value="QueueJndi"/>
4341   </bean>
4342
4343    <bean id="template" class="org.springframework.jms.core.JmsTemplate">
4344     <property name="connectionFactory" ref="cf"/>
4345     <property name="defaultDestination"  ref="q"/>
4346     </bean>
4347
4348    <bean id="ms"  class="MessageSender">
```

```
    <property name="jt"  ref="template"/>
  </bean>
</beans>
----------SendClient.java----------
import org.springframework.context.support.*;
public class SendClient
{
  public static void main(String args[])
  {
      FileSystemXmlApplicationContext ctx=new FileSystemXmlApplicationContext("spring.xml");
      MessageSender ms =(MessageSender)ctx.getBean("ms");
      ms.sendMessage();
      System.out.println("message sent");
  }
}
-----------------------------
ReceiverApp
============================
----------MessageReceiver.java----------
import javax.jms.*;
import org.springframework.jms.core.*;

public class MessageReceiver {
  private JmsTemplate jt;

    public MessageReceiver() {}

    public void setJt(JmsTemplate jt)
    {
       this.jt = jt;
    }

    public void receiveMessage()
    {
       Message message = jt.receive();
       TextMessage tmsg = null;
       if (message instanceof TextMessage)
       {
        tmsg=(TextMessage)message;
         try
         {
          System.out.println(tmsg.getText());
          System.out.println(tmsg.getStringProperty("text"));
         }
         catch (JMSException e)
         {
            e.printStackTrace();
         }
      }
    }
}
----------spring.xml----------
<!DOCTYPE beans PUBLIC  "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
<beans>

  <bean id="cf"  class="org.springframework.jndi.JndiObjectFactoryBean">
   <property name="jndiEnvironment">
    <props>
      <prop
      key="java.naming.factory.initial">com.sun.enterprise.naming.SerialInitContextFactory</prop>
```

```
4409        <prop key="java.naming.provider.url">iiop://localhost:4343</prop>
4410       </props>
4411      </property>
4412      <property name="jndiName" value="ConFactJndi"/>
4413     </bean>
4414
4415     <bean id="q"  class="org.springframework.jndi.JndiObjectFactoryBean">
4416      <property name="jndiEnvironment">
4417       <props>
4418        <prop
          key="java.naming.factory.initial">com.sun.enterprise.naming.SerialInitContextFactory</prop>
4419        <prop key="java.naming.provider.url">iiop://localhost:4343</prop>
4420       </props>
4421      </property>
4422      <property name="jndiName" value="QueueJndi"/>
4423     </bean>
4424
4425     <bean id="template" class="org.springframework.jms.core.JmsTemplate">
4426      <property name="connectionFactory" ref="cf"/>
4427      <property name="defaultDestination"  ref="q"/>
4428     </bean>
4429
4430     <bean id="ms"  class="MessageReceiver">
4431      <property name="jt"  ref="template"/>
4432     </bean>
4433    </beans>
4434    ------------------------ReceiverClient.java----------------------------
4435     import org.springframework.context.support.*;
4436      public class ReceiveClient
4437      {
4438       public static void main(String args[])
4439       {
4440        FileSystemXmlApplicationContext ctx=new FileSystemXmlApplicationContext("spring.xml");
4441        MessageReceiver mr =(MessageReceiver)ctx.getBean("ms");
4442           mr.receiveMessage();
4443           System.out.println("done");
4444       }
4445      }
4446    ----------------------------------------------------
4447    ====================================================================
4448    App 46) >>>>>>>>>>>>>>>>>>>>>>On Topic SenderApp<<<<<<<<<<<<<<<<<<<<<<
4449    ====================================================================
4450    ----------------------------MessageSender.java----------------------
4451    import javax.jms.*;
4452    import org.springframework.jms.core.*;
4453    public class MessageSender {
4454
4455       private JmsTemplate jt;
4456
4457       public MessageSender() {}
4458
4459       public void setJt(JmsTemplate jt)
4460       {
4461          this.jt = jt;
4462       }
4463
4464      public void sendMessage()
4465       {
4466       MessageCreator creator = new MessageCreator()
4467        {
4468        public Message createMessage(Session session)
```

```
4469      {
4470       TextMessage message = null;
4471       try
4472        {
4473         message = session.createTextMessage();
4474         message.setText("DurgaSoft1");
4475         message.setStringProperty("text", "Hello World");
4476        }
4477       catch (JMSException e)
4478      {
4479       e.printStackTrace();
4480       }
4481       return message;
4482     }};
4483     jt.send(creator);
4484     }
4485  }
4486  ----------spring.xml----------
4487  <?xml version="1.0" encoding="UTF-8"?>
4488  <!DOCTYPE beans PUBLIC  "-//SPRING//DTD BEAN//EN"
4489  "http://www.springframework.org/dtd/spring-beans.dtd">
4490  <beans>
4491   <bean id="cf"  class="org.springframework.jndi.JndiObjectFactoryBean">
4492    <property name="jndiEnvironment">
4493     <props>
4494       <prop
          key="java.naming.factory.initial">com.sun.enterprise.naming.SerialInitContextFactory</prop
          >
4495       <prop key="java.naming.provider.url">iiop://localhost:4343</prop>
4496     </props>
4497    </property>
4498    <property name="jndiName" value="ConFactJndi"/>
4499   </bean>
4500
4501   <bean id="q"  class="org.springframework.jndi.JndiObjectFactoryBean">
4502    <property name="jndiEnvironment">
4503     <props>
4504       <prop
          key="java.naming.factory.initial">com.sun.enterprise.naming.SerialInitContextFactory</prop>
4505       <prop key="java.naming.provider.url">iiop://localhost:4343</prop>
4506     </props>
4507    </property>
4508    <property name="jndiName" value="TopicJndi"/>
4509   </bean>
4510
4511    <bean id="template" class="org.springframework.jms.core.JmsTemplate">
4512     <property name="connectionFactory" ref="cf"/>
4513     <property name="defaultDestination"  ref="q"/>
4514    </bean>
4515
4516   <bean id="ms"  class="MessageSender">
4517    <property name="jt"  ref="template"/>
4518   </bean>
4519  </beans>
4520  ----------SendClient.java----------
4521  import org.springframework.context.support.*;
4522  public class SendClient
4523  {
4524  public static void main(String args[])
4525   {
4526       FileSystemXmlApplicationContext ctx=new FileSystemXmlApplicationContext("spring.xml");
```

```
4527        MessageSender ms =(MessageSender)ctx.getBean("ms");
4528        ms.sendMessage();
4529        System.out.println("message sent");
4530   }
4531
4532  }
4533  =========
4534  ReceiverApp
4535  =========
4536  ----------MessageReceiver.java----------
4537  import javax.jms.*;
4538  import org.springframework.jms.core.*;
4539
4540  public class MessageReceiver {
4541   private JmsTemplate jt;
4542
4543     public MessageReceiver() {}
4544
4545     public void setJt(JmsTemplate jt)
4546     {
4547        this.jt = jt;
4548     }
4549
4550     public void receiveMessage()
4551     {
4552       Message message = jt.receive();
4553       TextMessage tmsg = null;
4554       if (message instanceof TextMessage)
4555       {
4556        tmsg=(TextMessage)message;
4557        try
4558        {
4559         System.out.println(tmsg.getText());
4560         System.out.println(tmsg.getStringProperty("text"));
4561        }
4562        catch (JMSException e)
4563        {
4564          e.printStackTrace();
4565        }
4566       }
4567     }
4568  }
4569  ----------spring.xml----------
4570  <!DOCTYPE beans PUBLIC  "-//SPRING//DTD BEAN//EN"
4571  "http://www.springframework.org/dtd/spring-beans-2.0.dtd">
4572  <beans>
4573
4574   <bean id="cf"  class="org.springframework.jndi.JndiObjectFactoryBean">
4575    <property name="jndiEnvironment">
4576     <props>
4577      <prop
       key="java.naming.factory.initial">com.sun.enterprise.naming.SerialInitContextFactory</prop>
4578      <prop key="java.naming.provider.url">iiop://localhost:4343</prop>
4579     </props>
4580    </property>
4581    <property name="jndiName" value="ConFactJndi"/>
4582   </bean>
4583
4584   <bean id="q"  class="org.springframework.jndi.JndiObjectFactoryBean">
4585    <property name="jndiEnvironment">
4586     <props>
```

```
4587      <prop
          key="java.naming.factory.initial">com.sun.enterprise.naming.SerialInitContextFactory</prop>
4588      <prop key="java.naming.provider.url">iiop://localhost:4343</prop>
4589     </props>
4590    </property>
4591    <property name="jndiName" value="TopicJndi"/>
4592   </bean>
4593
4594    <bean id="template" class="org.springframework.jms.core.JmsTemplate">
4595     <property name="connectionFactory" ref="cf"/>
4596     <property name="defaultDestination"  ref="q"/>
4597   </bean>
4598
4599   <bean id="ms"  class="MessageReceiver">
4600    <property name="jt"  ref="template"/>
4601   </bean>
4602  </beans>
4603  ----------------------------------ReceiveClient .java----------------------
4604  import org.springframework.context.support.*;
4605  public class ReceiveClient
4606  {
4607   public static void main(String args[])
4608    {
4609     FileSystemXmlApplicationContext ctx=new FileSystemXmlApplicationContext("spring.xml");
4610        MessageReceiver mr =(MessageReceiver)ctx.getBean("ms");
4611        mr.receiveMessage();
4612        System.out.println("done");
4613    }
4614  }
4615  ------------------------------------------------------
4616  ================================================================================
4617  App 47)>>>>>>>>>>>>>>>>>>>>>>>>Spring  webservices application<<<<<<<<<<<<<<<<<<<<<<
4618  ================================================================================
4619  ==========
4620  ServerApp
4621  =========
4622  ----------------------------------CalculatorInter.java----------------------
4623  package com.ds;
4624  public interface CalculatorInter {
4625        public String sum(int x,int y);
4626
4627  }
4628  ------------------------------CalculatorImpl.java--------------------------
4629  package com.ds;
4630
4631  import javax.jws.WebMethod;
4632  import javax.jws.WebService;
4633  import javax.jws.soap.SOAPBinding;
4634  import org.springframework.stereotype.Service;
4635
4636  @SOAPBinding(style=SOAPBinding.Style.RPC, use=SOAPBinding.Use.LITERAL,
     parameterStyle=SOAPBinding.ParameterStyle.WRAPPED)
4637  @WebService(serviceName="CalcyWs")
4638  @Service("Calcy")
4639  public class CalculatorImpl implements CalculatorInter {
4640        @WebMethod
4641  public String sum(int x, int y) {
4642        return ""+(x+y);
4643        }
4644
4645  }
```

```
4646
4647  ----------------------------------servercfg.xml-------------------------
4648  <beans xmlns="http://www.springframework.org/schema/beans"
4649   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:p="http://www.springframework.org/schema/p"
4650   xmlns:aop="http://www.springframework.org/schema/aop"
      xmlns:context="http://www.springframework.org/schema/context"
4651   xmlns:jee="http://www.springframework.org/schema/jee"
      xmlns:tx="http://www.springframework.org/schema/tx"
4652   xmlns:task="http://www.springframework.org/schema/task"
4653   xsi:schemaLocation="http://www.springframework.org/schema/aop
      http://www.springframework.org/schema/aop/spring-aop-3.0.xsd
4654    http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
4655    http://www.springframework.org/schema/context
       http://www.springframework.org/schema/context/spring-context-3.0.xsd
4656    http://www.springframework.org/schema/jee
       http://www.springframework.org/schema/jee/spring-jee-3.0.xsd
4657    http://www.springframework.org/schema/tx
       http://www.springframework.org/schema/tx/spring-tx-3.0.xsd
4658    http://www.springframework.org/schema/task
       http://www.springframework.org/schema/task/spring-task-3.0.xsd">
4659   <context:component-scan base-package="com.ds"/>
4660     <bean class="org.springframework.remoting.jaxws.SimplejaxWsServiceExporter">
4661         <property name="baseAddress" value="http://localhost:5678"/>
4662    </bean>
4663   </beans>
4664   ----------------------------------ServiceExposer.java-------------------------
4665  package com.ds;
4666  import org.springframework.context.support.*;
4667  public class ServiceExposer {
4668    public static void main(String[] args)
4669    {
4670
4671     FileSystemXmlApplicationContext ctx=new FileSystemXmlApplicationContext("ServerCfg.xml");
4672     System.out.println("Service Exposed successfully");
4673
4674    }
4675  }
4676
4677  /* to access wsdl type the following
4678  http://localhost:5678/CalcyWs?WSDL
4679  */
4680  //jars needed
4681  /*
4682  org.sf.context.jar,
4683  beans.jar,
4684  core.jar,
4685  commons-logging.jar,-------->collect from spring2
4686  asm.jar,
4687  expression.jar,
4688  web.jar
4689  */
4690  ==========
4691  ClientApp
4692  ==========
4693  ----------------------------------ClientCfg.xml-------------------------
4694  <beans xmlns="http://www.springframework.org/schema/beans"
4695   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:p="http://www.springframework.org/schema/p"
4696   xmlns:aop="http://www.springframework.org/schema/aop"
```

```
      xmlns:context="http://www.springframework.org/schema/context"
4697  xmlns:jee="http://www.springframework.org/schema/jee"
      xmlns:tx="http://www.springframework.org/schema/tx"
4698  xmlns:task="http://www.springframework.org/schema/task"
4699  xsi:schemaLocation="http://www.springframework.org/schema/aop
      http://www.springframework.org/schema/aop/spring-aop-3.0.xsd
4700   http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
4701   http://www.springframework.org/schema/context
      http://www.springframework.org/schema/context/spring-context-3.0.xsd
4702   http://www.springframework.org/schema/jee
      http://www.springframework.org/schema/jee/spring-jee-3.0.xsd
4703   http://www.springframework.org/schema/tx
      http://www.springframework.org/schema/tx/spring-tx-3.0.xsd
4704   http://www.springframework.org/schema/task
      http://www.springframework.org/schema/task/spring-task-3.0.xsd">
4705
4706    <bean id="gs" class="org.springframework.remoting.jaxws.JaxWsPortProxyFactoryBean">
4707      <property name="wsdlDocumentUrl" value="http://localhost:5678/CalcyWs?WSDL"/>
4708      <property  name="serviceName" value="CalcyWs"/>
4709      <property name="namespaceUri" value="http://ds.com/"/>
4710      <property name="serviceInterface" value="p1.CalculatorInter"/>
4711      <property name="portName" value="CalculatorImplPort"/>
4712    </bean>
4713  </beans>
4714  ----------------------------------------ServiceClient.java---------------------
4715  package p1;
4716  import org.springframework.context.support.FileSystemXmlApplicationContext;
4717  public class ServiceClient {
4718  public static void main(String[] args) throws Exception
4719  {
4720   FileSystemXmlApplicationContext ctx=new FileSystemXmlApplicationContext("ClientCfg.xml");
4721   CalculatorInter obj=(CalculatorInter)ctx.getBean("gs");
4722   System.out.println(obj.sum(15, 10));
4723  }
4724  }
4725  /*
4726  jar files needed(collect from spring3 soft
4727  ***************
4728  context.jar
4729  beans.jar
4730  core.jar
4731  asm.jar
4732  expression.jar
4733  web.jar
4734  aop.jar
4735
4736  *********Collect separtely
4737
4738  commons-logging.jar(from spring2)
4739  aopalliance-1.0.jar (internet)
4740  */
4741   ----------------------------------------
4742   ============================================================
4743   App 48) >>>>>>>>>>>>>Timer Application  using  core java<<<<<<<<<<<<<<<<<<<<
4744  ============================================================
4745  ------------------------------task1.java----------------------------
4746  import java.util.*;
4747  public class  task1  extends TimerTask
4748  {
4749   public void run()
```

```
4750    {
4751     System.out.println("--------------------");
4752     System.out.println("welcome to durga ");
4753     System.out.println("--------------------");
4754     System.out.println("");
4755    System.out.println("");
4756     }
4757  }
4758  ----------------------------scheduledemo.java---------------------
4759  import java.util.*;
4760  public class scheduledemo
4761  {
4762   public static void main(String args[])
4763   {
4764   Timer t1=new Timer();
4765  //t1.schedule(new task1(),5000);
4766
4767   //t1.schedule(new task1(),5000,3000);
4768
4769   Calendar c1=Calendar.getInstance();
4770   c1.set(2005,Calendar.AUGUST,5,12,04,20);
4771  t1.schedule(new task1(),c1.getTime());
4772   }
4773  }
4774  ------------------------------------------------------------------------------------------------
4775  ================================================================
4776  App 49) >>>>>>>>>>>>>>>>>>>>>>Timer Appliation Using Spring<<<<<<<<<<<<<<<<<
4777  ================================================================
4778  -------------------------task1.java---------------------------------
4779  import java.util.*;
4780  public class  task1  extends TimerTask
4781  {
4782     public void run()
4783     {
4784        System.out.println("--------------------");
4785        System.out.println("welcome to durga");
4786        System.out.println("--------------------");
4787        System.out.println("");
4788        System.out.println("");
4789     }
4790  }
4791  ----------------------task2.java-----------------------------------
4792  import java.util.*;
4793  import java.io.*;
4794  public class  task2  extends TimerTask
4795  {
4796   public void run()
4797    { try{
4798        FileWriter  fp=new FileWriter("demofile.txt",true);
4799        fp.write(new Date().toString()+"\n");
4800        fp.close();
4801        System.out.println("--------------------");
4802        System.out.println("data is stored");
4803        System.out.println("--------------------");
4804        System.out.println("");
4805        System.out.println("");
4806        }catch(Exception e){}
4807     }
4808  }
4809  ----------------------------Timer.xml------ ----------------------------
4810  <?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="job1"  class="task1" />
<bean  id="job2"  class="task2" />
 <bean  id="schedule1"  class="org.springframework.scheduling.timer.ScheduledTimerTask" >
  <property name="delay"><value>5000</value></property>
  <property name="period"><value>2000</value></property>
  <property name="timerTask" ><ref bean="job1" /></property>
</bean>

<bean  id="schedule2"  class="org.springframework.scheduling.timer.ScheduledTimerTask" >
  <property name="delay"> <value>5000</value></property>
  <property name="period"><value>3000</value></property>
  <property name="timerTask" ><ref bean="job2" /></property>
</bean>

<bean   id="stt"  class="org.springframework.scheduling.timer.TimerFactoryBean">
 <property name="scheduledTimerTasks">
  <list>
    <ref  bean="schedule1" />
    <ref  bean="schedule2" />
  </list>
 </property>
 </bean>

</beans>
----------------------------------scheduledemo.java---------------------------------
import org.springframework.context.*;
import org.springframework.context.support.*;
import java.io.*;
public class   scheduledemo
{
 public static void main(String args[]) throws IOException
  {
  ApplicationContext  ctx=new FileSystemXmlApplicationContext("Timer.xml");
  System.in.read();
  }
}
-------------------------------------------------------------------------------------------------------
=========================================================================
App 50) >>>>>>Spring Based Timer Appliction (Applying Timer on user-defined method)<<<<<<
=========================================================================
----------------------------------task1.java-----------------------------------
public class  task1
{
  public void  method1()
   {
      System.out.println("--------------------");
      System.out.println("welcome to Durga");
      System.out.println("--------------------");
      System.out.println("");
      System.out.println("");
   }
}
-------------------------------task2.java-----------------------------------
import java.io.*;
import java.util.*;
public class  task2
{
   public void method2()
```

```
    {
     try{
       FileWriter  fp=new FileWriter("demofile.txt",true);
       fp.write(new Date().toString()+"\n");
       fp.close();
       System.out.println("--------------------");
       System.out.println("data is stored");
       System.out.println("--------------------");
       System.out.println("");
       System.out.println("");
       }catch(Exception e){}
    }
}
---------------------------------Timer.xml--------------------------------
<?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="job1"  class="task1" />
   <bean  id="job2"  class="task2" />

 <bean id="mt1"
 class="org.springframework.scheduling.timer.MethodInvokingTimerTaskFactoryBean">
   <property  name="targetObject"><ref bean="job1" /></property>
   <property  name="targetMethod"><value>method1</value> </property>
</bean>

<bean id="mt2"
class="org.springframework.scheduling.timer.MethodInvokingTimerTaskFactoryBean">
   <property  name="targetObject"><ref bean="job2" /> </property>
   <property  name="targetMethod"><value>method2</value></property>
</bean>

 <bean  id="schedule1"  class="org.springframework.scheduling.timer.ScheduledTimerTask" >
  <property name="delay"> <value>5000</value></property>
  <property name="period"><value>2000</value></property>
  <property name="timerTask" ><ref bean="mt1" /></property>
</bean>

<bean  id="schedule2"  class="org.springframework.scheduling.timer.ScheduledTimerTask" >
 <property name="delay"> <value>5000</value></property>
 <property name="period"><value>3000</value></property>
 <property name="timerTask" ><ref bean="mt2" /></property>
</bean>

<bean  id="stt"  class="org.springframework.scheduling.timer.TimerFactoryBean">
 <property name="scheduledTimerTasks">
  <list>
    <ref  bean="schedule1" />
    <ref  bean="schedule2" />
  </list>
 </property>
</bean>
</beans>
-----------------------------------scheduledemo.java-----------------------------------
import org.springframework.context.*;
import org.springframework.context.support.*;
import java.io.*;
public class  scheduledemo
{
  public static void main(String args[])
```

```
4931    {
4932    ApplicationContext  ctx=new FileSystemXmlApplicationContext("Timer.xml");
4933    }
4934  }
4935  ----------------------------------------------
4936  ================================================================================
4937  App 51)>>>>>>>>>>>>>>Struts with Spring  Using Approach No:1<<<<<<<<<<<<<<<<<<<
4938  ================================================================================
4939  ----------------------------------index1.jsp---------------------------
4940  <%@ taglib  prefix="html"  uri="struts-html"  %>
4941  <html>
4942  <body bgcolor=#ffffcc text=green>
4943  <h1><center>struts framewrok</h1><hr><br><h3>
4944  <html:form action="/demo">
4945    enter user name :<html:text  property="uname"  /> <br><br>
4946    <html:submit  value="wish" />
4947  </html:form>
4948  </body>
4949  </html>
4950  ----------------------------------web.xml-------------------------------------
4951  <!DOCTYPE web-app
4952   PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
4953   "http://java.sun.com/j2ee/dtds/web-app_2_3.dtd">
4954
4955  <web-app>
4956   <servlet>
4957       <servlet-name>action</servlet-name>
4958       <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
4959       <init-param>
4960            <param-name>config</param-name>
4961            <param-value>/WEB-INF/struts-config.xml</param-value>
4962       </init-param>
4963       <load-on-startup>1</load-on-startup>
4964    </servlet>
4965    <servlet-mapping>
4966       <servlet-name>action</servlet-name>
4967       <url-pattern> *.do </url-pattern>
4968    </servlet-mapping>
4969    <welcome-file-list>
4970       <welcome-file>index1.jsp</welcome-file>
4971    </welcome-file-list>
4972    <taglib>
4973       <taglib-uri>struts-html</taglib-uri>
4974       <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
4975    </taglib>
4976  </web-app>
4977  ----------------------------struts-config.xml---------------------------
4978  <?xml version="1.0" encoding="UTF-8"?>
4979  <!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts Configuration
     1.1//EN"
4980                    "http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">
4981  <struts-config>
4982   <form-beans>
4983    <form-bean  name="demofrm"   type="demopack.DemoForm" />
4984   </form-beans>
4985   <action-mappings>
4986    <action  path="/demo"  name="demofrm"  type="demopack.DemoAction">
4987       <forward  name="success"  path="/result.jsp" />
4988    </action>
4989  </action-mappings>
4990   <plug-in  className="org.springframework.web.struts.ContextLoaderPlugIn">
```

```
4991  </plug-in>
4992  </struts-config>
4993  ----------DemoForm.java----------
4994  package demopack;
4995  import org.apache.struts.action.*;
4996  public class DemoForm  extends ActionForm
4997  {
4998   private  String uname;
4999
5000    public void setUname(String s){uname=s;}
5001    public String getUname(){return uname;}
5002  }
5003  ----------DemoAction.java----------
5004  package demopack;
5005  import javax.servlet.*;
5006  import javax.servlet.http.*;
5007  import java.io.*;
5008  import org.apache.struts.action.*;
5009  import org.springframework.web.struts.*;
5010  import org.springframework.web.context.*;
5011
5012  public class DemoAction  extends ActionSupport
5013  {
5014   public ActionForward execute(ActionMapping mapping,ActionForm form,HttpServletRequest
      request,HttpServletResponse response)
5015   throws ServletException,IOException
5016    {
5017   System.out.println("In execute method of Action class");
5018   DemoForm   f1=(DemoForm)form;
5019   String  uname=f1.getUname();
5020
5021  WebApplicationContext context=getWebApplicationContext();
5022   ModelInter m1=(ModelInter)context.getBean("d1");
5023   String msg=m1.getWish()+"   "+uname;
5024   request.setAttribute("message",msg);
5025
5026   return mapping.findForward("success");
5027   }
5028  }
5029  ----------result.jsp----------
5030  <%@ page language="java" %>
5031  <%@ taglib  prefix="html"  uri="struts-html"  %>
5032  <html>
5033  <body bgcolor=#ffffcc text=green>
5034  <h1><center>struts framewrok</h1><hr><br><h3>
5035  <%=request.getAttribute("message")  %>
5036  <br><br>
5037  <html:link  href="index1.jsp" >home</html:link>
5038  </body>
5039  </html>
5040  ----------ModelInter.java----------
5041  package demopack;
5042  public interface ModelInter
5043  {
5044        public  String getWish();
5045  }
5046  ----------ModelImpl.java----------
5047  package demopack;
5048  import java.util.*;
5049  public class ModelImpl implements ModelInter
5050  {
```

```
5051  public  String getWish()
5052   {
5053   System.out.println("getWish() ModelImpl");
5054       Calendar c1=Calendar.getInstance();
5055        int h=c1.get(Calendar.HOUR_OF_DAY);
5056
5057        if(h  < 12)
5058        return "Good Morning";
5059        else if( h < 17)
5060        return "Good Afternoon";
5061        else
5062        return "Good Evening";
5063   }
5064  }
5065  ----------------------------------action-servlet.xml---------------------------------
5066  <?xml version="1.0" encoding="UTF-8"?>
5067  <!DOCTYPE beans PUBLIC  "-//SPRING//DTD BEAN//EN"
5068  "http://www.springframework.org/dtd/spring-beans.dtd">
5069  <beans>
5070    <bean   id="d1" class="demopack.ModelImpl"/>
5071  </beans>
5072  ==========================================================================
5073  App  52)>>>>>>>>>>>>>>>>>struts-spring (Approach No:2)<<<<<<<<<<<<<<<<<<<<<<<<<
5074  ==========================================================================
5075  ---------------------------------------index1.jsp-----------------------------------
5076  <%@ taglib  prefix="html"  uri="struts-html"  %>
5077  <html>
5078  <body bgcolor=#ffffcc text=green>
5079  <h1><center>struts framewrok</h1><hr><br><h3>
5080  <html:form action="/demo">
5081    enter user name :<html:text  property="uname"  /> <br><br>
5082    <html:submit  value="wish" />
5083  </html:form>
5084  </body>
5085  </html>
5086  ---------------------------------web.xml---------------------------------
5087  <!DOCTYPE web-app
5088   PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
5089   "http://java.sun.com/j2ee/dtds/web-app_2_3.dtd">
5090  <web-app>
5091   <servlet>
5092        <servlet-name>action</servlet-name>
5093        <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
5094        <init-param>
5095          <param-name>config</param-name>
5096          <param-value>/WEB-INF/struts-config.xml</param-value>
5097        </init-param>
5098
5099        <load-on-startup>1</load-on-startup>
5100    </servlet>
5101
5102    <servlet-mapping>
5103        <servlet-name>action</servlet-name>
5104        <url-pattern> *.do </url-pattern>
5105    </servlet-mapping>
5106
5107    <welcome-file-list>
5108        <welcome-file>index1.jsp</welcome-file>
5109    </welcome-file-list>
5110    <taglib>
5111        <taglib-uri>struts-html</taglib-uri>
```

```
5112        <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
5113     </taglib>
5114  </web-app>
5115  ----------------------------------------struts-config.xml----------------------------------
5116  <?xml version="1.0" encoding="UTF-8"?>
5117  <!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts Configuration
      1.1//EN"
5118                          "http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">
5119  <struts-config>
5120   <form-beans>
5121     <form-bean  name="demofrm"   type="demopack.DemoForm" />
5122   </form-beans>
5123    <action-mappings>
5124      <action  path="/demo"  name="demofrm"
        type="org.springframework.web.struts.DelegatingActionProxy" >
5125        <forward  name="success"  path="/result.jsp" />
5126     </action>
5127    </action-mappings>
5128     <plug-in className="org.springframework.web.struts.ContextLoaderPlugIn"></plug-in>
5129  </struts-config>
5130  -------------------action-servlet.xml-------------------------------
5131  <?xml version="1.0" encoding="UTF-8"?>
5132  <!DOCTYPE beans PUBLIC  "-//SPRING//DTD BEAN//EN"
5133  "http://www.springframework.org/dtd/spring-beans.dtd">
5134  <beans>
5135   <bean  name="/demo"   class="demopack.DemoAction" >
5136     <property  name="model"><ref bean="d1" /></property>
5137    </bean>
5138    <bean   id="d1" class="demopack.ModelImpl" >
5139   </bean>
5140  </beans>
5141  ----------------------------DemoForm.java-----------------------------------
5142  package demopack;
5143  import org.apache.struts.action.*;
5144  public class DemoForm  extends ActionForm
5145  {
5146    private  String uname;
5147    public void setUname(String s)
5148     {
5149        uname=s;
5150     }
5151      public String getUname()
5152      {
5153        return uname;
5154     }
5155  }
5156  ------------------------------DemoAction.java---------------------------------
5157  package demopack;
5158  import javax.servlet.*;
5159  import javax.servlet.http.*;
5160  import java.io.*;
5161  import org.apache.struts.action.*;
5162  public class DemoAction  extends Action
5163  {
5164     private    ModelInter  model;
5165
5166    public void setModel(ModelInter i1){ model=i1;}
5167
5168    public ActionForward execute(ActionMapping mapping,ActionForm form,
5169                          HttpServletRequest request,HttpServletResponse response)
5170                         throws ServletException,IOException
```

```
5171    {
5172    DemoForm   f1=(DemoForm)form;
5173    String  uname=f1.getUname();
5174
5175    String msg=model.getWish()+" "+ uname;
5176
5177     request.setAttribute("message",msg);
5178     return mapping.findForward("success");
5179    }
5180  }
5181  ------------------------------ModelInter.java------------------------------
5182  package demopack;
5183  public interface ModelInter
5184  {
5185        public  String getWish();
5186  }
5187  ------------------------------------------ModelImpl.java----------------------------
5188  package demopack;
5189  import java.util.*;
5190  public class ModelImpl implements ModelInter
5191  {
5192  public  String getWish()
5193   {
5194     Calendar c1=Calendar.getInstance();
5195     int h=c1.get(Calendar.HOUR_OF_DAY);
5196     if(h  < 12)
5197        return "Good Morning";
5198     else if( h < 17)
5199        return "Good Afternoon";
5200    else
5201        return "Good Evening";
5202    }
5203  }
5204
5205  ------------------------------------result.jsp--------------------------
5206  %@  page language="java" %>
5207  <%@ taglib  prefix="html"  uri="struts-html"  %>
5208  <html>
5209  <body bgcolor=#ffffcc text=green>
5210  <h1><center>struts framewrok</h1><hr><br><h3>
5211    <%= request.getAttribute("message")  %>
5212  <br><br>
5213  <html:link  href="index1.jsp" >home</html:link>
5214  </body>
5215  </html>
5216  ---------------------------------------------------------------------------------
5217  ================================================================================
5218  App  53)>>>>>>>>>>>>>>>>>>>>>>>>>>
      (struts-spring-hibernate)<<<<<<<<<<<<<<<<<<<<<<<<<<
5219  ================================================================================
5220  ------------------------------------index.jsp-----------------------------
5221  <%@  page language="java" %>
5222  <%@ taglib  prefix="html"  uri="struts-html"  %>
5223  <html>
5224  <body bgcolor=#ffffcc text=green>
5225  <h1><center>new user information</h1><hr><br><h3>
5226  <html:form action="/user">
5227    enter user name :<html:text  property="uname"  /> <br><br>
5228    select role:
5229  <table>
5230  <tr><td>
```

```
5231   <html:radio  property="role" value="Administrator">Administrator</html:radio>
5232   <td><html:radio  property="role" value="User">User</html:radio>
5233  </tr>
5234  <tr>
5235   <td><html:radio  property="role" value="Salesman">Salesman</html:radio>
5236  <td>  <html:radio  property="role" value="Manager">Manager</html:radio>
5237  </tr>
5238  </table>
5239    <html:submit  value="save" />
5240  </html:form>
5241  </body>
5242  </html>
5243  ---------------------------------result.jsp---------------------------
5244  %@  page language="java" %>
5245  <%@ taglib  prefix="html"  uri="struts-html"  %>
5246  <html>
5247  <body bgcolor=#ffffcc text=green>
5248  <h1><center>welcome user</h1><hr><br><h3>
5249  </body>
5250  </html>
5251  ---------------------------------------------web.xml-----------------------
5252  <!DOCTYPE web-app
5253   PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
5254   "http://java.sun.com/j2ee/dtds/web-app_2_3.dtd">
5255  <web-app>
5256   <servlet>
5257    <servlet-name>action</servlet-name>
5258    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
5259    <init-param>
5260        <param-name>config</param-name>
5261        <param-value>/WEB-INF/struts-config.xml</param-value>
5262    </init-param>
5263    <load-on-startup>1</load-on-startup>
5264   </servlet>
5265
5266    <servlet-mapping>
5267     <servlet-name>action</servlet-name>
5268     <url-pattern> *.do </url-pattern>
5269    </servlet-mapping>
5270
5271    <welcome-file-list>
5272     <welcome-file>index.jsp</welcome-file>
5273    </welcome-file-list>
5274
5275    <taglib>
5276     <taglib-uri>struts-html</taglib-uri>
5277     <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
5278    </taglib>
5279  </web-app>
5280  ---------------------------------struts-config.xml-----------------------------------------
5281  <?xml version="1.0" encoding="ISO-8859-1" ?>
5282  <!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts Configuration
      1.1//EN"
5283                          "http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">
5284  <struts-config>
5285   <form-beans>
5286    <form-bean  name="userform"   type="demo.UserForm" />
5287   </form-beans>
5288   <action-mappings>
5289    <action  path="/user"  name="userform"
       type="org.springframework.web.struts.DelegatingActionProxy" >
```

```
5290        <forward name="success" path="/result.jsp" />
5291        <forward name="failure" path="/index.jsp" />
5292     </action>
5293    </action-mappings>
5294    <plug-in  className="org.springframework.web.struts.ContextLoaderPlugIn">
5295     </plug-in>
5296   </struts-config>
5297   ------------------------------UserForm.java--------------------------------
5298   package demo;
5299   import org.apache.struts.action.*;
5300   public class UserForm  extends ActionForm
5301   {
5302    private  String uname,role;
5303
5304    public void setUname(String s){uname=s;}
5305    public String getUname(){return uname;}
5306    public void setRole(String s){ role=s; }
5307    public String getRole(){ return role; }
5308   }
5309   ---------------------------------UserAction.java----------------------------
5310   package demo;
5311   import javax.servlet.*;
5312   import javax.servlet.http.*;
5313   import java.io.*;
5314   import org.apache.struts.action.*;
5315   public class UserAction  extends Action
5316   {   private   ModelInter  model;
5317
5318   public void setModel(ModelInter i1) { model=i1;  }
5319    public ActionForward execute(ActionMapping mapping,ActionForm form,
5320                          HttpServletRequest request,HttpServletResponse response)
5321                         throws ServletException,IOException
5322    {
5323    UserForm   f1=(UserForm)form;
5324    String uname=f1.getUname();
5325    String role=f1.getRole();
5326
5327     if(model.insertInfo(uname,role))
5328       return mapping.findForward("success");
5329      else
5330       return mapping.findForward("failure");
5331     }
5332   }
5333   ---------------------------------ModelInter.java---------------------------------
5334   package demo;
5335   public interface ModelInter
5336   {
5337        public  boolean insertInfo(String uname,String role);
5338   }
5339   ---------------------------ModelImpl.java-------------------------------
5340   package demo;
5341   import java.util.*;
5342   import org.hibernate.*;
5343   public class ModelImpl implements ModelInter
5344   {
5345    private  SessionFactory sesfact;
5346
5347   public void setSesfact(SessionFactory sf)
5348   {
5349        sesfact=sf;
5350    }
```

```
5351  public  boolean insertInfo(String uname,String role)
5352  {     Session ses=sesfact.openSession();
5353        Transaction tx=ses.beginTransaction();
5354
5355        boolean flag=false;
5356        try{
5357          User u1=new User();
5358           u1.setUname(uname);
5359        u1.setRole(role);
5360        ses.save(u1);
5361        tx.commit();
5362        flag=true;
5363        }catch(Exception e)
5364        {     tx.rollback();
5365        System.out.println(e);
5366   }
5367        ses.close();
5368        return flag;
5369  }
5370  }
5371  ------------------------------action-servlet.xml-----------------------
5372  <?xml version="1.0" encoding="UTF-8"?>
5373  <!DOCTYPE beans PUBLIC  "-//SPRING//DTD BEAN//EN"
5374  "http://www.springframework.org/dtd/spring-beans.dtd">
5375  <beans>
5376
5377   <bean  name="/user"   class="demo.UserAction" >
5378    <property  name="model"><ref bean="d1" /></property>
5379   </bean>
5380
5381   <bean   id="d1" class="demo.ModelImpl" >
5382    <property  name="sesfact"><ref bean="mySessionFactory" /></property>
5383   </bean>
5384   <bean id="myDataSource"
      class="org.springframework.jdbc.datasource.DriverManagerDataSource">
5385     <property name="driverClassName"><value>oracle.jdbc.driver.OracleDriver</value></property>
5386     <property name="url">value>jdbc:oracle:thin:@localhost:1521:orcl</value></property>
5387     <property name="username"><value>scott</value></property>
5388     <property name="password"><value>tiger</value></property>
5389    </bean>
5390    <bean id="mySessionFactory"
      class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
5391     <property name="dataSource" ref="myDataSource"/>
5392      <property name="mappingResources">
5393       <list>
5394         <value>User.hbm.xml</value>
5395       </list>
5396     </property>
5397     <property name="hibernateProperties">
5398      <props>
5399        <prop key="hibernate.dialect">org.hibernate.dialect.Oracle9Dialect</prop>
5400     </props>
5401     </property>
5402    </bean>
5403  </beans>
5404  ------------------------------User.java-----------------------------
5405  package demo;
5406  public class User
5407  {
5408    private int uid;
5409    private String uname,role;
```

```
5410
5411    public void setUid(int n){uid=n;}
5412    public int getUid(){ return uid; }
5413    public void setUname(String s){ uname=s; }
5414    public String getUname(){ return uname; }
5415    public void setRole(String r){ role=r; }
5416    public String getRole(){  return role; }
5417  }
5418  ----------------------------------------user.hbm.xml---------------------------------
5419  <?xml version="1.0"?>
5420  <!DOCTYPE hibernate-mapping PUBLIC
5421  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
5422  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
5423  <hibernate-mapping>
5424    <class    name="demo.User"   table="users" >
5425      <id   name="uid"  column="userid"  >
5426         <generator  class="increment" />
5427      </id>
5428      <property   name="uname" />
5429      <property   name="role" />
5430    </class>
5431  </hibernate-mapping>
5432  ------------------------------------------------------------------------------------------------------
5433  ============================spring MVC================================
5434  App 54)>>>>>>>>>>>>>>>>>Application on spring MVC<<<<<<<<<<<<<<<<<<<<<<<<
5435  ============================================================
5436  ---------------------index.jsp---------------------
5437  <jsp:forward   page="/newpro.htm"  />
5438  ---------------------------newproduct.jsp--------------------------------
5439  <%@ page language="java" %>
5440  <%@ taglib  prefix="spring"  uri="spring-tld"  %>
5441
5442  <html>
5443  <body bgcolor=#ffffcc text=green>
5444  <h1><center>new product information</h1><hr><br><h3>
5445  <form method="post" >
5446
5447   enter product id :
5448   <spring.bind  path="p.pid" >
5449     <input type=text name=pid />
5450    </spring.bind>
5451  <br><br>
5452
5453  enter product name :
5454  <spring.bind  path="p.pname" >
5455     <input type=text name=pname />
5456  </spring.bind>
5457  <br><br>
5458
5459  enter product price :
5460  <spring.bind  path="p.price" >
5461     <input type=text name=price />
5462  </spring.bind>
5463  <br><br>
5464
5465  <input type=submit value=send />
5466  </form>
5467  </body>
5468  </html>
5469  ----------------------------------web.xml---------------------------------
5470  <?xml version="1.0" encoding="ISO-8859-1"?>
```

```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
   http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
   version="2.4">
 <servlet>
  <servlet-name>productaction</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
 </servlet>
 <servlet-mapping>
   <servlet-name>productaction</servlet-name>
   <url-pattern>*.htm</url-pattern>
 </servlet-mapping>
 <taglib>
  <taglib-uri>spring-tld</taglib-uri>
   <taglib-location>/WEB-INF/spring.tld</taglib-location>
 </taglib>
</web-app>
----------------------------------productaction-servlet.xml-------------------------------
<?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="simpleUrlMappings"
 class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping" >
   <property  name="mappings">
     <props>
      <prop  key="/newpro.htm">newproController</prop>
     </props>
   </property>
 </bean>

 <bean  id="newproController"  class="NewProductController" >
   <property name="sessionForm"> <value>true</value></property>
   <property name="commandName"> <value>p</value></property>
   <property name="commandClass"> <value>Product</value></property>
   <property name="formView"> <value>newproduct</value></property>
   <property name="successView"> <value>showproduct</value></property>
 </bean>

 <bean id="viewResolver" class =
 "org.springframework.web.servlet.view.InternalResourceViewResolver">
   <property
   name="viewClass"><value>org.springframework.web.servlet.view.JstlView</value></property>
   <property name ="prefix"><value>/</value></property>
   <property name ="suffix"><value>.jsp</value></property>
 </bean>
</beans>
-----------------------------Product.java--------------------------
public class Product
{
   private  int pid;
   private String pname;
   private double price;
   public void setPid(int n){ pid=n; }
   public int getPid(){ return pid; }
   public void setPname(String s){ pname=s; }
   public String getPname(){ return pname; }
   public void setPrice(double p){ price=p; }
   public double getPrice(){ return price; }
```

```
5528  }
5529
5530  ----------------------------NewProductController.java----------------------------------
5531  import org.springframework.web.servlet.mvc.*;
5532  import org.springframework.web.servlet.*;
5533  import   org.springframework.validation.*;
5534  import javax.servlet.http.*;
5535  public class   NewProductController  extends SimpleFormController
5536  {
5537    public ModelAndView  onSubmit(HttpServletRequest request,HttpServletResponse response,
5538                            Object command,BindException e) throws Exception
5539     {
5540      Product p=(Product)command;
5541      String str="Product id :"+p.getPid()+" <br>  "
5542          +"product name :"+p.getPname()+"<br>  "
5543         +"Price :"+p.getPrice();
5544
5545       if(p.getPrice()>=10000)
5546          request.setAttribute("msg","costly");
5547        else
5548          request.setAttribute("msg","cheap");
5549
5550
5551      System.out.println(str);
5552      request.setAttribute("productinfo",str);
5553      return new ModelAndView(getSuccessView());
5554     //return new ModelAndView("showproduct");
5555      }
5556  }
5557  ----------------------------showproduct.jsp----------------------------------
5558  <html>
5559  <body bgcolor=#ffffcc text=green>
5560  <h1><center>new product information</h1><hr><br><h3>
5561   <%=request.getAttribute("productinfo") %> <br>
5562   Item is < <%=request.getAttribute("msg") %>
5563  </body>
5564  </html>
5565  ------------------------------------------------------------------------------------------
5566  ==========================================================================
5567  App 55 )>>>>>>>>>>>>>>Application on SpringMVC having form validations <<<<<<<<<<<<<
5568  ==========================index.jsp=========================================
5569  <jsp:forward   page="/newpro.htm"  />
5570  ==========================newproduct.jsp=====================================
5571  <%@ taglib  prefix="spring"  uri="spring-tld"  %>
5572  <%@ taglib  prefix="c"  uri="core" %>
5573  <html>
5574  <body bgcolor=#ffffcc text=green>
5575  <h1><center>new product information</h1><hr><br><h3>
5576  <form method="post" >
5577
5578  enter product id :
5579  <spring.bind  path="p.pid" >
5580    <input type=text name=pid />
5581  </spring.bind>
5582  <br><br>
5583
5584  enter product name :
5585  <spring.bind  path="p.pname" >
5586    <input type=text name=pname />
5587  </spring.bind>
5588  <br><br>
```

```
5589
5590
5591  enter product price :
5592  <spring.bind  path="p.price" >
5593    <input type=text name=price />
5594  </spring.bind>
5595  <br><br>
5596   <input type=submit value=send />
5597
5598  </form>
5599
5600   <spring:hasBindErrors name="p">
5601      <ul>
5602        <c:forEach var="e" items="${errors.allErrors}">
5603          <li>
5604            <spring:message text="${e.defaultMessage}"/>
5605          </li>
5606      </c:forEach>
5607    </ul>
5608   </spring:hasBindErrors>
5609  </body>
5610  </html>
5611  ------------------------------web.xml---------------------------------------
5612  <?xml version="1.0" encoding="ISO-8859-1"?>
5613  <web-app xmlns="http://java.sun.com/xml/ns/j2ee"
5614     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5615     xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
5616     http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
5617     version="2.4">
5618
5619      <servlet>
5620         <servlet-name>productaction</servlet-name>
5621         <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
5622         <load-on-startup>1</load-on-startup>
5623      </servlet>
5624
5625      <servlet-mapping>
5626         <servlet-name>productaction</servlet-name>
5627         <url-pattern>*.htm</url-pattern>
5628      </servlet-mapping>
5629      <taglib>
5630         <taglib-uri>spring-tld</taglib-uri>
5631         <taglib-location>/WEB-INF/spring.tld</taglib-location>
5632       </taglib>
5633      <taglib>
5634         <taglib-uri>core</taglib-uri>
5635         <taglib-location>/WEB-INF/c.tld</taglib-location>
5636      </taglib>
5637
5638  </web-app>
5639
5640  ------------------------------productaction-servlet.xml---------------------------------
5641  <?xml version="1.0" encoding="UTF-8"?>
5642  <!DOCTYPE beans PUBLIC  "-//SPRING//DTD BEAN//EN"
5643  "http://www.springframework.org/dtd/spring-beans.dtd">
5644  <beans>
5645   <bean id="simpleUrlMappings"
      class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping" >
5646     <property name="mappings">
5647       <props>
5648         <prop key="/newpro.htm">newpro</prop>
```

```
5649      </props>
5650     </property>
5651    </bean>
5652
5653    <bean  id="newpro"   class="NewProductController" >
5654       <property name="validator"><bean class="ProductValidator"/></property>
5655       <property name="sessionForm" > <value>true</value></property>
5656       <property name="commandName" > <value>p</value></property>
5657       <property name="commandClass" > <value>Product</value></property>
5658        <property name="formView" > <value>newproduct</value></property>
5659     <property name="successView" > <value>showproduct</value></property>
5660    </bean>
5661
5662   <bean id="viewResolver" class =
      "org.springframework.web.servlet.view.InternalResourceViewResolver">
5663    <property
      name="viewClass"><value>org.springframework.web.servlet.view.JstlView</value></property>
5664     <property name ="prefix"><value>/</value></property>
5665     <property name ="suffix"><value>.jsp</value></property>
5666    </bean>
5667  </beans>
5668  ----------Product.java----------
5669  public class Product
5670  {
5671     private  int pid;
5672     private String pname;
5673     private double price;
5674
5675     public void setPid(int n){ pid=n; }
5676     public int getPid(){ return pid; }
5677     public void setPname(String s){ pname=s; }
5678     public String getPname(){ return pname; }
5679     public void setPrice(double p){ price=p; }
5680    public double getPrice(){ return price; }
5681  }
5682  ----------ProductValidator.java----------
5683  import org.springframework.validation.*;
5684
5685  public class  ProductValidator implements Validator
5686  {
5687
5688  public boolean supports(Class c)  //verifies weather appropriate cmd class is passed passed
5689                                    //as the argument validate () or not.
5690  {
5691   return c.equals(Product.class);
5692  }
5693
5694  public void validate(Object cmd,Errors errs)
5695  {
5696   System.out.println("validate of ProductValidator");
5697   Product p=(Product)cmd;
5698   ValidationUtils.rejectIfEmptyOrWhitespace(errs, "pid", "","id is requied");
5699   ValidationUtils.rejectIfEmptyOrWhitespace(errs, "pname", "","name is requied");
5700   ValidationUtils.rejectIfEmptyOrWhitespace(errs, "price", "","price is requied");
5701
5702   String pna=p.getPname();
5703   if( pna.trim().length() <5)
5704   errs.rejectValue("pname","","name must have at least 5 characters");
5705   }
5706  }
5707
```

```
5708 ----------------------------------NewProductController.java-----------------------------------
5709 import org.springframework.web.servlet.mvc.*;
5710 import org.springframework.web.servlet.*;
5711 import  org.springframework.validation.*;
5712 import javax.servlet.http.*;
5713 public class  NewProductController  extends SimpleFormController
5714 {
5715  public ModelAndView  onSubmit(HttpServletRequest request,HttpServletResponse response,
5716                Object command,
5717                BindException e) throws Exception
5718  {
5719   Product p=(Product)command;
5720   String str="Product id :"+p.getPid()+" <br>  "
5721        +"product name :"+p.getPname()+"<br>  "
5722       +"Price :"+p.getPrice();
5723       System.out.println(str);
5724
5725     request.setAttribute("productinfo",str);
5726    return new ModelAndView(getSuccessView());
5727  //  return new ModelAndView("showproduct");
5728  }
5729 }
5730 ----------------------------showproduct.jsp----------------------------
5731 <%@ page language="java" %>
5732 <html>
5733 <body bgcolor=#ffffcc text=green>
5734 <h1><center>new product information</h1><hr><br><h3>
5735  <%=request.getAttribute("productinfo") %>
5736  <br>
5737 </body>
5738 </html>
5739 ------------------------------------
5740 ================================================================================
5741 App 56)>>>>>>>>>>>>>>>>>Example on spring mvc with annotations<<<<<<<<<<<<<<<<<<<<
5742 ================================================================================
5743 ----------------------index.jsp--------------------
5744
5745 <jsp:forward page="pro.htm"/>
5746 ----------------------userForm.jsp----------------------------
5747
5748 <form method="POST" action="pro.htm">
5749  User Name :<input type=text  name="name" /><br>
5750  Password :<input type=password name=password /><br>
5751  <input type="submit">
5752
5753 </form>
5754 ------------web.xml!--------------------
5755 <web-app>
5756   <servlet>
5757       <servlet-name>dispatcher</servlet-name>
5758       <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
5759   </servlet>
5760    <servlet-mapping>
5761       <servlet-name>dispatcher</servlet-name>
5762       <url-pattern>*.htm</url-pattern>
5763    </servlet-mapping>
5764 </web-app>
5765 ------------------dispatcher-servlet.xml-------------
5766 <?xml version="1.0" encoding="UTF-8"?>
5767 <beans xmlns="http://www.springframework.org/schema/beans"
5768       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
        xmlns:p="http://www.springframework.org/schema/p"
5769    xmlns:context="http://www.springframework.org/schema/context"
5770    xsi:schemaLocation="http://www.springframework.org/schema/beans
5771                http://www.springframework.org/schema/beans/spring-beans.xsd
5772            http://www.springframework.org/schema/context
5773            http://www.springframework.org/schema/context/spring-context.xsd">
5774
5775   <context:component-scan base-package="p1" />
5776  </beans>
5777  ----------User.java----------
5778  package p1;
5779
5780  public class User {
5781
5782  private String name;
5783  private String password;
5784
5785   //write getXxx() and setXxx(-) methods
5786
5787  public String getName()
5788  {
5789      System.out.println("getName()");
5790      return name;
5791   }
5792  public void setName(String name)
5793  {
5794      System.out.println("setName(-)");
5795      this.name = name;
5796   }
5797  public String getPassword()
5798  {
5799    System.out.println("getPassword()");
5800    return password;
5801  }
5802  public void setPassword(String password)
5803  {
5804    System.out.println("setPassword(-)");
5805      this.password = password;
5806   }
5807  }
5808  ----------UserController.java----------
5809  package p1;
5810
5811  import org.springframework.stereotype.Controller;
5812  import org.springframework.ui.ModelMap;
5813  import org.springframework.web.bind.annctation.*;
5814  //import java.util.Date;
5815
5816  @Controller
5817  @RequestMapping("/pro.htm")
5818  //@SessionAttributes("msg")
5819
5820  public class UserController
5821  {
5822
5823  @RequestMapping(method = RequestMethod.GET)
5824  public String showUserForm(ModelMap model)
5825  {
5826      System.out.println("showUserForm(-) executed");
5827      User u1 = new User();
5828      model.addAttribute(u1);
```

```
5829        return "userForm.jsp";
5830    }
5831
5832    @RequestMapping(method = RequestMethod.POST)
5833    public String mySubmit(@ModelAttribute("user") User user)
5834    {
5835        System.out.println("mySubmit(-)");
5836         //read form data from Command class obj
5837        String uname=user.getName();
5838        String pass=user.getPassword();
5839        //write b.logic
5840        if(uname.equals("durga")&& pass.equals("soft"))
5841        {
5842         return "userSuccess.jsp";
5843        }
5844        else
5845        {
5846        return "userFailure.jsp";
5847        }
5848    }//mySubmit(-)
5849
5850    }//class
5851    ----------userSuccess.jsp----------
5852    ValidCredentials
5853    ----------userFailure.jsp----------
5854    InValidCredentials
5855    ------------------------------------------------------------
5856    ============================================================================
5857    App 57)>>>>>>>>>>>>>>>>>Example on spring mvc MultiActionController<<<<<<<<<<<<<<<<
5858    ============================================================================
5859    ----------user.jsp----------
5860    <html>
5861      <body>
5862        <center>
5863        <form method="POST" action="abc.cpp">
5864          Sno :<input type=text name="no"><br>
5865          SName:<input type=text name="name" /><br>
5866          Sadd:<input type=text name="add" /><br>
5867          Email:<input type=text name="email"/><br>
5868
5869          <input type="submit" name="btn" value="insert"/>
5870          <input type="submit" name="btn" value="update"/>
5871          <input type="submit" name="btn" value="delete"/>
5872          <input type="submit" name="btn1" value="abc"/>
5873       </form>
5874     </center>
5875    </body>
5876    </html>
5877    ----------web.xml----------
5878
5879    <web-app>
5880       <servlet>
5881          <servlet-name>disp</servlet-name>
5882          <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
5883       </servlet>
5884
5885      <servlet-mapping>
5886           <servlet-name>disp</servlet-name>
5887           <url-pattern>*.cpp</url-pattern>
5888      </servlet-mapping>
5889
```

```
5890    <welcome-file-list>
5891      <welcome-file>user.jsp</welcome-file>
5892     </welcome-file-list>
5893
5894  </web-app>
5895  ----------------------------disp-servlet.xml----------------------
5896  <?xml version="1.0" encoding="UTF-8"?>
5897  <beans
5898  xmlns="http://www.springframework.org/schema/beans"
5899  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5900  xmlns:p="http://www.springframework.org/schema/p"
5901  xsi:schemaLocation="http://www.springframework.org/schema/beans
5902  http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">
5903
5904  <bean name="daobean" class="p1.UserDAO"/>
5905
5906  <bean id="mnr"
      class="org.springframework.web.servlet.mvc.multiaction.ParameterMethodNameResolver">
5907   <property name="paramName" value="btn"/>
5908    <property name="defaultMethodName" value="requestNotFound"/>
5909  </bean>
5910  <bean name="/*.cpp" class="p1.UserController">
5911   <property name="udao" ref="daobean"/>
5912   <property name="methodNameResolver" ref="mnr"/>
5913  </bean>
5914
5915  </beans>
5916  ----------------------------------UserCommand.java----------------------------
5917  package p1;
5918  public class UserCommand
5919  {
5920  int no;
5921  String name;
5922  String add;
5923  String email;
5924
5925  public UserCommand()
5926  {
5927   System.out.println("UserCommand()");
5928   }
5929
5930  public void setNo(int x){  no=x;}
5931  public void setName(String s1){ name=s1;}
5932  public void setAdd(String add){ this.add=add;}
5933  public void setEmail(String mail){   email=mail;}
5934
5935   public int getNo(){return no;}
5936   public String getName(){return name;}
5937   public String getAdd(){       return add;}
5938   public String getEmail(){return email;}
5939  }
5940  ---------------------------------UserController.java-----------------------------------
5941  package p1;
5942
5943  import java.io.IOException;
5944  import javax.servlet.http.*;
5945  import org.springframework.web.servlet.ModelAndView;
5946  import org.springframework.web.servlet.mvc.multiaction.MultiActionController;
5947
5948  public class UserController extends MultiActionController
5949  {
```

```
5950
5951  UserDAO udao;
5952
5953  public UserController()
5954  {
5955   System.out.println("my multi controller is executed");
5956  }
5957
5958  public void setUdao(UserDAO udao)
5959  {this.udao=udao;}
5960
5961
5962  public ModelAndView insert(HttpServletRequest request,HttpServletResponse
      response,UserCommand uc)
5963  {
5964    System.out.println("insert() called");
5965    int i=0;
5966    try{
5967    i=udao.add(uc);
5968   }catch(Exception e){}
5969
5970    return new ModelAndView("success.jsp","result",i+" Record Inserted Successfully");
5971  }
5972
5973  public ModelAndView update(HttpServletRequest request,HttpServletResponse
      response,UserCommand uc)
5974  {
5975   System.out.println("update() called");
5976    int i=0;
5977
5978     try{
5979         i=udao.edit(uc);
5980      }catch(Exception e){}
5981
5982    return new ModelAndView("success.jsp","result",i+"Record updated Successfully");
5983  }
5984
5985  public ModelAndView delete(HttpServletRequest request,HttpServletResponse
      response,UserCommand uc)
5986  {
5987
5988  System.out.println("delete() called");
5989  int i=0;
5990  try{
5991        i=udao.remove(uc);
5992   }catch(Exception e){}
5993
5994    return new ModelAndView("success.jsp","result",i+"Record Deleted Successful");
5995  }
5996
5997  public ModelAndView requestNotFound(HttpServletRequest request,HttpServletResponse
      response,UserCommand uc){
5998        return new ModelAndView("success.jsp","result","Invalid Request");
5999     }
6000  }
6001  ------------------------------UserDAO.java------------------------------
6002  package p1;
6003  import java.sql.*;
6004  /*
6005  create table student1
6006  (
```

```
6007    sno number(5),
6008     sname varchar2(20),
6009    sadd varchar2(20),
6010    email varchar2(20)
6011   );
6012
6013   */
6014   public class UserDAO
6015   {
6016    Connection con;
6017    PreparedStatement pst;
6018    ResultSet rs;
6019
6020   public Connection getConn()
6021   {
6022    try
6023    {
6024     Class.forName("oracle.jdbc.driver.OracleDriver");
6025     con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
6026     return con;
6027    }catch(Exception e){System.out.println("Exception occured"+e);return null;}
6028    }
6029    public void closeConn()
6030    {
6031     try{
6032      con.close();
6033     }catch(Exception e){System.out.println("Error while closing connection");}
6034     }
6035
6036
6037    public int add(UserCommand uc)throws Exception
6038   {
6039    con=getConn();
6040    pst=con.prepareStatement("insert into student1 values(?,?,?,?)");
6041
6042    pst.setInt(1,uc.getNo());
6043    pst.setString(2,uc.getName());
6044    pst.setString(3,uc.getAdd());
6045    pst.setString(4,uc.getEmail());
6046
6047    int i=pst.executeUpdate();
6048
6049     return i;
6050    }
6051
6052   public int edit(UserCommand uc)throws Exception
6053    {
6054    con=getConn();
6055    pst=con.prepareStatement("update student1 set sname=?,sadd=?,email=? where sno=? ");
6056    pst.setString(1,uc.getName());
6057    pst.setString(2,uc.getAdd());
6058    pst.setString(3,uc.getEmail());
6059    pst.setInt(4,uc.getNo());
6060
6061    int i=pst.executeUpdate();
6062    return i;
6063   }
6064
6065   public int remove(UserCommand uc)throws Exception
6066    {
6067     con=getConn();
```

```
  pst=con.prepareStatement("delete from student1 where sno=?");
  pst.setInt(1,uc.getNo());
  int i=pst.executeUpdate();
 return i;
 }


}
----------success.jsp----------
<%=  request.getAttribute("result")
%>
==========================================
App 58)Example on spring mvc to render pdf/excel type response
==========================================
----------input.html----------
<form action="spring.do">
    <input type="submit">
 </form>
----------web.xml----------
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
   http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
   version="2.4">

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

   <servlet-mapping>
      <servlet-name>productaction</servlet-name>
      <url-pattern>*.do</url-pattern>
   </servlet-mapping>


</web-app>
----------productaction-servlet.xml----------
<?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="url" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
  <property name="mappings">
   <props>
      <prop key="spring.do">controller</prop>
   </props>
  </property>
 </bean>
 <bean id="controller" class="TestController"/>
 <bean id="bnvr" class="org.springframework.web.servlet.view.BeanNameViewResolver"/>
<!-- <bean id="viewbean" class="myexcelview"/> -->
<bean id="viewbean" class="mypdfview"/>

</beans>
----------TestController.java----------
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

```

```
6129 import org.springframework.web.servlet.ModelAndView;
6130 import org.springframework.web.servlet.mvc.Controller;
6131 import java.util.*;
6132
6133 public class TestController implements Controller
6134 {
6135
6136 public ModelAndView handleRequest(HttpServletRequest req,
6137             HttpServletResponse res) throws Exception {
6138       List l = new ArrayList();
6139       l.add("abcd");
6140       l.add("xyz");
6141       l.add("test");
6142       return new ModelAndView("viewbean","result",l);
6143  }
6144 }
6145 ----------myexcelview.java----------
6146 import java.util.*;
6147
6148 import javax.servlet.http.HttpServletRequest;
6149 import javax.servlet.http.HttpServletResponse;
6150
6151 import org.apache.poi.hssf.usermodel.*;
6152 import org.springframework.web.servlet.view.document.*;
6153
6154 public class myexcelview extends AbstractExcelView {
6155
6156 protected void buildExcelDocument(Map m, HSSFWorkbook wb,
6157             HttpServletRequest req, HttpServletResponse res) throws Exception {
6158       HSSFSheet sheet=wb.createSheet("Names");
6159       ArrayList l =(ArrayList)m.get("result");
6160       getCell(sheet,0,0).setCellValue(new HSSFRichTextString("User Names"));
6161       getCell(sheet,1,0).setCellValue(new HSSFRichTextString(l.get(0)+" "));
6162       getCell(sheet,2,0).setCellValue(new HSSFRichTextString(l.get(1)+" "));
6163       getCell(sheet,3,0).setCellValue(new HSSFRichTextString(l.get(2)+" "));
6164       }
6165 }
6166 ----------mypdfview.java----------
6167 import java.util.*;
6168
6169 import javax.servlet.http.HttpServletRequest;
6170 import javax.servlet.http.HttpServletResponse;
6171 import org.springframework.web.servlet.view.document.AbstractPdfView;
6172 import com.lowagie.text.*;
6173 import com.lowagie.text.pdf.PdfWriter;
6174
6175 public class mypdfview extends AbstractPdfView {
6176
6177 protected void buildPdfDocument(Map arg0, Document arg1, PdfWriter arg2,
6178             HttpServletRequest arg3, HttpServletResponse arg4) throws Exception {
6179       Object o = arg0.get("result");
6180       ArrayList l =(ArrayList)o;
6181       Paragraph p = new Paragraph("User details");
6182       p.setAlignment("center");
6183       arg1.add(p);
6184
6185       Table t = new Table(1);
6186       t.addCell(l.get(0)+" ");
6187       t.addCell(l.get(1)+" ");
6188       t.addCell(l.get(2)+" ");
6189       arg1.add(t);
```

```
6190
6191    }
6192  }
6193
```

Object Relational Mapping (ORM) data access

The Spring Framework provides integration with Hibernate, JDO, Oracle TopLink, iBATIS SQL Maps and JPA: in terms of resource management, DAO implementation support, and transaction strategies. All of these support packages for O/R (Object Relational) mappers comply with Spring's generic transaction and DAO exception hierarchies.

There are usually two integration styles: either using Spring's DAO 'templates' or coding DAOs against plain Hibernate/JDO/TopLink/etc APIs. In both cases, DAOs can be configured through Dependency Injection and participate in Spring's resource and transaction management.

Some of the benefits of using the Spring Framework to create your ORM DAOs include:

- Ease of testing. Spring's IoC approach makes it easy to swap the implementations and config locations of Hibernate SessionFactory instances, JDBC DataSource instances, transaction managers, and mappes object implementations (if needed). This makes it much easier to isolate and test each piece of persistence-related code in isolation.
- Common data access exceptions. Spring can wrap exceptions from your O/R mapping tool of choice, converting them from proprietary (potentially checked) exceptions to a common runtime DataAccessException hierarchy. This allows you to handle most persistence exceptions, which are non-recoverable, only in the appropriate layers, without annoying boilerplate catches/throws, and exception declarations. You can still trap and handle exceptions anywhere you need to. Remember that JDBC exceptions (including DB specific dialects) are also converted to the same hierarchy, meaning that you can perform some operations with JDBC within a consistent programming model.
- General resource management. Spring application contexts can handle the location and configuration of Hibernate SessionFactory instances, JDBC DataSource instances, iBATIS SQL Maps configuration objects, and other related resources. This makes these values easy to manage and change. Spring offers efficient, easy and safe handling of persistence resources. For example: related code using Hibernate generally needs to use the same Hibernate Session for efficiency and proper transaction handling. Spring makes it easy to transparently create and bind a Session to the current thread, either by using an explicit 'template' wrapper class at the Java code level or by exposing a current Session through the Hibernate SessionFactory (for DAOs based on plain Hibernate API). Thus Spring solves many of the issues that repeatedly arise from typical Hibernate usage, for any transaction environment (local or JTA).
- Integrated transaction management. Spring allows you to wrap your O/R mapping code with either a declarative, AOP style method interceptor, or an explicit 'template' wrapper class at the Java code level. In either case, transaction semantics are handled for you, and proper transaction handling (rollback, etc) in case of exceptions is taken care of. As discussed below, you also get the benefit of being able to use and swap various transaction managers, without your Hibernate/JDO related code being affected: for example, between local transactions and JTA, with the same full services (such as declarative transactions) available in both scenarios. As an additional benefit, JDBC-related code can fully integrate transactionally with the code you use to do O/R mapping. This is useful for data access that's not suitable for O/R mapping, such as batch processing or streaming of BLOBs, which still needs to share common transactions with ORM operations.

As of Spring 2.5, Spring requires Hibernate 3.1 or higher. Neither Hibernate 2.1 nor Hibernate 3.0 are supported anymore.

DAO definition in a Spring container - HibernateTemplate method

The HibernateTemplate class provides many methods that mirror the methods exposed on the HibernateSession interface, in addition to a number of convenience methods such as the one shown below.

```
<beans>
    <bean id="myProductDao" class="product.ProductDaoImpl">
        <property name="sessionFactory" ref="mySessionFactory"/>
    </bean>
</beans>
public class ProductDaoImpl implements ProductDao {
    private HibernateTemplate hibernateTemplate;
    public void setSessionFactory(SessionFactory sessionFactory) {
        this.hibernateTemplate = new HibernateTemplate(sessionFactory);
    }
    public Collection loadProductsByCategory(String category)
        throws DataAccessException {
        return this.hibernateTemplate.find("from test.Product product
                    where product.category=?", category);
    }
}
```

If you need access to the Session to invoke methods that are not exposed on the HibernateTemplate, you can always drop down to a callback-based approach

```
public class ProductDaoImpl implements ProductDao {
    private HibernateTemplate hibernateTemplate;
    public void setSessionFactory(SessionFactory sessionFactory) {
        this.hibernateTemplate = new HibernateTemplate(sessionFactory);
    }
    public Collection loadProductsByCategory(final String category)
            throws DataAccessException {
        return this.hibernateTemplate.execute(new HibernateCallback() {
            public Object doInHibernate(Session session) {
                Criteria criteria = session.createCriteria(Product.class);
                criteria.add(Expression.eq("category", category));
                criteria.setMaxResults(6);
                return criteria.list();
```

```
                }
            };
        }
}
```

A callback implementation effectively can be used for any Hibernate data access. HibernateTemplate will ensure that Session instances are properly opened and closed, and automatically participate in transactions. The template instances are thread-safe and reusable, they can thus be kept as instance variables of the surrounding class. For simple single step actions like a single find, load, saveOrUpdate, or delete call, HibernateTemplate offers alternative convenience methods that can replace such one line callback implementations. Furthermore, Spring provides a convenient HibernateDaoSupport base class that provides a setSessionFactory(..) method for receiving a SessionFactory, and getSessionFactory() and getHibernateTemplate()for use by subclasses. In combination, this allows for very simple DAO implementations for typical requirements:

```
public class ProductDaoImpl extends HibernateDaoSupport implements ProductDao {
    public Collection loadProductsByCategory(String category)
            throws DataAccessException {
            return this.getHibernateTemplate().find(
                    "from test.Product product where product.category=?", category);
    }
}
```

Implementing Spring-based DAOs without callbacks

As alternative to using Spring's HibernateTemplate to implement DAOs, data access code can also be written in a more traditional fashion, without wrapping the Hibernate access code in a callback, while still respecting and participating in Spring's generic DataAccessException hierarchy. The HibernateDaoSupport base class offers methods to access the current transactional Session and to convert exceptions in such a scenario; similar methods are also available as static helpers on the SessionFactoryUtils class. Note that such code will usually pass 'false' as the value of the getSession(..) methods 'allowCreate' argument, to enforce running within a transaction (which avoids the need to close the returned Session, as its lifecycle is managed by the transaction).

```
public class HibernateProductDao extends HibernateDaoSupport
                implements ProductDao {
    public Collection loadProductsByCategory(String category)
            throws DataAccessException, MyException {
            Session session = getSession(false);
            try {
                    Query query = session.createQuery("from test.Product product
                                    where product.category=?");
                    query.setString(0, category);
```

```
                List result = query.list();
                if (result == null) {
                        throw new MyException("No search results.");
                }
                return result;
        }
        catch (HibernateException ex) {
                throw convertHibernateAccessException(ex);
        }
    }
}
```

The advantage of such direct Hibernate access code is that it allows any checked application exception to be thrown within the data access code; contrast this to the HibernateTemplate class which is restricted to throwing only unchecked exceptions within the callback. Note that you can often defer the corresponding checks and the throwing of application exceptions to after the callback, which still allows working with HibernateTemplate. In general, the HibernateTemplate class' convenience methods are simpler and more convenient for many scenarios.

Implementing DAOs based on plain Hibernate 3 API

Hibernate 3 provides a feature called "contextual Sessions", where Hibernate itself manages one current Session per transaction. This is roughly equivalent to Spring's synchronization of one Hibernate Session per transaction. A corresponding DAO implementation looks like as follows, based on the plain Hibernate API:

```
public class ProductDaoImpl implements ProductDao {
    private SessionFactory sessionFactory;
    public void setSessionFactory(SessionFactory sessionFactory) {
        this.sessionFactory = sessionFactory;
    }
    public Collection loadProductsByCategory(String category) {
        return this.sessionFactory.getCurrentSession()
                .createQuery("from test.Product product
                                where product.category=?")
                .setParameter(0, category).list();
    }
}
```

This style is very similar to what you will find in the Hibernate reference documentation and examples, except for holding the SessionFactory in an instance variable. We strongly recommend such an instance-based setup over the old-school static HibernateUtil class. (In general, do not keep any resources in static variables unless absolutely necessary.)

The above DAO follows the Dependency Injection pattern: it fits nicely into a Spring IoC container, just like it would if coded against Spring's HibernateTemplate. Of course, such a DAO can also be set up in plain Java (for example, in unit tests): simply instantiate it and call setSessionFactory(..) with the desired factory reference. As a Spring bean definition, it would look as follows:

```
<beans>
    <bean id="myProductDao" class="product.ProductDaoImpl">
        <property name="sessionFactory" ref="mySessionFactory"/>
    </bean>
</beans>
```

The main advantage of this DAO style is that it depends on Hibernate API only; no import of any Spring class is required. This is of course appealing from a non-invasiveness perspective, and will no doubt feel more natural to Hibernate developers.However, the DAO throws plain HibernateException (which is unchecked, so does not have to be declared or caught), which means that callers can only treat exceptions as generally fatal - unless they want to depend on Hibernate's own exception hierarchy. Catching specific causes such as an optimistic locking failure is not possible without tieing the caller to the implementation strategy.

This tradeoff might be acceptable to applications that are strongly Hibernate-based and/or do not need any special exception treatment. Fortunately, Spring's LocalSessionFactoryBean supports Hibernate's SessionFactory.getCurrentSession() method for any Spring transaction strategy, returning the current Spring-managed transactional Session even with HibernateTransactionManager. Of course, the standard behavior of that method remains: returning the current Session associated with the ongoing JTA transaction, if any (no matter whether driven by Spring's JtaTransactionManager, by EJB CMT, or by JTA).

In summary: DAOs can be implemented based on the plain Hibernate 3 API, while still being able to participate in Spring-managed transactions.

Spring Remoting

Spring provides integration classes for remoting support using various technologies. The remoting support eases the development of remote-enabled services, implemented by your usual (Spring) POJOs.

Currently, Spring supports the following remoting technologies:

- **Remote Method Invocation (RMI)**. Through the use of the RmiProxyFactoryBean and the RmiServiceExporter Spring supports both traditional RMI (with java.rmi.Remote interfaces and java.rmi.RemoteException) and transparent remoting via RMI invokers (with any Java interface).
- **Spring's HTTP invoker**. Spring provides a special remoting strategy which allows for Java serialization via HTTP, supporting any Java interface (just like the RMI invoker). The corresponding support classes are HttpInvokerProxyFactoryBean and HttpInvokerServiceExporter.
- **Hessian**. By using Spring's HessianProxyFactoryBean and the HessianServiceExporter you can transparently expose your services using the lightweight binary HTTP-based protocol provided by Caucho.
- **Burlap**. Burlap is Caucho's XML-based alternative to Hessian. Spring provides support classes such as BurlapProxyFactoryBean and BurlapServiceExporter.
- **JAX-RPC**. Spring provides remoting support for web services via JAX-RPC (J2EE 1.4's web service API).
- **JAX-WS**. Spring provides remoting support for web services via JAX-WS (the successor of JAX-RPC, as introduced in Java EE 5 and Java 6).
- **JMS**. Remoting using JMS as the underlying protocol is supported via the JmsInvokerServiceExporter and JmsInvokerProxyFactoryBean classes.

The major advantage of RMI, compared to Hessian and Burlap, is serialization. Effectively, any serializable Java object can be transported without hassle. Hessian and Burlap have their own (de-)serialization mechanisms, but are HTTP-based and thus much easier to setup than RMI. Alternatively, consider Spring's HTTP invoker to combine Java serialization with HTTP-based transport.

org.springframework.remoting.rmi.RmiServiceExporter

RMI exporter that exposes the specified service as RMI object with the specified name. Such services can be accessed via plain RMI or via RmiProxyFactoryBean. Also supports exposing any non-RMI service via RMI invokers, to be accessed via RmiClientInterceptor / RmiProxyFactoryBean's automatic detection of such invokers.

With an RMI invoker, RMI communication works on the RmiInvocationHandler level, needing only one stub for any service. Service interfaces do not have to extend java.rmi.Remote or throw java.rmi.RemoteException on all methods, but in and out parameters have to be serializable.

RMI makes a best-effort attempt to obtain the fully qualified host name. If one cannot be determined, it will fall back and use the IP address. Depending on your network configuration, in some cases it will resolve the IP to the loopback address. To ensure that RMI will use the host name bound to the correct network interface, you should pass the java.rmi.server.hostname property to the JVM that will export the registry and/or the service using the "-D" JVM argument.

For example: -Djava.rmi.server.hostname=myserver.com

Using the RmiServiceExporter, we can expose the interface as RMI object. The interface can be accessed by using RmiProxyFactoryBean, or via plain RMI in case of a traditional RMI service. The RmiServiceExporter explicitly supports the exposing of any non-RMI services via RMI invokers.

org.springframework.remoting.rmi.RmiProxyFactoryBean

FactoryBean for RMI proxies, supporting both conventional RMI services and RMI invokers. Exposes the proxied service for use as a bean reference, using the specified service interface. Proxies will throw Spring's unchecked RemoteAccessException on remote invocation failure instead of RMI's RemoteException.

The service URL must be a valid RMI URL like "rmi://localhost:1099/myservice". RMI invokers work at the RmiInvocationHandler level, using the same invoker stub for any service. Service interfaces do not have to extend java.rmi.Remote or throw java.rmi.RemoteException. Of course, in and out parameters have to be serializable.

With conventional RMI services, this proxy factory is typically used with the RMI service interface. Alternatively, this factory can also proxy a remote RMI service with a matching non-RMI business interface, i.e. an interface that mirrors the RMI service methods but does not declare RemoteExceptions. In the latter case, RemoteExceptions thrown by the RMI stub will automatically get converted to Spring's unchecked RemoteAccessException.

NOTE When using RMI, it's not possible to access the objects through the HTTP protocol, unless you're tunneling the RMI traffic. RMI is a fairly heavy-weight protocol in that it supports full-object serialization which is important when using a complex data model that needs serialization over the wire. However, RMI-JRMP is tied to Java clients: It is a Java-to-Java remoting solution.

org.springframework.remoting.RemoteAccessException

Generic remote access exception. A service proxy for any remoting protocol should throw this exception or subclasses of it, in order to transparently expose a plain Java business interface.

When using conforming proxies, switching the actual remoting protocol e.g. from Hessian to Burlap does not affect client code. Clients work with a plain natural Java business interface that the service exposes. A client object simply receives an implementation for the interface that it needs via a bean reference, like it does for a local bean as well.

A client may catch RemoteAccessException if it wants to, but as remote access errors are typically unrecoverable, it will probably let such exceptions propagate to a higher level that handles them generically. In this case, the client code doesn't show any signs of being involved in remote access, as there aren't any remoting-specific dependencies.

Even when switching from a remote service proxy to a local implementation of the same interface, this amounts to just a matter of configuration. Obviously, the client code should be somewhat aware that it might be working against a remote service, for example in terms of repeated method calls that cause unnecessary roundtrips etc. However, it doesn't have to be aware whether it is actually working against a remote service or a local implementation, or with which remoting protocol it is working under the hood.

Following are its sub-classes →

- org.springframework.remoting.RemoteConnectFailureException
- org.springframework.remoting.RemoteInvocationFailureException
- org.springframework.remoting.RemoteLookupFailureException
- org.springframework.remoting.RemoteProxyFailureException

```
public class SearchSubmit extends ActionSupport {   |(1)
  public ActionForward execute(
    ActionMapping mapping,
    ActionForm form,
    HttpServletRequest request,
    HttpServletResponse response)
    throws IOException, ServletException {
          DynaActionForm searchForm = (DynaActionForm) form;
          String isbn = (String) searchForm.get("isbn");

          //the old fashion way
          //BookService bookService = new BookServiceImpl();

          ApplicationContext ctx =
            getWebApplicationContext();    |(2)
          BookService bookService =
            (BookService) ctx.getBean("bookService");   |(3)

         Book book = bookService.read(isbn.trim());
          if (null == book) {
            ActionErrors errors = new ActionErrors();
            errors.add(ActionErrors.GLOBAL_ERROR,new ActionError
              ("message.notfound"));
            saveErrors(request, errors);
            return mapping.findForward("failure") ;
        }
          request.setAttribute("book", book);
          return mapping.findForward("success");
      }
}
```

Let's quickly consider what's happening here. At (1), I create an Action by extending from the Spring ActionSupport class rather than the Struts Action class. At (2), I use the getWebApplicationContext() method to obtain an ApplicationContext. To obtain the business service, I use the context obtained at (2) to look up a Spring bean at (3).

This technique is simple and easy to understand. Unfortunately, it couples the Struts action to the Spring framework. If you ever decide to replace Spring, you would have to rewrite the code. Moreover, because the Struts action isn't under Spring's control, it can't reap the benefits of Spring AOP. This technique may be useful when using multiple independent Spring contexts, but for the most part it's not as desirable a solution as the other two choices.

Recipe 2. Override the RequestProcessor

Decoupling Spring from the Struts action is a much smarter design choice. One way to do this is to override the Struts RequestProcessor processor with the org.springframework.web.struts.DelegatingRequestProcessor class, as shown in Listing 2:

Listing 2. Integration via Spring's DelegatingRequestProcessor

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE struts-config PUBLIC
      "-//Apache Software Foundation//DTD Struts Configuration 1.1//EN"
      "http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">
<struts-config>
 <form-beans>
   <form-bean name="searchForm"
    type="org.apache.struts.validator.DynaValidatorForm">
          <form-property name="isbn"    type="java.lang.String"/>
   </form-bean>
   </form-beans>
<global-forwards type="org.apache.struts.action.ActionForward">
    <forward   name="welcome"              path="/welcome.do"/>
    <forward   name="searchEntry"          path="/searchEntry.do"/>
    <forward   name="searchSubmit"          path="/searchSubmit.do"/>
</global-forwards>
<action-mappings>
   <action    path="/welcome" forward="/WEB-INF/pages/welcome.htm"/>
   <action    path="/searchEntry" forward="/WEB-INF/pages/search.jsp"/>
   <action    path="/searchSubmit"
          type="ca.nexcel.books.actions.SearchSubmit"
          input="/searchEntry.do"
          validate="true"
          name="searchForm">
          <forward name="success" path="/WEB-INF/pages/detail.jsp"/>
          <forward name="failure" path="/WEB-INF/pages/search.jsp"/>
   </action>
</action-mappings>
<message-resources parameter="ApplicationResources"/>
<controller processorClass="org.springframework.web.struts.
 DelegatingRequestProcessor"/> |(1)
```

```
 <plug-in className="org.apache.struts.validator.ValidatorPlugIn">
   <set-property property="pathnames"
    value="/WEB-INF/validator-rules.xml,/WEB-INF/validation.xml"/>
 </plug-in>
 <plug-in className="org.springframework.web.struts.ContextLoaderPlugIn">
   <set-property property="csntextConfigLocation" value="/WEB-INF/beans.xml"/>
 </plug-in>
</struts-config>
```

Here, I've used the <controller> tag to override the default Struts RequestProcessor with the DelegatingRequestProcessor. My next step is to register the action in my Spring config file, as shown in Listing 3:

Listing 3. Registering an action in the Spring config file

```
<?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="bookService" class="ca.nexcel.books.business.BookServiceImpl"/>
 <bean name="/searchSubmit"
  class="ca.nexcel.books.actions.SearchSubmit"> |(1)
   <property name="bookService">
     <ref bean="bookService"/>
   </property>
 </bean>
</beans>
```

Note that at (1), I've registered a bean using the name attribute to match the struts-config action mapping name. The SearchSubmit action exposes a JavaBean property, allowing Spring to populate the property at run time, as shown in Listing 4:

Listing 4. A Struts action with a JavaBean property

```
package ca.nexcel.books.actions;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
```

```
import org.apache.struts.action.DynaActionForm;
import ca.nexcel.books.beans.Book;
import ca.nexcel.books.business.BookService;
public class SearchSubmit extends Action {
 private BookService bookService;
 public BookService getBookService() {
  return bookService;
 }
 public void setBookService(BookService bookService) { | (1)
  this.bookService = bookService;
 }
 public ActionForward execute(
  ActionMapping mapping,
  ActionForm form,
  HttpServletRequest request,
  HttpServletResponse response)
  throws IOException, ServletException {
  DynaActionForm searchForm = (DynaActionForm) form;
  String isbn = (String) searchForm.get("isbn");
 Book book = getBookService().read(isbn.trim());  |(2)
  if (null == book) {
    ActionErrors errors = new ActionErrors();
    errors.add(ActionErrors.GLOBAL_ERROR,new ActionError("message.notfound"));
    saveErrors(request, errors);
    return mapping.findForward("failure") ;
 }
    request.setAttribute("book", book);
    return mapping.findForward("success");
 }
}
```

In Listing 4, you can see how to build the Struts action. At (1), I create a JavaBean property. This property is automatically populated by the DelegatingRequestProcessor. This design protects the Struts action from knowing it's being managed by Spring while giving you all the benefits of Spring's action management framework. Because your Struts actions are oblivious to the existence of Spring, you can swap out Spring for some other inversion of control container without refactoring your Struts code.

While the DelegatingRequestProcessor approach is definitely better than the first one, it does have some problems. If you were using a different RequestProcessor, then you would need to integrate the Spring DelegatingRequestProcessor manually. The added code would become a maintenance hassle and would also reduce your application's flexibility going forward. Moreover,

there has been some talk of replacing the Struts RequestProcessor with a chain of command. Such a change would negatively impact the longevity of this solution.

Recipe 3. Delegate action management to Spring

A much better solution is to delegate Struts action management to the Spring framework. You can do this by registering a proxy in the struts-config action mapping. The proxy is responsible for looking up the Struts action in the Spring context. Because the action is under Spring's control, it populates the action's JavaBean properties and leaves the door open to applying features such as Spring's AOP interceptors.

In Listing 5, the Action class is the same as it was in Listing 4. However, the struts-config is a little different:

Listing 5. The delegation method of Spring integration

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE struts-config PUBLIC
        "-//Apache Software Foundation//DTD Struts Configuration 1.1//EN"
        "http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">
<struts-config>
 <form-beans>
   <form-bean name="searchForm"
    type="org.apache.struts.validator.DynaValidatorForm">
          <form-property name="isbn"    type="java.lang.String"/>
   </form-bean>
 </form-beans>
<global-forwards type="org.apache.struts.action.ActionForward">
    <forward   name="welcome"          path="/welcome.do"/>
    <forward   name="searchEntry"      path="/searchEntry.do"/>
    <forward   name="searchSubmit"      path="/searchSubmit.do"/>
 </global-forwards>
<action-mappings>
   <action    path="/welcome" forward="/WEB-INF/pages/welcome.htm"/>
   <action    path="/searchEntry" forward="/WEB-INF/pages/search.jsp"/>
   <action    path="/searchSubmit"
         type="org.springframework.web.struts.DelegatingActionProxy" |(1)
         input="/searchEntry.do"
         validate="true"
         name="searchForm">
         <forward name="success" path="/WEB-INF/pages/detail.jsp"/>
         <forward name="failure" path="/WEB-INF/pages/search.jsp"/>
   </action>
```

```
</action-mappings>
<message-resources parameter="ApplicationResources"/>
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
  <set-property
  property="pathnames"
  value="/WEB-INF/validator-rules.xml,/WEB-INF/validation.xml"/>
</plug-in>
<plug-in
  className="org.springframework.web.struts.ContextLoaderPlugIn">
  <set-property property="contextConfigLocation" value="/WEB-INF/beans.xml"/>
</plug-in>
</struts-config>
```

Listing 5 is a typical struts-config.xml file, except for one small difference. Instead of declaring the action's class name, it registers the name of Spring's proxy class, as shown at (1). The DelegatingActionProxy class uses the action mapping name to look up the action in the Spring context. This is the context that was declared with ContextLoaderPlugIn.

Registering a Struts action as a Spring bean is very straightforward, as shown in Listing 6. I simply create a bean using the name of the action mapping using the <bean> tag's name attribute (in this case, "/searchSubmit"). The action's JavaBean properties are populated like any Spring bean:

Listing 6. Register a Struts action in the Spring context

```
<?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="bookService" class="ca.nexcel.books.business.BookServiceImpl"/>
 <bean name="/searchSubmit"
      class="ca.nexcel.books.actions.SearchSubmit">
   <property name="bookService">
     <ref bean="bookService"/>
   </property>
 </bean>
</beans>
```

The benefits of action delegation

The action-delegation solution is the best of the three. The Struts action has no knowledge of Spring and could be used in non-Spring applications without changing a single line of code. It's not at the mercy of a change to the RequestProcessor, and it can take advantage of Spring's AOP features.

The benefits of action delegation don't stop there, either. Once you have your Struts action under Spring's control, you can leverage Spring to give them more pizzazz. For example, without Spring, all Struts actions must be threadsafe. If you set the <bean> tag's singleton attribute to

"false," however, your application will have a newly minted action object on every request. You might not need this feature, but it's nice to know you have it in your back pocket. You can also take advantage of Spring's lifecycle methods. For example, the <bean> tag's init-method attribute is used to run a method when the Struts action is instantiated. Similarly, the destroy-method attribute executes a method just before the bean is removed from the container. These methods are a great way to manage expensive objects in much the same way as the Servlet lifecycle does.

Intercepting Struts

As previously mentioned, one of the chief advantages of combining Struts and Spring, and doing it by delegating Struts actions to the Spring framework, is that you can apply Spring's AOP interceptors to your Struts actions. By applying Spring interceptors to Struts actions, you can tackle cross-cutting concerns with minimal effort.

Spring offers a few built-in interceptors, but I'll show you how to create your own and apply it to a Struts action. To use an interceptor, you need to do three things:

1. Create the interceptor.
2. Register it.
3. Declare where it will intersect the code.

This is pretty simple stuff but also very powerful. For example, in Listing 7, I create a logging interceptor for a Struts action. This interceptor prints out a statement before every method call:

Listing 7. A simple logging interceptor

```
package ca.nexcel.books.interceptors;
import org.springframework.aop.MethodBeforeAdvice;
import java.lang.reflect.Method;
public class LoggingInterceptor implements MethodBeforeAdvice {
  public void before(Method method, Object[] objects, Object o) throws Throwable {
     System.out.println("logging before!");
  }
}
```

This interceptor is very simple. The before() method is executed before every method in its intersection. In this case, it prints out a statement, but it could do anything you like. The next step is to register the interceptor in the Spring configuration file, shown in Listing 8:

Listing 8. Registering the interceptor in the Spring config file

```
<?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="bookService" class="ca.nexcel.books.business.BookServiceImpl"/>
 <bean name="/searchSubmit"
     class="ca.nexcel.books.actions.SearchSubmit">
   <property name="bookService">
     <ref bean="bookService"/>
   </property>
```

```
  </bean>
  <!-- Interceptors -->
  <bean name="logger"
   class="ca.nexcel.books.interceptors.LoggingInterceptor"/> |(1)
  <!-- AutoProxies -->
  <bean name="loggingAutoProxy"
      class="org.springframework.aop.framework.autoproxy.
       BeanNameAutoProxyCreator"> |(2)
   <property name="beanNames">
       <value>/searchSubmit</value> |(3)
   </property>
   <property name="interceptorNames">
      <list>
       <value>logger</value> |(4)
      </list>
   </property>
  </bean>
</beans>
```

As you've probably noticed, Listing 8 extends the application shown in Listing 6 to include an interceptor. Details are as follows:

- At (1), I register the interceptor.
- At (2), I create a bean name autoproxy describing how the interceptor is applied. There are other ways to define intersections, but this approach is common and easy to do.
- At (3), I register the Struts action as the bean that will be intercepted. If you wanted to intersect other Struts actions, then you could simply create additional <value> tags under "beanNames."
- At (4), when the interception occurs, I execute the name of the interceptor bean created at (1). All the interceptors listed here are applied against the "beanNames."

That's it! As this example shows, putting your Struts actions under control of the Spring framework opens up a whole new set of options for handling your Struts applications. In the case of this example, action delegation makes it easy to utilize Spring interceptors for better logging in Struts applications.

Conclusion

In this article, you learned three recipes for integrating Struts actions into the Spring framework. Using Spring's ActionSupport to integrate Struts (as I did in the first recipe) is quick and easy but couples your Struts actions to the Spring framework. If you ever needed to port the application to a different framework you would need to rewrite the code. The second solution of delegating the RequestProcessor cleverly decouples your code, but it doesn't necessarily scale well and may not last long if the Struts RequestProcessor is revised to a chain of command. The third approach is the best of the three: delegating Struts actions to the Spring framework results in decoupled code that lets you utilize Spring's features (such as logging interceptors) in your Struts applications.

Aspect Oriented Programming

1) Introduction

- One of the major features available in the Spring Distribution is the provision for separating the ***cross-cutting concerns*** in an Application through the means of ***Aspect Oriented Programming***. Aspect Oriented Programming is sensibly new and it is not a replacement for ***Object Oriented Programming***. In fact, ***AOP*** is another way of organizing your Program Structure.

2.1) The Real Problem

Since AOP is relatively new, this section devotes time in explaining the need for ***Aspect Oriented Programming*** and the various terminologies that are used within. Let us look into the traditional model of before explaining the various concepts.

Consider the following sample application,

Account.java

```
public class Account{

   public long deposit(long depositAmount){

      newAmount = existingAccount + depositAccount;
      currentAmount = newAmount;
      return currentAmount;

   }

   public long withdraw(long withdrawalAmount){

      if (withdrawalAmount <= currentAmount){
         currentAmount = currentAmount - withdrawalAmount;
      }
      return currentAmount;

   }
}
```

The above code models a simple Account Object that provides services for deposit and withdrawal operation in the form of Account.deposit() and Account.withdraw() methods. Suppose say we want to add some bit of the security to the Account class, telling that only users with BankAdmin privilege is allowed to do the operations. With this new requirement being added, let us see the modified class structure below.

Account.java

```
public class Account{

   public long deposit(long depositAmount){

      User user = getContext().getUser();
      if (user.getRole().equals("BankAdmin"){
         newAmount = existingAmount + depositAmount;
         currentAmount = newAmount;
      }
      return currentAmount;

   }

   public long withdraw(long withdrawalAmount){

      User user = getContext().getUser();
      if (user.getRole().equals("BankAdmin"){
         if (withdrawalAmount <= currentAmount){
            currentAmount = currentAmount - withdrawalAmount;
         }
      }
      return currentAmount;
   }
}
```

Assume that getContext().getUser() someway gives the current User object who is invoking the operation. See the modified code mandates the use of adding additional if condition before performing the requested operation. Assume that another requirement for the above Account class is to provide some kind of ***Logging*** and ***Transaction Management*** Facility. Now the code expands as follows,

Account.java

```
public class Account{
   public long deposit(long depositAmount){
      logger.info("Start of deposit method");
      Transaction trasaction  = getContext().getTransaction();
      transaction.begin();
```

```
        try{
            User user = getContext().getUser();
            if (user.getRole().equals("BankAdmin"){
                newAmount = existingAamount + depositAmount;
                currentAmount = newAmount;
            }
            transaction.commit();
        }catch(Exception exception){
            transaction.rollback();
        }
        logger.info("End of deposit method");
        return currentAmount;

    }

    public long withdraw(long withdrawalAmount){

        logger.info("Start of withdraw method");
        Transaction trasaction = getContext().getTransaction();
        transaction.begin();
        try{
            User user = getContext().getUser();
            if (user.getRole().equals("BankAdmin"){
                if (withdrawalAmount <= currentAmount){
                    currentAmount = currentAmount - withdrawalAmount;
                }
            }
            transaction.commit();
        }catch(Exception exception){
            transaction.rollback();
        }
        logger.info("End of withdraw method");
        return currentAmount;

    }
}
```

The above code has so many dis-advantages. The very first thing is that as soon as new requirements are coming it is forcing the methods and the logic to change a lot which is against the ***Software Design***. Remember every piece of newly added code has to undergo the Software Development Lifecycle of Development, Testing, Bug Fixing, Development, Testing, ... This, certainly cannot be encouraged in particularly big projects where a single line of code may have multiple dependencies between other Components or other Modules in the Project.

2.2) The Solution through AOP

Let us re-visit the Class Structure and the Implementation to reveal the facts. The Account class provides services for depositing and withdrawing the amount. But when you look into the implementation of these services, you can find that apart from the normal business logic, it is doing so many other stuffs like ***Logging***, ***User Checking*** and ***Transaction Management***. See the pseudo-code below that explains this.

```
public void deposit(){

   // Transaction Management
   // Logging
   // Checking for the Privileged User
   // Actual Deposit Logic comes here

}

public void withdraw(){

   // Transaction Management
   // Logging
   // Checking for the Privileged User
   // Actual Withdraw Logic comes here

}
```

From the above pseudo-code, it is clear that ***Logging***, ***Transaction Management*** and ***User Checking*** which are never part of the Deposit or the Service functionality are made to embed in the implementation for completeness. Specifically, AOP calls this kind of logic that ***cross-cuts*** or overlaps the existing business logic as ***Concerns*** or ***Cross-Cutting Concerns***. The main idea of AOP is to isolate the cross-cutting concerns from the application code thereby modularizing them as a different entity. It doesn't mean that because the cross-cutting code has been externalized from the actual implementation, the implementation now doesn't get the required add-on functionalities. There are ways to specify some kind of relation between the original business code and the Concerns through some techniques which we will see in the subsequent sections.

3) AOP Terminologies

It is hard to get used with the ***AOP terminologies*** at first but a thorough reading of the following section along with the illustrated samples will make it easy. Let us look into the majorly used AOP jargons.

3.1) Aspects

An ***Aspect*** is a functionality or a feature that ***cross-cuts over*** objects. The addition of the functionality makes the code to ***Unit Test*** difficult because of its dependencies and the availability of the various components it is referring. For example, in the below example, ***Logging*** and ***Transaction Management*** are the aspects.

```
public void businessOperation(BusinessData data){

   // Logging
   logger.info("Business Method Called");

   // Transaction Management Begin
   transaction.begin();

   // Do the original business operation here

   transaction.end();
}
```

3.2) JoinPoint

Join Points defines the various ***Execution Points*** where an ***Aspect*** can be applied. For example, consider the following piece of code,

```
public void someBusinessOperation(BusinessData data){

   //Method Start -> Possible aspect code here like logging.

   try{
      // Original Business Logic here.
   }catch(Exception exception){
      // Exception -> Aspect code here when some exception is raised.
   }finally{
      // Finally -> Even possible to have aspect code at this point too.
   }

   // Method End -> Aspect code here in the end of a method.
}
```

In the above code, we can see that it is possible to determine the various points in the execution of the program like ***Start of the Method***, ***End of the Method***, the ***Exception Block***, the ***Finally Block*** where a particular piece of ***Aspect*** can be made to execute. Such ***Possible Execution Points*** in the Application code for embedding ***Aspects*** are called ***Join Points***. It is not necessary that an ***Aspect*** should be applied to all the possible ***Join Points***.

3.3) Pointcut

As mentioned earlier, ***Join Points*** refer to the ***Logical Points*** wherein a particular ***Aspect*** or a ***Set of Aspects*** can be applied. A ***Pointcut*** or a ***Pointcut Definition*** will exactly tell on which ***Join Points*** the ***Aspects*** will be applied. To make the understanding of this term clearer, consider the following piece of code,

```
aspect LoggingAspect {}

aspect TransactionManagementAspect {}
```

Assume that the above two declarations declare something of type Aspect. Now consider the following piece of code,

```
public void someMethod(){

   //Method Start

   try{
      // Some Business Logic Code.
   }catch(Exception exception){
      // Exception handler Code
   }finally{
      // Finally Handler Code for cleaning resources.
   }

   // Method End
}
```

In the above sample code, the possible execution points, i.e. ***Join Points***, are the start of the method, end of the method, exception block and the finally block. These are the possible points wherein any of the aspects, ***Logging Aspect*** or ***Transaction Management Aspect*** can be applied. Now consider the following ***Point Cut*** definition,

```
pointcut method_start_end_pointcut(){
   // This point cut applies the aspects, logging and transaction, before the
   // beginning and the end of the method.

}

pointcut catch_and_finally_pointcut(){
   // This point cut applies the aspects, logging and transaction, in the catch
   // block (whenever an exception raises) and the finally block.

}
```

As clearly defined, it is possible to define a ***Point Cut*** that binds the ***Aspect*** to a particular ***Join Point*** or some ***Set of Join Points***.

3.4) Advice

Now that we are clear with the terms like ***Aspects***, ***Point Cuts*** and ***Join Points***, let us look into what actually ***Advice*** is. To put simple, Advice is the code that implements the **Aspect**. In general, an ***Aspect*** defines the functionality in a more abstract manner. But, it is this ***Advice*** that provides a ***Concrete code Implementation*** for the ***Aspect***.

4) Creating Advices in Spring

As mentioned previously, ***Advice*** refers to the actual implementation code for an ***Aspect***. Other Aspect Oriented Programming Languages also provide support for ***Field Aspect***, i.e. intercepting a field before its value gets affected. But Spring provides support only ***Method Aspect***. The following are the different types of aspects available in Spring.

- Before Advice
- After Advice
- Throws Advice
- Around Advice

4.1) Before Advice

Before Advice is used to intercept before the method execution starts. In AOP, Before Advice is represented in the form of org.springframework.aop.BeforeAdvice. For example, a System should make security check on users before allowing them to accessing resources. In such a case, we can have a Before Advice that contains code which implements the User Authentication Logic.

Consider the following piece of Code,

Authentication.java

```
public class Authentication extends BeforeAdvice{

   public void before(Method method, Object[] args, Object target) throws Throwable{

      if (args[0] instanceof User){
         User user = (User)args[0];
         // Authenticate if he/she is the right user.
      }

   }

}
```

The above class extends ***BeforeAdvice***, thereby telling that before() method will be called before the execution of the method call. Note that the java.lang.reflect.Method method object represents target method to be invoked, Object[] args refers to the various arguments that are passed on to the method and target refers to the object which is calling the method.

System.java

```
public class System{

   public void login(){
      // Apply Authentication Advice here.
   }

   public void logout(){
      // Apply Authentication Advice here too.
   }
}
```

Note that, till now we have not seen any code that will bind the Advice implementation to the actual calling method. We will see how to do that in the Samples section.

4.2) After Advice

After Advice will be useful if some logic has to be executed before ***Returning the Control*** within a method execution. This advice is represented by the interface org.springframework.aop.AfterReturningAdvice. For example, it is common in Application to Delete the Session Data and the various information pertaining to a user, after he has logged out from the Application. These are ideal candidates for ***After Advice***.

CleanUpOperation.java

```
public class CleanUpOperation implements AfterReturningAdvice {

   public void afterReturning(Object returnValue, Method method, Object[] args,
   Object target) throws Throwable{
      // Clean up session and user information.
   }

}
```

Note that, afterReturning() will be method that will be called once the method returns normal execution. If some exception happens in the method execution the afterReturning() method will never be called.

4.3) Throws Advice

When some kind of exception happens during the execution of a method, then to handle the exception properly, ***Throws Advice*** can be used through the means of org.springframework.aop.ThrowsAdvice. Note that this interface is a ***marker interface*** meaning that it doesn't have any method within it. The method signature inside the ***Throws Advice*** can take any of the following form,

```
public void afterThrowing(Exception ex)

public void afterThrowing(Method method, Object[] args, Object target,
Exception exception)
```

For example, in a File Copy program, if some kind of exception happens in the mid-way then the newly created target file has to be deleted as the partial content in the file doesn't carry any sensible meaning. It can be easily achieved through the means of Throws Advice.

DeleteFile.java

```
public class DeleteFile implements ThrowsAdvice{

    public void afterThrowing(Method method, Object[] args, Object target,
    IOException exception){

        String targetFileName = (String)args[2];
        // Code to delete the target file.
    }

}
```

Note that the above method will be called when an Exception, that too of type IOException is thrown by the File Copy Program.

4.4) Around Advice

This Advice is very different from the other types of Advice that we have seen before, because of the fact that, this Advice provides finer control whether the target method has to be called or not. Considering the above advices, the return type of the method signature is always void meaning that, the Advice itself cannot change the return arguments of the method call. But ***Around Advice*** can even change the return type, thereby returning a brand new object of other type if needed.

Consider the following code,

```
pubic void relate(Object o1, Object o2){

    o1.establishRelation(o2);

}
```

Assume that we have a method that provides an Association link between two objects. But before that, we have to we want to ensure that the type of the Objects being passed must conform to a standard, by implementing some interfaces. We can have this arguments check in the ***Around Advice*** rather than having it in the actual method implementation. The Around Advice is represented by org.aopalliance.intercept.MethodInterceptor.

ValidateArguments.java

```
public class ValidateArguments implements MethodInterceptor {

    public Object invoke(MethodInvocation invocation) throws Throwable {
```

```
        Object arguments [] = invocation.getArguments()
        if ((arguments[0] instanceof Parent) && (arguments[1] instanceof Child) ){
            Object returnValue = invocation.proceed();
            return returnValue;
        }
        throw new Exception ("Arguments are of wrong type");

    }
}
```

In the above code, the validation happens over the arguments to check whether they implement the right interface. It is important to make a call to MethodInvocation.proceed(), if we are happy with the arguments validation, else the target method will never gets invoked.

5) Creating Point Cuts in Spring

Point Cuts define where exactly the Advices have to be applied in various ***Join Points***. Generally they act as ***Filters*** for the application of various Advices into the real implementation. Spring defines two types of ***Point Cuts*** namely the ***Static*** and the ***Dynamic Point Cuts***. The following section covers only about the Static Point Cuts as Dynamic Point Cuts are rarely used.

5.1) The Point Cut Interface

Point Cuts in Spring are represented by org.springframework.aop.Pointcut. Let us look into the various components of this interface. Looking at the interface definition will have something like the following,

Pointcut.java

```
public interface Pointcut{

    ClassFilter getClassFilter()
    MethodMatcher getMethodMatcher()

}
```

The getClassFilter() method returns a ClassFilter object which determines whether the classObject argument passed to the matches() method should be considered for giving ***Advices***. Following is a typical implementation of the ClassFilter interface.

MyClassFilter.java

```
public class MyClassFilter implements ClassFilter{

    public boolean matches(Class classObject){

        String className = classObject.getName();
        // Check whether the class objects should be advised based on their name.
        if (shouldBeAdviced(className) ){
```

```
            return true;
        }
        return false;
    }
}
```

The next interface in consideration is the MethodMatcher which will filter whether various methods within the class should be given ***Advices***. For example, consider the following code,

MyMethodMatcher.java

```
class MyMethodMatcher implements MethodMatcher{

    public boolean matches(Method m, Class targetClass){

        String methodName = m.getName();
        if (methodName.startsWith("get")){
            return true;
        }
        return false;
    }

    public boolean isRuntime(){
        return false;
    }

    public boolean matches(Method m, Class target, Object[] args);

        // This method wont be called in our case. So, just return false.
        return false;
    }
}
```

In the above code, we have 3 methods defined inside in MethodMatcher interface. The isRuntime() method should return true when we want to go for Dynamic Point cut Inclusion by depending on the values of the arguments, which usually happens at run-time. In our case, we can return false, which means that matches(Method method, Class target, Object[] args) wont be called. The implementation of the 2 argument matches() method essentially says that we want only the getter methods to be advised.

We have convenient concrete classes' implementation the ***Point Cut*** classes which are used to give advices to methods statically. They are

- NameMatchMethod Pointcut
- Regular Expression Pointcut

5.2) NameMatchMethod Pointcut

Here the name of the methods that are too be given advices can me directly mentioned in the Configuration File. A '*' represents that all the methods in the class should be given Advice. For example consider the following class,

MyClass.java

```
public class MyClass{
   public void method1(){}
   public void method2(){}
   public void getMethod1()
   public void getMethod2()
}
```

Suppose we wish that only the methods getMethod1() and getMethod2() should be given ***Advice*** by some aspect. In that case, we can have the following Configuration file that achieves this,

```
<bean id="getMethodsAdvisor"
   class="org.springframework.aop.support.NameMatchMethodPointcutAdvisor">

   <property name="mappedName">
      <value>get*</value>
   </property>

</bean>
```

The Expression get* tells that all method names starting with the method name get will be given ***Advices***. If we want all the methods in the MyClass to be adviced, then the 'value' tag should be given '*' meaning all the methods in the Class.

5.3) Regular Expression Pointcut

This kind is used if you want to match the name of the methods in the Class based on ***Regular Expression***. Spring distribution already comes with two supported flavors of Regular Expression namely ***Perl Regular Expression*** (represented by org.springframework.aop.support.Perl5RegexpMethodPointcut) and ***Jdk Regular Expression*** (represented by org.springframework.aop.support.JdkRegexpMethodPointcut).

Considering the following class,

MyClass.java

```
public class MyClass{

   public void method1(){}

   public void method11(){}

   public void method2(){}
```

```
    public void getMethod1()

    public void getMethod11()

    public void getMethod2()
}
```

The Expression 'm*1' matches method1() and method11(), 'getMethod.' matches only getMethod1() and getMethod2(). The ***Configuration File*** should be populated with the following information for gaining this kind of support.

```
<bean id="regExpAdvisor"
    class="org.springframework.aop.support.RegExpPointcutAdvisor">

    <property name="pattern">
        <value>m*1<value>
    </property>

</bean>
```

6) Sample Application

6.1) Introduction

Let is illustrate the various types of Advices (***Before Advice***, ***After Advice***, ***Throws Advice*** and ***Around Advice***) that we saw before in this sample Application. For this sample application let us define Adder Service which provides logic for adding two numbers. The various classes involved in Application along with the Advices in the subsequent sections.

6.2) Addder.java

This is the interface definition for the Add Service. The interface name is Adder and it has one single method called add() taking two arguments both of type int.

Adder.java

```
package net.javabeat.spring.aop.introduction.test;

public interface Adder {

    public int add(int a,int b);

}
```

6.3) AdderImpl.java

The implementation class for the Add Service. The logic is as simple as it returns the summation of the two numbers given as arguments. Note that, in the later section we will see how Advices get bound with this Implementation Class.

AdderImpl.java

```
package net.javabeat.spring.aop.introduction.test;

public class AdderImpl implements Adder {

   public int add(int a, int b){
      return a+b;
   }

}
```

6.4) Before Advice Implementation

This is the ***Before Advice*** for the ***Adder*** Implmentation class. This class implements the before() method in the MethodBeforeAdvice interface by simply outputting a message telling that this advice is called.

LogBeforeCallAdvice.java

```
package net.javabeat.spring.aop.introduction.test;

import java.lang.reflect.Method;
import org.springframework.aop.MethodBeforeAdvice;

public class LogBeforeCallAdvice implements MethodBeforeAdvice{

   public void before(Method method, Object[] args, Object target) {
      System.out.println("Before Calling the Method");
   }

}
```

6.5) After Advice Implementation

The After Method Call Advice implements the AfterReturningAdvice interface providing implementation for the afterReturning() method. Like the ***Before Advice*** implementation, this Advice also outputs a simple message to the console.

LogAfterReturningAdvice.java

```
package net.javabeat.spring.aop.introduction.test;

import java.lang.reflect.Method;
import org.springframework.aop.AfterReturningAdvice;
```

```
public class LogAfterReturningAdvice implements AfterReturningAdvice{

    public void afterReturning(Object returnValue, Method method, Object[] args,
    Object target) throws Throwable {

        System.out.println("After Normal Return from Method");
    }

}
```

6.6) Throws Advice Implementation

This Advice will be called when some kind of Exception is caught during the method invocation. We have added a simple logic to simlate the exception when the user inputs are 0 and 0.

LogAfterThrowsAdvice.java

```
package net.javabeat.spring.aop.introduction.test;

import java.lang.reflect.Method;
import org.springframework.aop.ThrowsAdvice;

public class LogAfterThrowsAdvice implements ThrowsAdvice{

    public void afterThrowing(Method method, Object[] args, Object target,
    Exception exception){

        System.out.println("Exception is thrown on method " + method.getName());
    }

}
```

6.7) Around Advice Implementation

This Advice takes the entire control during the ***Method Execution***. It decides whether the add() method should be called or not based on the user inputs. Note that, only if the user inputs are not 0 and 0, then the add() method will be called through MethodInvocation.proceed().

LogAroundAdvice.java

```
package net.javabeat.spring.aop.introduction.test;

import org.aopalliance.intercept.*;
```

```
public class LogAroundAdvice implements MethodInterceptor{

    public Object invoke(MethodInvocation methodInvocation) throws Throwable {

        Object arguments[] = methodInvocation.getArguments();
        int number1 = ((Integer)arguments[0]).intValue();
        int number2 = ((Integer)arguments[1]).intValue();

        if (number1 == 0 && number2 == 0){
            throw new Exception("Dont know how to add 0 and 0!!!");
        }
        return methodInvocation.proceed();
    }
}
```

6.8) Configuration File

The Configuration File has 3 sections. One section is the ***Advice Bean Definition Section*** which is the definition set for all the 4 advices which we saw before. All the advices are given identifiers like 'beforeCall', 'afterCall', 'throwCall' and 'aroundCall'. Then contains the Bean Definition for the Add implementation class which is giving the identifier 'adderImpl'.

The next interesting section is how to ***bind these advices to the implementation code***. For this, we have to depend on ***ProxyFactory Bean***. This Bean is used to create Proxy objects for the Add Implementation class along with the Advice implementation. Note that the property 'proxyInterfaces' contains the Interface Name for which the proxy class has to ge generated. In our case, it is going to be the Adder interface. The 'interceptorNames' property takes a list of Advices to be applied to the dynamically generated proxy class. We have given all the 4 advices to this property. Finally the implementation class for the Adder service is given in the 'target' property.

aop-test.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">

    <!-- Advices -->
    <bean id = "beforeCall"
    class = "net.javabeat.spring.aop.introduction.test.LogBeforeCallAdvice" />

    <bean id = "afterCall" .
    class = "net.javabeat.spring.aop.introduction.test.LogAfterReturningAdvice" />

    <bean id = "throwCall"
```

```
    class = "net.javabeat.spring.aop.introduction.test.LogAfterThrowsAdvice" />

    <bean id = "aroundCall"
    class = "net.javabeat.spring.aop.introduction.test.LogAroundAdvice" />

    <!-- Implementation Class -->
    <bean id = "adderImpl"
    class = "net.javabeat.spring.aop.introduction.test.AdderImpl" />

    <!-- Proxy Implementation Class -->
    <bean id = "adder"
    class = "org.springframework.aop.framework.ProxyFactoryBean">

        <property name = "proxyInterfaces">
            <value>net.javabeat.spring.aop.introduction.test.Adder</value>
        </property>

        <property name = "interceptorNames">
            <list>
                <value>beforeCall</value>
                <value>afterCall</value>
                <value>throwCall</value>
                <value>aroundCall</value>
            </list>
        </property>

        <property name = "target">
            <ref bean = "adderImpl"/>
        </property>
    </bean>

</beans>
```

6.9) Test Class

Following is the test class for the Adder Service. The code loads the ***Bean Definition File*** by depending on the BeanFactory class. Watch carefully in the output for the various ***Advices*** getting called. Also, we have made to activate the ***Simulated Exception*** by passing 0 and 0 as arguments to the add() method call thereby making use of the ***Throws Advice***.

AdderTest.java

```
package net.javabeat.spring.aop.introduction.test;

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

public class AdderTest {

   public static void main(String args[]){

      Resource resource = new FileSystemResource("./src/aop-test.xml");
      BeanFactory factory = new XmlBeanFactory(resource);
      Adder adder = (Adder)factory.getBean("adder");
      int result = adder.add(10,10);
      System.out.println("Result = " + result);

      result = adder.add(0,0);
      System.out.println("Result = " + result);
   }
}
```

7) Conclusion

This article provided information on how to use ***Spring AOP*** for programming the ***Aspects*** in an Application. It started with defining what ***Aspects*** are and what are problems in having the ***Aspects*** directly embedded into the Application and how to separate them using ***AOP***. It then looked briefly into the various ***AOP Terminologies*** like ***Advice***, ***Point Cut***, ***Join Points*** etc. Then it moved on into the various support for creating ***Advices*** using ***Spring AOP***. Also covered in brief are the ***Static Point Cuts*** like ***Name Method Match*** and ***Regular Expression Point Cuts***. Finally the article concluded with a Sample Application that illustrates the usage of different Advices.

```
1  ==========================================================================
2  App 58)>>>>>>>>>>Demonstrates the behaviour of the AroundAdvice<<<<<<<<<<<<<<<<<<
3  ==========================================================================
4  ------------------------IBusinessLogic.java----------------------
5  public interface IBusinessLogic
6  {
7    public void method1();
8  }
9  -------------------------------BusinessLogic.java-----------------------------
10 public class BusinessLogic implements IBusinessLogic
11 {
12     public void method1()
13     {
14       System.out.println("");
15       System.out.println("execution of method1");
16       System.out.println("");
17     }
18 }
19 ---------------------------------------AroundAdvice.java-----------------------------------------
20  import org.aopalliance.intercept.MethodInvocation;
21  import org.aopalliance.intercept.MethodInterceptor;
22  public class AroundAdvice implements MethodInterceptor
23 {
24   public Object invoke(MethodInvocation i1) throws Throwable
25   {
26    System.out.println("good evening everybody..")
27    i1.proceed();
28    System.out.println("Goodbye!");
29    return null;
30     }
31   }
32 -------------------------------springconfig.xml-----------------------------------
33 <beans>
34   <bean id="pfb" class="org.springframework.aop.framework.ProxyFactoryBean">
35     <property name="proxyInterfaces">
36       <value>IBusinessLogic</value>
37     </property>
38     <property name="target">
39       <ref local="targetobj"/>
40     </property>
41     <property name="interceptorNames">
42       <list>
43         <value>advr</value>
44       </list>
45     </property>
46   </bean>
47   <bean id="targetobj" class="BusinessLogic"/>
48
49   <bean id="advr" class="org.springframework.aop.support.RegexpMethodPointcutAdvisor">
50     <property name="advice">
51       <ref local="adv"/>
52     </property>
53     <property name="pattern">
54       <value>.*</value>
55     </property>
56   </bean>
57
58   <bean id="adv" class="AroundAdvice"/>
59 </beans>
60
61
```

```
----------MainApplication.java----------
import org.springframework.context.ApplicationContet;
import org.springframework.context.support.FileSystemXmlApplicationContext;
public class MainApplication
{
 public static void main(String[]args)
 {
  ApplicationContext ctx = new FileSystemXmlApplicationContext("springconfig.xml");
  IBusinessLogic proxyobj=(IBusinessLogic) ctx.getBean("businesslogicbean");
  proxyobj.method1()
  }
}
--------------------------------------------------------------------------------
================================================================================
App59 (Aop)>>>>>>>>>>>>>>>>>>>>>Based on Return Value<<<<<<<<<<<<<<<<<<<<<
================================================================================
----------IBusinessLogic.java----------
public interface IBusinessLogic
{
  public int method1();
}
----------BusinessLogic.java----------
public class BusinessLogic implements IBusinessLogic
{
   public int method1()
   {
     System.out.println("");
     System.out.println("execution of method1");
     System.out.println("");
     return 1000;
   }
}
----------AroundAdvice.java----------
import org.aopalliance.intercept.MethodInvocation;
import org.aopalliance.intercept.MethodInterceptor;
public class AroundAdvice implements MethodInterceptor
{
  public Object invoke(MethodInvocation i1) throws Throwable
  {
    System.out.println("good evening everybody..");

    Integer n=((Integer)i1.proceed()).intValue();
    System.out.println("Goodbye! ");
    if(n  <=100)
      return new Integer(0);
    else
      return new Integer(1);
  }
}
----------springconfig.xml----------
<?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="pfb" class="org.springframework.aop.framework.ProxyFactoryBean">
    <property name="proxyInterfaces">
      <value>IBusinessLogic</value>
    </property>
    <property name="target">
      <ref local="beanTarget"/>
    </property>
```

```
123       <property name="interceptorNames">
124         <list>
125           <value>theAroundAdvisor</value>
126         </list>
127       </property>
128     </bean>
129     <bean id="beanTarget" class="BusinessLogic"/>
130
131     <bean id="theAroundAdvisor"
        class="org.springframework.aop.support.RegexpMethodPointcutAdvisor">
132       <property name="advice">
133         <ref local="theAroundAdvice"/>
134       </property>
135       <property name="pattern">
136         <value>.*</value>
137       </property>
138     </bean>
139
140     <bean id="theAroundAdvice" class="AroundAdvice"/>
141   </beans>
142   ----------MainAppliation.java----------
143   import org.springframework.context.ApplicationContext;
144   import org.springframework.context.support.FileSystemXmlApplicationContext;
145   public class MainApplication
146   {
147     public static void main(String [] args)
148     {
149       ApplicationContext ctx = new FileSystemXmlApplicationContext("springconfig.xml");
150       IBusinessLogic proxyobj = (IBusinessLogic) ctx.getBean("businesslogicbean");
151      int n=proxyobj.method1();
152      System.out.println("test result :"+n);
153     }
154   }
155   ------------------------------------------------------------------------
156   ========================================================================
157   App 60) >>>>>>>>>>>>>>>>>>>>Based on argument values<<<<<<<<<<<<<<<<<<<<<<
158   ========================================================================
159   ----------IBusinessLogic.java----------
160   public interface IBusinessLogic
161   {
162     public void method1(int id,String name);
163   }
164   ----------BusinessLogic.java----------
165   public class BusinessLogic implements IBusinessLogic
166   {
167       public void method1(int id,String name)
168       {
169         System.out.println("");
170         System.out.println("execution of method1");
171         System.out.println(" id :"+id);
172         System.out.println("name :"+name);
173         System.out.println("");
174       }
175   }
176   ----------AroundAdvice.java----------
177   import org.aopalliance.intercept.MethodInvocation;
178   import org.aopalliance.intercept.MethodInterceptor;
179
180   public class AroundAdvice implements MethodInterceptor
181   {
182     public Object invoke(MethodInvocation i1) throws Throwable
```

```
    {
      System.out.println("good evening everybody..");

   int x=((Integer)i1.getArguments()[0]).intValue();
    String y=(String)i1.getArguments()[1];
    if( x <= 0  )
       i1.getArguments()[0]=new Integer(101);
     if(  y.length() <= 3)
        i1.getArguments()[1]="durga soft";

     i1.proceed();

      System.out.println("Goodbye! ");

      return null;
    }
}

----------springconfig.xml----------
<?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="pfb" class="org.springframework.aop.framework.ProxyFactoryBean">
    <property name="proxyInterfaces">
      <value>IBusinessLogic</value>
    </property>
    <property name="target">
      <ref local="beanTarget"/>
    </property>
    <property name="interceptorNames">
      <list>
        <value>advisor</value>
      </list>
    </property>
  </bean>
  <bean id="beanTarget" class="BusinessLogic"/>

  <bean id="advisor" class="org.springframework.aop.support.RegexpMethodPointcutAdvisor">
    <property name="advice">
      <ref local="ad"/>
    </property>
    <property name="pattern">
      <value>.*</value>
    </property>
  </bean>

  <bean id="ad" class="AroundAdvice"/>
</beans>
==========MainApplication.java==========
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
public class MainApplication
{
  public static void main(String [] args)
  {
    ApplicationContext ctx = new FileSystemXmlApplicationContext("springconfig.xml");
    IBusinessLogic proxyobj = (IBusinessLogic) ctx.getBean("proxyobj");
    proxyobj.method1(22,"ramesh");
    proxyobj.method1(-2,"rao");
  }
```

```
243 }
244 ---------------------------------------------------------------------------
245 ===========================================================================
246 App 61)>>Application that configures multiple advisors on the business methods of spring bean
    <<
247 ===========================================================================
248 ------------------------------IBusinessLogic.java--------------------------------
249 public interface IBusinessLogic
250 {
251   public void method1();
252 }
253 ------------------------------------BusinessLogic.java-------------------------------
254 public class BusinessLogic implements IBusinessLogic
255 {
256     public void method1()
257     {
258       System.out.println("");
259       System.out.println("execution of method1");
260       System.out.println("");
261
262     }
263 }
264 ---------------------------BeforeAdvice1.java----------------------------
265 import org.springframework.aop.*;
266 import java.lang.reflect.*;
267 public class BeforeAdvice1 implements MethodBeforeAdvice
268 {
269
270  public void before(Method method, Object[] args, Object target) throws Throwable
271  {
272   System.out.println("before advice");
273  }
274
275 }
276 -------------------------AroundAdvice2.java-----------------------------
277 import org.aopalliance.intercept.MethodInvocation;
278 import org.aopalliance.intercept.MethodInterceptor;
279 public class AroundAdvice2 implements MethodInterceptor
280 {
281   public Object invoke(MethodInvocation i1) throws Throwable
282   {
283     System.out.println(" AroundAdvice : good evening everybody..");
284
285     i1.proceed();
286
287     System.out.println("AroundAdvice : Goodbye! ");
288
289   return null;
290
291   }
292 }
293
294 ---------------------------springconfig.xml--------------------------
295 <?xml version="1.0" encoding="UTF-8"?>
296 <!DOCTYPE beans PUBLIC  "-//SPRING//DTD BEAN//EN"
    "http://www.springframework.org/dtd/spring-beans.dtd">
297 <beans>
298   <bean id="pfb" class="org.springframework.aop.framework.ProxyFactoryBean">
299     <property name="proxyInterfaces">
300       <value>IBusinessLogic</value>
301     </property>
```

```
    <property name="target">
      <ref local="targetobj"/>
    </property>
    <property name="interceptorNames">
      <list>
        <value>advr1</value>
         <value>advr2</value>
      </list>
    </property>
  </bean>

  <bean id="targetobj" class="BusinessLogic"/>

  <bean id="advr1"
  class="org.springframework.aop.support.NameMatchMethodPointcutAdvisor">
    <property name="advice">
      <ref local="adv1"/>
    </property>
    <property name="mappedName">
      <value>method1</value>
    </property>
  </bean>

 <bean id="advr2" class="org.springframework.aop.support.RegexpMethodPointcutAdvisor">
    <property name="advice">
      <ref local="adv2"/>
    </property>
    <property name="pattern">
      <value>.*</value>
    </property>
  </bean>

  <bean id="adv1" class="BeforeAdvice1"/>
  <bean id="adv2" class="AroundAdvice2"/>
</beans>
----------------------------------MainApplication.java---------------------------------
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
public class MainApplication
{
  public static void main(String [] args)
  {
    ApplicationContext ctx = new FileSystemXmlApplicationContext("springconfig.xml");
    IBusinessLogic testObject = (IBusinessLogic) ctx.getBean("pfb");
    testObject.method1();
  }
}
-------------------------------------------------------------------------------
===============================================================================
App 61)>>>>>>Example application that deals with all the 4 types of advices<<<<<<<<<<<<<<
===============================================================================
----------------------------Adder.java-------------------------------
package aop;
public interface Adder {
   public int add(int a,int b);
}
----------------------------AdderImpl.java-----------------------------------
package aop;

public class AdderImpl implements Adder {

```

```
    public int add(int a, int b){
       return a/b;
    }

}
---------------------------------aop-test.xml-------------------------------
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC  "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">

<beans>
   <!-- Advices -->
   <bean id = "beforeCall" class = "aop.LogBeforeCallAdvice" />
   <bean id = "afterCall" class = "aop.LogAfterReturningAdvice" />
   <bean id = "throwCall" class = "aop.LogAfterThrowsAdvice" />
   <bean id = "aroundCall" class = "aop.LogAroundAdvice" />

   <!-- Implementation Class(SpringBean) -->
   <bean id = "adderImpl" class = "aop.AdderImpl" />

   <!-- Proxy Implementation Class -->
   <bean id = "pfb" class = "org.springframework.aop.framework.ProxyFactoryBean">
      <property name = "proxyInterfaces">
         <value>aop.Adder</value>
      </property>
      <property name = "interceptorNames">
         <list>
            <value>beforeCall</value>
            <value>afterCall</value>
            <value>throwCall</value>
            <value>aroundCall</value>
         </list>
      </property>
      <property name = "target">
         <ref bean = "adderImpl"/>
      </property>
   </bean>
</beans>

-------------------------LogAftgerReturningAdvice.java-----------------------------
package aop;

import java.lang.reflect.Method;
import org.springframework.aop.AfterReturningAdvice;
import org.apache.log4j.*;

public class LogAfterReturningAdvice implements AfterReturningAdvice{
 Logger logger = Logger.getLogger(LogAfterReturningAdvice.class);

 public   LogAfterReturningAdvice()
  {
 SimpleLayout layout = new SimpleLayout();
 FileAppender appender = null;
 try
 {
   appender = new FileAppender(layout,"output1.txt",false);
 }
 catch(Exception e) {}
  logger.addAppender(appender);
  logger.setLevel((Level) Level.DEBUG);
 }
```

```
422
423   public void afterReturning(Object returnValue, Method method, Object[] args,
424    Object target) throws Throwable {
425       logger.info("After Normal Return from Method");
426    }
427  }
428  ----------------------------LogAftgerThrowsAdvice.java--------------------------
429  package aop;
430
431  import java.lang.reflect.Method;
432  import org.springframework.aop.ThrowsAdvice;
433
434  public class LogAfterThrowsAdvice implements ThrowsAdvice{
435
436     public void afterThrowing(Method method, Object[] args, Object target,
437     Exception exception){
438
439        System.out.println("Exception is thrown on method " + method.getName());
440     }
441
442  }
443  ----------------------------LogAroundAdvice.java-----------------------
444  package aop;
445
446  import org.aopalliance.intercept.*;
447  public class LogAroundAdvice implements MethodInterceptor{
448
449     public Object invoke(MethodInvocation i1) throws Throwable {
450
451        Object arguments[] = i1.getArguments();
452        int number1 = ((Integer)arguments[0]).intValue();
453        int number2 = ((Integer)arguments[1]).intValue();
454
455        if (number1 == 0 && number2 == 0){
456           throw new Exception("Dont know how to add 0 and 0!!!");
457        }
458        return i1.proceed();
459     }
460  }
461  ----------------------------LogBeforeCallAdvice.java--------------------
462  package aop;
463
464  import java.lang.reflect.Method;
465  import org.springframework.aop.MethodBeforeAdvice;
466  import org.apache.log4j.*;
467
468  public class LogBeforeCallAdvice implements MethodBeforeAdvice{
469
470   Logger logger = Logger.getLogger(LogBeforeCallAdvice.class);
471
472   public   LogBeforeCallAdvice()
473    {
474    SimpleLayout layout = new SimpleLayout();
475    FileAppender appender = null;
476     try
477     {
478       appender = new FileAppender(layout,"output1.txt",false);
479     }
480     catch(Exception e) {}
481     logger.addAppender(appender);
482    logger.setLevel((Level) Level.DEBUG);
```

```
483     }
484
485     public void before(Method method, Object[] args, Object target) {
486       logger.info("Before Calling the Method");
487     }
488  }
489
490  ------------------------------AopTestClient.java------------------------------
491  package aop;
492
493  import org.springframework.beans.factory.BeanFactory;
494  import org.springframework.beans.factory.xml.XmlBeanFactory;
495  import org.springframework.core.io.*;
496
497  public class AopTestClient {
498     public static void main(String args[]){
499
500        Resource resource = new FileSystemResource("aop-test.xml");
501        BeanFactory factory = new XmlBeanFactory(resource);
502        Adder adder = (Adder)factory.getBean("pfb");
503        int result = adder.add(10,5);
504        System.out.println("Result = " + result);
505        result = adder.add(10,0);
506        System.out.println("Result = " + result);
507     }
508  }
509
510  ============================================================================
511  App 62)(AspectJ With schema support for before and after advices)
512  ============================================================================
513  --------FooService.java--------------------
514  public interface FooService {
515  FooService getFoo(String fooName,int age);
516  void getAfter();
517  void getBefore(String myName);
518  }
519  --------DefaultFooService.java---------------
520  public class DefaultFooService implements FooService {
521        public FooService getFoo(String fooName, int age) {
522              System.out.println("getFoo(fooName,age) business logic");
523              return null;
524        }
525        public void getAfter() {
526              System.out.println("getAfter() business logic");
527        }
528        final public void getBefore(String myName) {
529              System.out.println("getBefore(myName) business logic");
530        }
531  }
532  --------SimpleProfiler.java----------------
533  public class SimpleProfiler {
534  public void afterMethod() throws Throwable {
535  System.out.println("After the method call");
536  }
537  public void beforeMethod(String myName){
538  System.out.println("My name is "+myName);
539  }
540  }
541  ---------spring.xml-----------------------------
542  <beans xmlns="http://www.springframework.org/schema/beans"
543  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-2.5.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">
<!-- this is the object that will be proxied by Spring's AOP infrastructure -->
 <bean id="fooService" class="DefaultFooService"/>

 <!-- this is the actual advice itself -->
 <bean id="profiler" class="SimpleProfiler"/>

 <aop:config>
 <aop:aspect ref="profiler">
  <aop:pointcut id="aopafterMethod"  expression="execution(* FooService.*(..))"/>
   <aop:after pointcut-ref="aopafterMethod"  method="afterMethod"/>
   <aop:pointcut id="aopBefore" expression="execution(* FooService.getBefore(String)) and
   args(myName)"/>
   <aop:before pointcut-ref="aopBefore" method="beforeMethod"/>
  </aop:aspect>
</aop:config>
</beans>
----------client.java----------
import org.springframework.beans.factory.*;
import org.springframework.context.*;
import org.springframework.context.support.*;
public class client {
   public static void main(final String[] args) throws Exception {
      BeanFactory ctx = new ClassPathXmlApplicationContext("spring.xml");
      FooService foo = (FooService) ctx.getBean("fooService");
      foo.getFoo("raja", 12);
      foo.getAfter();
      foo.getBefore("raja");
    }
 }
==========================================================================
App 63)(AspectJ With schema support for throws and around advices)
==========================================================================
----------TestInter.java----------
public interface TestInter
{
  void  demo(String name);
  long findfact(int k);
}
----------TestBean.java----------
public class TestBean implements TestInter
{
 public void demo(String name)
  {
       int k =Integer.parseInt(name);
   }
 public long findfact(int k)
 {
       long f=1;
       for(int i=1; i<=k; i++)
       {
       f = f * i;
       }
       return f;

  }
}
```

```
----------MyAdvices.java----------
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.JoinPoint;
public class MyAdvices {
public void throwsMethod(JoinPoint jp,NumberFormatException nfe) throws Throwable
{
      System.out.println("this advice is applied for "+jp.getSignature().getName());
      System.out.println("The exception occured is : "+nfe.getMessage());
      System.out.println("advice from throws method");
}

public Object aroundMethod(ProceedingJoinPoint pjp) throws Throwable
{
      System.out.println("this advice is applied for "+pjp.getSignature().getName());
      long x = System.currentTimeMillis();
      Object retval=pjp.proceed();
      long y = System.currentTimeMillis();
      System.out.println("The method execution taken "+(y-x)+" milliseconds");
      System.out.println("The above service is from around advice");
      return retval;
 }
}
----------spring.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-2.5.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">

 <bean id="id1" class="TestBean"/>
 <bean id="id2" class="MyAdvices"/>
 <aop:config>
  <aop:aspect ref="id2">
    <aop:pointcut id="pt1" expression="execution(* TestInter.demo(..))"/>
    <aop:after-throwing pointcut-ref="pt1" method="throwsMethod" throwing="nfe"/>

    <aop:pointcut id="pt2" expression="execution(long TestInter.*(..))"/>
    <aop:around pointcut-ref="pt2" method="aroundMethod"/>

  </aop:aspect>
 </aop:config>
</beans>
----------client.java----------
import org.springframework.beans.factory.*;
import org.springframework.context.*;
import org.springframework.context.support.*;
public class client {
  public static void main(final String[] args)  {
    BeanFactory ctx = new ClassPathXmlApplicationContext("spring.xml");
    TestInter ti=(TestInter)ctx.getBean("id1");
    try
    {
     ti.demo("durga");
    }
    catch(Exception e)
    { }
    System.out.println("---------");

    long m1 = ti.findfact(20);
```

```
665       System.out.println(m1);
666       System.out.println("----------");
667
668       long m2 = ti.findfact(58);
669       System.out.println(m2);
670
671     }
672   }
673   ============================================================
674   App 64)>>>>>>>>>>>>>>>> AspectJ  Annotations<<<<<<<<<<<<<<<<<<<<
675   ============================================================
676   ----------TestInter.java--------------------
677   public interface TestInter
678   {
679       void  demo(String name);
680       long findfact(int k);
681   }
682   ----------TestBean.java--------------------
683   public class TestBean implements TestInter
684   {
685   public void demo(String name)
686   {
687       int k =Integer.parseInt(name);
688   }
689   public long findfact(int k)
690   {
691       long f=1;
692       for(int i=1; i<=k; i++)
693       {
694            f = f * i;
695       }
696    return f;
697    }
698   }
699   ----------MyAspect.java----------------------
700   import org.aspectj.lang.annotation.*;
701   @Aspect
702   public class MyAspect
703   {
704   @Pointcut("execution(* TestInter.*(..))")
705   private void testinterOk(){}
706
707   @Before("testinterOk()")
708   public void beforeMethod()
709   {
710    System.out.println("i am before advice");
711   }
712
713    @AfterReturning("testinterOk()")
714    public void afterReturningMethod()
715    {
716     System.out.println("i am after returning advice");
717    }
718   }
719   ----------spring.xml--------------------------
720   <beans xmlns="http://www.springframework.org/schema/beans"
721   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
722   xmlns:aop="http://www.springframework.org/schema/aop"
723   xsi:schemaLocation="http://www.springframework.org/schema/beans
724   http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
725   http://www.springframework.org/schema/aop
```

```
http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">

 <bean id="id1" class="TestBean"/>
  <bean class="MyAspect"/>
  <aop:aspectj-autoproxy/>
 </beans>
----------client.java-------------------------
import org.springframework.beans.factory.*;
import org.springframework.context.*;
import org.springframework.context.support.*;
public class client {
   public static void main(final String[] args)  {
      BeanFactory ctx = new ClassPathXmlApplicationContext("spring.xml");
      TestInter ti=(TestInter)ctx.getBean("id1");
      try
      {
      ti.demo("durga");
      }
      catch(Exception e)
      { }
      System.out.println("---------");

      long m1 = ti.findfact(20);
      System.out.println(m1);
      System.out.println("----------");

      long m2 = ti.findfact(58);
      System.out.println(m2);

    }
 }
-------------------------------------------------------------------------------
======================Spring transactions(using spring DAO)=======================
App 65)>>>>>>>>>>>>>>>>>Programmatic Spring transactions(local,flat Tx)<<<<<<<<<<<<<<<<
=============================================================================
-----------------------------Demo.java-----------------------

public interface Demo
{
      public void mymethod();
}
-----------------------------DemoBean.java--------------------

import org.springframework.transaction.support.*;
import org.springframework.transaction.*;
import org.springframework.jdbc.core.*;

public class DemoBean implements Demo
{
  TransactionTemplate tt;
  JdbcTemplate jt;
  public void setJt(JdbcTemplate jt)
   {
       this.jt=jt;
   }
  public void setTt(TransactionTemplate tt)
   {
       this.tt=tt;
   }
  public void mymethod()
   {
```

```
787     tt.execute(new TransactionCallback()
788     {
789        public Object doInTransaction(TransactionStatus ts)
790        {
791
792        try
793        {
794        int r1=jt.update("insert into dept values(14,'mydept12','hyd12')");
795        int r2=jt.update("update emp set sal=10000 where empno=7499");
796
797        if(r1==0 || r2==0)
798        {
799        ts.setRollbackOnly();
800        System.out.println("Tx is rolledback");
801        }
802        else
803        System.out.println("Tx is committed");
804        }
805        catch(Exception e)
806        {
807        ts.setRollbackOnly();
808        System.out.println("Tx is rolledback");
809        }
810        return null;
811     }
812  });
813  }
814 }
815 ----------------------------------------------demo.xml----------------------------
816 <?xml version="1.0" encoding="UTF-8"?>
817 <!DOCTYPE beans PUBLIC  "-//SPRING//DTD BEAN//EN"
818 "http://www.springframework.org/dtd/spring-beans.dtd">
819 <beans>
820   <bean id="drds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
821    <property name="driverClassName"><vaiue>oracle.jdbc.driver.OracleDriver</value> </property>
822     <property name="url"><value>jdbc:oracle:thin:@localhost:1521:orcl</value></property>
823     <property name="username">  <value>scott</value></property>
824     <property name="password"> <value>tiger</value></property>
825    </bean>
826
827    <bean id="dst" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
828      <property name="dataSource"><ref bean="drds"/></property>
829    </bean>
830
831    <bean id="tt1" class="org.springframework.transaction.support.TransactionTemplate">
832      <property name="transactionManager"><ref  bean="dst"/></property>
833    </bean>
834
835    <bean id="jt1" class="org.springframework.jdbc.core.JdbcTemplate">
836      <property name="dataSource"><ref bean="drds"/> </property>
837     </bean>
838
839     <bean id="db" class="DemoBean">
840      <property name="tt"> <ref bean="tt1"/> </property>
841      <property name="jt"><ref bean="jt1"/>  </property>
842     </bean>
843    </beans>
844 ----------------------------------------------ClientApp.java----------------------------
845 import org.springframework.context.support.*;
846 public class ClientApp
847 {
```

```
 public static void main(String args[ ])
 {
     FileSystemXmlApplicationContext ctx=new FileSystemXmlApplicationContext("demo.xml");
     Demo  beanref =(Demo)ctx.getBean("db");
     beanref.mymethod();
 }
}
```

==========

App 66) Declarative Spring transactions(local,flat Tx)

==========

----------Demo.java----------

```
public interface Demo
{
 public void bm1();
}
```

----------DemoBean.java----------

```
import org.springframework.jdbc.core.*;
public class DemoBean implements Demo
{
 JdbcTemplate jt;
  public void setJt(JdbcTemplate jt)
  {
     this.jt=jt;
  }

 public void bm1(){
  int r2=jt.update("insert into dept values(32,'abc','hyd')");
  int r1=jt.update("update emp set sal=5000 where empno=74001");

  if(r1==0 || r2==0)
  {
    System.out.println("Tx is rolledback");
    throw new java.lang.RuntimeException();
  }
  else
   System.out.println("Tx is commited");
 }
}
```

----------demo.xml----------

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

 <bean id="dts" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
  <property name="dataSource"><ref bean="drds"/></property>
 </bean>


  <bean id="jt1" class="org.springframework.jdbc.core.JdbcTemplate">
    <property name="dataSource"><ref bean="drds"/></property>
   </bean>

   <bean id="db" class="DemoBean">
```

```
    <property name="jt"><ref bean="jt1"/></property>
   </bean>

   <bean id="tas"
   class="org.springframework.transaction.interceptor.NameMatchTransactionAttributeSource">
     <property name="properties">
      <props>
       <prop key="bm1">PROPAGATION_REQUIRED</prop>
      </props>
     </property>
   </bean>

   <bean id="tfb"
   class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">
     <property name="target"><ref bean="db"/></property>
     <property name="transactionManager"><ref bean="dts"/></property>
     <property name="transactionAttributeSource"><ref bean="tas"/></property>
   </bean>
  </beans>
  ----------------------------------------ClientApp.java----------------------------------------
import org.springframework.context.support.*;
public class ClientApp
{
 public static void main(String args[ ]) throws Exception
  {
      FileSystemXmlApplicationContext ctx=new FileSystemXmlApplicationContext("demo.xml");
      Demo beanref=(Demo)ctx.getBean("tfb");
      beanref.bm1();
 }
}
-----------------------------------------
=========================================
App 67) Declarative Transactions(Flat,Local) Using Annotations
=========================================
--------------------Demo.java----------------------
public  interface  Demo
{
      public void bm1()throws RuntimeException;
}
-------------------------DemoBean.java--------------------
import  org.springframework.transaction.annotation.*;
import org.springframework.jdbc.core.*;
public class DemoBean  implements Demo{
 JdbcTemplate jt;
 public void setJt(JdbcTemplate jt) {
      this.jt = jt;
 }

 @Transactional(propagation=Propagation.REQUIRED)
 public void bm1()throws RuntimeException
 {
      int res1=jt.update("update emp set sal=9999 where empno=74991");
      int res2=jt.update("update dept set dname='payroll1'  where deptno=31");

      if(res1==0 || res2==0)
      {
       System.out.println("Tx rolled back");
       throw new RuntimeException();
      }
       else
     {
```

```
968       System.out.println("Tx Committed");
969       }
970    }//bm1
971   }//class
972   ----------------------------------spcfg.xml-------------------------
973   <beans xmlns="http://www.springframework.org/schema/beans"
974   xmlns:aop="http://www.springframework.org/schema/aop"
975   xmlns:tx="http://www.springframework.org/schema/tx"
976   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
977   xsi:schemaLocation="
978   http://www.springframework.org/schema/beans
979   http://www.springframework.org/schema/beans/spring-beans.xsd
980   http://www.springframework.org/schema/tx
981   http://www.springframework.org/schema/tx/spring-tx.xsd
982   http://www.springframework.org/schema/aop
983   http://www.springframework.org/schema/aop/spring-aop.xsd">
984
985    <bean id="drds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
986     <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
987     <property name="url" value="jdbc:oracle:thin:@localhost:1521:orcl"/>
988     <property name="username"  value="scott"/>
989     <property name="password"  value="tiger"/>
990    </bean>
991
992    <bean  id="txmgr"
      class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
993     <property name="dataSource"  ref="drds"/>
994    </bean>
995
996     <bean id="template" class="org.springframework.jdbc.core.JdbcTemplate">
997     <constructor-arg  ref="drds"/>
998    </bean>
999
1000   <bean id="db"  class="DemoBean">
1001     <property name="jt" ref="template"/>
1002  </bean>
1003
1004   <tx:annotation-driven   transaction-manager="txmgr"/>
1005   </beans>
1006   ----------------------------------ClientApp.java--------------------------------
1007  import org.springframework.context.*;
1008  import org.springframework.context.support.*;
1009  public class ClientApp {
1010   public static void main(String args[])throws Exception
1011   {
1012       ApplicationContext ctx=new ClassPathXmlApplicationContext("spcfg.xml");
1013       Demo proxy =(Demo)ctx.getBean("db");
1014       proxy.bm1();
1015   }
1016  }
1017
1018  ========================================================================
1019  App 68)>>>>>>Spring programatic transactions(using Hibernate Transaction
      manager)(Flat,Local)<<<<<<<<<<<<<<<<<<<<
1020  ========================================================================
1021  --------------------Demo.java-----------------------
1022  public interface Demo
1023  {
1024       public boolean transferMoney(int srcid,int destid,float amt);
1025  }
1026
```

```
1027 ----------DemoBean.java----------
1028 import org.springframework.transaction.support.*;
1029 import org.springframework.transaction.*;
1030 import org.springframework.orm.hibernate3.*;
1031
1032 public class DemoBean implements Demo
1033 {
1034 TransactionTemplate tt;
1035 HibernateTemplate ht;
1036 public void setHt(HibernateTemplate ht)
1037 {
1038       this.ht=ht;
1039 }
1040 public void setTt(TransactionTemplate tt)
1041 {
1042  this.tt=tt;
1043 }
1044
1045 public boolean transferMoney(final int srcid,final int destid,final float amt)
1046 {
1047
1048  Boolean result=(Boolean)tt.execute(new TransactionCallback()
1049       {
1050       public Object doInTransaction(TransactionStatus ts)
1051       {
1052       boolean status=false;
1053
1054    try
1055    {
1056      //transferMoney Logic
1057       int r1=ht.bulkUpdate("update Account a1 set a1.balance=a1.balance-? where a1.acid=?",
1058                        new Object[]{new Float(amt),new Integer(srcid)});
1059
1060       int r2=ht.bulkUpdate("update Account a1 set a1.balance=a1.balance+? where a1.acid=?",
1061                        new Object[]{new Float(amt),new Integer(destid)});
1062
1063       if(r1==0 || r2==0)
1064       {
1065        status=false;
1066        ts.setRollbackOnly();
1067        System.out.println("Tx is rolledback");
1068       }
1069       else
1070       {
1071        System.out.println("Tx is committed");
1072        status=true;
1073       }
1074    }//try
1075   catch(Exception e)
1076   {
1077       status=false;
1078       ts.setRollbackOnly();
1079       System.out.println("Tx is rolledback");
1080   }
1081       return new Boolean(status);
1082     }//doInTransaction()
1083  });
1084
1085  return result.booleanValue();
1086 }//transferMoney()
1087
```

```
}//class
----------Account.java (HB pojo class)----------

public class Account
{
 int acid;
 String holdername;
 float balance;

 public void setAcid(int acid){this.acid=acid;}
 public int getAcid(){    return acid;}
 public void setHoldername(String holdername){this.holdername=holdername;}
 public String getHoldername(){  return holdername;}
 public void setBalance(float balance){     this.balance=balance;}
 public float getBalance(){return balance;}
}
----------Account.hbm.xml----------
<!DOCTYPE hibernate-mapping PUBLIC
      "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
      "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
 <class name="Account"  table="HB_Account">
  <id name="acid" column="ACID" >
   <generator class="increment"/>
  </id>
  <property name="holdername" column="HOLDERNAME"/>
  <property name="balance" column="BALANCE"/>
 </class>
</hibernate-mapping>
----------mycfg.xml----------
<!DOCTYPE hibernate-configuration PUBLIC
      "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
      "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
 <session-factory>
  <property
  name="hibernate.connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
  <property name="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:orcl</property>
  <property name="hibernate.connection.username">scott</property>
  <property name="hibernate.connection.password">tiger</property>
  <property name="hibernate.dialect">org.hibernate.dialect.OracleDialect</property>
  <property name="show_sql">true</property>
  <mapping resource="Account.hbm.xml"/>
 </session-factory>
</hibernate-configuration>
----------SpringCfg.xml----------
<!DOCTYPE beans PUBLIC  "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">

<beans>
 <bean id="mySessionFactory"
 class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
  <property name="configLocation"><value>classpath:mycfg.xml</value></property>
 </bean>

 <bean id="template" class="org.springframework.orm.hibernate3.HibernateTemplate">
   <constructor-arg><ref bean="mySessionFactory"/></constructor-arg>
 </bean>
```

```
<bean id="hbt" class="org.springframework.orm.hibernate3.HibernateTransactionManager">
   <property name="sessionFactory" ref="mySessionFactory"/>
 </bean>

 <bean id="tt1" class="org.springframework.transaction.support.TransactionTemplate">
   <property name="transactionManager"><ref  bean="hbt"/></property>
 </bean>
 <bean id="db" class="DemoBean">
    <property name="ht"><ref bean="template"/></property>
    <property name="tt"><ref bean="tt1"/></property>
  </bean>
</beans>
----------------------------------------------ClientApp.java-------------------------------
import org.springframework.context.support.*;
public class ClientApp
{
 public static void main(String args[ ])
  {
     FileSystemXmlApplicationContext ctx=new
     FileSystemXmlApplicationContext("SpringCfg.xml");
     Demo  bean =(Demo)ctx.getBean("db");
     boolean res=bean.transferMoney(102,101,3000);
     if(res)
      System.out.println("Money Transfered");
     else
      System.out.println("Money not Transfered");
  }
}
==================================================
App69)>>>>>>Spring Declarative transactions(using Hibernate Transaction
manager)(Flat,Local)<<<<<<<<<<<<<
==================================================
-------------------------------Demo.java-------------------------
public interface Demo
{
     public boolean transferMoney(int srcid,int destid,float amt)throws RuntimeException;
}
-----------------------DemoBean.java-----------------

import org.springframework.transaction.support.*;
import org.springframework.transaction.*;
import org.springframework.orm.hibernate3.*;

public class DemoBean implements Demo
{
 HibernateTemplate ht;

 public void setHt(HibernateTemplate ht){this.ht=ht;}

 public boolean transferMoney( int srcid, int destid, float amt)throws RuntimeException
 {
   boolean status=false;

   int r1=ht.bulkUpdate("update Account a1 set a1.balance=a1.balance-? where a1.acid=?",
                        new Object[]{new Float(amt),new Integer(srcid)});
   int r2=ht.bulkUpdate("update Account a1 set a1.balance=a1.balance+? where a1.acid=?",
                        new Object[]{new Float(amt),new Integer(destid)});
    if(r1==0 || r2==0)
    {
     status=false;
    }
```

```
        else
        {
         System.out.println("Tx committed");
         status=true;
        }
        if(status==false){
         throw new RuntimeException();
        }
        return status;
   }//transferMoney()
}//class
----------Account.java----------
public class Account
{
  int acid;
  String holdername;
  float balance;

  public void setAcid(int acid) {this.acid=acid;}
  public int getAcid(){    return acid;}
  public void setHoldername(String holdername){    this.holdername=holdername;}
  public String getHoldername(){ return holdername;}
  public void setBalance(float balance){this.balance=balance;}
  public float getBalance(){   return balance;}
}
----------Account.hbm.xml----------
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
  <class name="Account"  table="HB_Account">
   <id name="acid" column="ACID" >
     <generator class="increment"/>
   </id>
   <property name="holdername"/>
   <property name="balance"/>
  </class>
</hibernate-mapping>
----------mycfg.xml----------
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>
    <property
    name="hibernate.connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
    <property name="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:orcl</property>
    <property name="hibernate.connection.username">scott</property>
    <property name="hibernate.connection.password">tiger</property>
    <property name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</property>
    <property name="show_sql">true</property>
    <mapping resource="Account.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
----------SpringCfg.xml----------
<!DOCTYPE beans PUBLIC  "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">

<beans>
```

```
<bean id="mySessionFactory"
class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
  <property name="configLocation"><value>classpath:mycfg.xml</value></property>
</bean>

<bean id="template" class="org.springframework.orm.hibernate3.HibernateTemplate">
   <constructor-arg><ref bean="mySessionFactory"/></constructor-arg>
</bean>

<bean id="hbt" class="org.springframework.orm.hibernate3.HibernateTransactionManager">
    <property name="sessionFactory" ref="mySessionFactory"/>
</bean>

<bean id="db" class="DemoBean">
    <property name="ht"><ref bean="template"/></property>
</bean>

 <bean id="tas"
 class="org.springframework.transaction.interceptor.NameMatchTransactionAttributeSource">
  <property name="properties">
     <props>
       <prop key="transferMoney">PROPAGATION_REQUIRED</prop>
     </props>
  </property>
 </bean>

 <bean id="tfb"
 class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">
   <property name="target"><ref bean="db"/></property>
   <property name="transactionManager"><ref bean="hbt"/></property>
   <property name="transactionAttributeSource"><ref bean="tas"/></property>
  </bean>
</beans>
----------------------------------------ClientApp.java----------------------------------
import org.springframework.context.support.*;
public class ClientApp
{
 public static void main(String args[ ])
 {
     FileSystemXmlApplicationContext ctx=new
     FileSystemXmlApplicationContext("SpringCfg.xml");
     Demo  bean =(Demo)ctx.getBean("tfb");
     boolean res=bean.transferMoney(102,101,6000);
     if(res)
      System.out.println("Money Transfered");
     else
      System.out.println("Money not Transfered");
 }
}
-------------------------------------------------------------
=============================================================================
App 70)>>>>>>>>>>>>>Spring Application on Global /Distributed Transactions <<<<<<<<<<<<
=============================================================================
-----------------------------AccountInter.java-----------------------------------
public  interface  AccountInter
{
vpublic void  transferMoney(int acc1,int acc2,int amount);
}
-------------------------------Account.java-----------------------------------------
import  org.springframework.jdbc.core.JdbcTemplate;
import  org.springframework.transaction.annotation.Transactional;
```

```
1323  public  class  Account  implements AccountInter
1324  {
1325   private  JdbcTemplate  jt1;
1326   private  JdbcTemplate  jt2;
1327   public  void  setJt1(JdbcTemplate jt1){this.jt1=jt1;}
1328   public  void  setJt2(JdbcTemplate jt2){this.jt2=jt2;}
1329
1330   @Transactional
1331    public  void  transferMoney(int acc1,int  acc2,int amount)throws RuntimeException
1332    {
1333       int s1=jt1.queryForInt("select  bal  from  Account1 where  accno=?",new Object[]{acc1});
1334       int s2 = s1-amount;
1335
1336       int s3=jt2.queryForInt("select  bal  from  Account2 where  accno=?",new Object[]{acc2});
1337       int s4 = s3+amount;
1338
1339       int r1=jt1.update("update Account1  set bal=?  where accno=?",new Object[]{s2,acc1});
1340       int r2=jt2.update("update Account2  set bal=?  where  accno=?",new Object[]{s4,acc2});
1341       if(r1==0 || r2==0)
1342       {
1343            System.out.println("Tx rolled back (Money not transffered");
1344            throw new RuntimeException();
1345       }//if
1346       else
1347       {
1348        System.out.println("Tx Committed(Money  transffered");
1349       }
1350    }//method
1351  }//class
1352  ----------spring.xml----------
1353  <?xml version="1.0" encoding="UTF-8"?>
1354  <beans xmlns="http://www.springframework.org/schema/beans"
1355  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1356  xmlns:tx="http://www.springframework.org/schema/tx"
1357  xsi:schemaLocation="http://www.springframework.org/schema/beans
1358  http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
1359  http://www.springframework.org/schema/tx
1360  http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">
1361  <bean id="dataSourceA" class="com.atomikos.jdbc.AtomikosDataSourceBean"
1362     init-method="init" destroy-method="close">
1363   <property name="uniqueResourceName"><value>XADBMSA</value></property>
1364   <property
      name="xaDataSourceClassName"><value>oracle.jdbc.xa.client.OracleXADataSource</value></pr
      operty>
1365   <property name="xaProperties">
1366    <props>
1367    <prop key="databaseName">ORCL</prop>
1368     <prop key="user">scott</prop>
1369      <prop key="password">tiger</prop>
1370      <prop key="URL">jdbc:oracle:thin:@localhost:1521:orcl</prop>
1371    </props>
1372   </property>
1373   <property name="poolSize"><value>10</value></property>
1374  </bean>
1375
1376  <bean id="dataSourceB" class="com.atomikos.jdbc.AtomikosDataSourceBean"
1377         init-method="init" destroy-method="close">
1378    <property name="uniqueResourceName"><value>XADBMSB</value></property>
1379    <property
      name="xaDataSourceClassName"><value>com.mysql.jdbc.jdbc2.optional.MysqlXADataSource<
      /value>
```

```
</property>

<property name="xaProperties">
 <props>
   <prop key="databaseName">mydb1</prop>
   <prop key="user">root1</prop>
    <prop key="password">root1</prop>
  </props>
 </property>
<property name="poolSize"><value>10</value></property>
</bean>

<!-- Construct Atomikos UserTransactionManager, needed to configure Spring -->
<bean id="atomikosTransactionManager"
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</value></property>
</bean>

<!-- Also use Atomikos UserTransactionImp, needed to configure Spring -->
<bean id="atomikosUserTransaction" class="com.atomikos.icatch.jta.UserTransactionImp">
  <property name="transactionTimeout"><value>300</value></property>
</bean>

<!-- Configure the Spring framework to use JTA transactions from Atomikos -->
<bean id="tm" class="org.springframework.transaction.jta.JtaTransactionManager">
   <property name="transactionManager"> <ref
   bean="atomikosTransactionManager"/></property>
    <property name="userTransaction"><ref bean="atomikosUserTransaction"/></property>
</bean>

<tx:annotation-driven transaction-manager="tm"/>

<bean id="jt1"  class="org.springframework.jdbc.core.JdbcTemplate">
  <constructor-arg  ref="dataSourceA"/>
</bean>

<bean id="jt2"  class="org.springframework.jdbc.core.JdbcTemplate">
  <constructor-arg  ref="dataSourceB"/>
</bean>

<bean  id="ac1"  class="Account">
  <property  name="jt1"  ref="jt1"/>
  <property  name="jt2"  ref="jt2"/>
</bean>

</beans>
----------------------------Client.java-------------------------
import  org.springframework.context.*;
import  org.springframework.context.support.*;
class  Client
{  public static void main(String[] args)
  {
     ApplicationContext  ctx=new  ClassPathXmlApplicationContext("spring.xml");
     AccountInter  ac =(AccountInter)ctx.getBean("ac1");
     ac.transferMoney(101,102,1000);
  }
}
```

```
1439 ==========================================================================
1440 App 71) >>>>>>>>>>>>>>>>>Application  on SpringSecurity Plugin (Acegi)<<<<<<<<<<<<<<<<
1441 ==========================================================================
1442 ----------------------------index.jsp----------------------------
1443 Anyone can view this page.
1444 <p><a href="secure/Admin.jsp">Admin Page</a></p>
1445 <p><a href="secure/Faculty.jsp">Faculty Page</a></p>
1446 ----------------------------web.xml----------------------------------
1447 <web-app>
1448  <listener>
1449    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
1450  </listener>
1451  <context-param>
1452      <param-name>contextConfigLocation</param-name>
1453      <param-value>/WEB-INF/mySecurityConfig.xml</param-value>
1454  </context-param>
1455 <welcome-file-list>
1456     <welcome-file>index.jsp</welcome-file>
1457  </welcome-file-list>
1458 <filter>
1459   <filter-name>springSecurityFilterChain</filter-name>
1460   <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
1461 </filter>
1462
1463 <filter-mapping>
1464  <filter-name>springSecurityFilterChain</filter-name>
1465  <url-pattern>/*</url-pattern>
1466 </filter-mapping>
1467 </web-app>
1468 ----------------------------Admin.jsp----------------------------
1469 <h1>Admin Page</h1>
1470  Welcome Mr:<%=request.getUserPrincipal().getName()%><br><br><br>
1471  Wanna goto Faculty Page <a href="Faculty.jsp">click here</a>
1472 <p><a href="../j_spring_security_logout">Logout</a>
1473 ----------------------------Faculty.jsp----------------------------
1474 <h1> Faculty Page </h1>
1475  Welcome Mr:<%=request.getUserPrincipal().getName()%><br><br><br>
1476  Wanna goto Admin Page <a href="Admin.jsp">click here</a>
1477 <p><a href="../j_spring_security_logout">Logout</a>
1478 ----------------------------mySecurityConfig.xml----------------------
1479 <beans:beans xmlns="http://www.springframework.org/schema/security"
1480    xmlns:beans="http://www.springframework.org/schema/beans"
1481    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1482    xsi:schemaLocation="http://www.springframework.org/schema/beans
1483   http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
1484   http://www.springframework.org/schema/security
1485   http://www.springframework.org/schema/security/spring-security-3.0.3.xsd">
1486
1487  <http use-expressions="true">
1488     <intercept-url pattern="/index.jsp" access="permitAll" />
1489     <intercept-url pattern="/secure/Admin.jsp" access="hasRole('admin')" />
1490     <intercept-url pattern="/secure/Faculty.jsp" access="hasRole('faculty')" />
1491     <form-login />
1492     <logout />
1493     <remember-me />
1494     <session-management invalid-session-url="/logout.jsp">
1495        <concurrency-control max-sessions="3" error-if-maximum-exceeded="true" />
1496     </session-management>
1497   </http>
1498
1499  <authentication-manager>
```

```
1500        <authentication-provider>
1501          <user-service>
1502            <user name="durga" password="soft" authorities="admin,faculty" />
1503            <user name="nataraz" password="nataraz" authorities="faculty" />
1504          </user-service>
1505        </authentication-provider>
1506      </authentication-manager>
1507    </beans:beans>
1508    ------------------------logout.jsp---------------------
1509    You have successfully LoggedOut<br>
1510    <a href="index.jsp">start again</a>
1511    ---------------------------------------------------------------------------------------
1512
```

What's new in Spring 2.0 and 2.5?

2.1. Introduction

If you have been using the Spring Framework for some time, you will be aware that Spring has undergone two major revisions: Spring 2.0, released in October 2006, and Spring 2.5, released in November 2007.

Java SE and Java EE Support

The Spring Framework continues to be compatible with all versions of Java since (and including) Java 1.4.2. This means that Java 1.4.2, Java 5 and Java 6 are supported, although some advanced functionality of the Spring Framework will not be available to you if you are committed to using Java 1.4.2. Spring 2.5 introduces dedicated support for Java 6, after Spring 2.0's in-depth support for Java 5 throughout the framework.

Furthermore, Spring remains compatible with J2EE 1.3 and higher, while at the same time introducing dedicated support for Java EE 5. This means that Spring can be consistently used on application servers such as BEA WebLogic 8.1, 9.0, 9.2 and 10, IBM WebSphere 5.1, 6.0 and 6.1, Oracle OC4J 10.1.3 and 11, JBoss 3.2, 4.0 and 4.2, as well as Tomcat 4.1, 5.0, 5.5 and 6.0, Jetty 4.2, 5.1 and 6.1, Resin 2.1, 3.0 and 3.1 and GlassFish V1 and V2.

This chapter is a guide to the new and improved features of Spring 2.0 and 2.5. It is intended to provide a high-level summary so that seasoned Spring architects and developers can become immediately familiar with the new Spring 2.x functionality. For more in-depth information on the features, please refer to the corresponding sections hyperlinked from within this chapter.

2.2. The Inversion of Control (IoC) container

One of the areas that contains a considerable number of 2.0 and 2.5 improvements is Spring's IoC container.

2.2.1. New bean scopes

Previous versions of Spring had IoC container level support for exactly two distinct bean scopes (singleton and prototype). Spring 2.0 improves on this by not only providing a number of additional scopes depending on the environment in which Spring is being deployed (for example, request and session scoped beans in a web environment), but also by providing integration points so that Spring users can create their own scopes.

It should be noted that although the underlying (and internal) implementation for singleton- and prototype-scoped beans has been changed, this change is totally transparent to the end user... no existing configuration needs to change, and no existing configuration will break.

2.2.2. Easier XML configuration

Spring XML configuration is now even easier, thanks to the advent of the new XML configuration syntax based on XML Schema. On a related note, there is a new, updated DTD for Spring 2.0 that you may wish to reference if you cannot take advantage of the XML Schema-based configuration. The DOCTYPE declaration is included below for your convenience, but the interested reader should definitely read the 'spring-beans-2.0.dtd' DTD included in the 'dist/resources' directory of the Spring 2.5 distribution.

```
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
                "http://www.springframework.org/dtd/spring-beans-2.0.dtd">
```

2.2.3. Extensible XML authoring

Not only is XML configuration easier to write, it is now also extensible.

What 'extensible' means in this context is that you, as an application developer, or (more likely) as a third party framework or product vendor, can write custom tags that other developers can

then plug into their own Spring configuration files. This allows you to have your own domain specific language (the term is used loosely here) of sorts be reflected in the specific configuration of your own components.

Implementing custom Spring tags may not be of interest to every single application developer or enterprise architect using Spring in their own projects. We expect third-party vendors to be highly interested in developing custom configuration tags for use in Spring configuration files.

2.2.4. Annotation-driven configuration

Spring 2.0 introduced support for various annotations for configuration purposes, such as @Transactional, @Required and @PersistenceContext /@PersistenceUnit.

Spring 2.5 introduces support for a complete set of configuration annotations: @Autowired in combination with support for the JSR-250 annotations @Resource, @PostConstruct and @PreDestroy .

2.2.5. Autodetecting components in the classpath

Spring 2.5 introduces support component scanning: autodetecting annotated components in the classpath. Typically, such component classes will be annotated with stereotypes such as @Component, @Repository, @Service, @Controller. Depending on the application context configuration, such component classes will be autodetected and turned into Spring bean definitions, not requiring explicit configuration for each such bean.

2.3. Aspect Oriented Programming (AOP)

Spring 2.0 has a much improved AOP offering. The Spring AOP framework itself is markedly easier to configure in XML, and significantly less verbose as a result; and Spring 2.0 integrates with the AspectJ pointcut language and @AspectJ aspect declaration style. The chapter entitled

2.3.1. Easier AOP XML configuration

Spring 2.0 introduces new schema support for defining aspects backed by regular Java objects. This support takes advantage of the AspectJ pointcut language and offers fully typed advice (i.e. no more casting and Object[] argument manipulation).

2.3.2. Support for @AspectJ aspects

Spring 2.0 also supports aspects defined using the @AspectJ annotations. These aspects can be shared between AspectJ and Spring AOP, and require (honestly!) only some simple configuration.

2.3.3. Support for bean name pointcut element

Spring 2.5 introduces support for the bean(...) pointcut element, matching specific named beans according to Spring-defined bean names.

2.3.4. Support for AspectJ load-time weaving

Spring 2.5 introduces explicit support AspectJ load-time weaving, as alternative to the proxy-based AOP framework. The new context:load-time-weaver configuration element automatically activates AspectJ aspects as defined in AspectJ's META-INF/aop.xml descriptor, applying them to the current application context through registering a transformer with the underlying ClassLoader. Note that this only works in environments with class transformation support.

2.4. The Middle Tier

2.4.1. Easier configuration of declarative transactions in XML

The way that transactions are configured in Spring 2.0 has been changed significantly. The previous 1.2.x style of configuration continues to be valid (and supported), but the new style is markedly less verbose and is the recommended style. Spring 2.0 also ships with an AspectJ aspects library that you can use to make pretty much any object transactional - even objects not created by the Spring IoC container.

Spring 2.5 supports convenient annotation-driven transaction management in combination with load-time weaving, through the use of context:load-time-weaver in combination with tx:annotation-driven mode="aspectj".

2.4.2. Full WebSphere transaction management support

Spring 2.5 explicitly supports IBM's WebSphere Application Server, in particular with respect to WebSphere's transaction manager. Transaction suspension is now fully supported through the use of WebSphere's new UOWManager API, which is available on WAS 6.0.2.19+ and 6.0.1.9+.

So if you run a Spring-based application on the WebSphere Application Server, we highly recommend to use Spring 2.5's WebSphereUowTransactionManager as your PlatformTransactionManager of choice. This is also IBM's official recommendation.

For automatic detection of the underlying JTA-based transaction platform, consider the use of Spring 2.5's new tx:jta-transaction-manager configuration element. This will autodetect BEA WebLogic and IBM WebSphere, registering the appropriate PlatformTransactionManager.

2.4.3. JPA

Spring 2.0 ships with a JPA abstraction layer that is similar in intent to Spring's JDBC abstraction layer in terms of scope and general usage patterns.

Spring 2.5 upgrades its OpenJPA support to OpenJPA 1.0, with support for advanced features such as savepoints.

2.4.4. Asynchronous JMS

Prior to Spring 2.0, Spring's JMS offering was limited to sending messages and the *synchronous* receiving of messages. This functionality (encapsulated in the JmsTemplate class) is great, but it doesn't address the requirement for the *asynchronous* receiving of messages.

Spring 2.0 now ships with full support for the reception of messages in an asynchronous fashion..

As of Spring 2.5, the JCA style of setting up asynchronous message listeners is supported as well, through the GenericMessageEndpointManager facility. This is an alternative to the standard JMS listener facility, allowing closer integration with message brokers such as ActiveMQ and JORAM.

Spring 2.5 also introduces an XML namespace for simplifying JMS configuration, offering concise configuration of a large numbers of listeners. This namespace supports both the standard JMS listener facility as well as the JCA setup style, with minimal changes in the configuration.

2.4.5. JDBC

There are some small (but nevertheless notable) new classes in the Spring Framework's JDBC support library. The first, NamedParameterJdbcTemplate, provides support for programming JDBC statements using named parameters (as opposed to programming JDBC statements using only classic placeholder ('?') arguments.

Another of the new classes, the SimpleJdbcTemplate, is aimed at making using the JdbcTemplate even easier to use when you are developing against Java 5+ (Tiger).

Spring 2.5 significantly extends the functionality of SimpleJdbcTemplate and introduces SimpleJdbcCall and SimpleJdbcInsert operation objects.

2.5. The Web Tier

The web tier support has been *substantially* improved and expanded in Spring 2.0, with annotation-based controllers introduced in Spring 2.5.

2.5.1. Sensible defaulting in Spring MVC

For a lot of projects, sticking to established conventions and having reasonable defaults is just what the projects need... this theme of convention-over-configuration now has explicit support in

Spring MVC. What this means is that if you establish a set of naming conventions for your Controllers and views, you can *substantially* cut down on the amount of XML configuration that is required to setup handler mappings, view resolvers, ModelAndView instances, etc. This is a great boon with regards to rapid prototyping, and can also lend a degree of (always good-to-have) consistency across a codebase.

2.5.2. Portlet framework

Spring 2.0 ships with a Portlet framework that is conceptually similar to the Spring MVC framework.

2.5.3. Annotation-based controllers

Spring 2.5 introduces an annotation-based programming model for MVC controllers, using annotations such as @RequestMapping, @RequestParam, @ModelAttribute, etc. This annotation support is available for both Servlet MVC and Portlet MVC. Controllers implemented in this style do not have to extend specific base classes or implement specific interfaces. Furthermore, they do not usually have direct dependencies on Servlet or Portlet API's, although they can easily get access to Servlet or Portlet facilities if desired.

2.5.4. A form tag library for Spring MVC

A rich JSP tag library for Spring MVC was *the* JIRA issue that garnered the most votes from Spring users (by a wide margin).

Spring 2.0 ships with a full featured JSP tag library that makes the job of authoring JSP pages much easier when using Spring MVC; the Spring team is confident it will satisfy all of those developers who voted for the issue on JIRA.

2.5.5. Tiles 2 support

Spring 2.5 ships support for Tiles 2, the next generation of the popular Tiles templating framework. This supersedes Spring's former support for Tiles 1, as included in Struts 1.x.

2.5.6. JSF 1.2 support

Spring 2.5 supports JSF 1.2, providing a JSF 1.2 variant of Spring's DelegatingVariableResolver in the form of the new SpringBeanFacesELResolver.

2.5.7. JAX-WS support

Spring 2.5 fully supports JAX-WS 2.0/2.1, as included in Java 6 and Java EE 5. JAX-WS is the successor of JAX-RPC, allowing access to WSDL/SOAP-based web services as well as JAX-WS style exposure of web services.

2.6. Everything else

This final section outlines all of the other new and improved Spring 2.0/2.5 features and functionality.

2.6.1. Dynamic language support

Spring 2.0 introduced support for beans written in languages other than Java, with the currently supported dynamic languages being JRuby, Groovy and BeanShell..

Spring 2.5 refines the dynamic languages support with autowiring and support for the recently released JRuby 1.0.

2.6.2. Enhanced testing support

Spring 2.5 introduces the *Spring TestContext Framework* which provides annotation-driven unit and integration testing support that is agnostic of the actual testing framework in use. The same techniques and annotation-based configuration used in, for example, a JUnit 3.8 environment can also be applied to tests written with JUnit 4.4, TestNG, etc.

In addition to providing generic and extensible testing infrastructure, the *Spring TestContext Framework* provides out-of-the-box support for Spring-specific integration testing functionality such as context management and caching, dependency injection of test fixtures, and transactional test management with default rollback semantics.

2.6.3. JMX support

The Spring Framework 2.0 has support for Notifications; it is also possible to exercise declarative control over the registration behavior of MBeans with an MBeanServer.

- Section 20.7, "Notifications"
- Section 20.2.5, "Controlling the registration behavior"

Furthermore, Spring 2.5 provides a context:mbean-export configuration element for convenient registration of annotated bean classes, detecting Spring's @ManagedResource annotation.

2.6.4. Deploying a Spring application context as JCA adapter

Spring 2.5 supports the deployment of a Spring application context as JCA resource adapter, packaged as a JCA RAR file. This allows headless application modules to be deployed into J2EE servers, getting access to all the server's infrastructure e.g. for executing scheduled tasks, listening for incoming messages, etc.

2.6.5. Task scheduling

Spring 2.0 offers an abstraction around the scheduling of tasks.

The TaskExecutor abstraction is used throughout the framework itself as well, e.g. for the asynchronous JMS support. In Spring 2.5, it is also used in the JCA environment support.

2.6.6. Java 5 (Tiger) support

Find below pointers to documentation describing some of the new Java 5 support in Spring 2.0 and 2.5.

- Section 3.11, "Annotation-based configuration"
- Section 25.3.1, "@Required"
- Section 9.5.6, "Using @Transactional"
- Section 11.2.3, "SimpleJdbcTemplate"
- Section 12.6, "JPA"
- Section 6.2, "@AspectJ support"
- Section 6.8.1, "Using AspectJ to dependency inject domain objects with Spring"

2.7. Migrating to Spring 2.5

This final section details issues that may arise during any migration from Spring 1.2/2.0 to Spring 2.5.

Upgrading to Spring 2.5 from a Spring 2.0.x application should simply be a matter of dropping the Spring 2.5 jar into the appropriate location in your application's directory structure. We highly recommend upgrading to Spring 2.5 from any Spring 2.0 application that runs on JDK 1.4.2 or higher, in particular when running on Java 5 or higher, leveraging the significant configuration conveniences and performance improvements that Spring 2.5 has to offer.

Whether an upgrade from Spring 1.2.x will be as seamless depends on how much of the Spring APIs you are using in your code. Spring 2.0 removed pretty much all of the classes and methods previously marked as deprecated in the Spring 1.2.x codebase, so if you have been using such classes and methods, you will of course have to use alternative classes and methods (some of which are summarized below).

With regards to configuration, Spring 1.2.x style XML configuration is 100%, satisfaction-guaranteed compatible with the Spring 2.5 library. Of course if you are still using the Spring 1.2.x DTD, then you won't be able to take advantage of some of the new Spring 2.0 functionality (such as scopes and easier AOP and transaction configuration), but nothing will blow up.

The suggested migration strategy is to drop in the Spring 2.5 jar(s) to benefit from the improved code present in the release (bug fixes, optimizations, etc.). You can then, on an incremental basis, choose to start using the new Spring 2.5 features and configuration. For example, you could choose to start configuring just your aspects in the new Spring 2 style; it is perfectly valid to have 90% of your configuration using the old-school Spring 1.2.x configuration (which references the 1.2.x DTD), and have the other 10% using the new Spring 2 configuration (which references the 2.0/2.5 DTD or XSD). Bear in mind that you are not forced to upgrade your XML configuration should you choose to drop in the Spring 2.5 libraries.

2.7.1. Changes

For a comprehensive list of changes, consult the 'changelog.txt' file that is located in the top level directory of the Spring Framework distribution.

2.7.1.1. Supported JDK versions

As of Spring 2.5, support for JDK 1.3 has been removed, following Sun's official deprecation of JDK 1.3 in late 2006. If you haven't done so already, upgrade to JDK 1.4.2 or higher.

If you need to stick with an application server that only supports JDK 1.3, such as WebSphere 4.0 or 5.0, we recommend using the Spring Framework version 2.0.7/2.0.8 which still supports JDK 1.3.

2.7.1.2. Jar packaging in Spring 2.5

As of Spring 2.5, Spring Web MVC is no longer part of the 'spring.jar' file. Spring MVC can be found in 'spring-webmvc.jar' and 'spring-webmvc-portlet.jar' in the lib/modules directory of the distribution. Furthermore, the Struts 1.x support has been factored out into 'spring-webmvc-struts.jar'.

Note: The commonly used Spring's DispatcherServlet is part of Spring's Web MVC framework. As a consequence, you need to add 'spring-webmvc.jar' (or 'spring-webmvc-portlet/struts.jar') to a 'spring.jar' scenario, even if you are just using DispatcherServlet for remoting purposes (e.g. exporting Hessian or HTTP invoker services).

Spring 2.0's 'spring-jmx.jar' and 'spring-remoting.jar' have been merged into Spring 2.5's 'spring-context.jar' (for the JMX and non-HTTP remoting support) and partly into 'spring-web.jar' (for the HTTP remoting support).

Spring 2.0's 'spring-support.jar' has been renamed to 'spring-context-support.jar', expressing the actual support relationship more closely. 'spring-portlet.jar' has been renamed to 'spring-webmvc-portlet.jar', since it is technically a submodule of Spring's Web MVC framework. Analogously, 'spring-struts.jar' has been renamed to 'spring-webmvc-struts.jar'.

Spring 2.0's 'spring-jdo.jar', 'spring-jpa.jar', 'spring-hibernate3.jar', 'spring-toplink.jar' and 'spring-ibatis.jar' have been combined into Spring 2.5's coarse-granular 'spring-orm.jar'.

Spring 2.5's 'spring-test.jar' supersedes the previous 'spring-mock.jar', indicating the stronger focus on the test context framework. Note that 'spring-test.jar' contains everything 'spring-mock.jar' contained in previous Spring versions; hence it can be used as a straightforward replacement for unit and integration testing purposes.

Spring 2.5's 'spring-tx.jar' supersedes the previous 'spring-dao.jar' and 'spring-jca.jar' files, indicating the stronger focus on the transaction framework.

Spring 2.5 ships its framework jars as OSGi-compliant bundles out of the box. This facilitates use of Spring in OSGi environments, not requiring custom packaging anymore.

2.7.1.3. XML configuration

Spring 2.0 ships with XSDs that describe Spring's XML metadata format in a much richer fashion than the DTD that shipped with previous versions. The old DTD is still fully supported, but if possible you are encouraged to reference the XSD files at the top of your bean definition files.

One thing that has changed in a (somewhat) breaking fashion is the way that bean scopes are defined. If you are using the Spring 1.2 DTD you can continue to use the 'singleton' attribute. You can however choose to reference the new Spring 2.0 DTD which does not permit the use of the 'singleton' attribute, but rather uses the 'scope' attribute to define the bean lifecycle scope.

2.7.1.4. Deprecated classes and methods

A number of classes and methods that previously were marked as @deprecated have been removed from the Spring 2.0 codebase. The Spring team decided that the 2.0 release marked a fresh start of sorts, and that any deprecated 'cruft' was better excised now instead of continuing to haunt the codebase for the foreseeable future.

As mentioned previously, for a comprehensive list of changes, consult the 'changelog.txt' file that is located in the top level directory of the Spring Framework distribution.

The following classes/interfaces have been removed as of Spring 2.0:

- ResultReader : Use the RowMapper interface instead.
- BeanFactoryBootstrap : Consider using a BeanFactoryLocator or a custom bootstrap class instead.

2.7.1.5. Apache OJB

As of Spring 2.0, support for Apache OJB was *totally removed* from the main Spring source tree. The Apache OJB integration library is still available, but can be found in it's new home in the Spring Modules project.

2.7.1.6. iBATIS

Please note that support for iBATIS SQL Maps 1.3 has been removed. If you haven't done so already, upgrade to iBATIS SQL Maps 2.3.

2.7.1.7. Hibernate

As of Spring 2.5, support for Hibernate 2.1 and Hibernate 3.0 has been removed. If you haven't done so already, upgrade to Hibernate 3.1 or higher.

If you need to stick with Hibernate 2.1 or 3.0 for the time being, we recommend to keep using the Spring Framework version 2.0.7/2.0.8 which still supports those versions of Hibernate.

2.7.1.8. JDO

As of Spring 2.5, support for JDO 1.0 has been removed. If you haven't done so already, upgrade to JDO 2.0 or higher.

If you need to stick with JDO 1.0 for the time being, we recommend to keep using the Spring Framework version 2.0.7/2.0.8 which still supports that version of JDO.

2.7.1.9. UrlFilenameViewController

Since Spring 2.0, the view name that is determined by the UrlFilenameViewController now takes into account the nested path of the request. This is a breaking change from the original contract of the UrlFilenameViewController, and means that if you are upgrading from Spring 1.x to Spring 2.x and you are using this class you *might* have to change your Spring Web MVC configuration slightly. Refer to the class level Javadocs of the UrlFilenameViewController to see examples of the new contract for view name determination.

2.8. Updated sample applications

A number of the sample applications have also been updated to showcase the new and improved features of Spring 2.0. So do take the time to investigate them. The aforementioned sample applications can be found in the 'samples' directory of the full Spring distribution ('spring-with-dependencies.[zip|tar.gz]').

Spring 2.5 features revised versions of the PetClinic and PetPortal sample applications, reengineered from the ground up for leveraging Spring 2.5's annotation configuration features. It also uses Java 5 autoboxing, generics, varargs and the enhanced for loop. A Java 5 or 6 SDK is now required to build and run the sample. Check out PetClinic and PetPortal to get an impression of what Spring 2.5 has to offer!

2.9. Improved documentation

The Spring reference documentation has also substantially been updated to reflect all of the above features new in Spring 2.0 and 2.5. While every effort has been made to ensure that there are no errors in this documentation, some errors may nevertheless have crept in. If you do spot any typos or even more serious errors, and you can spare a few cycles during lunch, please do bring the error to the attention of the Spring team by raising an issue.

Difference b/w Spring 2.5 and Spring 3.0

Spring framework 3.0	**Spring framework 2.5**
Spring framework 3.0 is compatible with Java 5 and higher versions	Spring framework 2.5 is compatible with java 1.4 and higher versions
It introduces Spring expression language which defines bean definitions based XML spring expression and annotation	In this native expression language exists which is less powerful than language of spring 3.0 and it has different parsing rules
spring 3.0 has type converting system and field formatting	does not support type conversions and field formatting
fully supports the JSR303 bean validation	does not offer complete support API
Comprehensive REST support is available	does not
Spring 3.0 automatically validates the @Controller inputs	Does not validate the @Controller inputs

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: Spring has layed architecture. Use what you need and leave you don't need now.

1. Spring Enables POJO Programming. There is no behind the scene magic here. POJO programming enables contineous integration and testability.
2. Dependency Injection and Inversion of Control Simplifies JDBC (Read the first question.)
3. 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 DB, 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 benifits of Hibernate?

A: These are the key benifits 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.cfg.xml.

```
// Initialize the Hibernate environment
      Configuration cfg = new Configuration().configure();
// Create the session factory
      SessionFactory factory = cfg.buildSessionFactory();
// Obtain the new session object
      Session session = factory.openSession();
```

The call to Configuration().configure() loads the hibernate.cfg.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.cfg.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 (ex: 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 ?

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.

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 reffered 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 JSF and Spring together. This variable resolver is called as DelegatingVariableResolver

18. How to integrate Java Server Faces (JSF) with Spring?

JSF and Spring do share some of the same features, most noticeably in the area of IOC services. By declaring JSF managed-beans in the faces-config.xml configuration file, you allow the FacesServlet to instantiate that bean at startup. Your JSF pages have access to these beans and all of their properties.We can integrate JSF and Spring in two ways:

•DelegatingVariableResolver: Spring comes with a JSF variable resolver that lets you use JSF 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.jsf.DelegatingVariableResolver
      </variable-resolver>
    </application>
  < /faces-config>
```

The DelegatingVariableResolver will first delegate value lookups to the default resolver of the underlying JSF implementation, and then to Spring's 'business context' WebApplicationContext. This allows one to easily inject dependencies into one's JSF-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 (JSF) 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 JSF- Spring integration ?

Spring - JSF 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 transform 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 theSpring 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 springâ€™s 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 jdbcTemplate;

public void setJdbcTemplate(JdbcTemplate 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.

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" />
```