CoreJava | JSP | Servlets | JDBC | Struts | Spring | Hibernate

Projects | FAQs | Sample Programs | eBooks | Certification Stuff

Question Banks | Communities | Tutorials | Softwares | Sample Resumes

Interview Tips | Forums | Discussions | Online Test Engines | Jobs

www.JavaEra.com

A Perfect Place For All Java Resources

Framework: Framework is special software that is built on the top of core technologies having the ability to generate the common logics of the applications dynamically while working with Framework softwares.

1. Programmer just develops application specific logics because the Framework software automatically generates the common logics of application dynamically by using core technologies internally.
2. Framework software provides abstraction layer core technologies internally generate some common logic but they never make programmer bothering or knowing about these core technologies.

EX: Hibernate, Struts, Spring.

Spring: Developed by Interface21 company.

Type → Java EE Framework software

Version: 2.5(Compatable with jdk 1.5+)

Version 3.1 (Compatable with jdk 1.5+)

Open source software(free).

Creator → Mr. Rod Jhonson.

To download software→ www.springsource.org.

As a zip file we have to download.

For documentation and help: www.springframework.org

Good online tutorial: www.roseindia.net

Books → Spring in Action (Manning publisher)

Spring Live

Installation process:

Extract spring-framework-2.5.1-with dependencies.zip software extract spring-framework-2.5.1-with dependencies.zip file to a folder.

<spring home>\List\spring.jar file represents whole Spring API.

This jar file having one dependent jar file called Spring home\lib\jarkartha-commonds\commonds-loging.jar

The classes of spring.jar are using the classes and interfaces of commonds-loging.jar. So commonds-logging.jar file is dependent jar file to Spring.jar

Spring home\dist\spring-sources.jar file represents source code of Spring software.

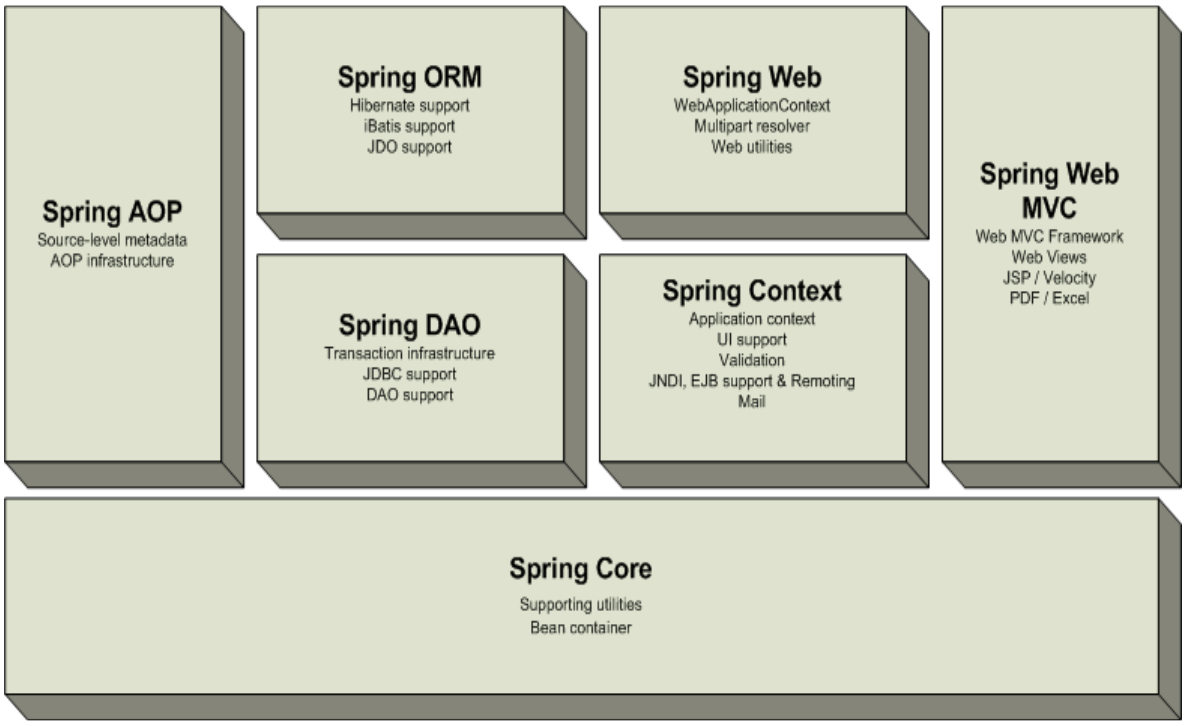
Changelog.txt: which is used to obtain version details.

Spring home\docs folder gives Spring API documentation and pdf files Spring home\lib folder gives helper jar files required for Spring application development. Spring home\mock\samples\test\tiger folders give sample projects and applications.

Spring home\changelog.txt file talks about the release dates and enhancements done in various versions of Spring software.

Spring1.X Framework Overview:

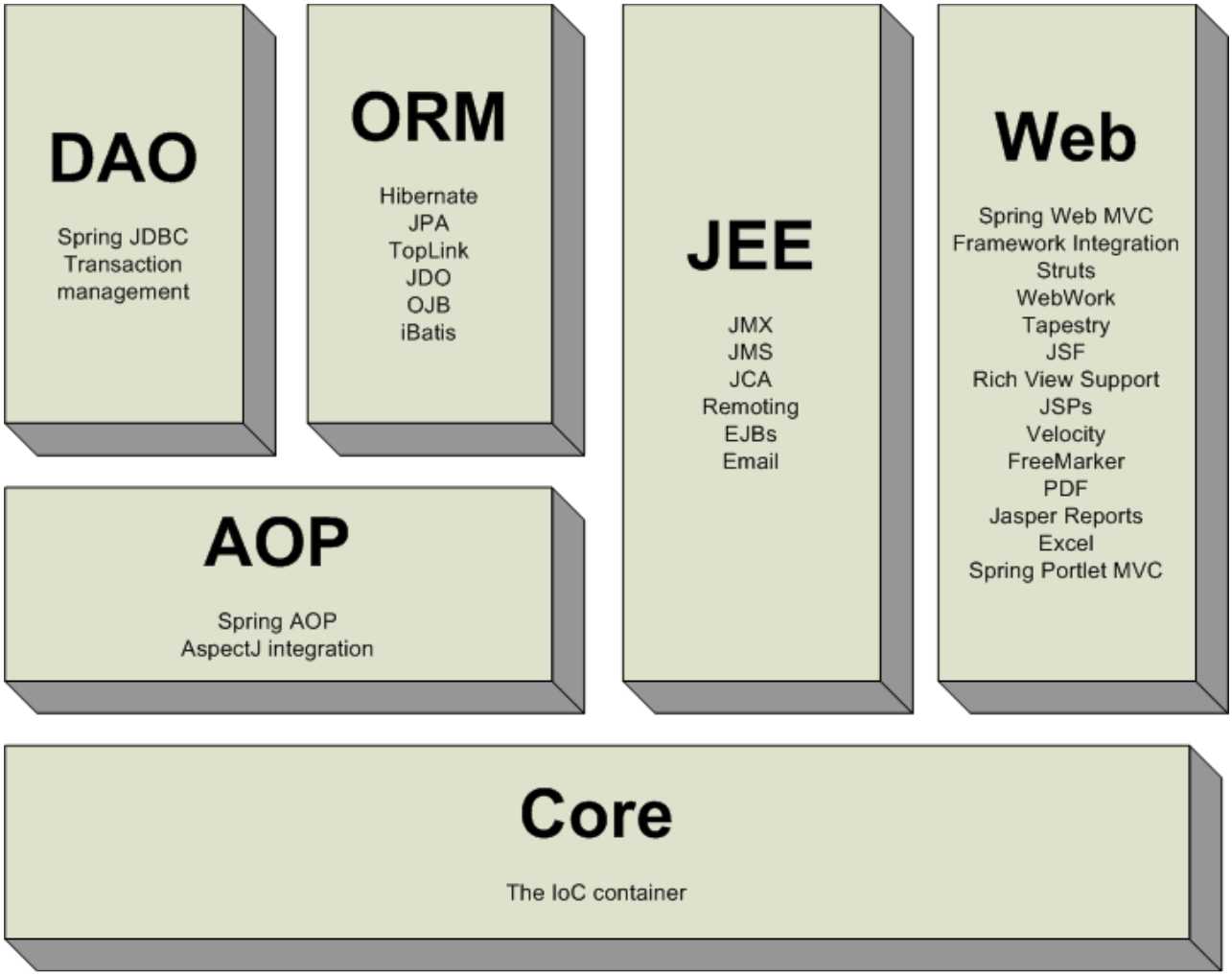
Spring contains a lot of functionality and features, which are well-organized in seven modules shown in the diagram below.



Overview of the Spring Framework

Spring 2.X Overview:

The Spring Framework contains a lot of features, which are well-organized in seven modules shown in the diagram below.



Overview of the Spring Framework

In Spring 2.x:

Web module: Spring .x+ Spring 1.x web MVC module.

JEE Module is nothing but spring 1.x context module.

Core Module: Base module for all Spring modules.

1. Which provides Spring containers(Light weight)

2. Takes care Spring Beans life cycle.

3. The resources in Spring application that are there in the form of Java are called as Spring Beans.

4. Container is a software application which can take care the whole life cycle of given resources.

Spring DAO Module (Data Access object): Provides abstraction layer on Core JDBC technology and allows us to develop JDBC style persistence logic.

ORM Module: Provides abstraction layer on multiple ORM softwares and allows us to develop o-r mapping persistence logic. It provides abstraction layer on Hibernate, Ibatis, JDO, EJB and etc., ORM software

Spring AOP (Aspect oriented programming): It is no way related with OOP. Provides facilities to apply middleware services like security, transaction management and etc., on Spring applications. Middleware services are not minimum logics of application development. They are additional configurable and optional logics to apply on the applications to make applications running smoothly and perfectly in all situations.

Web module:

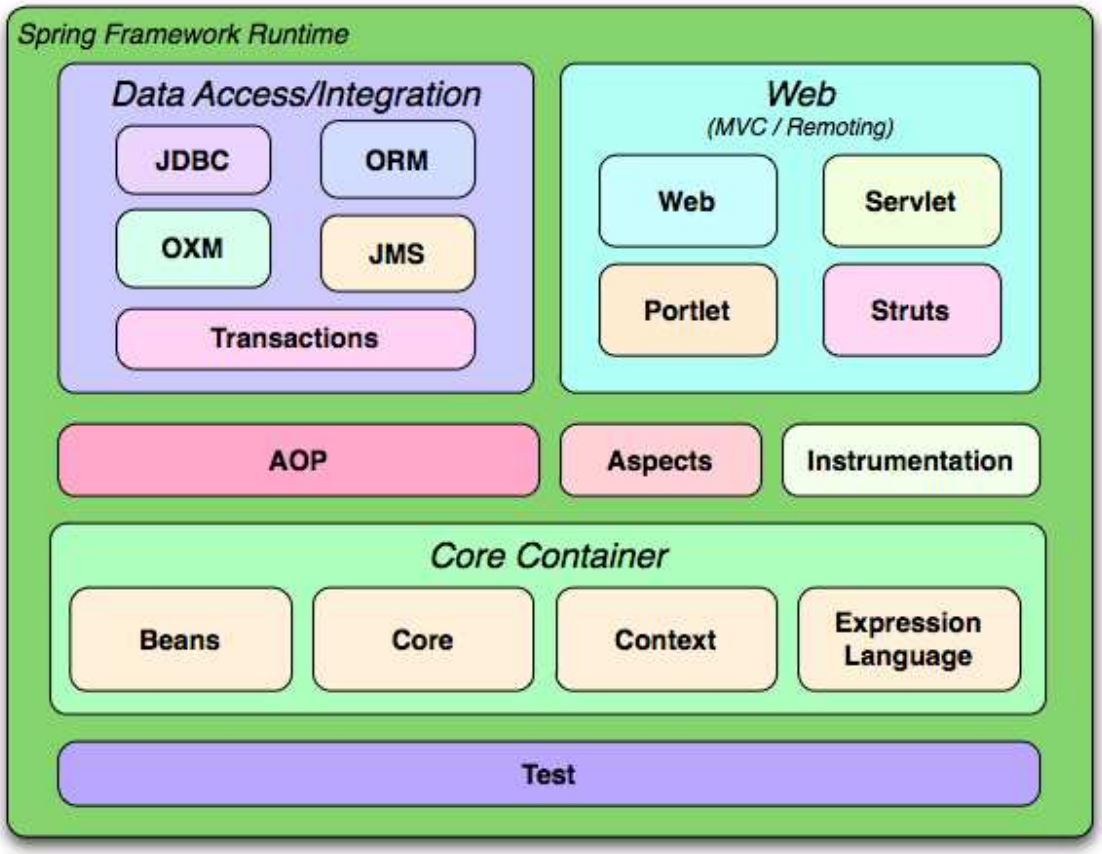
Part-I: It gives facilities to make Spring application communicate from other web framework software based applications like JSF, Struts etc.,

Part-II: It gives Spring web MVC as Spring's own web framework software to develop web applications.

Spring JEE/Context module: Provides abstraction layer on JMS, RMI, EJB, Java Mail and etc., JEE technologies and allows us to develop large scale and complex applications of enterprise.

EX: Banking applications, Shopping websites, credit/debit card processing etc.,

Architecture diagram of Spring 3.x:



Spring def: Spring is an open source, light weight, loosely coupled, aspect oriented and dependency injection based Java-JEE framework software to develop all kinds of Java, JEE applications.

Open source: When we install Spring software its source code will be supplied. So Spring is called an open source software.

Light weight: Spring is light weight software because Spring supplies built-in containers in the form of predefined Java classes by creating objects for classes. We can activate Spring containers.

Note:

1. Servlet, JSP containers are heavy weight containers because we need to start servers to activate them where as Spring containers are light weight containers.
2. The resources of Spring application we can be develop without using Spring API.
3. Spring can be integrated easily with other Java technologies.

Loosely Coupled: Spring software and Spring applications are loosely coupled the reasons are

1. Data required for the resources of the Spring application can be passed through XML files or properties files instead of hard coding approach.
2. While developing Spring applications we can use either few modules or all modules for Spring software.
3. We can use Spring along with other Java technologies in application development like Struts with Spring, Spring with Hibernate etc.,
4. If the degree of dependency is less between two resources/components then they are called as loosely coupled components otherwise they are called as tightly coupled components.

Aspect oriented: Middleware services (Aspects) are not minimum logics of application development. They are additional, optional and configurable logics on the applications to make our applications more perfect and accurate in all situations.

EX:

1. Transaction Management (Do every thing or nothing)
2. Security Logging(Keeps track of application flow)
3. Connection Pooling etc.,

Spring gives a new methodology called Spring AOP and Aspect J to apply middleware services on Spring applications.

Dependency injection or Inverse of Control (IOC):

Q) What is the Dependency Lookup and Dependency Injection?

Dependency Lookup: If resource of the application gathers its dependent values from other resources of the application is called Dependency Lookup in Dependency Lookup resource full the values from other resources.

1. Corse material is dependent value for Student. If student is getting course material only by demanding. It is called dependency lookup.
2. If values are assigned to variables only after calling explicitly is called Dependency Lookup.

EX: The way client application gathers data source object reference from registry software is called Dependency Lookup.

Advantage: Resource can gather its needed values. That means here resource has choice to elect values.

Disadvantage: Resource can need to spend explicit time to search and gather dependent values.

Dependency Injection:

If underlying container or Framework or server or Runtime environment or special resource is dynamically assigning dependent values to resource of the application then it is called dependency injection or ioc.

In dependency injection underlying container/framework and etc., dynamically phases the values to resource.

EX: Assigning material to student immediately after his registration for course is called dependency injection the way Action servlet writes from data to Form Bean class object comdes under dependency injection. The way JRE calls constructor automatically to assign default values to member variables when objet is created comes under dependency injection.

Advantage: Resource need not to spend time to gather input values before utilizing them.

Disadvantage: Underlying framework/container and etc., may inject both necessary and unnecessary values to resource.

Spring supports three modes of dependency injections those are

1. Setter injection (Utilizing setXxx(-))

2. Constructor injection (by using parameterized constructor)

3. Interface injection

Interface injection implementing special interfaces. Any predefined or user defined or third party supplied concrete Java classes can be taken as Spring Bean (Resource in the Spring Application).

MVC2 Architecture: MVC2 architecture is industry defracto standard to develop projects.

M→ Model Layer → business logic +Presentation logic

V → View Layer → Presentation logic

C→ Controller Layer → Integaration locgic/ connectivity

The integration layer talks about view and model layer resources.

Combination of various technologies to develop MVC2 architecture based projects:

Jsp (view) → servlets (controller) → JavaBean/Java Classes (model) → Databse software

The Java class that contains only persistency logic ans separates that logic from other logic s of the application is called as DAO.

Jsp(view) → servlet→java class/ Java Bean → DAO (Persistency logic) → Database software.

1

Jsp(view) → servlket → EJB SessionBean component → EntiotyBean component (Persistency logic) → Database software

Struts Application (view) → EJB SessionBean component (b.logic) → Hibernate (persistency logic) → Database software

2

Struts Application (View & controller) → Spring JEE (business logic) → Hibernate (Persistency) → Database software.

1

Sprting → SpringJEE → Spring DAO → Spring ORM -> Database software

3

Note: We can't use more than 7 micro processors in Windows but we can use 64 micro processors in Linux, SUN Solaris so all enterprise applications using Java.

Posible technologies to develop view layer logic:

Jsp, Freemarker, HTML, Velocity etc.,

Possible technologies to develop controller layer logics:

Servlet

ServletFilter

Possible web framework technologies to develop view and controller Layer logics:

Struts, JSF, Webwork, Spring MVC (Spring web MVC) and etc.,

Possible technologies to develop business logic of model layer:

Java class, Java Bean, RMI, EJD, EJB SessionBean, webservices, SPRING JEE and etc.,

Possible technologies to develop persistency logic of Model Layer:

JDBC, Hibernate, Ibatis, EJB EntityBeans, JDO, TOPLINK etc.,

Note: We can actually use Spring to develop all logics of the project but industry I sunder utilizing. Industry is using Spring in Model layer to jsu develop business logic..

Spring supports POJO/POJO model programming when Java class is taken as resource of certain Java technology based application and that class is not extending and not implementing that technology specific predefined class and interface is called as POJO class.

bEX: If Java class is taken as resource of Spring application and it that class is not extending, implementing the predefined Spring API classes and interfaces then that class is called as POJO class.

EX: Class Test extends Demo

{

}

Class Demo extends HttpServlet

{

}

Test is not POJO

EX2:

class Test implements java.io.Serializable

{

}

Test is POJO

POJI: When Java interface is taken as resource of certain Java technology based application and if that interface is not extending predefined interfaces of htat technology specific interfaces then it is called POJI.

EX: When Java interface is taken as the resource of Spring application and if that interface is not extending resources of other technology specific interfaces such interface is called POJI.

EX:

Public interface xyz

{

}

Interface ABC

{

}

“ABC” is POJI

“XYZ” I POJI

EX2:

Public interface xyz extends java.rmi.Remote

{

}

XYZ is not POJI

Public interface XYZ extends ABC

{

}

Public interface ABC extends java.rmi.Remote

{

}

ABC is not POJI Struts 2.x, Spring all versions, Hibernate all versions , EJB 3.x, JSF all versions gives support for POJO and POJI model programming.

Features of Spring:

1. Supports POJO/ POJI model programming.
2. Gives two ligh weight containers.
    a) BeanFactory container
    b) ApplicationContext container

   Note: Creating object for certain predefined classes we can active containers.

3. Provides abstraction layers on multiple Core technologies Java, JEE technologies and simplifies the process of application development.
4. Can be used to develop all kinds of applications in Framework style like standalone applications, 2tier applications, web applications, Enterprise applications.

5. Gives built-in middleware services like Transaction management, security and etc., and also allows to work with third party supplied or user defined middleware services.
6. Can be used to develop all kinds of logics like presentation, business, persistence and other logics.
7. Provides environment to perform Unit Testing on the application.

Note: UNIT testing means programmer testing on his own piece of code is called UNIT Testing. BEAN Testing means one programmers code will test by another programmer.

8. Gives its own direct technologies to develop the application.

EX: Spring's own distributed technology is **HTTP Involker.**

9. Easy to learn and easy to apply.

10. Easy to integrate with other technologies based applications.

11. Supports all three models of dependency injection

BeanFactory is basic Spring container by creating object for a Java class that implements org.springframework.beans.BeanFactory

We can activate BeanFactory container.

To activate BeanFactory container:

//locate Spring cfg file

FileSystemResource resource=new FileSystemResource

..activate BeanFactory container

XmlBeanFactory factory=new XmlBeanFactory(resource);

In the above code XmlBeanFactory means class of org.sf.beans.factory.XMLPackage implementing BeanFactory interface.

Factory object represents the activated Spring container.

FileSystemResource is class internally uses java.io.File class to locate Spring configuration file.

Spring Core Module:

1. It is base module for other modules of Spring.
2. Designed to support dependency injections and supporting BeanLife cycle operations.
3. Allows us to develop standalone applications.
4. Provides Event Listeners to perform Event Handling on Spring containers.
5. For Core module Spring application we can work local client for JEE module.
6. We can develop both Local and Remote client applications.
7. If application and its client both reside on same JVM then that client is called Local client to application. If application and its client reside in two different computers or same computer then that client is called as Remote client to application.
8. Core module allows interacting only local clients not remote clients.
9. Core module applications are normal Java applications so they allow only local clients.
10. JEE module applications are distributed applications so they allow both local and runtime clients.

Note: IDE softwares are not Frameworks even though they simplify application development because IDEs are not software technologies. Every Framework software technology providing abstraction layer on other core software technologies.

Spring Core module is for developing large scale and web applications. The application that shares the burden of the business logic with multiple server applications and provides location transparency with client the applications is called distributed applications.

EX: Credit card/debit card processing applications and all websites of internet if client applications are able to detect the location change of server applications they are called as distributed applications. Spring core modules applications are there demonstrate dependency injection and Spring Bean life cycle.

The resources of Spring Core Applications:

1. Spring interface (can be a POJI)

2. Spring Bean class (can be a POJO class)

3. Spring Bean configuration file (XML file DTD/ schema(XSD) rules)

4. Client application XSD (XML Schema definition) spring interface.

5. Can be a POJI (plain old java interface)

6. Contains declaration of business methods.

7. Acts as common understanding document between Spring Bean class and client application.

8. Spring Bean class can be a POJO class.

9. Implements business methods having business logic by implementing Spring interface.

10. Contains supporting code for dependency injection (like setXxx methods, param constructors and etc.,)

11. Contains Spring Bean life cycle methods

12. Contains other utility and helper methods.

Spring Bean Configuration File:

1. XML file that can be developed either based on DTD rules or Schema rules. The XML file of certain software technology based applications either based on DTD or Schema rules supplied by that technology.
2. Contains Spring Bean configuration to make Spring Containers to recognize Spring.
3. Contains configuration for setter injections and constructor injections.
4. Any filename.xml can be taken as Spring Bean configuration file name and also contains some misecllanious configurations related to Spring Beans.

Cleint Applciaiton:

1. Must be local to Spring Bean class.
2. Activates Spring containers by specifying Spring configuration file name
3. Gets Spring Bean class object from Spring container calls business methods on Spring Bean class objects.
4. Spring supports three modes of dependency injection.
5. Spring containers uses setXxx methods constructor injections(Spring container uses param constructor for it) Interface injection (Spring container uses special predefined Spring interfaces for it).

Understanding setter injection:

```
public class TestBean

{

//bean property

private String msg;

//write setXxx method supporting setter injection

public void setMsg(String msg)

{
```

this.msg=msg;

}

//business method

public void bm1()

{

}

}

Spring cfg file(XML file) demo.xml:

<bean id=”tb” class =TestBean”>

<property name=”msg”>

<value> hello </value>

Value to be injected

</property>

</bean>

When this Spring configuration file given to container the container loads TestBean class and create object for it having name “id” and calls setMsg(“hello”). setMsg(“hello”) method is called that to inject hello value to the bean property “msg”

Note:

1. The member variables of Spring Bean class are called as Bean properties in real time projects this dependency injection will be used to inject special objects to Bean properties like JDBC connection object data source object and etc.,
2. Here Bean property values can be changed through XML files. This shows loose coupling between Bean properties and its values.
3. Any concrete class (User defined or predefined or third party supplied). Java classes can be taken as Spring Bean.
4. Every Spring Bean need not be a Java Bean but we take Java Bean also as Spring Bean

Understanding Constructor Injection:

Every Spring Bean is not Java Bean

```
public class TestBean
{
//bean property
private String msg;
//param constructor
public TestBean(String msg)
{
this,.msg=msg;
}
public void bm1()
{
------
------
}
}
```

When Spring cfg file is given to Spring container. The container loads TestBean constructor. In this process the value hello will be injected to bean property “msg”

Note: In setter injection the Spring container calls setXxx methods to assign values to bean properties on constructor injection. Spring container uses parameterized constructor to create Spring Bean class object and to assign values to bean properties file.

Procedure to develop and execute Spring application:

1. First develop configuration file of Spring. Don't develop configuration file your own. Copy configuration file from Spring software and paste it in Notepad. Open it with Notepad++ or Edit Plus and change the values according to following procedure.

How to find configuration file (DTD file) in Spring software

Simple just go to Spring software and select search option and select files and folders option and type .xml then we will get lot of file copy one of the file and save it.

spring.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>

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

<beans>

        <bean id="db" class="spr.DemoBean">

                <property name="msg">

                        <value>hello</value>

                </property>

                </bean>

</beans>
```

Now Develop Demo.java interface

```
package spr;

public interface Demo

{

 public String generateWishMsg(String unmae);

}
```

DemoBean.java class and DemoClient.java classes.

Now set path

Now Develop DemoBean.java

```
package spr;
import java.util.Calendar;
public class DemoBean implements Demo
{
   //bean property
   private String msg;
   public DemoBean()
   {
      System.out.println("Demo Bean: 0-Parameter Constructor");
   }
   //setter methods supporting setter injection
   public void setMsg(String msg)
   {
      this.msg=msg;
      System.out.println("Demo Bean: setMsg()");
   }
   //implemets business methods.
  public String generateWishMsg(String uname)
  {
     //get current Date and time
     Calendar cl=Calendar.getInstance();
     //get current hour of day(24 hours)
     int h=cl.get(Calendar.HOUR_OF_DAY);
     //get message
```

```
    if(h<=12)
    {
       return msg+"Good Morning"+uname;
    }
    else if(h<=16)
    {
       return msg+"Good Afternootn"+uname;
    }
    else if(h<=20)
    {
       return msg+"Good Evening"+uname;
    }
    else
    {
       return msg+"Good Night"+uname;
    }
  }
}
```

Now Develop DemoClient.java

```
package spr;

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

public class DemoClient
```

```
{
   public static void main(String[] args)
   {
      //local Spring Cfg file configuration
    FileSystemResource res=new FileSystemResource("C:\\Demo.cfg.xml");
      //Activate Spring Container
      XmlBeanFactory factory=new XmlBeanFactory(res);
       /* get Spring Bean class object from Spring Container*/
            Object obj=factory.getBean("db");
            //type casting
            System.out.println("obj is Object of"+obj.getClass());

            Demo bobj=(Demo)obj;
            //call Business methods
            System.out.println("Result is"+bobj.generateWishMsg("Bhargav"));
   }

   }
```

How To Run the application

Set following jar file locations to class path

C:\Spring-framework-2.5.1\dist;C:\Spring-framework-2.5.1\lib\jakarta-commons.

Now run the application.

For complete program please refer Spring Core Module\NetBeans 7.1(Spring)\FirstSpringApplication folder

Spring container can perform dependency injections only on the Spring container created Spring Bean class objects that means Spring container can't perform dependency injections on user defined Spring Bean class objects

When Bean Container is activated it reads and verifies the entries of Spring configuration file. Spring caontaiuners internally uses SAX parser of XMl to read and process Spring configuration file entries. XML parser is a software application that can validate XML document against dtd/Schema rules and can read, process XML documents.

List of Parasers:

DOM Parser (Document Object Model)

SAX Parser (Standard API for XML processing)

JDOM Parser (Java Document Object Model)

DOM4J Parser (Document Object Model for Java)

Observations on First Spring application:

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

1. Makes SpringBean Factory container to load the Spring Bean class DemoBean whose value is db.
2. Makes BeanFactory container to create DemoBean class object having name by using zero parameter constructor.
3. Makes Spring container to perform setter injection on msg property to assign the value hello. This is used to <property> that is used to configure "msg" property.
4. Makes BeanFactory container to return the above created DemoBean class object we are referring that object and we are referring that object through java.lang.Object reference variable.
5. When super class reference variable points one of its sub class object we can't use that reference variable to call the direct methods of sub class to make this operation possible we need to type cast the reference variable either to sub class or interface by sub class having these direct method declarations it is not recommended to expose SpringBean class to client application instead of that is recommended to expose Spring interface.

```
Object obj=fatory.getBean("db");
Demo bobj=(Demo) obj' //good
    (or)
DemoBean bobj=(DemoBean); //bad
```

OOPS Recap:

```
interface ABC
   {
      public void k();
}
class Demo implements ABC
   {
      public void k()
      {
         System.out.println("I am from k()");
      }
      public void l()
      {
         System.out.println("I am from l()");
      }
   }
class prs
{
   public static void main(String[] args)
   {
Demo d=new Demo();
d.k();
d.l();

ABC a=(ABC)d;
a.k();
   }
}
```

When Spirng bean is configured without Bean id then either Bean class name or Bean class name#0 will be taken as default bean id. To configure simple values as the value to be injected we can use values attribute instead value tag.

```
<?xml version="1.0" encoding="UTF-8"?>

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

```
<beans>
        <bean  class="spr.DemoBean">
        <!--no bean id-->
                <property name="msg">
                        <value>hello</value>
                </property>
                </bean>
</beans>
```

When sample Bean class is configuring for multiple times bean id then container uses <bean class name>#<n> as Bean ids.

For complete program Please refer Spring Core Module\TestAppWithoutHavingBeanID folder

Note: This program is not Working in NetBeans IDE

Example application on Constructor injection:

demo.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
"http://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>
        <bean id="db" class="spr.DemoBean2">
                <property name="msg">
                        <value>hello</value>
                </property>
                </bean>
</beans>
```

For Complete Program Please refer Spring Core Module\NetBeans 7.1(Spring)\FirstSpringApplication→ Here we have to refer ParamConstructor Program.

Q) What happens if same bean property is configured for setter and constructor injection with different values?

A) Since setter(-) executes after constructor will be execute so the value passed through setter injection will override the value injected by constructor injection but Bean object will be created using parameterized constructor.

For Complete Program Please refer Spring Core Module\NetBeans 7.1(Spring)\FirstSpringApplication→ Here we have to refer ConstructorSetterProgram

This value will be applied if all bean properties are configured for setter injection then Spring constructor creates Spring Bean class object using zero parameterized constructor. If any Bean property configured for constructor injection then Spring container creates object using parameterized constructor and Spring configuration file is mandatory **but** its configuration is optional.

We can locate Spring cfg file for BeanFactory container in two ways

1. Using File System Resources
2. Using classpath resource class

1. **Using File System Resources:** Locates the Spring configuration file from the specified path of the class.
2. **Using classpath resource class:** locates spring configuration file from the directories and jar files are added to class path environment variable.

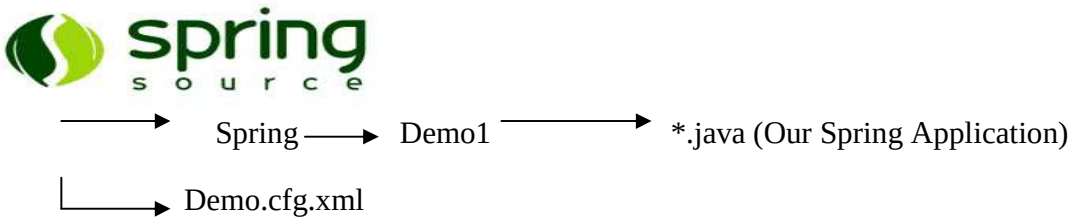
All files and folders maintain by OS together is called as File System . While working with FileSystemResource class we can pass Spring configuration file either by using absolute path or by using relative path.

Example to work with FileSystemResource:

FileSystemResource fs=new FileSystemResource(“Democfg.xml”);

Locates the Spring cfg file from current catageory

C:\ Spring program



In DemoClient.java

FileSystemResource res=new FileSystemResource("C:\\Apps\\Demo.cfg.xml");

(Absolute Path)

FileSystemResource res=new FileSystemResource("----//---//Demo.cfg.xml");

Relative path

Working with ClassPath Resource:

My Compute→Environment variables → System/user variables

In DemoClient.java:

ClassPathResource =new ClassPathResource("Demo.cfg.xml");

XmlBeanFactory factory=new XmlBeanFactory(res);

Note:

While Spring module application is using BeanFactory container it is recommended to work with FileSystemResource.

The <property> that can hold object as value is called as referenced type property.

String, Primitive data type properties are called simple properties.

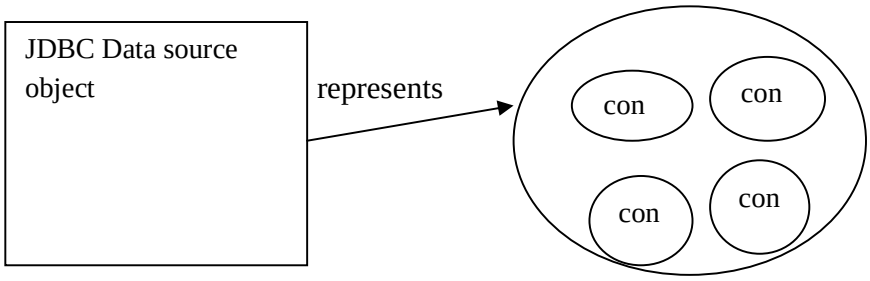
To configure dependent values for Simple property use <value> or value attribute. To configure dependent values for reference properties use reference tag key or ref attribute.

Example application on injecting Objects to Bean properties as dependent values through setter injection.

For Complete Program Please refer Spring Core Module\NetBeans 7.1(Spring)\FirstSpringApplication here verify RefVariable.java program

Adding JDBC properties as ref variables: JDBC Connection pool is a factory that contains set of readily available JDBC connection objects. JDBC Datasource objects represent JDBC connection pool and client applications to use this data source object to access JDBC Connection

Connection objects from Connection Pool JDBC Datasource object means it is the object of Java class that implements javax.sql.DataSource interface.



Spring API gives org.sf.jdbc.datasource.DriverDataSource class gives JDBC Datasource object representing spring container managed JDBC connection objects in this Connection Pool will be created based on the JDBC driver details supplied by the programmer as Bean property values. The bean properties are

i) driver class name

ii) url

iii) user name

iv) password

To gather bean property names predefined classes that can be taken as refer their setXxx methods.

For complete program please refer Spring Core Module\NetBeans 7.1(Spring)\JDBCRefVariable folder\SelectClient.java

In DriverManagerDataSource class the type of Connection property is java.util.Properties class so we can use <props> to configure this bean property to specify the Database user name and password as shown below

```
<property name="connectionProperties">
        <props>
         <prop key ="username">scott</prop>
         <prop key ="password">tiger</prop>
          </props>
      </property>
```

For complete program please refer Spring Core Module\NetBeans 7.1(Spring)\JDBCRefVariable folder\SelectClient1.java

MyEclipse: MyEclipse=EclipseIDE+built in plugin

Plug-in: Plug-in is a patch software or software application that can be used to enhance the functionalities of existing software and software application. In Java environment plugins comes as jar files. Plugins of IDE simplifies the process of working with software technologies.

Eclipse	MyEclipse
1. Open source 2. Does not provide built-in plug-ins to work with advanced technologies but allows to add plug-ins. 3. Suitable for small scale projects. 4. Gives basic JDK setup to develop the applications. 5. Allows adding external plug-ins to work with advanced technologies.	1. Commercial 2. Provide built in plug-ins and allows to add plug-ins externally. 3. Suitable for large scale projects. 4. Gives basic JDK setup and advanced setup to develop the applications. 5. Allows adding external plug-ins to work with advanced technologies.

Note: Easy Eclipse, Eclipse Galilio and etc., are alternative for MyEclipse which are developed based on Eclipse.

MyEclipse Details:

Type: IDE software fo Java Environment.

Vendor: Eclipse org

Version: 8.x(compatable with jdk 1.5,1.6)
Commercial IDE.

For help details: www.myeclipseide.com

Gives Tomcat as default server and allows configuring other servers.

List of tags to perform dependency injection on different types of bean properties:

bean property type	Tag to use

simple datatype, String	<value>
bean reference	<ref> or ref attribute
array	<list>
java.util.List	<list>
java.util.Set	<set>
java.util.Map	<map>
java.util.Properties	<props>

Procedure to develop Spring Core module application by using MyEclipse IDE:

For complete process please refer Spring Core module\WorkingWithMyEclipseIDE

For Complete project please refer Spring Core Module\My Eclipse8.x projects(Spring)\MyTestProj folder

In IDE environment File System resource class looks for Spring configuration file in project folder by default. So we need to specify relative or absolute path of the file in client application as shown below.

In TestClient.java:

```
"C:\\Documents and Settings\\"
        + "Administrator\\My
Documents\\NetBeansProjects\\FirstSpringApplication\\src\\Demo3.cfg.xml");
```

Example application that shows setter injection on java.util.Set, java.util.Properties class type bean properties.

For complete program please refer Spring Core Module\NetBeans 7.1(Spring)\SetMapPropertiesSetterInjection folder

In the bean configuration file keeping <!DOCTYPE---→ statement and XML schema related statements are mandatory.

Understanding different models of constructor injection: If values are injected to bean properties by spring container through parameterized constructor then that is called Constructor injection.

For example application please refer ConstructorInjection.java in Spring Core Module\NetBeans 7.1(Spring)\ConstructorInjection

Note: The parameterized constructor that is required to perform constructor injection will be picked up based on the number of <constructor-arg> placed under <bean>. In the above example

<constructor-arg> is placed for three times so Spring container uses three param constructor for Spring Bean class Object creation and for Constructor Injection.

In Spring cfg file we need to specify value to the constructor parameters in the order they are available in parameterized constructor as shown in the above application. If we want change this order of passing values then we need to identify/resolve the parameters either based on type or index based.

Resolving/identifying constructor parameters through their datatypes:

For example application please refer IdentifyingThroughthrierDatatypes.java in Spring Core Module\NetBeans 7.1(Spring) \ConstructorInjection

If multiple parameters of construcotrs contains same type then it is recommended to identify them based on their indexes.

Resolving/identifying constructor parameters based on indexes:

For example application please refer IdentifyingThroughIndex.java in Spring Core Module\NetBeans 7.1(Spring)\ConstructorInjection

Q) Can we analyze when to go setter injection?

A) While developing our Spring Bean class if there is only one bean property then we can go for constructor injection because it is quicker than setter injection. If our bean class contains multiple bean properties then design the class supporting setter injection because by using minimum setter methods we can allow the programmer to configure injections either or all bean properties or his/her choice bean properties.

Example scenario: If Spring Bean class having five bean properties then five setter methods are enough to allow setter injection in any angle.

In the above scenario we need 120+(5!) parameter constructors are required for these five bean properties to allow constructor injection on Bean properties in any angle but writing 120+ constructors for the sake of five bean properties is meaningless so it is recommended to go for Setter Injection.

Most of the Spring classes given by Spring API are designed supporting Setter Injection.

Example Application on Constructor injection on different types of bean properties:

For example application please refer ConInjDifferentTypesOfBeanProps.java in Spring Core Module\NetBeans 7.1(Spring)\ConstructorInjection

Note: When all bean properties are configured for setter injection then Spring Container uses a parameter constructor to create Spring Bean class object. If any property is configured gor Constructor Injection then Spring container creates Spring Bean class object by using parameterized constructor.

Q) Can we place only parameterized constructor in Spring Bean class when all bean properties are configured for setter injection?

A) Not possible because Spring container looks to use zero parameter constructor in the above situation to create Bean class Object and it is not available and also not coming as default constructor.

We can develop Spring configuration file based on existing Spring configuration file for this we need to activate two Bean Factory containers as parent, Child Containers.

Example application to work with parent, child Bean properties:

For complete program please refer Spring\NetBeans7.1(Spring)\twoProperties

Q) What is the difference between "local attribute and Bean attribute" of ref tag?

A) local attribute looks for the given id based dependent Spring Bean only in the current container file where as Bean attribute searches for dependent bean in the Spring configuration file of current container and in the Spring configuration file of parent container.

<property name="d">

<ref bean="dt"/>

</property>

In the above syntax <ref bean="dt"/> looks for this dependent Bean in the current container related Spring configuration file if not available it looks in parent container related Spring configuration file.

<property name="d" ref="dt"/>

In the above syntax <ref ="dt"/> is same as >ref bean="dt"/>

<property name="d">

<ref =local="dt">

</property>

Here <ref =local=”dt”> looks for Spring (local=”dt”) this dependent bean only in the current container related Spring configuration file.

<property name=”d” local=”dt”/>

Here local=”dt”/ is invalid attribute. In <constructor-arg> ref attribute is there but there is no local attribute.

Note: This process is rare in real time.

Q) When do go for Constructor Injection and when to go for Setter Injection while developing real time projects?

A) If Spring Bean class is providing appropriate parameterized constructors then we have to go for constructor injection otherwise its best to go for Setter Injection.

Bean Wiring: Most of the Spring API supplied Spring Beans are designed supporting Setter Injection. The process of configuring Spring Bean and configuring Bean properties with dependent values in Spring Configuration file is called as Bean Wiring.

Singleton Java class: The Java class that allows to create only one object per JVM is called as Singleton Java class. If underlying container or run time environment creates only one object even though class allows to create multiple objects then that class is not called as Singleton Java class. Our Servlet program class is not Singleton class in any angle. When programmer tries to create multiple objects for Java class if Java class is allowing create only one Object then that class is called as Singleton Java class. Instead of creating multiple objects having same data it is recommended to create only one object and using for multiple times. Our Spring Beans are not singleton classes and container never makes them Singleton classes.

Q) How to affirm your Servlet is not Singleton class? Servlet container is creating only one object so it is Singleton?

A) Servlet class doesn’t contain any private constructor and also Factory method. So I can affirm Servlet is not singleton class.

We can make Spring Container to create Spring Bean class object in one of the following four scopes.

1. Singleton (default)
2. Prototype
3. Request
4. Session

Note: If we use toString() on object then one output value will come that is not Object Hash Code. It is Hexadecimal value of Hash Code.

Note: For execution of each application one JVM will start and stop.

Singleton: Spring Container creates only one object for Spring Bean on one per JVM basis but it never makes Spring Bean as Singleton Java class. When scope Singleton the Spring container creates and returns one Spring Bean class object for multiple factory.getBean method class.

In Demo.cfg.xml of XML of Application1

Note: When Object is created by Cloning or Deserialization then Constructor never execute.

Prototype: Spring container creates and returns multiple Spring Bean class objects for multiple factory.getBean method calls on one per method call basis.

For complete Program please refer Spring Core Module\NetBeans 7.1(Spring)\ObjectHashCode folder

request: In web environment keep Spring Bean class object as request attribute value that means creates Spring Bean class object on one per request basis.

session: In web environment keeping Spring Bean class Object as session attribute value. That means Spring Bean class Object one per browser window basis.

Examples of Static Factory Methods:

Class c=Class.forName(“Test”);

Thread t=Thread.currentThread();

Examples on instance factory method:

String s=new String(“hello”);

String s1=s.concat(“fine”);

OUTPUT: Hello Fine

The method of Java class that is capable of creating and returning objects is called as Factory method. There are two types of Factory methods.

1. **Static Factory Method:** When class is having only private constructors then these methods are useful to create objects outside to the class.

EX:

i. Class c=Class.forName("Test");

ii. Thread t=Thread.currentThread();

2. **Instance Factory Method:** To create new object for Java class by using existing object and its data we use Instance Factory methods.

i. String s=new String("hello");

String s1=s.concat("fine");

Output is hello fine

ii. Stirng s=new String("Sathya Tech");
String s1=s.subString(2,5);

Note: The Factory method of a class returns either its own class object or different class object.

Some more examples on static Factory methods:

EX: Calendar cl=calender.getInstance();

Here getInstance does bot return Calender class object because Calendar is an abstract class. This method is returning one sub class object(Gregorian Calendar) of Calendar class. So we are able to refer object with Calendar reference variable.

Some more examples on Instance methods:

Date d=new Date();

String s=d.toString();

Here d.toString is Instance Factory method. We can make Spring Container to create Spring bean class Object either by using static Factory method or instance factory method for this we need to use Factory method, Factory bean attributes.

To specify static Factory method we can use Factory method attribute similarly to specify Instance Factory method we need to use Factory method attribute along with Factory bean attribute. We can use <constructor-arg> either pass parameter values to constructors or pass argument values to specified methods.

For complete program please refer Spring Core Module\NetBeans 7.1(Spring)\StaticFactoryMethod folder

Note: While working with java.util.List, Set, Map, Properties we can configure the dependent values directly because Spring supplies <list> <set> <map> <Props> for this. For renaming types of reference properties direct tags are not given so first we need to configure Spring Bean that

gives dependent object and configure that object as dependent values to bean property using <ref> or “ref” attribute.

Application Context Container: It is an endhancement for BeanFactory Container to activate Application Context Container to create object for the Java class that extends Appliation Cotnext Container. This is sub interface of BeanFactory(Interface)

There are three important implementation classes for ApplicationContext(Interface). They are

1. FileSystemXmlApplicationContext(org.sf.context.Support)

It activates Application, Context Container by locating Spring configuration file for the specified path of the FileSystem.

2. ClassPathXmlApplicatinContext

Org.sf.contxt.support.pkg

Activates Application Context Container by usingSpring cfg file from entries of class path environment variable.

3. XmlWebApplicationcontext

org.springframework.web.context.support

Acitvates Application Context Container by using Spring cfg file in web applications.

It activates ApplicationContextContainer in web application environemt.

Activating and using Application Context Container with respect to first Spring Application

Application Context Container can perform every activity of BeanFactory container but it also gives the following additional features.

Note: Application Context container is originally part of Spring context/spring JEE module.

1. Ability to work with multiple Spring cfg files.
2. Pre-Installation of Singleton class.
3. Ability to work with properties files.
4. Support to work with I18n(Internationalization)
5. Ability to work with Event Listeners and etc.,

Note: Application Context Container is originally part of Spring Context/Spring JEE module. By activating BeanFactory container for one time we can’t make that container dealing multiple Spring configuration files but this activity is possible by using Application Context Container.

For complete program refer Spring Core Module\NetBeans 7.1(Spring)\ApplicationContextContainerPrograms folder

BeanFactory container does not create Objects for Spring Beans when it is just activated but it creates bean class object and completes dependency injection on those objects when Factory.getBean(-) method is called. When Application Context container is created it creates object for ll singleton scope Spring Beans and also completes dependency injection on those beans this is called pre installation of Spring Beans.

Note: Application context container can perform pre instantiation only one Singleton scope Beans. Working with Singleton scope Spring Bean

factory.getBean(-) creates and returns Spring Bean class object.

Ctx.getBean(-) returns already instantiated Spring Bean class Object.

Ctx returns Application Context Container.

Q) When there are 10 Spring Beans in Spring Configuration files how can we make Application Context Container performing pre instantiation only on four beans?

A) Configure four beans as Singleton scope beans and configure beans with prototype scope.

Q) What happens when prototype scope bean is dependent to Singleton scope bean while working with Application Context Container?

A) Container creates prototype scope bean class object along with singleton class object through pre instantiation process but does not change prototype to Singleton scope.

<bean id=”tb” class=”TestBean” scope=”prototype”/>

<property name=”msg” value=”hello”/>

<bean id=”db” class=”DemoBean” scope=”singleton”>

<property name=”name” value=”hello”/>

</bean>

<bean id=”db” class=”DemoBean” scope=”singlecon”/>

<property name=”t1” ref=”tb”/>

</bean>

</beans>

Dealing with two configuration files:

For Two configuration files program please refer Spring Core Module\NetBeans7.1 (Spring)\TwoConfigurationFilesOfAplicationContextContainer

Properties File: The text file that maintains the entries in the form of key=value pairs is called as text properties file.

Myfile.properties:

#properties file(comment)

name=raja

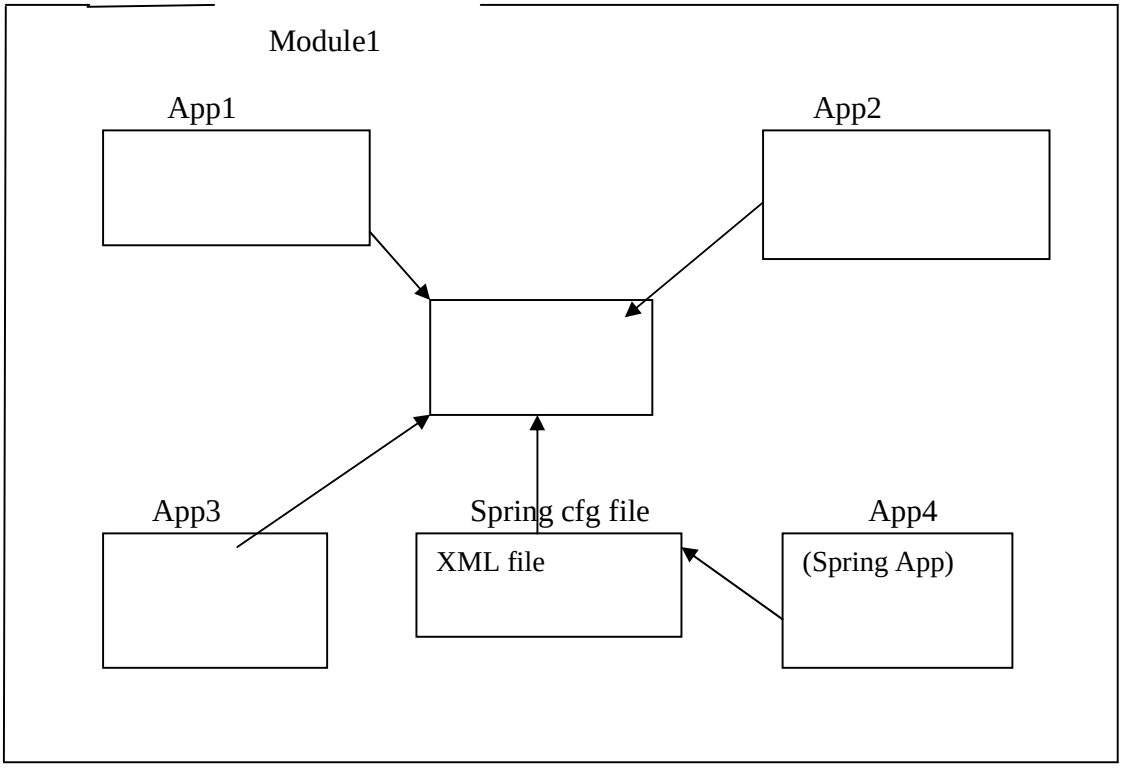
age=30

Here name,age are keys and raja,30 are values.

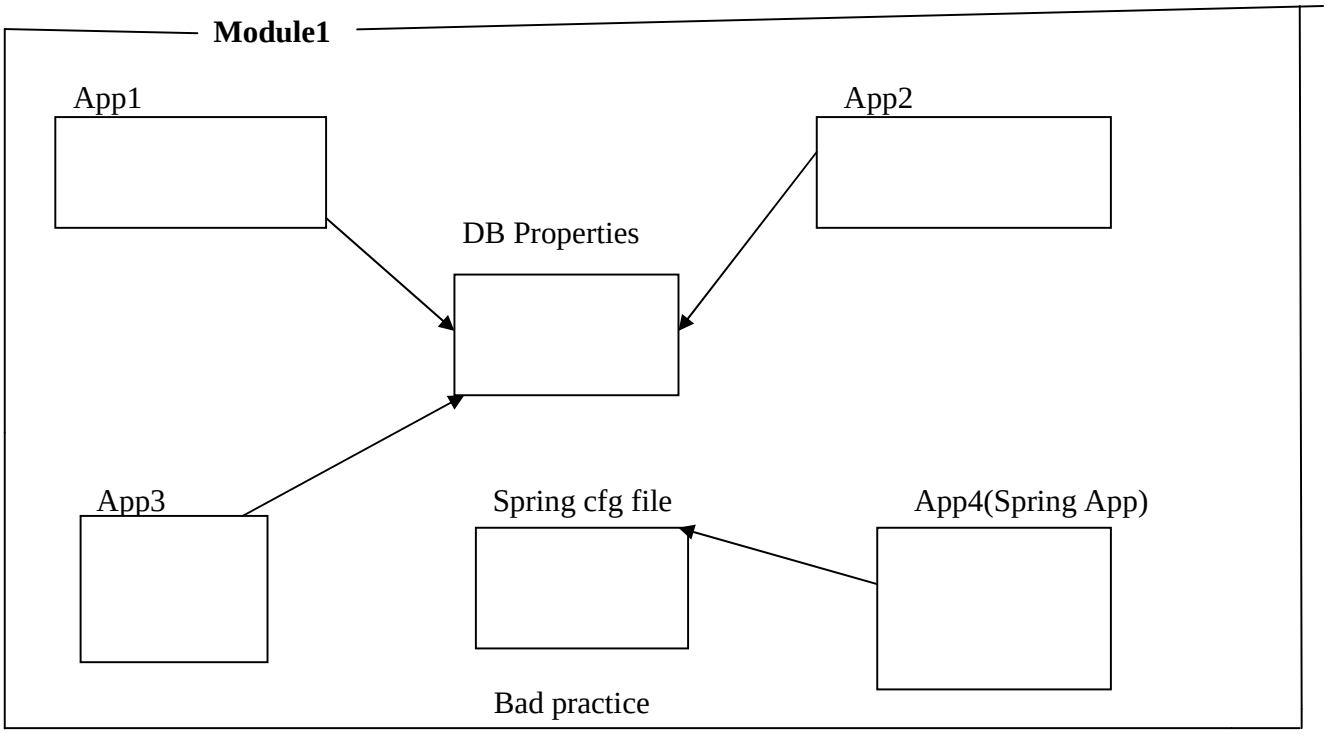
In JDBC applications use properties file to pass Database, JDBC driver details to applications from outside the applications to get the flexibility of modification.

In Spring environment this can be done through spring configuration file but if **other applications of the module** are dealing with properties file then it is recommended to see Spring application also dealing with properties file through XML file.

Good Practice:



Bad Practice:



Centralization of Database properties were missed.

App1, App2, App3 are non spring Apps(like JDBC applications)

Note: Spring application can't directly communicate with the properties file it is done by using XML file only.

Database properties and JDBC driver properties nothing but driver name, url, uname, password.

To make Spring configuration file collecting bean property values from properties file we need to apply place holders having key names of properties files to make Application Context container recognizing these place holders. We need to configure one special Spring bean that is org.sf.beans.factory.config.PropertyPlaceHolderConfigurer

Syntax of placeholder:

${<key name>}

Bean Factory container can't work with properties files. Appliation Context container can work with properties

For complete program please refer Spring Core Module\NetBeans 7.1(Spring)\PropertiesFile folder

For dealing with multiple properties files

Please refer Spring Core Module\NetBeans 7.1(Spring)\twoProperties folder

Internationalization (I18N):

Making our application working for different locales(countries+languages) by doing modifications only in presentation logics is nothing but enabling I18N on the application Locale means Country + Language

en-US

fr – FR (French as it Speak in France)

fr – CA

Note: when i18n is enable our project can be used by multiple client organizations belonging to multiple locales.

To enable I18N on the application we need to take multiple properties files for multiple locales including base files.

Native to ASCII: When no properties files file found base properties file will be utilized. The base properties file docs not contain country code and language code but the remaining locale specific properties files contain country code and language code. We can also place Unicode numbers in properties files to render the tables in different languages.

Note: Keys must match in all properties files

I18N does not deal with business and persistence logics it just deals with presentation logic by gathering labels from an appropriate locale specific properties file Bean Factory container does not give support for I18N where as Application Context gives support. To make Application Context container ready for I18N we need to configure special Spring Bean resource bundle message source with fixed bean id message source and specifying set of property file names as the values of base name property files.

I18N based spring example application:

Spring Core Module\My Eclipse8.x projects(Spring)\MyTestProj folder

For complete program please refer Spring Core Module\Spring Programs(with out IDE)\I18N folder

To set command arguments for NetBeans IDE please right click on the project → properties → run → set arguments here those are fr Fr like this. Don't put coma.

Q) What are the commonalitites and difference between BeanFactory and ApplicationContext container.

Feature	BeanFactory	Application
Dependency injection/writing	No	Yes
Bean Instantiation	No	Yes
Pre instantiation	No	Yes
Ability to work with multile spring configuration file in single container environment	No	Yes
Support for properties files	No	Yes
Ability to stop the container explicitly	No	Yes

Event publishing	No	Yes
Automatic registration of Bean post processor Factory	No	Yes

Q) When to go for bean factory and when to go for Application factory container?

A) While executing Application if the memory is the issue like Applets, mobile applications then we may think about using Bean Factory container. In all other medium and large scale appliations use Application context container even though it needs few KB extra memory during execution to Bean Factory container.

Event Handling feature: Event is an action performed on the object/component. Event handling is the process of executing some logics when event is posible and it allows us to keep track of when that container is activated/stopped. This allows to know the amount of time that is consumed to execute Spring based business logic. For this Event Handling we need

Source obj: Application Context Container (Bean Factory container does not support)

Event: Application Event

Event Listener: Application Listener(org.sf.context)

Event Handling Method: public void onApplicationEvent(-)

(This method executes when container is started and stopped)

For Complete program please refer Spring Core Module\NetBeans(Spring)\SpringListener folder

Auto wiring:

1. The bean properties of Spring bean class can be taken as static or non static member variables but the setter methods taken for setter injection must be non static methods.
2. Spring Container tries to perform injections with respect to specific bean class objects and static methods are command for multiple bean class objects.
3. Configuring bean properties explicitly for injections is called as explicit wiring for this we use property <constructor-arg>.
4. Making Spring container deciding the dependent values dynamically and automatically based on configurations done in Spring configuration file is called **Auto wiring.** Here property <constructor-arg> will not be used but container injects the dependent values. (It is also not industry recommonded approach)

Limitations with autowiring

1. Auto wriing is posible only on reference type bean properties that means auto wiring is not posible on simple, String , Collection Framework type properties.
2. Autowiring kills the readability of Spring configuration file.
3. To configure Autowiring we need to use “autowire” attribute of <bean>
4. This autowiring is posible in multiple modes those are

i. byName(autowire=”byName”)
ii. byType(autowire=”byType”)
iii. constructor (autowire=”constructor”)
iv. autodetect(autowire=”autoDetect”)

Example application on ByName mode autowiring: byName mode of autowiring performs setter injection for this dependent bean class object name(id attribute value) must be bean property name)

For Complete Project Please refer Spring Core Module\NetBeans(Spring)\BeanWiring folder

Container injects the TestBean class object tb(having value hello) to the “tb” property of DemoBean class through setter injection.

byType mode Autowiring: Performs setter injection on Bean proprties for this the type of bean proeprty and the type of dependent of bean object must match.

While working with byType mode of autowiring there is a posibility of getting ambiguity problem.

Cosntructor mode of Autowiring: Performs constructor injection by using parameterized constructor. When ambiguios state rises the injection will be ignore.

autoDetect or autoWiring mode: Perofrms setter injection(by type mode) on bean property if bean class contains 0-parameterized constructor otherwise it performs constructor injection.)

We can disable autowiring on bean properties of Spring Bean class then we use configure=”no”

If we configure both explicit, autowiring on same bean property and both performing setter injection or constructor ibnjection then the values given by explicit wiring will be affected.

If we configure both explicit autowiring on same bean property one performing constructor injection and other one performing setter injection then setter injection values will be affected if there is no ambiguity problem.

Life Cycle methods: Life cycle methods are methods which executes automatically based on Life cycle events. Spring Bean allows the programmer to specify two user defined methods as life cycle methods. These methods must be configured in spring configuration file during spring bean configuration. Those are

init method: Which is user defined method which will be called by spring container right after bean instantiation and dependency injections.

In generally place the logic to verify whether required values are injected or not to bean properties.

destroy method: This user defined method will be called by Spring container when Spring bean class is about to destroy.. We place generally logic to modify bean properties in this method. This method is also useful to release non-Java resources like JDBC Connection object from Spring Bean.

For working process of NetBeans please refer Spring Core Module\Working With NetBeans IDE document.

Both BeanFactory and ApplicationContext container support life cycle methods. We can't stop BeanFactory container explicitly we need to call factory.destroySingletons() at the end of above client application. When the container is BeanFactory.

Limitations with above custom/user defined life cycle methods:

1. Programmer must configure life cycle methods explicitly otherwise container will not recognize.
2. While working with predefined classes as Spring Beans identifying life cycle methods is difficult. It is recommended to implement two special interfaces called

i. Initializing bean(org.sf.beans.factory package)
ii. Disposable bean(org.sf.beans.factory.package)

Procedure to develop the above life cycle methods based Spring application by suing NetBeans IDE:

For complete program please refer Spring Core Module\NetBeans 7.1(Spring)\LifeCycleMethods folder.

A Spring bean can be developed as POJO or non POJO class. When our Spring bean class implements Spring API interfaces then it is called non POJO class.

Initializing Bean(I): It gives afterPropertiesSet() (here afterPropertiesSet is same as custom init life cycle method)

Disposable Bean(I): It gives destroy() this method is same as custom destroy life cycle method.

Instead of custom init, custom destroy methods in our Spring application it is recommended to work with these two interfaces(Initialization Bean and Disposable Bean) most of the Spring supplied predefined classes implements initializing bean interface, Disposable bean interface to check whether proper values are set or not to bean properties and to nullify bean properties respectively.

Example on initializing Disposable Bean implementation:

For complete program please refer Spring Core Module\NetBeans 7.1(Spring)\InitializingDisposableBeans folder

If Spring Bean class contains both afterPropertiesSet(), custom init method. The Spring container executes both methods(order is afterPropertiesSet(), customInit method)

If Spring bean class contains both destroy(), custom destroy methods then the Spring container executes both methods (order is destroy(), custom destroy())

Conclusion: When working with jdk, third party supplied classes look for custom life cycle methods configuration with user defined bean classes make them implementing Initializing, Disposable bean interfaces. While working with Spring supplied classes which are not implementing initializing bean, disposable, bean interfaces then look for custom life cycle methods configuration.

AwareXxx methods: When Spring bean class implements special Xxx aware interfaces then the Spring container can inject special values to bean properties through interface injection process.

The Xxx Aware interfaces are:

1. org.springframework.beans.factory.BeanNameAware injects underlying bean id of current bean id.
2. org.springframework.beans.factory.BeanFactoryAware injects underlying Bean Factory container to Spring Bean class property.
3. org.springframework.context.ApplicationContextAware injects underlying Application Context container to Spring Bean class property.
4. When Spring container injects SpringBean class then the bean class can use that container to get other bean class objects and call their methods.
5. Application Context container internally contain bean factory container.
6. If the underlying container is ApplicationContext container then the interface injection process can inject both BeanFactory, Application Context container because every Application Context container internally contains Bean Factory container.

Example Application on interface injection:

For Complete program please refer Spring Core Module\NetBeans 7.1(Spring)\InterfaceInjection folder

In the above application Container calls setBeanName, setBeanFactory, setApplicationCopntext method dynamically by seeing the implementation of XxxAware interfaces(nothing but interface injection).

BeanPostProcessor: Custom life cycle methods, Dependency injection, initiating bean, disposable bean interfaces, and implementation are specific to single Spring bean. In order to execute same logic for multiple beans configured in the Sprign cfg file we can think about working with BeanPostProcessor. For this we need to separate Java class implementing org.sppringframework.beans.factory.config.BeanPostProcessor interface. PostProcessor allows the programmer to keep and execute the logics as common logics outside from the Spring Bean classes.

Example Application on BeanPostProcessor: the BeanPostProcessor interface contains two methods.

1. **PostProcessorBeforeInitialization(-,-):** This method executes before initializing bean's , afterPropertiesSet() and custom init method.
2. **PostProcessAfterInitialization(-,-):** This method executes after initializing beans afterPropertiesSet(-,-) and custom init method.

For complete program please refer Spring Core Module\NetBeans 7.1(Spring)\PostProcessor folder

ApplicationContext container recognizes BeanPostProcessor automatically by using its configuring in Spring Configuration file refer **Demo.cfg.xml** where as shown below.

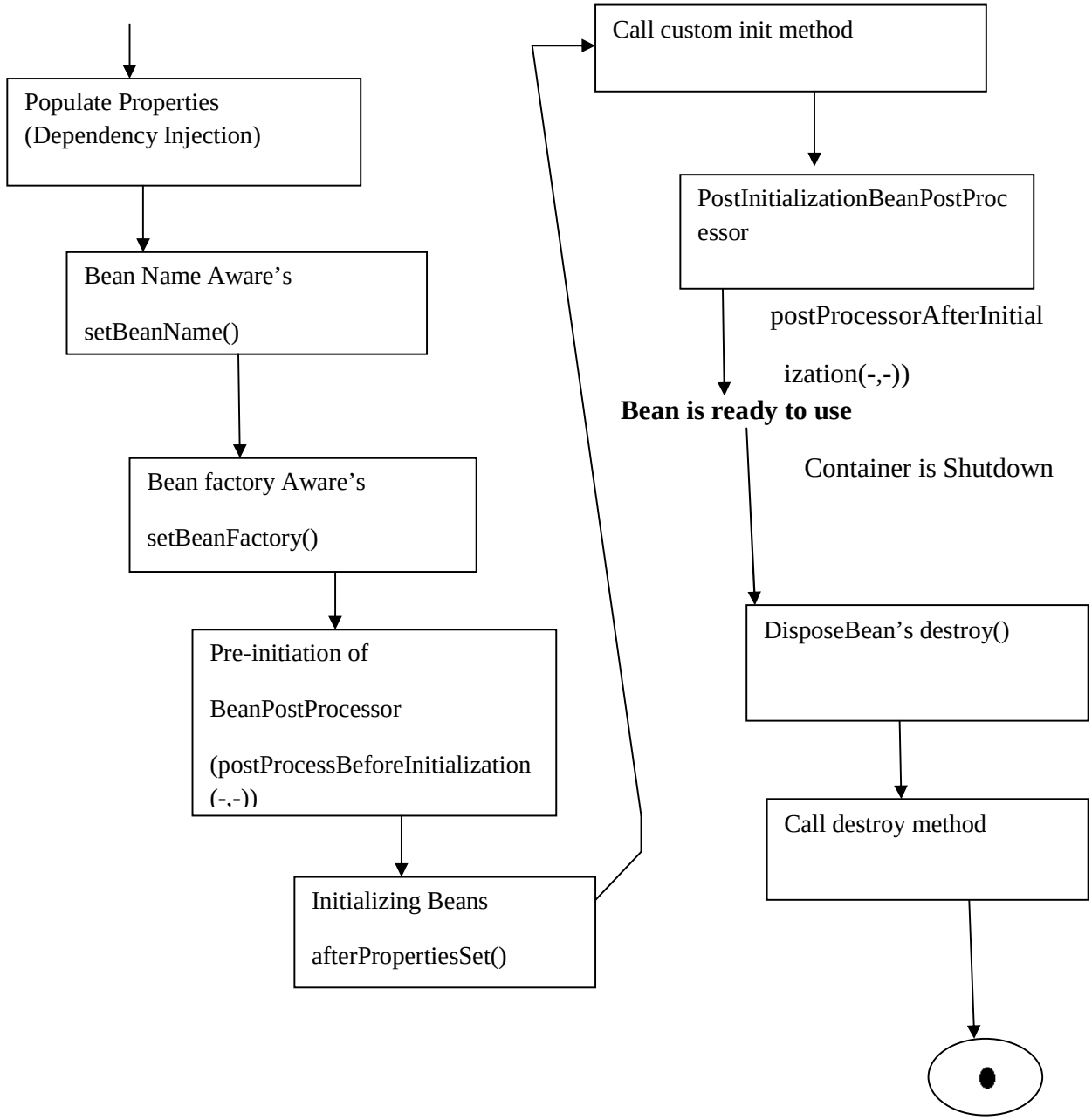
MyProcessor mp=new MyProcessor()

Factory.addBeanPostProcessor(mp);

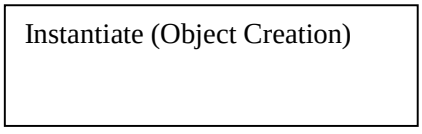
Note: No need of configuring MyProcessor class in psing cfg file.

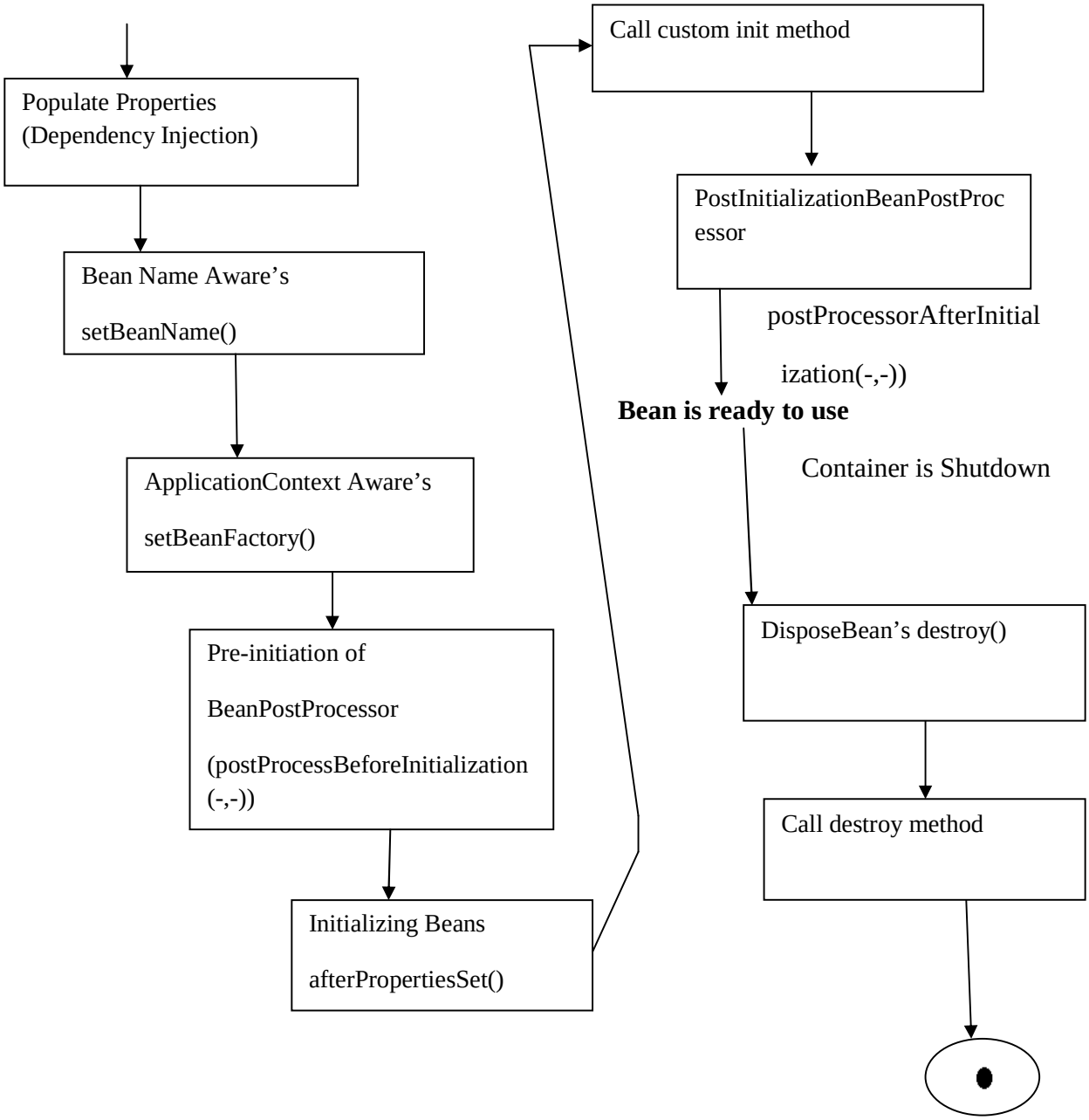
Spring Bean life cycle diagram with respect to BeanFactory Container:

Instantiate (Object Creation)



Spring Bean life cycle diagram with respect to AppliaitonContext Container:





Factory Class: Factory class is a class that can create and return objects there are two types of Spring Beans

1. Normal Spring Bean
2. Factory Spring Bean

When normal Bean is configured as dependent Bean then normal bean object itself will be injected to bean property when Factory Bean is configured as Dependent Bean then Factory bean object won't be injected dependent bean object. The resultant object given by Factory bean will be injected as dependent object to bean property. Normal beans are like selfish beans. Factory beans are selfless beans if Java class implements org.springframework.beans.factoryFactoryBean interface then it is called as Factory Bean class. Methods of Factory bean interface are

1. **getObject:** Returns the resultant object.
2. **getOjectType():** Returns the class name of resultant Object.
3. **isSingleton():** Returns true to specify that resultant object is Singleton object

<beans>

<bean id="tb" class="TestBean"/>

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

<property name="t1" ref="tb"/>

</bean>

</bean>

If the TestBean is normal Bean the 't1' property of DemoBean class will be injected with TestBean class object. If TestBean is Factory Bean then "t1" property of DemoBean class will be injected with the **resultant object** given by TestBean class. The resultant object of getObject method.

Example Application:

For Complete program please refer Spring Core Module\NetBeans 7.1(Spring)\FactoryBeanProgram folder.

Note: We can create more than one FactoryBean according to our requirement.

For complete program on this process please refer

Advantages of FactoryBean:

1. Container need not to create object separately. So automatically FactoryBean reduces the burden on Container.
2. Application execution is little bit faster.

3. Application memory size will reduce.

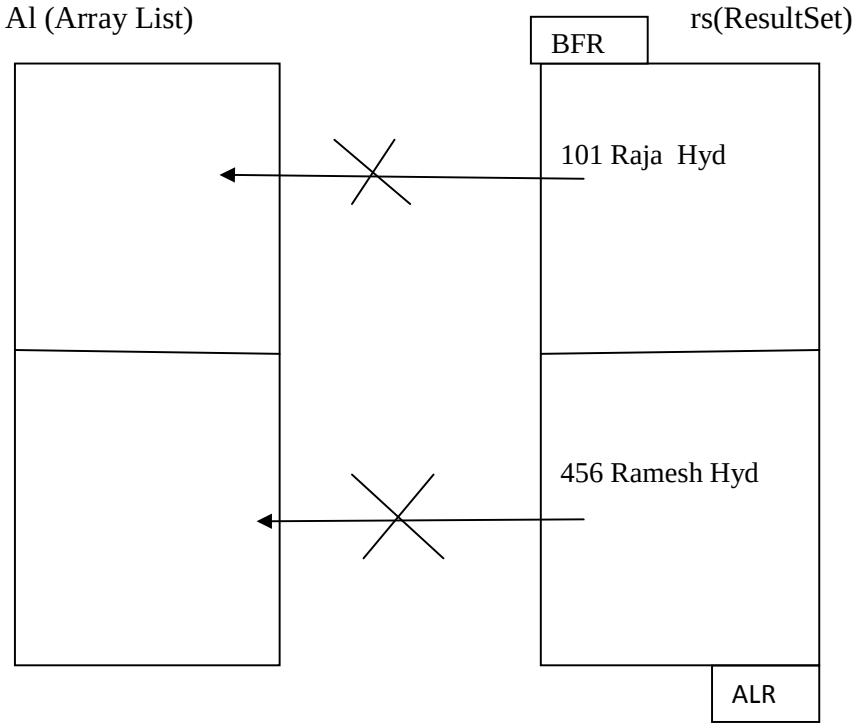
Where to keep Spring Container activation logic in various technologies:

While developing application/component as client to Spring Bean it is recommended to keep the logic of activating Spring container and getting Spring Bean class object from container in onetime execution block and use that Spring Bean object to call business methods in repeatedly executing blocks for better performance.

Application/component acting as Spring Client	**Place to activate Spring Container and to get Spring Bean object**	**Place to call business methods on Spring Bean object**
1. AWT/SWING Framework.	Constructor(object level)/static block(class level)	Event Handling
2. Applet/JAPpplet	Init()/constructor/static block	Event Handling
3. Servlet Component/program	Init()/constructor/static block	goGet(-,-)/doPost(-,-) service
4. Jsp program/copmponent	<%!public void jspInit() {----------}%>	<%------%>
5. Struts Application	static block /constructor of Action class	In the execute method of Action class
6. JSF Application	Static block/constructor of managed bean class	In the business methods of managed bean

Mini Project: We can't send ResultSet object over the network because it is not a serializable Object. So copy the records of ResultSet to Collection Framework Datasource like Arraylist and send that data source over the network. All Collection Framework data structures are serializable objects by default.

Each record of ResultSet contains multiple values including multiple objects where as each element of ArrayList allows only the object so we can't copy records of ResultSet object to the elements of ArrayList directly.



To solve the above problem copy the records of ResultSet object ot multiple objects of user defined class and add those objects to the elements of ArrayList this user defined class is called as DTO class or VO class.

EX:

Student.java/D.T.O class/V.O class

Public class Student

{

private int no;

private String name;

private String addr;

//write setXxx(-) and getXxx(-)

}

Logic to copy the records of ResultSet object to ArrayList:

ResultSet rs=st.executeQuery("select * from student");

ArrayList al=new ArrayList();

{

While(rs.next)

{

Student st1=new Student();

st1.setNo(rs.getInt(1));

st1.setName(rs.getString(2));

st1.setAddr(rs.getString(3));

}

Note: Objects added to the data structure must be serializable to send over the network. That means the Java Bean Class student must implement Serializable properties.

Note: Objects added to the data structure must be serializable objects to send Datastructure over the network.

Note: Irrespective of network availability it is not recommended to send ResultSet object from one to another layer because to receive Resultset object we need JDBC code in destination layer and keep in JDBC code in every layer is not a recommended process.

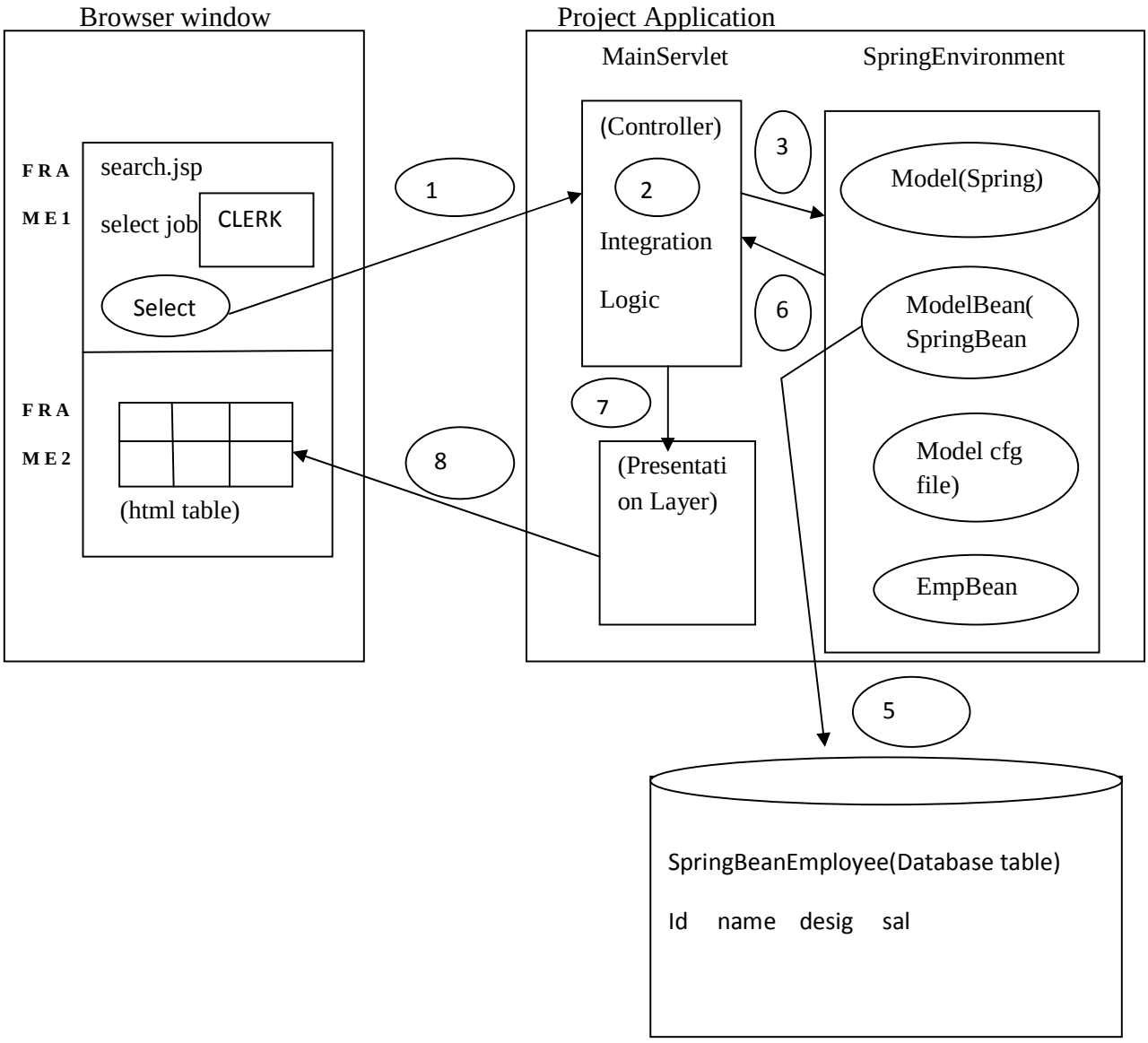
Q) Where did you use Java Bean in your Project?

1. As VO/DTO class while copying ResultSet records to ArrayList datastructure.
2. As model layer Business component in MVC1, MVC2 project.
3. As helper class in SessionTracking to store entire form data as single Bean class Object.
4. As persistence class/entity class in Hibernate.
5. As Spring Bean in Hibernate.
6. As FormBean in Struts 1.x

Q) Where did you use Collection Framework in your project?

To send ResultSet object data over the network we copy the Array List.

1. To send JDBC details from properties file we use java.util.Properties class.
2. We used them in SessionTracking.
3. We use them network algorithms implementation.
4. To maintain basic amount of data temporary without Database and files.
5. To remember application/game status when it is paused state.
6. To maintain JNDI properties we use Map data structure.
7. To maintain mail properties we use java .util.Properties class.



For complete project please refer Spring Core Module\NetBeans 7.1(Spring)\SpringMiniProject folder

Our mini project is based on MVC architecture:

M→ Model → (business logic and persistence logic)

(Spring core module based Spring Bean + Java Bean)

V→ View Layer (Presentation logic)

C→ Controller Layer(Integration logic/connectivity logic) (servlet program)

Q) What is container class back method or Container Life cycle method?

A) The method that is called by underlying container automatically based on the Event that is raised is called as container callback method or container Life cycle method.

CoreJava | JSP | Servlets | JDBC | Struts | Spring | Hibernate

Projects | FAQs | Sample Programs | eBooks | Certification Stuff

Question Banks | Communities | Tutorials | Softwares | Sample Resumes

Interview Tips | Forums | Discussions | Online Test Engines | Jobs

www.JavaEra.com

A Perfect Place For All Java Resources