Spring Core Module
==========
=>It is base module for other modules
=> if we use this module alone in App development .. we can develop only standalone Apps..
=> This module gives two spring containers/IOC containers called BeanFactory and Applicaiton Context to perform
(a) Spring bean life cycle management
(To manage spring bean from birth (obj creation) to death (obj destruction)
(b) Dependency Management
|---> i) Dependency lookup
|---> ii) Dependency injection...

we can develop spring Apps in 4 approaches
==========
(a) Using xml driven configurations (only in maitennce Projects)
(b) Using annotation driven configurations
(c) Using java code /100%Code driven configurations
(d) Using spring boot driven configurations

Different modes Dependency Injection that spring supports
==========
(a) setter Injection
(b) Constructor Injection
(c) Aware Injection/InterfaceInjection/Contextual Dependency Lookup/Injection
(d) Lookup Method Injecction
(e) MethodInjection/Method Replacer..

Setter Injection
==========
=>HEre IOC container calls setter method of Target class to assign Dependent class object to taget class object..

=> new operator creates the object at runtime .. but expects the presence of java class from compile time onwords.. i.e we can not use new oeprator to create objects for java classe whose class names comes to the app dynamically at run time..
Test t=new Test();

=> In Spring Apps we get SpringBean class names to SpringContainers dynamically at runtime from sprign bean cfg file (xml file) .. So new operator can not be used to create objects sprign bean classes.. So we need special Spring container that internally uses other concepts to create spring bean class object...

Sample code for Setter Injectiion (POC)
==========

SpringContainers simplifies /automates the Spring Beans Life cycle management and Dependency management . based on the inputs instructions collected from spring bean cfg file

WishMessageGenerator (target class) java.util.Date(Dependent class)

|----> It has to get System Date by using Date class object , so it can use current hour of the day to generate wish messages like "Good morning", "Good afternoon " and etc.. So "Date" is called as Dependent class and "WishMessageGenerator" is called as target class..

WishMessageGenerator.java (Spring bean --Target class)

```
package com.nt.beans;
  import java.util.Date;
public class WishMessageGenerator {
     private Date  date;    //spring bean property

      // setter  method to support setter injection process
      public void setDate(Date date){
         this.date=date;
      }

    //B.method using the injected Date class obj in the b.logic

    public  String  generateWishMessage(String user){
      int hour=0;
      //get current hour of the day
      hour=date.getHour();  // gives in 24 hour format

      //b.logic to geneate wish message
       if(hour<12)
          return "Good Morning::"+user;
       else if(hour<16)
          return "Good Afternoon::"+user;
       else if(hour<20)
           return "Good Evening::"+user;
       else
          return "Good Nigght::"+user;

     }
}
```

IOC :: Inversiion of Controller

applicationContext.xml (com.nt.cfgs)
========== dtd
import xsd rules

<bean ...>

<bean >---> for Spring bean class cfg
<property>--->To cfg bean property for setter injection

```
<!-- cfg Dependent class -->
<bean id="dt"  class="java.util.Date"/>
<!--cfg  Target class -->
<bean id="wmg"  class="com.nt.beans.WishMessageGenerator">
    <property  name="date"  ref="dt"/>
</bean>
</beans>
```

beand id — sprign bean class

Dependent spring bean class bean id

<property> tag instructs to perform setter Injection

bean propertyname
or
xxx word in setDate(-) method

=>Servlet comp is idetified with its urlpattern (at servlet container)
=>java obj is identified with its hashcode (at jvm)
=>Spring Bean is identified with its bean id (at spring container)

SetterInjectionTest.java (Client App)
========== (com.nt.test)

```
// import pkgs..
public calss SetterInjectionTest{
   p  s  v  main(String args[]){
    Resource res=null;          WishMessageGernator target=null;
    BeanFactory factory=null;
     //Hold the name and location spring bean cfg file in Resource object
    res=new FileSystemResource("<location>/applicationContext.xml");
    //  create IOC container/Spring container (BeanFactory)
     factory=new XmlBeanFactory(res);

    //get Target Bean class obj from BeanFActory cotainer
     Object obj =  factory.getBean("wmg");

     S.o.p(obj.generateWishMessage("raja"));
     target=(WishMesageGenerator)obj;

     String result=target.generateWishMessage("raja");
     S.o.p("Result is ::"+result);
   }
}
```

=>public Object getBean(String beanId)
if the java method return type java.lang.Object then that method can any java class object becoz java.lang.Object is the top most in the inheritence hierarchy any java class..

=> Using java.lang.Object class ref variable we can refer any java class object..

```
class Test{
  public void x(){
    ..
  }
```

```
class Demo extends Test{
  public void x(){
    ...
  }
  public void y(){
    ,,
  }
```

```
Test t=new Demo();
 t.x() ;  (valid)
 t.y();  (invalid)
 Demo d=(Demo)t;
  d.x() ; (valid)
  d.y() ; (valid)
```

clipse

d.x() ; (valid)
d.y() ; (valid)

Eclipse
=======
type :: IDE for java
version :: 2020-06 (compitable with jdk 1.8+) (2020-03 --->recomanded)
vendor :: Eclipse org
Open Source or setup file
To download s/w :: download as zip from eclipse.org
To install s/w :: extract the zip file.. or install setup file
Flavours :: Eclipse sdk (only for standalone Apps)
Eclipse JEE (for all types of Apps developmenmt)
https://www.eclipse.org/downloads/packages/release/neon/3/eclipse-ide-java-ee-developers

Procedure to devleope the above setter Injection example using Eclipse IDE?
==========

step1) Keep the following software setup ready
=>Spring latest version (zip extraction) -> 5.2.7
=> jdk 1.8+ version -> jdk 9
=> Eclipse 2019/20 versions -> 2019-12

step2) Launch eclipse IDE by choosing workspace folder.
The folder where Projects will be saved
(G:\Workspaces\Spring\NTSP713)

step3) create Java Project in Eclipse IDE
file menu---> project ---> java project ----> name ::IOCProj1-SEtterInjection-POC ---> .....

step4) Add Spring core module libraries to the Project..

spring-beans-<ver>.jar
spring-context-<ver>.jar
spring-context-support-<ver>.jar
spring-core-<ver>.jar
spring-expression-<ver>.jar
(collect from <spring_home>\libs folder)

commons-logging-<ver>.jar | collect from internet..

Right click on the Project ----> build path ----> configure build path ----> libraries tab ----> add external jars ----> .....

step5) make sure that following packages are cared to reosurces for application development

IOCProj1-SetterInjection-POC
|---->src
|--->com.nt.beans
|---->WishMessageGenerator.java (target class)
|--->com.nt.cfgs
|----->applicationContext.xml (spring cfg file)
|---->com.nt.test
|--->SelectterInjection.java (Client app)
|---->referenced libraries
|-----> 5+1 jar files..

Alt+shift+s,r ---> To get setters & getters or setters or getters

step6) run client app ....
using runAs -->java application or usisng run Button (or) ctrl +f11

DTD,XSD are Build blocks for constructing xml doc... Each DTD /XSD docucment /file contains bunch of rules having xml tag names, attributes names and their structure details to be used in the construction of xml files...

=> we need to import DTD /XSD rules at the top Xml files in order inform to frameworks /containers/server that we have developed xml file according to certain DTD/XSD rules..

DTD :: Document Type Definitation (old)

XSD :: Xml schema Definitation (latest)

In XSD /Xml schema

|------>Xml schema NameSpaces

(Schemal NameSpace is a library that contains set of xml tags and their rules)

(Every Schema Name Space idenfied with its namespace uri/url)

(Every Schema namepsace or bunch of namespaces togather will come as one xsd file)

The spring framework supplied multiple prefined Xml schema namespaces

They must be imported in spring bean cfg file by specifying their urls/uris

Examples

========

namespace	namespace uri/url	xsd file
beans	http://www.springframework.org/schema/beans <Website url>/schema/<namespace>	spring-beans.xsd
c	http://www.springframework.org/schema/c	spring-beans.xsd
p	http://www.springframework.org/schema/p	spring-beans.xsd
context	http://www.springframework.org/schema/context	spring-context.xsd

10 + namespaces are given in spring frameworks..

u can sample schema import statements of spring from <spring_home>\docs\springref...

All these xsd files are available spring libraries (jar files of <spring_home>\libs folder)

ts-corona.xml

```
<dhavakhanas>
   <dhavakhana>
      <nam>OSG</nam>
      <sankya>1000</sankya>
   </dhavakhana>
    <dhavakhana>
        <nam>GH</nam>
        <sankya>1000</sankya>
    </dhavakhana>
    ,,,,,
</dhavakhanas>
```

ap-corona.xml

```
<hospitals>
   <hospital>
      <name>RIMS</name>
      <size>500</size>
   </hospital>
   <hospital>
      <name>SIMS</name>
      <size>200</size>
   </hospital>
    .....
</hospitals>
```

(bad)

natarazjavaarena --->FB group

Solution

=======

Central gives corona.xsd

|----> namespace --->corona

|--->uri: http://www.nit.com/schema/corona

|--><hospitals>,

<hospital>, <name>,<size>

sysout ---> gives S.o.println(-)

systrace --->gives S.o.println(-) with message

=> Importing DTD/XSD rules in web.xml is optional... but mandatory in spring bean cfg file...

=>The Bean id given to spring bean becomes object name (nothing ref variable name pointing to spring bean class object) when Spring container instantiating given spring bean class object.

Limitation of "new" operator

```
Test t=new Test();
```

=> new operator creates the object at runtime .. but expects the presence of the java class from compile time onwords ... i.e it can not be used to create the object of java class whose class name comes to App dynamically at run time..

=>To overcome this problem

-->use newInstance() method of java.lang.Class (or) (can use only 0-param constructor)

-> use newInstance(...) methord of java.lang.reflect.Constructor (can use any param constructor)

Each object of java.lang.Class represents one class /interface/enum/annotation in the App execution..

String class object can hold string values ... numeric data type variable can hold numberic vlaues.. similarly the object of java.lang.Class can hold class name/ interface name/enum name/annotation name comming to App dynmaically at runtime..

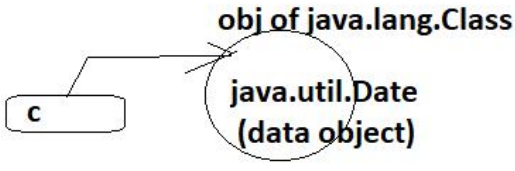
=>each object of java.lang.reflect.Constructor class hold the details of one constructor of a given java class..

working with newInstnace() of java.lang.Class

```
eg1::
//Load the java class dynamically run time
    Class c=Class.forName("Test");
              static          args[0]
              method of java.lang.Class

 // instantiate the class
Object obj=c.newInstance();
              creates "Test" class obj
              usisng 0-param constructor
  //typer casting
Test t=(Test) obj
```

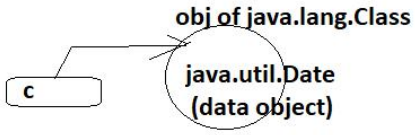
Class.forName(-) method makes the jvm to load the given java class dynamically at run time and returns the object of java.lang.Class having the loaded name as the data of the object...

```
eg2:
====
   //load the class
   Class c=Class.forName("java.util.Date");
                         args[0]

    // instantiate the class (create the object of loaded class)
     Object obj=c.newInstance();
      S.o.p(obj.toString()) (or) S.o.p(obj)
                system date and time will be printed..
```



eg2:
====

```
//load the class
Class c=Class.forName("java.util.Date");
                              args[0]

// instantiate the class (create the object of loaded class)
Object obj=c.newInstance();
S.o.p(obj.toString()) (or) S.o.p(obj)
        system date and time will be printed..
```



Spring Container/IOC container creates spring bean calss objects by using newInstance() of java.lang.Class or java.lang.reflect.Constructor dynamically at runtime becoz it is getting spring class names dynamically at rumtime from spring bean cfg file..
i.e spring container is not using "new" operator to create any spring bean class object..

alt+shift+s,s ---> to get toString()
main + ctrl+space ---> to main method
alt+shift+s,o --> to get constuctor

To pass Cmd line args in Eclipse App execution...
Go to App class (where main(-) is avaiable) --->right click ---> run as ---> run configurations -->choose project ---> choose main class name ---> go arguments tab --->

```
com.nt.Test   jav.util.Date
agrs[0]         args[1]
```

Working with newInstance() of java.lang.reflect.Constructor class

=>Very useful if dynamic object creation should happen by parameterized constructor

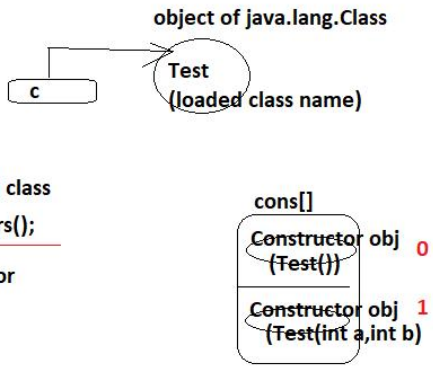
Test.java
=======

```
package com.nt.comp;

public calss Test{
  int a,b;
  public Test(int a,int b){
      this.a=a;
      this.b=b;
  }
  //toString
  ....
}
```

```
public Test(){
  }
```

Main class
=========

```
package com.nt.test;
public class NewIsntanceTest1{
   p s v main(String args[]){
      Class c1=null;
     try{
       //Load the class
        c1=Class.forName(args[0]);
                          Test
       //get access to all the constructor of the loaded class
         Constructor cons[]=c1.getDeclaredConstructors();
       // instantiate the class using 2-param constructor
         Object obj1=cons[1].newInstance(10,20);
          S.o.p(obj1);
       }
     catch(Exception e){
       e.prinstackTrace();
     }
  }
}
```



ctrl+shift+o :: to import the pkgs..

Spring First App Flow of execution

==========

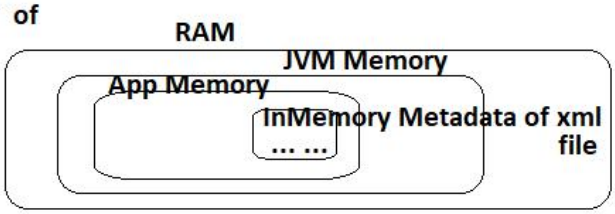
=>if the xml document/file is satisfying xml syntax rules then it is called well-formed xml document..
=>if the xml document /file is satisying the import DTD /XSD rules then it is called Valid Xml document..

=> XML parser is as software prg that load given xml file , can check wheather xml file is well-formed or not , valid or not . if well-formed and valid it can read and process the given xml file..

eg: Sax parser (Simple api xml parser)
eg: Dom parser (Document object model parser)
eg: Dom4J parser (DOM for java parser)
and etc...

Data about data is called MetaData....

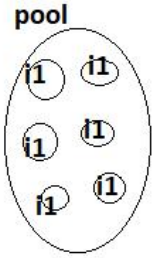
=> This Xml parser also creates "In Memory MEtaData " of the loaded xml file in the Memory where the current app ahving xml parser is running that is nothing but JVM Memory of the.. RAM .. So that App can use xml file info for multiple times from InMemory MetaData of JVM Memory it self..



=> Pool Vs Cache

note: Spring/IOC container are having built-in xml parsers...

pool

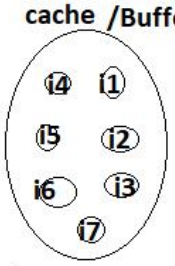


(set of same items)

(It is gives reusablitiy of same items)

(prefer List collection)

cache /Buffer



(Set of differnt items)
(Gives the reusability of different items)

(prefer Map Collection to costruct cache)

Flow of exection (since our First App is standlone App the flow start with main(-) method and end with main(-) method)

==========

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

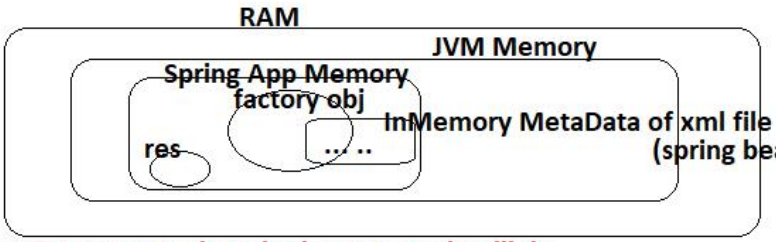
=>Internally uses java.io.File class to hold the name of location of spring bean cfg file.. At this line it will not try to locate the given spring bean cfg file... (it just hold filepath .it will not check the filepath location is correct or right)

=> There are Two Popular implementation classes for Resoruce(I)

a) FileSystemResource (Makes Spring container to locate the given spring bean file from the specified path of the File system)

b) ClassPathResource (Makes Spring container to locate the given spring bean file from the directories and jars added to CLASSPATH/BUILDPATH)

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



=> the above line creates BeanFactory container in that process it will do

=> the above line creates BeanFactory container in that process it will do

--> takes the given Resource obj(res) and locates spring bean cfg file (applicationContext.xml) from the specified path of File System (here filepath ,syntax,imported DTD/XSD checking will happen)

--> Loads the spring bean file from File system and checks wheather it well formed or not, valid or not , if not App throws exception.. if it well formed and valid then It creates InMemory MEtaData of Spring bean Cfg file in the Memory (JVM Memory of the app) where our Appp is running

-> create XmlBeanFactory class object representing BeanFactory container and returns that object back To App..

```
Object obj=factory.getBean("wmg");
```

-->getBean(-) takes the given "wmg" bean id and goes to InMemory MetaData of spring bean cfg file and checks the availability spring bean cfg having bean id "wmg" and gets "com.nt.beans.WishMessageGenerator" as spring bean class and also observers setter Injection eanbled on the spring bean class (based <property> tag.. so it loads and instantiates spring bean class (taget class) as shown below.

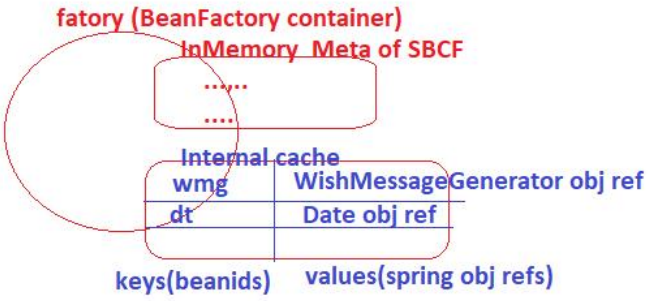
```
Class c1=Class.forName("com.nt.beans.WishMessageGenerator");
WishMessageGenerator wmg=(WishMessageGenerator)c1.newInstance();
```

-> collects name="date" , ref="dt" from <property> , checks the availability of setDate(-) method in target class using reflection api .. and also checkes spring bean class cfg having bean id "dt" and gets java.util.Date as spring bean class from the In Memory MetaData of springbean cfg file. since no injection is cfg on java.util.Date class (dependent) the Spring container loads and creats the object as shown below..

```
Class c2=Class.forName("java.util.Date");
Date dt=(Date)c2.newInstance();
```

-> Becoz of <property> the container performs setter injection by calling setter method on taget class obj (wmg) having Dependent class obj (dt) as the argument..

wmg.setDate(dt) (wmg) taget obj date:: depenetn obj (dt)



-> keeps all the created Spring bean class objs in the Internal cache of IOC container having bean ids as keys and Bean class obj refs as values... for the reusability the objects...

-> getBean(-) returns WishMessageGeneator class obj(target obj) back to client App... and we are reffering that obj with Object class ref variable (obj)

```
//typecasting
WishMessageGenerator generator=(WishMessageGenerator)obj;
```

=> Type casting/down casting to call direct methods (b.methods) of WishMessageGenerator class..

```
String result=generator.generateWishMessage("raja");
System.out.println("Message::"+result);
```

=>Invoking b.method on target class object.. and this b.method will use the injected Dependent class obj (Date obj) to generate the wish message based on the current hour of the day..

End of the main(-) method

=>Since all objects are local objects main(-) method , they will be vanished.. at the end of main(-) method , In that process IOC container(factory) and its internal cache and InMemory MetaData will also be vanished..

Constuctor Injection

==========

=>Here IOC container uses parameterized construtor to create target bean class object .. In that process it assigns/ injects dependdent class object to target class object..

=>In setter Injection , first target class object will be created ,next dependent class object will be created..

=>In constructor Injection , first dependent class object will be created ,next target class object will be created dependent class obj as the parameterized constructor arg value

=> use <property>tag for setter injection

=>use <constructor-arg> for contructor injection. if u place <constructor-arg> tags for "n" times under <bean> tag then the IOC container uses n-param/n-arg constructor to create spring bean class object..

Example

========

WishMessageGenerator.java (target class)

==========

```
package com.nt.beans;

public class WishMessageGenerator{
  private   Date date;

   // for constructor injection
   public  WishMessageGenerator(Date  date){
       this.date=date;
    }

    public  String generateWishMessage(String user){
        ...
        ... //same as previous
        ...
    }
```

java.util.Date (Dependent class)

applicationContext.xml

==========

```
<beans ..... >

  <!--Dependent class cfg -->
   <bean id="dt"  class="java.util.Date"/>

  <!-- target bean class cfg  -->
   <bean id="wmg"  class="com.nt.beans.WishMessageGenerator">
       <constructor-arg name="date"  ref="dt"/>
  </bean>
```

ConstructorInjetionTest.java

==========

same as previous..

note:: while cfg spring bean..if we enable only setter injection.. then IOC container uses 0-param constructor to create spring bean class object.. if enable construictor injection then it uses parameterized constructor for spring bean class object creation..

alt+shift +s , o --> To generate parameterized constructor

sample xsd import statements to import in spring bean cfg file can be collected from

G:\Spring5.2.7Soft\spring-framework-5.2.7.RELEASE\docs\spring-framework-reference foder reference docs.

what happens wrong bean ids in spring programming?

The IOC container throws

org.springframework.beans.factory.NoSuchBeanDefinitionException: No bean named 'xxx' available

beanid

In constructor Injection enabled Appliation the

generator=(WishMessageGenerator)factory.getBean("wmg"); **performs**

-->Takes the bean id "wmg" -->searches in Interanal Cache for bean class object (not available) ---> goes InMemory MetaData of spring bean cfg file and searches for bean calss cfg whose bean id is "wmg" and gets "com.nt.beans.WishMessageGenerator" as spring bean class but notices the constructor injection is enable by seeing <constructor-arg> tag under <bean> tag.

-> Goes to name="date" , ref="dt" of <constructor-arg> tag , then search in Internal cache for Bean class obj using bean id "dt" (not available) -->goes to InMemory MetaData of sprign bean cfg file gets "java.util.Date" class having bean id .. Since no injection is enabled .. IOc container loads and creates the object for java.util.Date having "dt" as object name (ref variable name)

```
Class c1=Class.forName("java.util.Date");
Date  dt=(Date)c1.newInstance();
```

-> IOC container get access 1 param constructor of taget class and uses that constrictor to create target class object(WishMessageGenerator) having Dependent class object (Date) as the argument value of cosntructor.

```
Class c2=Class.forName("com.nt.beans.WishMessageGenerator");
Constructor cons[]=c2.getDeclaredContructors();
WishMessager wmg=(WishMessageGenerator) cons[0].newInstance(dt);
```

cons[]
Constructor obj
0
(1-param constructor)

-> IOC container keeps both Target and dependent class objs in the Internal Cache

Internal Cache

(keys)	(values)
dt	Date class obj ref
wmg	WishMessageGenerator obj ref

-> retruns WishMessageGenerator class obj ref (target) back to Client App..

=> <ref>, "ref" attribute to cfg bean ids based spring bean class objs injection to target bean class properties
=> <value> ,"value" attribute to cfg "simple values" injection to taget bean class propertie

```
java.util.Date
      |--->year (int)
      |-->month (int)
      |--->date (int)
      |--->hours (int)
      |-->minutes (int)
      |--->seconds (int)
```

What is happens if we enable both setter Injection and cosntructor Injection on the bean property? tell me whose values will be injected as final values?

Ans) Since setter method always execute aftr constructor execution , we can say stter injection overrides the values injected by the constructor injection i.e the values injected by setter injection will become final values.

```
<!-- Dependent class cfg -->
    <bean id="dt"  class="java.util.Date"/>

    <bean id="dt1"  class="java.util.Date">
      <property name="year" value="90"/>
      <property name="month" value="09"/>
      <property name="date" value="4"/>
    </bean>


    <!-- Taget class cfg -->
    <bean id="wmg"  class="com.nt.beans.WishMessageGenerator">
          <property name="date"  ref="dt1"/>
       <constructor-arg name="date"  ref="dt"/>

    </bean>
```

note:: Bean id should be unique with in a IOC container....
note:: we can cfg two spring beans having same class names but we should take different beans ids...

Spring AOP ---> tommorow onwords
6pm ---> (only if we know
spring core already)

ctrl+shift +c --> for single line commenting enabling /disabling
select code ,ctrl+shift+/ --> for enabling multiline commenting
select code ,ctrl+shift+\ --> for disabling multiline commenting

```
<property name="month" value="09"/>
  is equal to
<property name="month">
  <value>9 </value>
</property>

<property name="date" ref="dt"/>
    is equal to
<property name="date">
    <ref bean="dt"/>
</property>
```

In the <property> tag, "name" value is not bean property name , it is xxx/Xxx word of setXxx(-) method.

In the <constructor-arg> tag, "name" value is not bean propety name , it is parameter name of parameterized constructor.

Eclipse Plugin --->STS (Spring Tool Suite)

=> Plugin is patch s/w , that provides addtional functionalities to existing s/w ...
=> STS plugin added to eclipse makes spring app development very easy in eclipse IDE...

To install third party supplied pluings use "Eclipse Market Place " of help menu
To install eclipse supplied pluings use "Install new Software " of help menu.

To install STS plugin
==========
Help menu --->Eclipse market place -----> search for sts (go) ----> select sts 3.9 ----> install --->

select all check boxes ---> accept terms and conditions ----> restart IDE...

What is the difference b/w FileSystemResource and ClassPathResource?

note: Both are implementation classes of org.springframework.core.io.Resource;

=>FileSystemResource class obj given to BeanFactory Container makes the BeanFactory container to locate the given spring bean cfg file from Specified of path File System .. we can give either relative path (recomanded) or absolute path of spring bean cfg file

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

=> ClassPathResource class obj given to BeanFactory container makes the BeanFactory container to search and locate spring bean cfg file from the directories and jar files that are added to CLASSPATH /BuildPath..
note:: In Every Eclipse Project "src" folder is in Classpath/Build path by default...

```
Resource res=new ClassPathResource("com/nt/cfgs/applicationContext.xml");
```
(Recomanded)

=>we can added any location of the Project to Claspath/Build path .. if we add "com/nt/cfgs" folder of project to Build path then we can write like this,,

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

To add "com/nt/cfgs" folder to Buid path ---> righ clikc on project --> build path --> configure build path ---> libraries tab ---> add Class Folder ---> select project and package/folder -> ...

---

java.lang.Class
==========

The object class of java.lang.Class can hold class name/interface name/annotation name/enum name in a running java App..

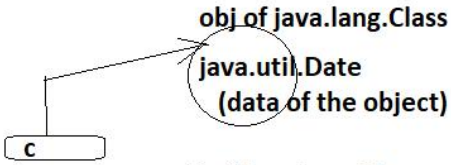
we can not create object for java.lang.Class using "new" operator becoz it having only one 0-param private constructor..

```
Class c=new Class();
```

Different ways of creating object for java.lang.Class
==========

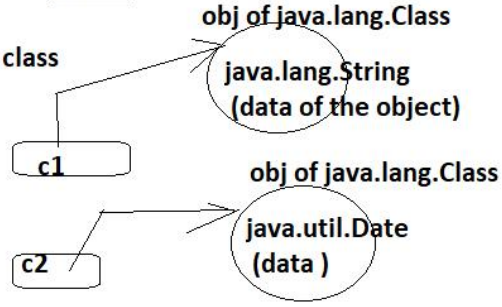
Approach1: Using Class.forName(-) method
==========

```
Class c=Class.forName("java.util.Date");
```



Approach2: Using getClass() method of java.lang.Object class
==========

```
String s=new String("ok");
Class c1=s.getClass();

Date d=new Date();
Class c2=d.getClass();
```



String s1="java.utl.Date";
using s1 we can not instantatiate "Date" class becoz "s1" holds "Date" class as as stirng value.

Using "c","c2" we can get any info about "Date" class like methods, construictors, fields and etc.. more over we can instantiate Date class also.

Approach3 :: using "class" property of java "implicit"
==========

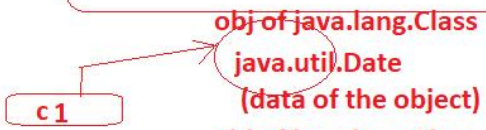
=>Two built-in properties of java language are "class" ,"length".

=> Two implicit ref variables in java are "this" ,"super".

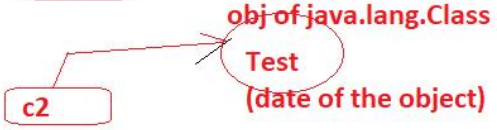
=> Two implicit threads in java are "main" ,"gc"

"class" -->built in property
"class" --> keyword
"java.lang.Class" ---> pre-defined class.
"class" -> oops terminology.

eg1: Class c1=java.util.Date.class;



eg2: Class c2= Test.class;



Compiler added /generated code in our java class
==========

-> java.lang.Object as super class if there is no super class for our class

-> public 0-param constructor , if there are no consturctor in our class

->public static property "class" of type java.lang.Class and etc...

imp

imp

Two overloaded Signatures of getBean() method in BeanFactory(I)

==========

(a) public Object getBean(String beanId)

=>Here type casting is required .. i.e code is not type safe... possibility of getting java.lang.ClassCastException.

eg:WishMessageGenerator wmg=(WishMessageGeneator)factory.getBean("wmg");

(b) public <T>T getBean(String beanId, Class<T> requiredType) (Recomanded)

=>Here typecasting not required.. i.e our code becomes type safe code..

=>Generic are given from java 1.5
=>They avoid typecasting in coding
=>They make our code as typesafe code

syntax < > (template/gerneic)

T-->Type , K-->key , V-->value
N-->Node , E-->Element

eg1:: Class c=WishMessageGenerator.class;
WishMessageGenerator generator=factory.getBean("wmg",c);

eg2: WishMessageGenerator generator=
factory.getBean("wmg",WishMessageGenerator.class);

working with

While (b) signature of getBean(-) if i give wrong class name as the 2nd argument then what happens?

org.springframework.beans.factory.BeanNotOfRequiredTypeException: Bean named 'xxx' is expected to be of type 'AAAA' but was actually of type 'BBBB '

Creating Libraries in Eclipse having jars linked api docs and source code

==========

https://www.youtube.com/watch?v=Y7E6hDic3xw&list=PLVlQHNRLflP-wlUj1MAuLwiMekHpP-yQu&index=12

DesignPatterns

==========

Def1 :: DesignPatterns of set of rules that are given as best solutions for reoccurring problems of application development.

def2:: Design pattern best pratices to use software languages, technologies and frameworks effectively in application development...

Two types patterns in JAva

==========

a) GoF Desing Pattern (Core Patterns) (Can be implemented any oop language)
-->Signlegon ,factory , abstract factory , strategy and etc.. 23 patterns

b) JEE Patterns (given by Sun Ms) (Can be implemented only in java)
DAO , Frontcontroller, View Helper ,Composite view and etc...

Factory Pattern
==========

=> It return one of the several related classes object based on data we supply.

problem:: different objects will be created using differen techniques.. making all developers knowing the creational process objects .. makes developers work bit heavy.

Solution:: Create a factory and provide abstraction on object creation process and return the object based on the data that is supplied... (nothing hide object creation process from clients)

eg:: `Connection con=DriverManager.getConnection(-,-,-);`

It is returning one of the several implementation class objects of java.sql.Connection(I) based on the data (url, user,pwd) details we passeed.. i.e returns jdbc con object by creating necessary dependent objs like socket objects and uses then in the creation of jdbc con object..

eg: `Object obj =factory.getBean("<beanId>");`

BeanFactory containener means it factory to create and manage spring bean class obejcts

It will create and return Spring bean class objects using different techniques and assigns dependent objects based on the bean id that we pass as input value.

Spring /IOC container is designed based on Flyweight Design Pattern which interally alos uses Factory Pattern

note:: To Keep multiple classes as related classes make them implementing common interface or extendign from Common super class..

Important points on methord return types

(a) if the java method return type is an interface , then it returns one of its implementation classes obj as the return value.

(b) if the java method return type is an abstract class , then it returns one of its sub classes obj as the return value.

(c) if java method return type is concrete class , then it can return either that concrete class object or one of its sub class object.

```
IOCProj4-FactoryPattern-CoreJava
   |------>src
         |--->com.nt.comp
               |---->Car.java (I)
               |---->SportsCar.java (IC)
               |--->LuxoryCar.java (IC)
               |--->BudgetCar.java (IC)
               |--->Tyre.java (I)
               |--->ApolloTyre.java (IC)
               |-->MRFTyre.java (IC)
               |--->CETATyre.java (IC)

         com.nt.factory
               |----->CarFactory.java  (Factory Pattern )
                                        having one factory method
                                              (The method that creates/gathers
                                              and returns object is called factory
                                              method)

         com.nt.test
               |---->FactoryPatternTest.java (Client)
```

Strategy DesignPattern
==========

|-->It is not spring Design Pattern

|--->It is GOF Design Pattern and can be implemented in any oop language.. including java

|--->It is has become popular becoz of Spring framework .. It is given to provide set of guidlines while keeping classes in dependency .. Since spring also support dependency management .. it is recomanded to design taget and dependent classes accorrding this design pattern.

|--->This desing pattern says .. i allow the classes of depedency management (target and dependent classes) as loosely coupled interchangable parts.

To implement STrategy DP, we need to implement 3 principles/rules

if the degree of dependecny is less b/w two comps then they are called Loosely coupled..
if the degree of dependency is more b/w two comps then they are called tightly coupled.

3 principles
==========

1) prefer Composition over Inheritence

2) Always code to interfaces/abstract classes i.e never code to concreate classes/ implementation classes/ sub classes..

3) Our code must be open for extension and should closed for modification.

1) prefer Composition over Inheritence

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

Inheritence (IS-A relation..)

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

Compoistion (HAS-A relation)

=>if one class wants to use entire state and behaviour of another class then design those classes having inheritence

=>if one class wants to use limited/some state and behaviour of another class then design those classes having composition

Limitations of Inheritence
---

i) Few oop languages like java do not support multiple inheritence (i.e we can not access the state and behaviour from multiple classes at a time)

ii) Code becomes fragile (Easily brekable)

iii) Testing of code is bit complex..

i) Few oop languages like java do not support multiple inheritence (i.e we can not access the state and behaviour from multiple classes at a time)

problem::

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

```
class B {
  ...
  ..
}
```

```
class C{
  ...
  ...
}
```

Not possilbe in Java ...

note:: Do not look at interfaces in java in the angel of inheritence ...becoz nothing to inherit from interfaces. Look at interfaces in the angel of polymorphism and loose coupling

Soltution:

```
class A {
  private B b=new B();
  private C c=new C();
  ...
  ..
}
```

```
class B{
  ...
  ..
}
```

```
class c{
  ...
  ...
}
```

Using composition we can make One class using the state and behaviour of multiple classes..

ii) Code becomes fragile (Easily brekable)

Problem::

```
class X{
      float
  public int m1(){
    ..
    return 100;
  }
}
```

```
class Y extends X{

  public int m1(){
    ...
    return 1000;
  }
}
```

```
class Z extends Y{
  ...
  ...
}

class Z1 extends Y{
  ....
  ....
}
```

note:: if we modify the singnature of any method that is there in java.lang.Object class , that will distrub the entire classes of the inheritence hierarchy..

if we change the signature of m1() method in "X" class then all the classes of the inheritence hierarchy will be distrubed.

solution:: (using composition)

==========

```
class X{
      float
  public int m1(){
    ...
    retrun 100;
  }
}
```

```
class Y{
  private X x=new X();

  public int m1(){
    ...
    int val=x.m1();
    return val+100;
  }
}
```

```
class Z extends Y{
  ...
  ...
}

class Z1 extends Y{
  ....
  ....
}
```

Math.round(x.m1());

note:: HEre When m1() method signature is chanaged (int to float) then it distrubs only one line of Y class .. not other classes that inheriting and using Y class.. This indicates code is not easily breakable..

iii) Testing of code is bit complex..

problem::

```
class X{
  public int m1() {
     ....
    return ..
  }
  public int m2(){
      ....
     return ..
    }
}
```

```
class Y extends X{

 public int m3(){
    ....
    ..  return ..
  }
}
```

```
class Z extends Y{
    public int m4(){
       ....
       return ;
     }
}
```

Client App

```
Z z=new Z();
  z.m1(); valid()
  z.m2(); valid()
  z.m3(); valid()
  z.m4(); valid()
```

unitTesting:: Programmer's testing on his own piece of code is called UnitTesting...

=>Testing is all about matching expected results with actual results..

if matched --> Test results are positive

if not matched ->Test result are negetive

note:: while testing "Z" class , we need to test "Z" class direct methods (m4) and inherited methods (m1,m2,m3) .. This improves burden on the programmer..

Solution::

```
calss  X{
  public int m1(){
     ....
     ...
   return
  }

 public int m2(){
     ....
     ...
   return
  }
}
```

```
class Y{

 pubic int m3(){
    ...
    ..
  }
}
```

```
class Z{
 private X x=new X();
 private Y y=new Y();

 public  int m4(){
    int val1=x.m1();
    int val2=x.m2();
    int val3=y.m3();
   return val1+val2+val3+100;
   }
 }
```

```
Z z=new Z();
  z.m4();  (valid)
  z.m1() ; (invalid)
 z.m2() ; (invalid)
 z.m3() ; (invalid)
```

=> Here we can only m4() method on "Z" class object and we can not call m1(),m2(),m3() methods on the same object.. becoz composition.. So we need to unittesting only on "m4()" method .. In that process unit testing m1(),m2() ,m3() methods also completed indirectly..

2) Always code to interfaces/abstract classes i.e never code to concreate classes/ implementation classes/

Problem:

```
========
        (target class)
class  Flipkart {
  private DTDC dtdc=new DTDC();

  public  String  shopping(float[] items, float[] prices){
        ....
         ....
        dtdc.deliver();
       return ..;
    }
}
```

```
 //dependent class 1
public class   DTDC{

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

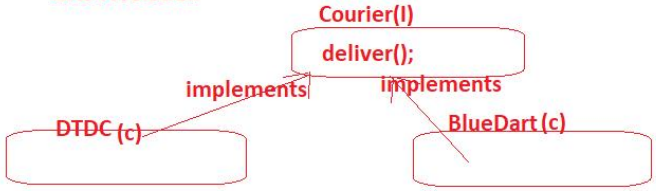
```
                //Dependent class2
        public class BlueDart {

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

=>Here Flipkart and DTDC/BlueDart (taget class ,dependent classes) are in tight coupling.. i.e

(a) if Flipkart wants to use BlueDart services instead of DTDC services then we need to modify the code.. of Flipkat class (Tight coupling)

(b) if the method names in DTDC/BlueDart (Dependent classes) are changed (like from deliver() to supply()) then we need to modify of code of Taget class (Tight coupling)

Solution:: make all Dependent classes implementing common Interface having common methods Declaration and use that Common Interface referece variable in Taget class while creating HAS-A relation



code::

```
=====                 interface Courier{
                        public String deliver(int oid);
                      }
    //Dependent class1
public class DTDC  implements  Courier{

   public   String   deliver(int oid){


       .... //delivery logic
       ....
     return status;
   }
}
```

```
                //Dependent class2
                BlueDart
public class          implements  Courier{

   public   String   deliver(int oid){


       .... //delivery logic
       ....
     return status;
   }
}
```

HEre DTDC,BlueDart classes can not change deliver(-) method name..becoz they are getting it from the common interface called Courier.

```
//target class
==========
public class Flipkart {
 private Courier courier=new DTDC(); (bad pratice)
  private  Courier courier;
  public void setCourier(Courier courier){
        this.courier=courier;
   }
  public  String  shopping(float[] items, float[] prices){
       ....  //bill calc logic
        ....
       ... ... //logic to generate oid  (order id)
    String status= courier.deliver(oid)
     return    billAmt+status;
  }
}
```

```
//Client app
==============
Flipkart fpkt=new Flipkart();
Courier  cr1=new DTDC();
fpkt.setCourier(cr1);
-----------------------
Courier cr2=new BlueDart();
fpkt.setCourier(cr2);
```

note:: Here we can change Dependent of the taget with out touching the source code of target class (Looseocupling)

---

3) Our code must be open for extension and should be closed for modification.

note1:: Principle2/rule2 implementation indirectly makes our code open for extension becoz we can add more implementation classes for Courier(I) to keep more possible Dependents ready..
eg:: By adding more Depedent classes to Courier (I) like DHL, FirstFlight ande etc.. classes we can keep more Dependents ready to work with Targt class (Flipkart).

```
public class  DHL implements Courier{

     public  String  deliver(int oid){        // Code is open for extension..
         ....
         ...
        return status;
      }
}
```

note2: By making both target and Dependent classes as final classes or by taking their methods as final methods.. We can make our code Closed for modification becoz we can not sub class the final class and we can not override the final methods in the sub classes..

note:: In the implementation of Strategy DP , we also use FActory DP To provide abstraction on Dependent, Target classes object creation and to assign Dependent class object to targe..

```
IOCProj5-StrategyDP-CoreJava
  |----->src
      |--->com.nt.comp
            |----> Flipkart.java (target class)
            |--->Courier.java
            |--->DTDC.java , BlueDart.java,FirstFlight.java (dependent classes)
      |--->com.nt.factory
            |----->FlipkartFactory.java     (factory class)
      |--->com.nt.test
            |-----> StrategyDPTest.java     (client app)
```

IOCProj5-StrategyDP1-CoreJava

|---> It is implementing StrategyDP with suppport of Factory Pattern..
|--->This project is having following limitation
=> this project is not 100% loosely coupled .. the target and dependent classes are designed effectively ... but if new Dependent class is added or to change to another dependent class from current dependent class we need to modify Factory , client Apps code respectively..
=> By taking the support of properties file we can solve this problem upto some extend..
=>do not hard code target ,dependent class names in Factory class.. Get them dynamically from properties file --> load them dynamically ---> instantiate them and inject dependent to target class object

properties file

=>It is text file that maintain the entries in the form of key=value/ name=value pairs
=> we can take any extension , but recomanded to take .properties
=> "#" symbol can be used to comment the line
=> we can take "_" ,"." ,"-" symbols in the key naming
=> Duplicate keys are not allowed , but duplicate values can be taken..
=> we can read properties file info into Map collection like java.util.Properties..

myfile.properties

```
# maintains personal info
person.name=raja
person.age=30
person.addrs=hyd
```

java.util.Properties (Map Collection)
|--->Sub class of java.util.Hashtable
|--->HashMap supports generics and allows to take any type key ,any type value..
|-->"Properties" does not support generics , here key must be String and value must be String
|---> by using "load()" method "Properties" class we can store properties file data into "java.util.Properties" object element keys and values

java.util.Properties object (Map)

keys	values
person.name	raja
person.age	30
person.addrs	hyd

=>Byte Streams like InputStreams and OutputStream can read,write both text data, binary data (images,audio files ,vedio files and etc..)
=>Character streams like Reader,Writer streams can read,write only text data...
=>InputStream, Reader streams are useful for reading data.
=>OutputStream,Writer streams are useful for writing data..

Sample code

```
//Locate the file using Stream
InputStream is=new FileInputStream("<location of file>/myfile.properties");
// Read data from properties file and write into java.util.Properties class object
Properties props=new Properties();
props.load(is);

// To get specific value of specific key
S.o.p(" person.age key value is "+props.getProperty("person.age")); //30
//To print all keys and values
S.o.p("data is "+props); //gives entire data of the object by calling toString() method internally
```

improved Factory class of Strategy DP to achieve more Loosecoupliing

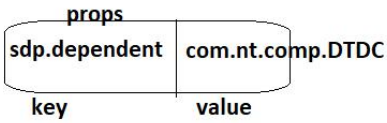
info.properties (com.nt.commons)

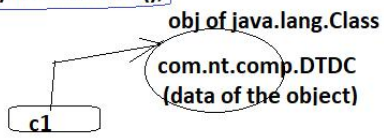
```
sdp.dependent=com.nt.comp.DTDC
```

FlipkartFactory.java

=>static block is class level one time executing block..
where as constructor is object level 1 time executing block.

```
public class FlipkartFactory{
   private static Properties props;

   static{
    try{
     //Locate properties file
      InputStream is=new FileInputStream("src/com/nt/commons/info.properites");
     //Load properties file info to java.util.Properties object..
        props=new Properties();                    props.load(is);
     }
     catch(Exception e){
       e.pristackTrace();
      }
   } //static block

   //factory method
   public static Flipkart getInstance(){
        Courier courier=null;  Class c1=null;  Flipkart fpkt=null;
      //get Dependent class name from Properties collection (props)
   try{
        , load that class and instantiate that class
      //courier= (Courier)   Class.forName(props.getProperty("sdp.dependent")).newInstance();
                                                                    (or)
            //Load dependent class
            c1=Class.forName(props.getProeperty("sdp.dependent"));
            //create Dependent class object
            courier=(Courier) c1.newInstance();

        // create Target class object
          fpkt=new Flipkart();
          //assign Dependent class object to target class object
          fpkt.setCourier(courier);
     } catch(Exception e){   e.printStackTrace(); }
        return fpkt;
      }
   }
```





ClientApp

```
public class StrategyDPTest{
   p s  v main(String args[]){
     Flipkart fpkt=null;
      //get Flipkart object from Factory..
      fpkt=FlikpartFactory.getInstance();
      //invoke b.method
      S.o.p( fpkt.shopping(new String[]{"---","---","..."},
                           new float[]{455,666,777}));

      }
     }
```

Limitations of developing Strategy Design Pattern using Core JAva to achieve more loose coupling

(a) PRogrammer should take care of Instantiations (object creations)
(b) PRogrammer should take care of dependency management (assinging Dependent class obj to target obj)
(c) To achieve more loose coupling we need write addtional code in factory class by taking support of properties file (bit complexity)
(d) No IOC container support
(e) No Internal Cache Support
and etc...

To overcome these problems implements strategy DP using Spring env...

Developing Strategy DesignPattern applicaiton using Spring

```
IOCProj7- StrategyDP-Spring
   |------>src
          |--->com.nt.comp
                 |-----> Flipkart.java
                 |----->Courier.java
                 |----->BlueDart.java
                 |----->DTDC.java
          |--->com.nt.cfgs
                 |----->applicaitonContext.xml
          |--->com.nt.test
                 |----->StrategyDPTest.java
```

Advantages of developing strategy Design Pattern using Spring

=> The IOC Container of spring gives the following benifits

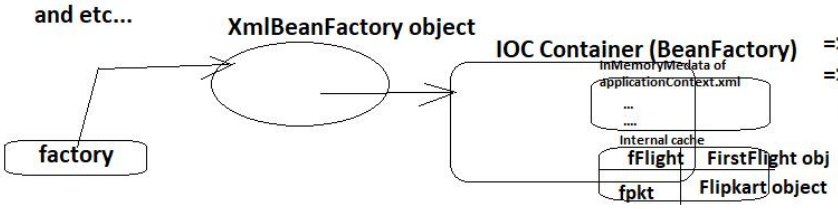
--->Takes care of Instantation of spring beans (objects cration)

-->Takes care of Dependency Management either setter Injection or constructor injection in our example (Assigning Dependent class object to target class object)

-->No need of developing Factory class seperetly becoz IOC container itself acts as Factory

--> The spring bean cfg file gives loosecoupling support to change from one dependent to another dependent

--> IOC contaner provides Internal Cache support for the resuability Spring bean class objects and etc...



=>IOC Container is create based Factory ,Flyweight DP

=> While designing Target and Dependent classes we need Strategy DP principles to design them best..

note: In Setter Injection the thumb rule is ..the IOC container first creates Taget class object and later creates Dependent class object ,calls setter method on target class object having dependent class object as the argument.

note: In constructor Injection the thumb rule is ..the IOC container first creates Dependent class object and later creates Target class object using Parameterized contructor having Dependent class object as the argument value..

When should we use Setter Injection and when should we use Constructor Injection?

Ans) if all properties of spring bean class are mandatory to partcipate in Injection the go for costructor Injection.. otherwise go for setter Injection...

```
public class Student{
  private int sno;
  private String sname;
  private String sadd;
  private float avg;
  //4 setter methods
   ....
   ...
}
```

in applicationContext.xml

```
<beans>
  <bean id=" st"  class="pkg.Student">
     <property name="sname" value="raja"/>
     <property name="sadd"  value="hyd"/>
   </bean>
</beans>
```

```
public class Student{
  private int sno;
  private String sname;
  private String sadd;
  private float avg;
 //4 param constructor
 public Student(int sno,String sname,String sadd, floa avg){
    this.sno=sno;
    this.sname=sname;
    this.sadd=sadd;
    this.avg=avg;
  }
}
```

4!=4*3*2*1 =24

10!=

applicationContext.xml

```
<beans ....>
  <bean id="st"  class="pkg.Student">
     <constructor-arg  name="sno" value="1001"/>
      <constructor-arg  name="sname" value="raja"/>
      <constructor-arg  name="sadd" value="hyd"/>
      <constructor-arg  name="avg" value="67.55"/>
   </bean>
</beans>
```

=> if u r using certain paramerized constuctor for constructor injection.. then we must invovle all params of constructor in the injection process by writing param count no.of <constructor-arg>tags under <bean> .. otherwise exception will be raised... no such restriction for setter injection.. we can involve our choice no.of setter methods for setter injection...

for example
==========
case1:: if i have 10 bean properties in want invovle my choice no.of properties in the Injection then we need just 10 setter methods to perform injection in all permitation and combination.. if we do some thing using constructor injection we need 10! (10 factorial -means lakhs) number construcotr.. which quite complex.. So prefere setter injection here...

case1:: if i have 10 bean properties i want to invovle all properties in the Injection as mandatry then we need just need one 10-param constructor to perform injection on all bean properties , if we do some thing using setter injection we need 10 setter methods but it does not keep the restriction of involving all properties in the Injection.. So prefer constructor injection here..

ctrl+shift+ / ---> To enable multiline comments
ctrl+shift+ \ ---> To disable multiline comments
ctrl+shift+ c ---> To enable/disable singleline comments

note:: Every 0-param constructor is not default constructor.. Only the java compiler generated 0-param cosntructor is called default constructor..
note:: java compiler generates 0-param constructor as default constructor when class not having any another user-defined constructor..

=>Spring IOC container uses 0-param constructor for spring bean instantiation in the following situations
=> when spring Bean is cfg with out any injections
=> when spring bean is cfg only having setter Injections..

=>Spring IOC container uses paramerized constructor for spring Bean instantiation in the following situation
=>when atleast one property is cfg for constructor injection..

```
public String toString() {
    return getClass().getName() + "@" + Integer.toHexString(hashCode());
}
```

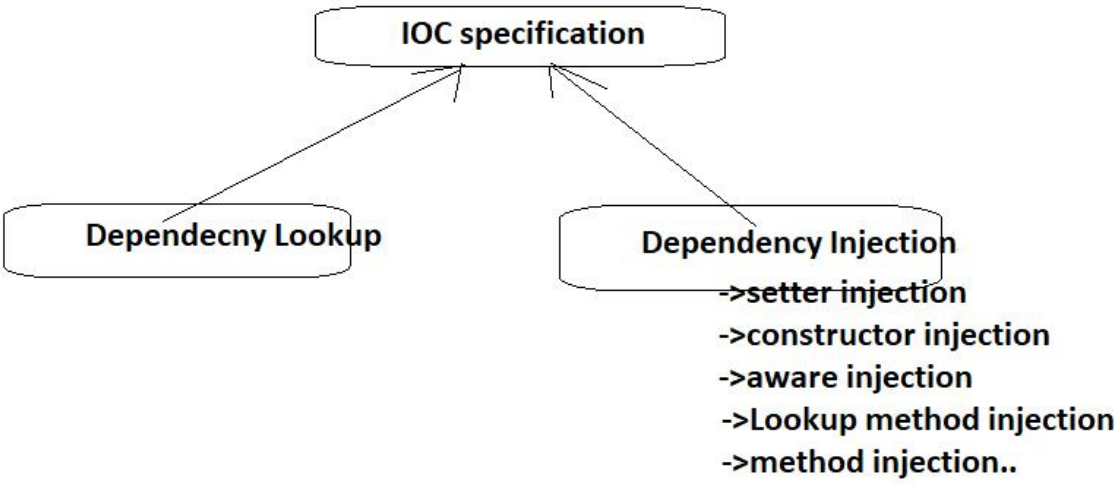
The toString() method java.lang.Object class displays
<fullyqualified current class name>@< hexa decimal string of hashcode>
by using the the above code..
F3 :: To the soruce code..

What the difference between IOC and Dependency Injection?

IOC :: Inversion of Control nothing but inverting control i.e taking control from programmer and giving to some else like container/framework and etc.. to create and manage objects.Here that some one is called IOC container.. ServletContainer,JspContainer,EJB Container , Spring Container can be callled as IOC container

IOC is specification or plan that is having set rules of guidlines asking other than programmer like IOC containers to creeate , manage objects and to keep objects/classes in dependency (as taget ,dependent classes/objects)..

=>Dependency Lookup and Depedency Injection are implementation techniques of IOC ... spring IOC containers support both techniques..



Resolving/Identifying Constructor Parameters

==========

=> Generally we do constructor injections cfgs .. in spring bean cfg file in the order the parameters are placed in cosntructor.. if we mismatch the order of parameters.. then we can indentify/resolve them

(a) using index

(b) using type

(c) using name (from spring 3.x --Best)

Using index (0- based index)

==========

Example

========

```
package com.nt.beans;
public class Marks {
        private int m1,m2,m3;

        public Marks(int m1, int m2, int m3) {
                System.out.println("Marks: 3-param constructor");
                this.m1 = m1;
                this.m2 = m2;
                this.m3 = m3;
        }

        @Override
        public String toString() {
                return "Marks [m1=" + m1 + ", m2=" + m2 + ", m3=" + m3 + "]";
        }
   }
```

applicationContext.xml

==========

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
https://www.springframework.org/schema/beans/spring-beans-4.3.xsd">

   <!-- Spring beans cfgs -->
   <bean id="mk"  class="com.nt.beans.Marks">
     <constructor-arg  index="2" value="70"/>

     <constructor-arg index="1"  value="80"/>
       <constructor-arg  index="0"  value="90"/>
   </bean>

</beans>
```

=>Giving duplicate index , out of range index will throw exceptions...

=> if we ignore to pass indexs for few params.. then they will be assinged with left over indexes..

Using type

```
// Employee.java
package com.nt.beans;
public class Employee {
	private int eno;
	private  float salary;
	private String ename;
	// alt+shift+s , o
	public Employee(int eno, String ename, float salary) {
		System.out.println("Employee:: 3-param construtor");
		this.eno = eno;
		this.ename = ename;
		this.salary = salary;
	}
	@Override
	public String toString() {
		return "Employee [eno=" + eno + ", ename=" + ename + ", salary=" + salary + "]";
	}
}
```

applicationContext.xml

```
<beans ....>
   <bean id="emp"  class="com.nt.beans.Employee">
     <constructor-arg value="raja"  type="java.lang.String"/>
     <constructor-arg  value="1001" type="int"/>
     <constructor-arg value="60000.55f"  type="float"/>
   </bean>
</beans>
```

data type

if multiple params of a constructor are having same then go for resolving the parameters either index or name (recomanded)

Resolving constructor parameter using "name"

```
package com.nt.beans;
public class Student {
	private int sno;
	private String sname;
	private String sadd;
	private  int total;
	public Student(int sno, String sname, String sadd, int total) {
		System.out.println("Student:: 4-param constructor");
		this.sno = sno;
		this.sname = sname;
		this.sadd = sadd;
		this.total = total;
	}
	@Override
	public String toString() {
		return "Student [sno=" + sno + ", sname=" + sname + ", sadd=" + sadd + ",
total=" + total + "]";
	}
  }
```

in applicationContext.xml

```
<beans .....>
  <bean id="st"  class="com.nt.beans.Student">
     <constructor-arg name="total"  value="270"/>
     <constructor-arg name="sno"  value="342"/>
     <constructor-arg name="sname" value="karan"/>
     <constructor-arg name="sadd"  value="hyd"/>
   </bean>
</beans>
```

What is happens if we specify name,type, index attributes in one <constructor-arg> at time?

=> if all these are pointing to same param of consturctor .. then the given value will inject to that param .. otherwise exception will be raised...

org.springframework.beans.factory.UnsatisfiedDependencyException:

note: Do not mismatch constructor params order in dependency injection configurations.. and do not resolve them using different techniques..

Cyclic Dependency Injection/Circular Dependency Injection

==>It is all about making two classes dependent on each other like A on B and B on A
==> It is not at all industry pratice.. (no real time use-case)
==> Setter injection supports cyclic DI and consturctor injection does not support
==> One side setter injection and another side Constructor injection also support cyclic DI

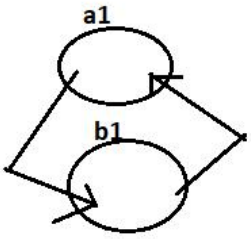
Using Setter Injection

```
class A{
 private B b;

 public void setB(B b){
   this.b=b;
 }
  //toString()
   ...
}
```

```
class B{
 private A a;

 public void setA(A a){
     this.a=a;
 }
 //toString()
   ...
}
```

applicationContext.xml

```
<beans ...>
  <bean id="a1" class="pkg.A">
      <property name="b" ref="b1"/>
  </bean>
  <bean id="b1" class="pkg.B">
      <property name="a" ref="a1"/>
  </bean>
```



Client App

```
A a1=factory.getBean("a1",A.class);
```

```
B b1=factory.getBean("b1",B.class);
```

Exception in thread "main" java.lang.StackOverflowError ---> both classes toString() methods deadlock situation...

Using Constructor Injection

```
public class A{
  private B b;

   public A(B b){
    this.b=b;
   }
   //toString
    ...
}
```

```
public class B{
  private A a;

   public B(A a){
    this.a=a;
   }
   //toString
    ...
}
```

applicationContext.xml

```
<beans .....>

   <bean id="a1" class="pkg.A">
       <constructor-arg ref="b1"/>
    </bean>
   <bean id="b1" class="pkg.B">
       <constructor-arg ref="a1"/>
    </bean>

</beans>
```

Client App

```
A  a1=factory.getBean("a1",A.class);
```

throws this exception

Exception in thread "main" org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'a1' defined in class path resource [com/nt/cfgs/applicationContext.xml]: Cannot resolve reference to bean 'b1' while setting constructor argument; nested exception is org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'b1' defined in class path resource [com/nt/cfgs/applicationContext.xml]: Cannot resolve reference to bean 'a1' while setting constructor argument; nested exception is org.springframework.beans.factory.BeanCurrentlyInCreationException: Error creating bean with name 'a1': Requested bean is currently in creation: Is there an unresolvable circular reference?

Cyclic Dependency Injection having 1 side setter injection and another side constructor Injection

note:: we must call factory.getBean(-,-) having that spring bean class bean id in which setter injection support is there...

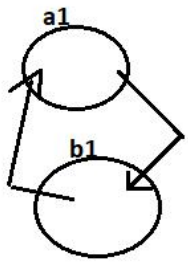
```
class A{
 private B b;

 public void setB(B b){
   this.b=b;
 }
  //toString()
   ...
}
```

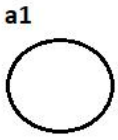
```
public class B{
  private A a;

  public B(A a){
   this.a=a;
  }
  //toString
   ...
}
```

natarazjavaarena
natarazjavaarena@gmail.com

applicationContext.xml

```
<beans ....>
   <bean id="a1"  class="pkg.A">
       <property name="b" ref="b1"/>
   </bean>
   <bean id="b1" class="pkg.B">
         <constructor-arg ref="a1"/>
   </bean>
```





Client App

```
==========
A  a1 =factory.getBean("a1", A.class) ;  // success

B  b1=factory.getBean("b1",B.class);  // error
```

throws

Exception in thread "main" org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'b1' defined in class path resource [com/nt/cfgs/applicationContext.xml]: Cannot resolve reference to bean 'a1' while setting constructor argument; nested exception is org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'a1' defined in class path resource [com/nt/cfgs/applicationContext.xml]: Cannot resolve reference to bean 'b1' while setting bean property 'b'; nested exception is org.springframework.beans.factory.BeanCurrentlyInCreationException: Error creating bean with name 'b1': Requested bean is currently in creation: Is there an unresolvable circular reference?

What is the difference between setter Injection and constructor Injection?

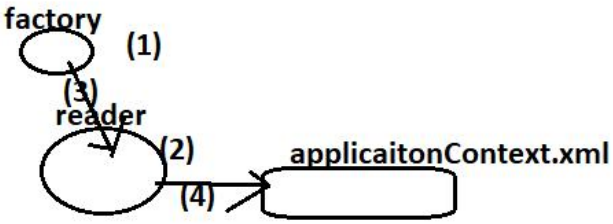
Setter Injection	constructor Injection
(a) uses setter method to inject dependent values/objs to target class object	(a) uses param constructor to instantiate target class and also to inject dependent values/ objects to target class object
(b) we need to use <property> tag for this	(b) we need to use <constructor-arg> for this
(c) we can resolve /identity setter method only passing its name like by passing xxx/Xxx word of setter method in "name" of <property>	(c) we can resolve/indenity constructor params using name/index/type.
(d) resolving setter method by using "name" of <property> tag is mandatory	(d) Resolving constructor param using name/index/type is optional if u write <constructor-arg> tags in the same order of constructor params..
(e) Supports Cyclic Dependency Injection	(e) does not support
(f) Bit Slow becoz injection happens after creating taget bean class object	(f) Bit faster becoz injection takes place while instantiating target bean class obj
(g) In this mode , IOC container first creates Target class object , then creates Dependent class object	(g) In this mode of injection, IOC container first creates dependent class object, then creates target class object
(h) Suitable when u want involve u r choice no.of properties in dependency injection	(h) Suitable when u want involve all bean properties in Dependency injection on mandatory basis
(i) To involve our choice no.of bean properties to setter injection we just need n setter methods for n properties	i)To involve our choice no.of bean properties to constructor injection we need n! overloaded constructors (very complex)
(j) if spring bean is cfg with no injection or only with setter injection then IOC container creates spring bean class object using 0-param constructor	(j) Once constructor injection is enabled, the IOC container creates Spring bean class object using parameterized constructor.
(k) we can use "p" namespace tags/attributes as alternate to <property> to cfg setter injections	(k) we can use "c" namespace tags/attributes as alternate to <constructor-arg> to cfg constructor injections

=>XmlBeanFactory class is deprecated from spring 3.1 .. and it is recomanded to use DefaultListableBeanFactory as alternate to create BeanFactory container...

Limitations of XmlBeanFactory
==========
(a) Does not allow to take multiple multiple xml files at a time as spring bean cfg files.
(b) It internally uses DefaultListableBeanFactory for registering every Spring bean..
(c) Xml parser used XmlBeanFactory to parse and process Xml files is not so good towards Performence
(d) Need to use Resource obj to hold the name of Location of Spring bean cfg file..

Creating BeanFactory container using DefaultListableBeanFactory
==========

```
DefaultListableBeanFactory factory=null;
XmlBeanDefinitationReader reader=null;

//create IOC container
factory=new DefaultLisableBeanFactory();
//create Reader object
reader=new XmlBeanDefinitationReader(factory);
//specify the name of location spring bean cfg
reader.loadBeanDefinitations("com/nt/cfgs/applicationContext.xml");
```

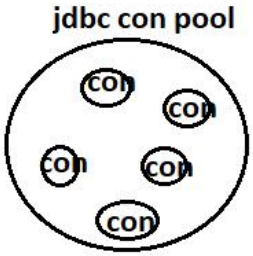


Advantages of DefaultLitableBeanFactory to create BeanFactory container
==========

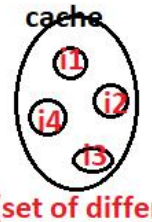
(a) It directly recognizes the spring beans.. with out passing this work to another classes....
(b) Allows to pass multiple xml files as spring bean cfgs files... becoz
loadBeanDefinitations() is designed taking var args..
(c) No need of taking Resource object.. to hold the name and location of spring bean cfg file..
we can pass the same info directly as string..
(d) The Perfomence towards reading and processing Xml files is good ..

Background preparation for developing Realtime DI , Realtime stategy DP based Layered Application (Mini Project Discussions)

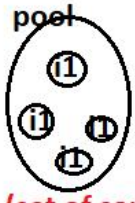
jdbc con pool
==========



initialsize ::10
max size ::30
maxWaitTime :: 100 ms
shrink frequency :: 900 secs



(set of diffent items)



(set of same items)

=>Jdbc con pool is a factory that contains set of readily available set jdbc con objects before actually being used...

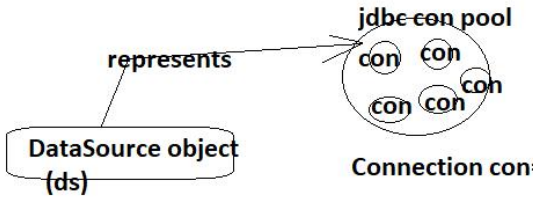
advantages
==========

(a) gives reusability of jdbc con objects..
(b) With minimum no.of jdbc con objects , we can make max requests/Apps talking to Db s/w
(c) Creating jdbc con object, managing jdbc con object and closing jdbc con will be taken care by jdbc con pool itself...

DataSource object
==========

=>DataSource object represents jdbc con pool and it acts entry point for jdbc con pool i.e each jdbc con object from jdbc con pool should be accessed through Datasource object..

=>DataSource object means it is the object of java class that implements javax.sql.DataSource(I)



Connection con=ds.getConnection();

gets one jdbc con object from jdbc con pool

Explain different types jdbc con objects?

(a) Direct jdbc con object

->created by the programmer manually

```
Class.forName(".....");
Connection con=DriverManager.getConnection(-,-,-);
```

(b) Pooled jdbc con object (recomanded)

-->collected from the jdbc con pool through DataSource object..

```
Connection con=ds.getConnection();
```

=> All jdbc con objects of a jdbc con pool represents connectivity with same Db s/w...

=> for example jdbc con pool for oracle means.. all jdbc con objects in the jdbc con pool represents connectivity with same oracle DB s/w..

=> we can create diff jdbc con pools for diff DB s/ws..

Two of types jdbc con pools
==========

(a) standalone jdbc con pool

->useful in standalone Apps

->runs outside the servers like Tomcat, weblogic and tc..

-> These are Independent jdbc con pool

eg:: apache DBCP , Prxool , Hikari CP(best), C3P0 and etc..

(b) server managed jdbc con pool

->created and managed in Severs like Tomcat , weblogic and etc..

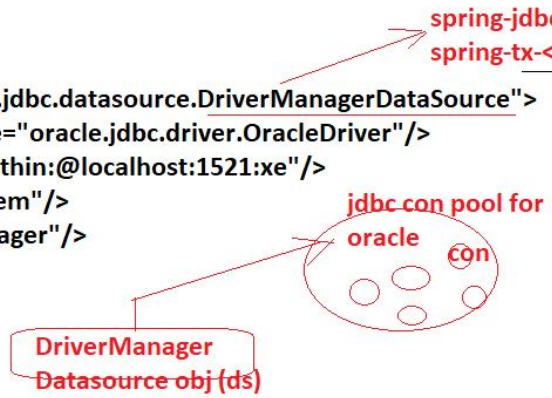
->useful in those Apps that are deployable and excutable in servers like web applications

eg:: tomcat managed jdbc con pool , weblogic managed jdbc con pool and etc..

note:: Spring f/w gives built-in jdbc con pool support.. i.e it is one pre-defined DataSource class like DriverManagerDataSource implementing javax.sql.DataSource(I). if we configure this class spring bean.. then Spring container /IOC container creates DataSource object representing jdbc con pool.

in applicationContext.xml

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



note:: we can get bean property names of pre-defined class by using api docs or source code of that class...

---

3 types of java beans (Based on the kind of data they hold)

==========

VO class (Value object class)
DTO class (DataTransfer Object class)
BO class (Business Object class)

VO class

========

=>The java bean whose obj holds either inputs or outputs is called VO class..
Generally in most of the Apps the inputs or outputs will be there as "text" values.So this class generally maintains all its properties as String properties...

StudentVO.java

```
public class StudentVO{
  //bean properties
   private String sno;
   private String sname;
   private String sadd,
    private String m1,m2,m3;
   //setters  && getters
        ... ....
    public  void setSno(String sno){
      this.sno=sno;
     }
     public String getSno(){
       return sno;
     }
      ...
      ....
      ...
  }
```

DTO class
=========
It is also called as TO class (Transfer Object)

=>It is java bean whose object holds shippable(transferab le) data from one class to another class with in a project or from project to another project. This class implements java.io.Serializable (I) and contains different types properties ..

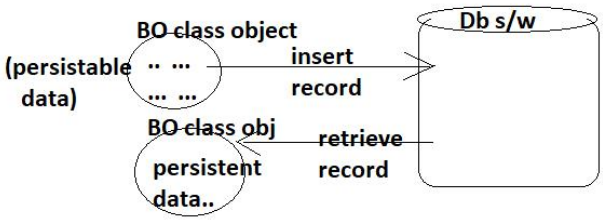
CardDetailsDTO.java

```
public class CardDetailsDTO implements  java.io.Serializable{
   private long cardNo;
   private String holderName;
   private String bankName;
   private  String paymentGateway;
   private  Date expiryDate;
   private   int  cvv;
   //setters  && getters
     ....
     ...
  }
```

note:: We can send only Serialiable objects data over the network.. To make object as Serializable object the class of the object must implement java.io.Serializable(I) (Marker Interface-Empty interface)

Serialiazation is the process of converting object data into bits and bytes.. or(Stream of bytes) .. These bits can be sent over the network or can be written to the file as needed...

BO class
========
It is the Java bean whose object holds persistable data (to inserted) or persistent data (already saved). Generally this class will be taken on 1 per db table basis .. This properties type , count must be compitable with Db table cols type and count..



BO class is also called as Entity class or model class or Domain class or Persistence class..

BO holds .. persistable /persistent data.
DAO holds .. PErsistence logic (jdbc code, hibernate code)

Oracle DB table
==========
Customer
|--->cno (n) (pk)
|--->cname (vc2)
|--->cadd (vc2)
|--->billAmt (float)

BO class/Entity class
==========
BO
Customer.java

```
public class CustomerBO{
  private int cno;
  private String cname;
  private  String cadd;
  private float billAmt;
   //setters & getters
     ...
     ...
  }
```

Writing multiple logics in single java class is a bad practice the reason are
==========
(a) Since we mixup multiple logics in a single class , the code looks very clumsy and the clean seperation b/w logics will not be possible..
(b) The modifications done one kind of logics effects other kind of logics
(c) The maintenance and Enhacement of the Project becomes difficult.
(d) Parellel developement is not possible , So the productivity is very poor..
(e) it is not industry standard..

To solve the above problem.. develop u r applications as layered applicaitons...
A layer is logical partition in the application reprsenting cerntain logics.. having 1 or more classes /files.
presentation layer contains perensetation logic
service class layer contains b.logics/serivce logics
persistence layer contains persistence logics.. and etc..
The different layers of layered Apps will interact .. with each other..

=>Presentation logic
The logic that gives user-interface either supply inputs or to display outputs..
eg:: html,css, jsp logics |
Scanner, sysout , awt/swing logics
=>business logics /service logics
The main logic of the Applicaiton that deals with calculations, analyzations and etc..
eg:: calc intreset amount , calc emp Gross, net salaries , calc student total,avg, rank
=>Persistence logic
The logic that interacts with Persistence store like DB s/w and manipulates its DAta .. by performing CURD operations
eg:: jdbc code , IO streamscode
hibernate code and etc..
spring jdbc/orm /data code..

Typical Standalone Layered in Java
==========
Client App --VO-->Controller class --DTO-->service class --BO-->DAO class -->DB s/w
(presentation logic) (Monitoring logic) (B.logic) (service logic) (Persistence logic)

DAO class (Data Access object class)
========
=>The java class that contains only persistence logic and makes that logics re usable logics and flexible logics to modify is called DAO class..
=> It is called DAO class becoz it having code that is developed by using DataAccess Technologies/frameworks like JDBC,hibernate and etc..

=>DB s/ws like oracle ,mysql, postgresql and etc.. falls under Data storage softwares ...
=>Jdbc, hibernate , spring jdbc/orm/data falls under DataAccess Technologies/frameworks..because we develop persistence logic to manipulate Db s/w data..

=> if no.of db tables in DB s/w are <100
then take 1 DAO class per 1 Db table
=> if no.of db tables in DB s/w are >=100
then take 1 DAO class per 4 or 5 related Db tables
=>Every DAO class contains
a) Query part (SQL queries used for persistnce logic)
-->Declare them at top of DAO class Stirng constant values having uppercase letters
-->It is recomanded to avoid "*" in the SQL Query .. place col names always..

In DAO class

```
private static final String GET_STUDENT_BY_SNO="SELECT SNO,SNAME,SADD FROM STUDENT
                                                WHERE SNO=?";
```

In DAO class

```
private static final  String GET_STUDENT_BY_SNO="SELECT SNO,SNAME,SADD FROM  STUDENT
                                                 WHERE SNO=?";
```

b) Code Part

=> contains multiple methods to perform multiple persistnce opertaions (CURD) operations on 1 method per each Persistnce operation.. These methods get inputs from service classs (<=3 values as normal params , >3 values as BO class objs)

In DAO class

==========

```
public  StudentBO  getStudentBySno(int sno){
    ....
    ...
 }
public  int insert (StudentBO  bo){
   ...
   ..
}
```

This DAO calss can use either DirectConnection or pooled Con to interact with Db s/w

---

SErvice class

==========

=>It is the java class that contains either b.logic or service logic.

=> b.logic means that main logics of the App which fullfills the main requirement of Application having calculations , analyzations with Transaction Mgmt support..

=>calculating total,avg, and generating results

=>calculating simple , compound intrest amounts

Executing logics by applying do every thing or nothing principle

=> we take this class on 1 per module basis... having multiple b.methods..

=> One service class can use 1 or more DAO classes internally.

=> This class gets more than 3 inputs as DTO obj from controller and gives more than 1 output value to Controller as another DTO object.

=>set of instructions is called 1 program

=>set of programs is called 1 App

=> set of Apps is called 1 module

=> set of modules is called 1 project

In Service class

==========

```
public  String  registerStudent(StudentDTO dto){
    ....
    ....  b.logic
    ...
  return ...;
 }
public  String  registerEmployee(EmployeeDTO dto){
    ...
    ...     //b.logic
    ...
 }
```

Controller class

==========

=> This is a java class that contains monitoring logics/controlling logics.. like passing appropriate Client App inputs to Appropriate service class and passing appropriate service class outputs to Appropriate Client App.

=> Gets inputs as VO from Client App and passes to Service class as DTO ...

=>This will be taken on 1 per Project basis..

In controller classs

```
public String processStudent(StudentVO vo){
   ....
   ....  return ...;
}

public String processEmployee(EmployeeVO vo){
   ....
   ....
   return ...;
}
```

ClientApp

==========

=> Contains User-Interface logic .. providing env.. to read inputs from enduser and display outputs for enduser..

=> There can be multiple client Apps .. to gather different of types inputs and to display their outputs (count will vary based on the requirement)

Client App ----VO----> controller class ----DTO----> service class ----BO---->DAO class -------->DB s/w

=>While dealing with pooled connections in DAO class.. it needs DataSource object as dependent ,So we can inject datasoure to DAO ,Similarly we can inject DAO object to SErvice class and Service class object to Controller.But ClientApps get Controller class objects by using factory.getBean(-) becoz we create IOC container in the Client App itself..

Controller is just act as Target class and not dependent to another class.. So Controller class need not to implement any interface... where as Service class, DAO class, DataSoruce classes should implement repspective interfaces to achieve loosely coupling.. becoz they acting as dependent classes to other classes..

Customer gives

cno ,cname, cadd , pamt,rate,time

wants to get Simpleintrest amount as the output after registering customer (By inserting customer details to DB table as record)

Exception propagration

==========

```
public  class EmployeeDAO{

    public int insert(EmployeeBO bo){
        try{
          ...
          .... //jdbc code
          ...
          ...
        }
      catch(Exception e){
        ...
       }
   }
```

when exception is raised in the insert(-) method ,the control automatically goes to catch(-) block and the expcetion will be eaten /supressed there.. But we need to propagate /pass this exception to Client App becoz Enduser excepts error /success output from Client App only.. To solve this problem do not place try/catch blocks in DAO ,Service, Controller classes and make them the declaring the exception to thrown using "throws" clause and make clientApp catching and handling the exception

HEre the exception raised DAO propagages Client App through Service ,Controler classes becoz they are also not caching and handling exception..and they are also propagating the exception using "throws".

```
public class  STudentDAO{

 public  int insert(StudentBO bo)throws Exception{
      ....
      ....
      ...
  }
}
```

Story Board of Layered Application

ClientApp-----VO----> MainController -----DTO--------> CustomerMgmtServiceImpl ----BO------> CustomerDAOImpl ---------->Db s/w
(presentation logic) (Monitoring logic) (b.logic) (Persistence logic)

Customer_tab
cno(pk) cname cadd pAmt intrest

CustomerDAO.java
```
public interface CustomerDAO{
  public int insert(CustomerBO bo)throws Exception;
}
```

CustomerDAOImpl.java
```
final
public class CustomerDAOImpl implements CustomerDAO{
   private DataSoruce ds;

  public CustomerDAOImpl(DataSource ds){
       this.ds=ds;
   }
  public int insert(CustomerBO bo)throws Exception{ (h)
    ...
    .... //jdbc code to insert record
    ....
    ....
    return 0/1; (i)
  }
}
```

CustomerMgmtService.java
```
public interface CustomerMgmtService{
  public String calculateSimpleIntrestAmount(CustomerDTO dto)throws Exception;
}
```

CustomerMgmtServiceImpl.java
```
final
public class CustomerMgmtServiceImpl implements CustomerMgmtService{
   private CustomerDAO dao;

   public CustomerMgmtServiceImpl(CustomerDAO dao){
       this.dao=dao;
    }
                                  (f)
```

note:: All DataSource classes implementing classes of javax.sql.DataSource(I) .
eg:: DriverManagerDataSource
HikariDataSource -->hikari cp
BasicDataSource --> Apache DBCp

applicationContext.xml
```
<beans ....>  (11)
  <bean id="drds" class="org.sf.jdbc.datasource.DriverManagerDataSource">
      ....
setter ....  //inject values jdbc properties (driverClassname,url,username,password)   11
      ...
  </bean>
  <bean id="custDAO" class="com.nt.dao.CustomerDAOImpl">   (9)
     <constructor-arg ref="drds"/>   (10)   12
  </bean>
  <bean id="custService" class="com.nt.service.CustomerServiceImpl">   (6)
     <costructor-arg ref="custDAO"/>   (7)   13
  </bean>
  <bean id="controller" class="com.nt.controller.MainController">   (3)
     <constructor-arg ref="custService"/>   (4)   14
  </bean>
```

Internal cache of IOC container (2?)

keys	values
drds	DriverManagerDataSoruce class obj ref (5?)
custDAO	CustomerDAOImpl obj ref (8?)
custService	CustomerMgtServiceImpl (10?) obj ref
controller	MainController obj ref

(15) (keeping beans in the cache)

```
                                    (f)
public String  calculateSimpleIntrestAmount(CustomerDTO dto)throws Exception{
  //calculate simple intrest amount  from DTO
      ...     float iamt=dto.getPamt()*dto.getRate()*dto.getTime()/100.0f;
      ...
  //prepare CustomerBO  having persistable Data like cno,cname,cadd,pamt, intrestamt
      ....
      ...
   //use DAO
(i) int count=dao.insert(bo);   (g)
   if(count==0)
      return "Customer registration failed --> intrest amount is ::"+iamt;
   else                                                                    (k)
    return "Customer registration succeded --> intrest amount is ::"+iamt;
  }
}
```

MainController.java

```
public final class  MainController {
  private  CustomerMgmtService service;

  public   MainController(CustomerMgmtService service){
       this.service=service;
   }
                                  (d)
  public   String    processCustomer(CustomerVO vo)throws Exception{
   //convert CustomerVO to  CustomerDTO
       ....
       ....
   //use service                          (e)
(l)  String result=service.caculateSimpleIntrestAmount(dto);
     return result;  (m)
    }
}
```

RealtimeDITest.java (ClientApp)

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

    //read inputs from enduser using scanner
       ....
       ....
       ....
    //store inputs into VO class object
       ...
       ...
    //create BeanFActory Container
          ....
    //get Controller class object...
                                                  (b)  (1)
(16) MainController controller=factory.getBean("controller",MainController.class);
    //invoke methods
     try{
                                (c)
(n)   String result=controller.processCustomer(vo);
      S.o.p(result);  (o)
     }
     catch(Exception e){
       S.o.p("Internal Problem ::"+e.getMessage());
       e.prinStacTrace();
     }

  }//main
}//class
```

Procedure to develop the above layered using Gradle

==========

step1) make sure that gradle buildship plugin is avaiable in Eclipse ...

step2) create GradleProject...

Fie -> project ---> gradle --->gradleproject ----> next ---> fill the details

Projectname :: IOCProj13-RealtimeDI,

choose gradle installation folder

select Autosynchronization

-> next --> finish.

step3) add the following dependencies info (jars info) in dependencies { } enclouser by collecting them from mvnrepository.com

spring-context-support-<ver>.jar ojdbc6.jar spring-jdbc-<ver>.jar	build.gradle

build.gradle

```
plugins {
   // Apply the java-library plugin to add support for Java Library
   id 'application'
}
repositories {
  mavenCentral()
}
dependencies {
    // https://mvnrepository.com/artifact/org.springframework/spring-context-support
implementation group: 'org.springframework', name: 'spring-context-support', version: '5.2.8.RELEASE'
// https://mvnrepository.com/artifact/org.springframework/spring-jdbc
implementation group: 'org.springframework', name: 'spring-jdbc', version: '5.2.8.RELEASE'
// https://mvnrepository.com/artifact/com.oracle.database.jdbc/ojdbc6
implementation group: 'com.oracle.database.jdbc', name: 'ojdbc6', version: '11.2.0.4'

}
```

step4) Develop resources as shown below ... by creating packages in src/main/java folder...

```
IOCProj13-LayeredApp
    |------>src/main/java
            |------>com.nt.vo
                    |----->CustomerVO.java
            |---->com.nt.dto
                    |---->CustomerDTO.java
            |---->com.nt.bo
                    |----->CustomerBO.java
            |--->com.nt.cfgs
                    |---->applicationContext.xml
            |--->com.nt.dao
                    |---->CustomerDAO.java
                    |--->CustomerDAOImpl.java
            |---->com.nt.service
                     |---->CustomerMgmtService.java
                     |--->CustomerMgmtServiceImpl.java

            |--->com.nt.controller
                     |------>MainController.java
            |---->com.nt.test
                     |------>RealtimeDITest.java


            >build.gradle
```

tep5) Keep DB table ready in Oracle DB s/w...

GUI DB tools

SQLDeveloper for oracle
Toad for oracle
Toad for mysql
Mysql Workbench
SQLyog for mysql
and etc...

i) create connection in sql developer
+ create new Db coinnection ---->
connection name: con
provide username: ---,password: --- --> Test --->
connect.

ii) create Db table

Right clikc Tables --->new table ----> and provide details like
table name , col names, data type and etc.. which generates this SQL

```
CREATE TABLE "SYSTEM"."SPRING_CUSTOMER"
(    "CNO" NUMBER(10,0) NOT NULL ENABLE,
     "CNAME" VARCHAR2(20 BYTE),
     "CADD" VARCHAR2(20 BYTE),
     "PAMT" FLOAT(126),
     "INTRAMT" FLOAT(126),
      CONSTRAINT "SPRING_CUSTOMER_PK" PRIMARY KEY ("CNO"));
```

---

ii) create sequence to generate values in "CNO" col Dynamically...

Right click sequences ---> create new sequence by providing sequence name ,
start with,increment , min value , max value details ----> that generates this SQL Query

```
CREATE SEQUENCE "SYSTEM"."CNO_SEQ1" MAXVALUE 10000 INCREMENT BY 1 START WITH 1;
```

step5) Develop the Soruce code and run the Client App either directly... or using gradle...

Where did u use Depdendency injection or StrategyDP in ur real Project development?

Ans) Every Project is Layered Application... having DataSource, DAO ,SErvice ,Controller and etc.. classes

Standalone Layered App

Client App ----------> Controller classs ----------> Service class ---------->DAO class --------> DB s/w

web based Layered App
---
html/jsp(UI) ---------->controller Servlet ------->Service class ---------->DAO class ------>DB s/w

if these Apps are developed in Spring env... we use Depedency Injection (mostly construction Injection) based on strategy DP for the following injections
=> DataSource will be injected to DAO
=> DAO will be injected to Service
=>Service will be injected to Controller

To run Layered App in Gradle env..

a) place the following additional lines in build.gradle

```
mainClassName="com.nt.test.RealtimeDITest"
run{
  standardInput=System.in
 }
```

b) place com.nt.cfgs package that is applicationContext.xml file in src/main/resource folder...

c) Run the Application refresh

Go gradle tasks ---> Project (our) ---> applicaiton ---> run --> run gradle task ---> go to console and give inputs...

MP+UP+Bihar+UK+Haryana+Delhi ::: Employee Regisration (calculating gross,netsalaries)

PJB+RAJ+GUJARAT + WB+Orissa+MH +HP+AP:: Corona Patient Registration (call Bill Amount on 3000 per day)

remaining states :: IPL Batsman registration (calculating batting Avg, Bolwing avg)

natarazjavaarena@gmail.com

MySQL DB s/w

==========

Maria DB (internally mysql)

type :: DB s/w

version :: 8.x

vendor :: Devx/Sun Ms /Oracle corp

Open Soruce

default port no :: 3306

admin username :: root

password :: root (will be chosen during the installation)

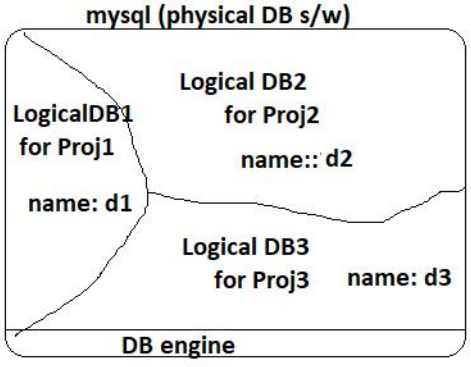
To dowload mysql s/w :: download as setup file

https://www.mysql.com/downloads/

(mysql-installer-community-8.0.16.0.ext -->setup file)

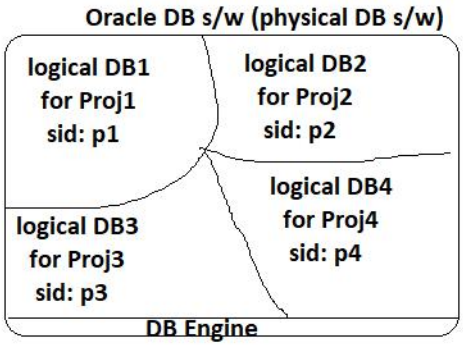
Gives built-in GUI DB tool called "mysql workbench"...

Physical DB s/w vs Logical DB



DBA will create these logical DBs..

flats :: Logical DBs



=>Oracle jdbc driver s/w (ojdbc6/7/8.jar)

=> To interact with mysql Db s/w .. we need mysql Connector/j JDBC driver and it comes in the form jar file (mysql-connector-java-<ver>.jar)

Use SQL Command prompt

==========

searach sql --> lauch run SQL command Line ---->

```
SQL> connect
Enter user-name: system
Enter password: manager
Connected.
SQL > type the queries here...
```

Collection Injection

==========

=> It is all about injecting values to array ,collection type bean properties through Dependency Injection..using diff tags in spring bean cfg file...

Property type	tag or attribute
simple /primitive (including String)	value attribute or <value> tag
Object type/reference type	ref attribute or <ref>
array	<array> or <list>
java.util.List	<list>
java.util.Set	<set>
java.util.Map	<map>
java.util.Properties	<props>

A spring bean can have 4 types of bean properties

(a) Simple bean properties
(primitive data type/wrapper data type/ String bean properties)

(b) Object type/reference type bean properties

(c) Array type bean properties

(d) Collection type bean properties

Injecting values to array type bean properties

==========

=>if the array is simple/String data type
then use <array> with <value> tag
to perform injections to elements.

=>if the array is reference/Object data type
then use <array> with <<ref> tag
to perform injections to elements.

=>arrays are HOMOGENUOUS
=>array elements needs memory continuosly
=>arrays are having fixed size
=>we can access array elements through indexes (0 based index)
=>when we removed element values from array, its size will not be decresed
=>arrays are objects in java ... having one built-in property called length.
=>preserves the insertion order of the array elements..

To change jdk /java version for Eclipse Gradle Project

==========

(a)Right click on JRE Systems --->properties ---> alternate JRE ---> Installed JRE ---> add --> choose jdk installation foflder ---> select jdk version ---> select workspace default --->
... ...

(b) Change java compiler version ---> open properites of the Project ---> java compiler ---> choose version

(b) change java version in Project facets --> open properites of the Project ---> project facets ---> java --> choose version..

```
int[] b , a;  //here a, b are int[] type
int  a[],b ;  here a is [] type , b is simple int type
```

example

example

==========

```
public class MarksInfo {
	private int marks[];

	public void setMarks(int[] marks) {
		this.marks = marks;
	}

	@Override
	public String toString() {
		return "MarksInfo [marks=" + Arrays.toString(marks) + "]";
	}
}
```

In applicationContext.xml

```
<bean id="mi"  class="com.nt.beans.MarksInfo">
   <property name="marks">
    <array>
      <value>10</value>
      <value>20</value>
      <value>30</value>
    </array>
   </property>
 </bean>
```

Internal code

==========

```
//create Bean class ohject
MarksInfo mi=(MarksInfo)Class.forName("com.nt.beans.MarksInfo").newInstance();
 int[] marks=new int[]{10,20,30};
 mi.setMarks(marks);
```

Why should we go for collections

==========

=>collections are hetrogenous

=>Collections are dynamically growable

=> collection elements do not allocate memory continusoly..

=> provides lots built-in methods to access and manipulate elemenent values..

=> internally uses multiple DataStructure algorithms to manage data ...

=>Collections with generics makes our code as type safe code..

and etc.. **(No type casting issues)**

->if we insert simple values , they will be converted into wrapper objects (autoboxing)

Injecting values to java.util.List collection

==========

use <list> with <value> or

<list> with <ref> for injections..

->insertion order preserved

-> duplicates are allowed

-> can access elements through index (0-based)

-> null insertion possible

-> List collection are ArrayList ,Vector, Stack, LinkedList

```
public class College {
	private List<String>  studNames;
	private  List<Date>  datesList;
	public void setStudNames(List<String> studNames) {
		this.studNames = studNames;
	}
	public void setDatesList(List<Date> datesList) {
		this.datesList = datesList;
	}

	@Override
	public String toString() {
		return "College [studNames=" + studNames + ", datesList=" + datesList + "]";
	}
}
```

in applicationContext.xml

==========

```
<!-- List Injection -->
<bean id="clg" class="com.nt.beans.College">
   <property name="studNames">
      <list>
        <value>raja</value>
        <value>rani</value>
        <value>raja</value>
        <value>suresh</value>
      </list>
   </property>
   <property name="datesList">
      <list>
        <ref bean="sysDate"/>
        <ref bean="dobDate"/>
      </list>
   </property>
</bean>
```

note:: if the bean property type is java.util.List then the IOC container internally uses java.util.ArrayList type ListCollection... and we can not change it ..
if want work with other ListCollections like Vector,Stack and etc.. then take concrete class type bean proerties as shown below

```
private Sta ck<String>  studNames;
      (or)
private  Vector<String> studNames;
```

Internal code for List<Date> datesList property

==========

```
//bean object creation
College clg=(College)Class.forName("com.nt.beans.College").newInstance();
//Dependent objs creation..
Date sysDate=new Date();
Date dobDate=new Date();
dobDate.setYear(1900+90);
dobDate.setMonth(11);
dobDate.setDate(22)

List<Date> datesList=new ArrayList();
datesList.add(sysDate);  dateList.add(dobDate);
    //setter injection...
clg.setDatesList(datesList);
```

Injecting values into java.util.Set type Collection

==========

=>use <set> with <value> or <set> with <ref> for this injection cfgs

Set collection

-->does not allow duplicates

--> does not maintain indexes for elements

-> insertion order not preseved

->To preserve use LinkedHashSet

->allows only one null value

ContactsInfo.java

```
public class ContactsInfo {
        private   Set<Long>  phoneNumbers;
        private  Set<Date>    dates;

        public ContactsInfo(Set<Long> phoneNumbers, Set<Date> dates) {
                System.out.println(phoneNumbers.getClass());
           this.phoneNumbers = phoneNumbers;
                this.dates = dates;
        }

        @Override
        public String toString() {
                return "ContactsInfo [phoneNumbers=" + phoneNumbers + ", dates=" + dates + "]";
        }
}
```

In applicationContext.xml

```
<bean id="cInfo"  class="com.nt.beans.ContactsInfo">
     <constructor-arg name="phoneNumbers">
        <set value-type="java.lang.Long">
           <value>999999999</value>
           <value>888888888</value>
           <value>777777777</value>
           <value>999999999</value>
        </set>
     </constructor-arg>
     <constructor-arg  name="dates">
        <set value-type="java.util.Date">
          <ref bean="sysDate"/>
          <ref bean="dobDate"/>
          <ref bean="sysDate"/>
        </set>
     </constructor-arg>
  </bean>
```

Internal code

==========

```
// creates Dependent classes objects
 Date  sysDate=(Date)Class.forName("java.util.Date").newInstance();
Date   dobDate=(Date)Class.forName("java.util.Date).newInstance();
  dobDate.setYear(1900+90);  dobDate.setMonth(11);  dobDate.setDate(22);

// create Set Collections having given data
  Set<Long> phoneNumbers =new LinkedHashSet();
   phoneNumbers.add(999999999 ); phoneNumbers.add(8888888888);  phoneNumbers.add(7777777777);

  Set<Date> dates=new LinkedHashSet();
    dates.add(sysDate);  dates.add(dobDate);

 //create target class object
   Class   c=Class.forName("com.nt.beans.ContactsInfo");
    Constructor cons[]=c.getDeclaredConstructors();
    ContactsInfo  cInfo=(ContactsInfo)cons[0].newInstance(phoneNumbers,dates);
```

Injecting values to java.util.Map Collection

==========

use <map> with <entry> for injecting vlaues..

In <entry> tag
- =>for simple keys use either
  - <key> with <value> or key attribute
- =>for simple values use either <value> tag
  - or "value" attribute
- => for object type keys use either
  - <key> with <ref> tags or key-ref attribute
- =>for object type vlaues use either
  - <ref> tag or value-ref attribute

Map Collection

--> can maintain elements having key-values pairs
--> keys can not be duplicated , but values can be duplicated
-> Insertion order not preserved by default
To preserve use LinkedHashMap
-> keys can not be null... but values can be null
implementations are HashMap, HashTable ,LinkedHashMap and etc..
->key can be any object and value can be any object..

```
public class UniversityInfo {
	private Map<Integer,String>  facultyDetails;
	private Map<String,Date>  datesInfo;

	public void setFacultyDetails(Map<Integer, String> facultyDetails) {
		System.out.println(facultyDetails.getClass());
		this.facultyDetails = facultyDetails;
	}
	public void setDatesInfo(Map<String, Date> datesInfo) {
		this.datesInfo = datesInfo;
	}


	@Override
	public String toString() {
		return "UniversityInfo [facultyDetails=" + facultyDetails + ", datesInfo=" + datesInfo + "]";
	}
```

```
}
```

in applicationContext.xml

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

```
<!-- Map Collection-->
   <bean id="uInfo"  class="com.nt.beans.UniversityInfo">
      <property name="facultyDetails">
         <map  key-type="java.lang.Integer" value-type="java.lang.String">
           <entry>  <!-- Element 0 -->
              <key><value>1001</value></key>  <!-- key -->
              <value>rajesh</value> <!-- value -->
           </entry>
           <entry key="1002"  value="ramesh"  />
            <entry  key="1003"  value="suresh"/>
         </map>
      </property>
      <property name="datesInfo">
        <map key-type="java.lang.String" value-type="java.util.Date">
            <entry>
              <key><value>toDay</value></key>
              <ref bean="sysDate"/>
            </entry>
            <entry key="dob"  value-ref="dobDate"/>
        </map>
      </property>

   </bean>
```

Injecting values to java.util.Properties Collection

==

use <props> tags with <prop> tags..

private Properties fruitsInfo;

in applcationContext.xml

```
  <property name="fruitsInfo">
        <props>
          <prop key="banana">yellow</prop>
     <!--  <prop key=" key data"> value data </prop> -->
          <prop key="grapes">green</prop>
          <prop key="apple">red</prop>
          <prop key="mango">yellow</prop>
        </props>
     </property>
```

java.util.Properites

->It is map collection

->sub class of Hashtable

->Here Elements allow only Strings as keys and Strings as values.

-> No Generics is required here..

->Can load element values (keys, values) from external properties file (text file)

->Keys can not be duplicated ,but values can be

->insertion order not preserved..

Collection Injection in Realtime DI Application

==========

DriverManagerDataSource (spring api supplied pre-defined class)

|--->driverClassName
|--->url
|--->username
|----->password

* String type bean properties (simple bean properties)

|---> connectionProperties ** java.util.Properties (collection type

|

|--->allows to pass db username, db password having fixed keys "user", "password"

```
<bean id="oraDrds"
        class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName"          value="oracle.jdbc.driver.OracleDriver" />
        <property name="url"              value="jdbc:oracle:thin:@localhost:1521:xe" />
    <!--   <property name="username" value="system" />
        <property name="password" value="manager" /> -->
        <property name="connectionProperties"> <!-- java.util.Properties type -->
            <props>
              <prop key="user">system</prop>
              <prop key="password">manager1</prop>
            </props>
        </property>
    </bean>
```

Collection Injection in real time Applications..

if ** is used to supply db username, password ..there is no need of using * properties.

if both are used * properties values will override ** bean properties values..

Null Injection

==========

In constructor injection , all params must participate in Injection process otherwise exception will be raised..

=> if constructor param type is object/reference type and we are not ready with value.. then we can go for null injection i.e injecting null value constructor param.. to statisfy the process...

note:: This null injection is not possible if param type is primitive data type..

=>This is very handy(useful) , while working with pre-defined classes as spring bean.. that are having limited no.of overloaded constructors and no setter injection support is available...

<constructor-arg><null/> </constructor-arg>

In applicationContext.xml

==========

```
<bean id="raviInfo"  class="com.nt.beans.PersonInfo">
    <constructor-arg value="545345554"/>
    <constructor-arg value="ravi"/>
     <constructor-arg><null/></constructor-arg>
    <constructor-arg><null/></constructor-arg>
    <constructor-arg  ref="doj"/>
 </bean>
```

(Null Injection)

Note:: Do not look at spring's Dependency Injection to inject enduser supplied non-technical inputs (either using Scanner or using html from) to spring bean properties.. It is there to inject only programmer supplied technical input values (like jdbc properties) either from xml file or properties file and also inject one spring bean class object another spring bean class object (like DS to DAO ,DAO to Service and etc...)

=> To make gradle Spring Application collecting soruce code and xml files from the packages of src/main/java folder location.. add the fllowing code in build.gradle.

```
sourceSets {
   main {
      resources {
         srcDirs = ["src/main/java"]
         includes = ["**/*.xml"]
      }
   }
}
```

Can u we do null injection while working with settr injection?

Ans) yes , but not required becoz while working setter injection there is no mandatory to inject value/object to bean property.. we can invovle our choice properties in setter injecton, more over the reference type/object type bean proeprties by default holds null values..

```
<property name="dob"><null /> </property>
```

Bean Inheritence

==========

=> This is no way related to Java classes level inheritence..

=> It is xml file level inheritence across the spring bean configurations to reuse bean properties related values injection .

note:: Both setter injection and consturctor injection supports Bean Inheritence ..

Problem::

==========

```
public class Car{
   private String regNo;
    private String engineNo;
    private String model;
    private String company;
    private String type;
    private int engineCC;
    private String color;
    private String owner;
    private String fuelType;
    //setter methods for setter injection
     ....
     ....
    //toString()
     ....
}
```

=>In both spring bean cfgs "*" bean property values are same ,but we are not able to use them across the multiple spring bean cfgs of xml file..

applicationContext.xm

```
<beans ... >
  <bean id="rajaCar1" class="pkg.Car">
     <p name="regNo" value="TSO7EN8909"/>
     <p name="engineNo" value="5454335345"/>
     <p name="type" value="hatchback"/>
     <p name="model" value="swift"/> *
     <p name="color" value="red"/>
     <p name="engineCC" value="1200"/> *
     <p name="owner" value="raja"/> *
     <p name="company" value="suzuki"/> *
     <p name="fueltype" value="diesel"/>
  </bean>
  <bean id="rajaCar2" class="pkg.Car">
     <p name="regNo" value="TSO7EN8709"/>
     <p name="engineNo" value="5454335315"/>
     <p name="type" value="hatchback"/>
     <p name="model" value="swift"/>
     <p name="color" value="blue"/>
     <p name="engineCC" value="1200"/>
     <p name="owner" value="raja"/>
     <p name="company" value="suzuki"/>
     <p name="fueltype" value="diesel"/>
  </bean>
```

Solution1:: (Bean Inheritence)

applicationContext.xml

```
<beans ... >
   <bean id="rajaCar1" class="pkg.Car">
      <p name="regNo" value="TSO7EN8909"/>
      <p name="engineNo" value="5454335345"/>
      <p name="type" value="hatchback"/>
      <p name="model" value="swift"/>
      <p name="color" value="red"/>
      <p name="engineCC" value="1200"/>
      <p name="owner" value="raja"/>
      <p name="company" value="suzuki"/>
      <p name="fueltype" value="diesel"/>
   </bean>

   <bean id="rajaCar2" class="pkg.Car" parent="rajaCar1">
      <p name="regNo" value="TSO7EN8709"/>
      <p name="engineNo" value="5454335315"/>
      <p name="color" value="blue"/>
   </bean>
```

Solution2: (improved solution1)

applicationContext.xml

```
<beans ... >
   <bean id="baseCar" class="pkg.Car" abstract="true">
      <p name="type" value="hatchback"/>
      <p name="model" value="swift"/>
      <p name="engineCC" value="1200"/>
      <p name="owner" value="raja"/>
      <p name="company" value="suzuki"/>
      <p name="fueltype" value="diesel"/>
   </bean>

   <bean id="rajaCar2" class="pkg.Car" parent="baseCar">
      <p name="regNo" value="TSO7EN8709"/>
      <p name="engineNo" value="5454335315"/>
      <p name="color" value="blue"/>
   </bean>

   <bean id="rajaCar1" class="pkg.Car" parent="baseCar">
      <p name="regNo" value="TSO7EN8909"/>
      <p name="engineNo" value="5454335345"/>
      <p name="color" value="red"/>
   </bean>

</beans>
```

In client app

```
factory.getBean("baseCar",Car.class);  //gives error
Car car1=factory.getBean("rajaCar1",Car.class);  // success
 S.o.p(car1);  // 9 properties  (6 inherited from "baseCar" , 3 direct)
Car car2=factory.getBean("rajaCar2",Car.class);  // success
 S.o.p(car2);  // 9 properties  (6 inherited from "baseCar" , 3 direct)
```

important points

---

a) This is not class level inheritence.. it is Spring bean cfg file level bean properties inheritence across the multiple spring bean cfgs..

b) <bean >tag abstract="true" does not make java class as abstract .. but make spring bean cfg as abstract.. i.e we can not call factory.getBean(-) having that bean id .. we can inject this bean id based spring bean class object to other spring beans..

gives Exception in thread "main" org.springframework.beans.factory.BeanIsAbstractException: Error creating bean with name 'baseCar': Bean definition is abstract

c) One spring bean can inherit and reuse bean proeprties only from 1 spring bean ..

d) The class names in base bean cfg and child bean cfg can be the same or can be the different either partcipating in the inheritence or not particpating in the inheritece.

ctrl+F :: To and find and replace content...

ctrl+shift+f :: To fromat code..

ctrl+a and ctrl+i :: To select all content and to indentation..

Bean Inheritence can be used even through constructor Injection

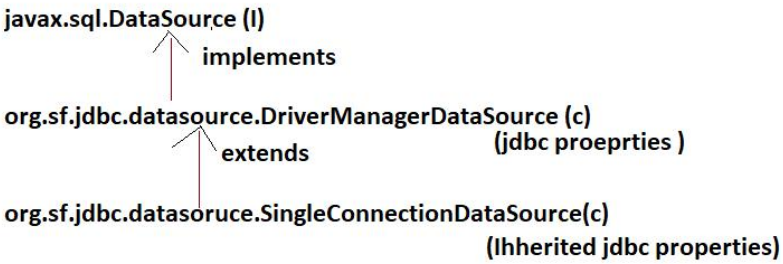
==========

```
<bean id="baseCar" class="com.nt.beans.Car" abstract="true">
        <constructor-arg name="engineCC" value="1500" />
        <constructor-arg name="model" value="swift" />
        <constructor-arg name="company" value="suzuki" />
        <constructor-arg name="fuelType" value="diesle" />
        <constructor-arg name="owner" value="raja" />
        <!-- <constructor-arg name="type" value="hatchback" />
        <constructor-arg name="type"><null/></constructor-arg:

    </bean>

    <bean id="rajaCar1" class="com.nt.beans.Car" parent="baseCar">
        <constructor-arg name="regNo" value="TS07EN4345" />
        <constructor-arg name="engineNo" value="12345" />
        <constructor-arg name="color" value="red" />
    </bean>

    <bean id="rajaCar2" class="com.nt.beans.Car" parent="baseCar">
        <constructor-arg name="regNo" value="TS07EN4344" />
        <constructor-arg name="engineNo" value="56789" />
        <constructor-arg name="color" value="white" />
        <constructor-arg name="owner" value="rani" />
    </bean>
```

In which situation we can pass less than available no.of bean property values for constructor injection?

Ans) a) while working with <null> injection

b) While working with constructor based Bean Inheritence



in applicationContext.xml

```
<!-- Configiure DataSource -->
    <bean id="oraDrds"    class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName"              value="oracle.jdbc.driver.OracleDriver" />
        <property name="url"        value="jdbc:oracle:thin:@localhost:1521:xe" />
        <property name="username" value="system" />
        <property name="password" value="manager" />
    </bean>
    <bean id="oraScds"  class="org.springframework.jdbc.datasource.SingleConnectionDataSource"  parent="oraDrds"/>
```

What is the jdbc con pool /dataSource that u used in u r Spring project?

=> Do not use DriverManagerDataSoruce becoz This class is not an actual connection pool; it does not actually pool Connections. It just serves as simple replacement for a full-blown connection pool, implementing the same standard interface, but creating new Connections on every call (ds.getConnection()).

=>Do not use SingleConnectionDataSource becoz it just creates only one jdbc con object and reuses that con object.. reusing single jdbc con object across the multiple requests coming to a web application raises Data inconsistency problems.. like if any requests calls con.rollback() then the simultaenous requests related persistence operations will also be rolledback.. (This is not suitable in multi-threaded standalone , web application env..)

note:: Both the above DataSources are not providing con pool parameters like initialPoolSize ,maxPoolSize and etc...

note:: if u r spring project is standalone Project then use third party standalone jdbc con pool s/w like apache DBCP , C3P0, Proxool , Vibur ,Hikari CP(best) and etc..

note:: if u r spring project is web application/website then use underlying server managed jdbc con pool .. like weblogic managed jdbc con pool , Tomcat managed jdbc con pool and etc...

HikariCP Details

==========

DataSoruce class name:: com.zaxxer.hikari.HikariDataSource

jar file ::: hikariCP-<ver>.jar

Gradle dependencies entry :: // https://mvnrepository.com/artifact/com.zaxxer/HikariCP

implementation group: 'com.zaxxer', name: 'HikariCP', version: '3.4.5'

```
<!-- Configiure DataSource -->
    <bean id="oraDrds"    class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName"              value="oracle.jdbc.driver.OracleDriver" />
        <!-- <property name="url"                  value="jdbc:oracle:thin:@localhost:1521:xe"  /> -->
        <property name="username" value="system" />
        <property name="password" value="manager" />
    </bean>

    <bean id="oraScds"  class="org.springframework.jdbc.datasource.SingleConnectionDataSource"  parent="oraDrds"/>

    <bean id="oraHkDs"  class="com.zaxxer.hikari.HikariDataSource" parent="oraDrds">
        <property name="jdbcUrl" value="jdbc:oracle:thin:@localhost:1521:xe"/>
        <property name="minimumIdle" value="10"/>  <!-- min pool size -->
        <property name="maximumPoolSize" value="20"/>
        <property name="connectionTimeout"  value="2000"/>
    </bean>
```

note:: In real time developers directly configures HikariDatasource with all its properties... becoz that is the best DataSource in standalone env...

Collection Merging

Collection Merging

==========

=> It should be used along with Bean Inheritence

=> It can be applied only on collection/arry type Bean properties..

=> It is all about adding more values collection/array type property in the child bean cfg by inheriting it from parent bean cfg..

```
public class EnggCourse{
   private  Set<String>  subjects;

    public   void  setSubjects(Set<String> subjects)
       this.subjects=subjects;
     }

     //toString()
       ....
  }
```

in applicationContext.xml

==========

```
<beans ...>

   <bean id="baseIstYear"  class="com.nt.EnggCourse"  abstract="true">
        <property name="subjects">
            <set>
              <value>C </value>
              <value>M1 </value>
              <value>English </value>
              <value>Drawing </value>
             </set>
           </property>
      </bean>

      <bean  id="ecIstYear"  class="pkg.EnggCourse"  parent="base1stYear">
           <property name="subjects">
              <set   merge="true">
                <value> EDC </value>
                <value>NT </value>
              </set>
          </property>
      </bean>

   </beans>
```

mergu attribute make to mergu bean property from spring cfgs..mergu="'faluse" (Default)makes override bean property

Client App

==========

```
EnggCourse engg=factory.getBean("ec1stYear",EnggCourse.class);
 S.o.p(engg);
```

gives 4+2 = 6 subjects...

Collection Merging important points

=> Base/parent bean collectin/array property name and type must match with
Child bean collection/array property name and type..

=> collection merging is possible only with collection,array bean properties .. not on
simple , object type bean properties..

=> "merge" attribute is available only in <list>,<set>,<map>,<props>, <array> tags..

=> The possible values for "merge" attribute are

a) false
b) true
c) default (default)

=>Both setter ,constructor injection supports the "collection merging".

merge="true" enables collection merging
merge="false" disables collection merging
merge="default" (default) --> fallbacks to "default-merge" attribute value of <beans> tag.

the possible values for default-merge attribute are

<beans default-merge="true" > -->enables collection merging on all collection properties of all spring beans belonging to current spring bean cfg file

<beans default-merge="false" > -->disables collection merging on all collection properties of all spring beans belonging to current spring bean cfg file

<beans default-merge="default" > The default is "default", indicating inheritance from outer 'bean' sections in case of nesting, otherwise falling back to "false".

Q) Can we perform collection merging with out taking merge="true" ?

Ans) possible but we need write default-merge="true" in <beans> tag (at top of spring bean cfg file)

Default Bean Ids

==========

=>The IOC container generates default bean id for spring bean class ... if we do not provide any bean id to it..

Generally that is <fully qualified classname>

```
<bean class="com.nt.beans.EnggCourse ">
    ....
    ....
</bean>
```

The default bean id :: "com.nt.beans.EnggCourse" (or) "com.nt.beans.EnggCourse#0"

---

in applicationContext.xml

```
<bean class="com.nt.beans.EnggCourse">
    ....
    ....
    ....
</bean>
```

default bean id :: "com.nt.beans.EnggCourse" (or) "com.nt.beans.EnggCourse#0"

```
<bean class="com.nt.beans.EnggCourse">
    ....
    ....
    ....
</bean>
```

default bean id :: "com.nt.beans.EnggCourse#1"

```
<bean class="com.nt.beans.EnggCourse">
    ....
    ....
    ....
</bean>
```

"com.nt.beans.EnggCourse#2"

default bean id syntax is :: <pkg>.<classname>#<n>

n is "O" based index..

Q) Can i have same bean id for two different spring beans cfg with in a IOC container?

ans) no , the bean ids must be unique with in the IOC container

Q) can i have same class names for diff spring bean cfgs ?

Ans) yes --- u must cfg them with diff unique bean ids..

inner beans

=> It is no way related to inner classes of java ...
=> It is all about configuring one spring bean inside another spring bean cfg..
=> for this, we need to place <bean> tag as the sub tag of <property> or <constructor-arg> tags of another <bean> (indirectly writing <bean> tag under another <bean> tag)

=> if want cfg bean as the dependent bean only for 1 target bean cfg then cfg it as inner bean to that target bean cfg ..
=> if want cfg bean as the dependent bean for multiple target bean cfgs either having same class names or different class names then cfg it as normal bean to inject to multiple target beans..

=> Cricket Bat to PRoffesional Crickter should be cfg as inner bean
=> Cricket Bat to Normal Crickter should be cfg as normal bean
=> Cricket Ball to Crickter should be cfg as normal bean
=> CarKey to Car should be cfg as inner bean
=>BankAccountDetails to Customer should be cfg as inner bean
=>DataSource to DAO should be cfg as normal bean
=> DAO to Service should be cfg as normal bean
=> Service to controller can be cfg as inner bean becoz every project contains only controller

```
<bean id="kohli" class="com.nt.beans.ProffesionalCrickter"> <!-- outer bean cfg -->
    <constructor-arg value="virat kohli"/>
    <constructor-arg>
       <bean id="bat1" class="com.nt.beans.CricketBat"/> <!-- inner bean cfg-->
    </constructor-arg>
</bean>
```

optional (in fact not required)

important points

=> giving bean id to inner bean cfg is optional (not required) becoz inner bean can not be injected to other than current outer bean and can not be accessed by calling factory.getBean(-) from Client app..
=> In Target class we can take both normal beans, inner beans based dependency injection..
=> One InnerBean can have other inner beans injection..
=> We can not access inner directly.. it should be accessed only through outer bean ...

Can we inject both normal bean and inner bean to a property of taget bean class ?
Ans) No .. becoz <property> ,<constructor-arg> tags can have only ref attribute or "value" attribute or sub tag (not multiple at a time).. This XSD /DTD level restriction ...

With respect to realtime DI

==========

```
<!-- cfg Controller class -->
    <bean id="controller" class="com.nt.controller.MainController">
        <constructor-arg >
            <!-- cfg service class as inner bean-->
            <bean       class="com.nt.service.CustomerMgmtServiceImpl">
                <!-- <constructor-arg ref="mysqlCustDAO" /> -->
                <constructor-arg ref="oraCustDAO"/>
            </bean>
        </constructor-arg>
    </bean>
```

Q)if we configure same class as multiple spring beans with different bean ids or no bean ids then can tell me how objects will be created for class.. when start using all the bean cfgs..?

Ans) Multiple objects will be created on 1 per bean id /default bean id basis.

```
<bean id="engg1" class="pkg.EnggCourse">
  ...
</bean>
<bean id="engg2" class="pkg.EnggCourse">
  ...
</bean>
```

if we call both factory.getBean(-,-) having both both bean ids.. then IOC Container creates two objects for EnggCourse class..

internal cache of IOC container

engg1	EnggCourse object
engg2	EnggCourse object

```
EnggCourse e1=factory.getBean("engg1",EnggCourse.class);
EnggCourse e2=factory.getBean("engg1",EnggCourse.class);
```

here e1,e2 refers to same object of EnggCourse class.

```
EnggCourse e1=factory.getBean("engg1",EnggCourse.class);
EnggCourse e2 =factory.getBean("engg2",EnggCourse.class);
EnggCourse e3=factory.getBean("engg2",EnggCourse.class);
```

Creates 2 objs for EnggCourse class..

IOC container creates and reuse bean object by creating it based on per class and per id..

Bean Alias

=========

=> providing nick names to bean id ....

=> Useful when bean id is very lengthy to refer in multiple places...

=> Generally there is a practice of taking class name as the bean id ... which is quite lengthy.. To give short names/nicknames to that bean id we should go for alias names..

```
<bean id="wishMessageGenerator"    name="wmg,wmg1"
            class="com.nt.beans.WishMessageGenerator"/>

 <alias name="wishMessageGenerator"  alias="msg1"/>
 <alias name="wmg"  alias="msg2"/>
 <alias name="msg2"  alias="msg3"/>
```

=>Upto spring 2.x , we can use only "name" attribute , from spring 3.x we can we can use both "name" attribute and <alias> tag to provide alias names/nick names...

=> Using one "name" we can provide multiple alias names.. at a time ..

=>Using one "<alias>" tag we can provide only one alias name/nick name at a time

=> we can also provide alias names to default bean ids..

=> we can provide alias names to alias name it self..

Dependency Lookup

=>Here Target class writes logics to search and get Dependent class object.. by spending some time..

Q)When should we go for dependency lookup and when should we go dependency injection?

=> if the dependent class obj is required only in the one method of target class then go for dependency lookup ...

=> if the dependent class object is required in multiple methods of target class then go for Dependency injection (setter injection and constructor injection)

Crickter(target) needs cricketBall(Dependent) , Bat(Dependent) , keeping Gloves(dependent)

|--->fielding()

|-->Batting()

|--->bolwing()

|--->keeping()

Since Cricket ball is required in all the 4 methods go for dependency injection

since Bat is required only in batting() method go for dependenyc lookup

Since keeping gloves are required only in keeping() go for dependency lookup..

note:: To perform dependency lookup .. create extra IOC container in the target class method and call factory.getBean(-) method.. having bean id of dependent bean to get Dependent Bean class object..

Vehicle (target class) needs Engine (dependent class)
|--> jounery(srcPlace,destPlace)
|---> entertainment()
|---> soundHorn()
|--->fillingFuel()

Here Engine is required only in one method of Target class (journey(-,-)) ,So go for Dependency lookup.

can

note:: In target class we place code supporting both Dependency lookup and injection..

```
<bean id="engg1" class="com.nt.beans.Engine"/>

<!-- target bean class -->
<bean id="vehicle" class="com.nt.beans.Vehicle">
  <!-- <constructor-arg value="engg2"/> --> <!-- Injecting bean id -->
  <constructor-arg>
     <idref bean="engg2"/>
  </constructor-arg>
</bean>
```

What is the difference b/w injecting beanid using <value> or "value" attribute and using <idref> tag?

Ans) <value> or "value" attribute injects the bean id as an ordinary string value.. with out checking the having that bean id spring bean is cfg or not .. So exception comes lately when that wrongly injected spring id is used in the target class.. (not recomanded)

```
<property name="beanId" value="engg"/>
      (or)                          dependent class bean id
<constructor name="beanId" value="engg"/>
```

=>we can use <idref> tag to inject the bean id .. it will check having bean id is there any spring bean configuration before injecting bean id to target class.. if not there it will throw exception.. i.e problem is detected very early...

```
<property name="beanId">
  <idref bean="engg"/>
</property>
  (or)
<constructor-arg name="beanId">
  <idref bean="engg"/>
</constructor-arg>
```

conclusion:: To Inject bean id to spring bean prefer using <idref> tag..

```
String s="java.util.Date";
Class c=java.util.Date.class;
```

Disadvantages of Traditional Dependnecy lookup

==========

(a) Taking extra IOC cotainer in the specific method of Target class is bad practice.
(b) The injected bean id of dependent class to the target class is visible in all the methods of target class though it is required in only in one method.

note:: The Soluton for above problems is Aware Injection or Lookup method Injection
(Both these are extension to Tradional dependency lookup)

AutoWiring
==========

Assiging dependent class object to target class object is called Dependency Injection or bean wiring..

Explicit wiring/ Manual Injection

Here we use <property> or <constructor-arg> to inject dependent value/objects to target class

(So far we are working on this)

Autowiring/ Auto Injection

=>Here that IOC container automatically detects and injects Dependent spring bean calss objs to target spring bean class object..

=>use "autowire" attribute of <bean> for this..

=> no need of writing <property> , <constructor-arg> tags here..

Limitations of autowiring
==========

(a) Autowiring is possible only on object type/ referene type bean properties .. not possible on simple ,array , collection type bean properties..

(b) There is a possibility of getting ambiguity Problem (It will confuse to choose one of the multiple dependents)

(c) kills the readability of spring bean cfg file..

note:: Autowirning is very useful .. for Rapid application devlelopement (RAD) -->Faster Developement..

Different modes of Autowiring
==========

(a) byName
(b) byType
(c) constructor
(d) autodetect (removed from spring 3.x)

autowire="byName"

==========

=>performs setter Injection

=> Detects/finds dependent spring bean class object based its bean id that is matching with target class proeprty name(variable name)

=> There is no possiblity of getting Ambiguity problem becoz the bean ids in IOC container are unique ids..

Flipart (target class)

|-->private Courier courier;

|-->p v setCourier(Couier courer){ this.courier=courier;}

DTDC implements Courier

BlueDart implements Courier

example

==========

```
<!--dependent beans cfgs -->
<bean id="courier" class="pkg.DTDC"/>
<bean id="courier1" class="pkg.BlueDart"/>

<bean id="fpkt" class="pkg.Flipkart" autowire="byName"/>
```

Here injects DTDC class object to the courier property of Flipkart class becoz the DTDC class bean id(courier) is matching with "courier" property name of Flipkart class..

autowire="byType"

==========

=>Performs setter injection

=> Target Bean class proeprty /variable type and Depednet spring bean type(Dependent class name or type) must match.

=>There is a possiblity of getting ambigutity Problem.. and we can solve it by using primary=-"true" in one of the dependent spring beans cfg..

throws

org.springframework.beans.factory.NoUniqueBeanDefinitionException for ambiguity problem

example

========

```
<!-- cfg all dependent classes as spring beans -->
    <bean id="dtdc" class="com.nt.comp.DTDC"/>
     <bean id="bDart" class="com.nt.comp.BlueDart" />
    <bean id="fFlight" class="com.nt.comp.FirstFlight" primary="true"/>

 <!-- Cft Target class as spring bean -->
 <bean id="fpkt" class="com.nt.comp.Flipkart" autowire="byType"/>
```

autowire="constructor"

==========

=>Performs constructor injection by using parameterized constructor

=> Here Constructor param type and Dependent bean class type must match in order to use that parameter constructor for autowiring

=> There is possiblity of getting ambiguity Problem and it can be solved in two ways

(a) by matching dependent spring bean class id with contsructor param name

(b) by keeping primary="true" in one of multiple possbible dependent spring bean classes cfg..

note:: if both are placed in two different possible dependent spring beans.. then primary="true" gets high priority.

```
<bean id="courier" class="com.nt.comp.DTDC" />
<bean id="bDart" class="com.nt.comp.BlueDart" primary="true" />
<bean id="fFlight" class="com.nt.comp.FirstFlight" />

<!-- Cft Target class as spring bean -->
<bean id="fpkt" class="com.nt.comp.Flipkart" autowire="constructor"/>
```

autowire="autodetect"

=> removed from spring 3.x , becoz it killing more and more redability

=> if spring bean is having 0-param constructor directly or indirectly then it goes for "byType" mode of autowiring .. if not there then it goes for "constructor" mode autowiring..

=> To work with this mode auto wiring chanage spring version to 4.x/3.x and Xsd version to 2.5/2.0

=>There is possibility of getting ambiguity problem and we resolve it by using the same solutions of "byType" , "constructor" mode auto wirings..

```
<!-- cfg all dependent classes as spring beans -->
    <bean id="courier" class="com.nt.comp.DTDC" />
     <bean id="bDart" class="com.nt.comp.BlueDart"  />
    <bean id="fFlight" class="com.nt.comp.FirstFlight" primary="true" />

  <!-- Cft Target class as spring bean -->
  <bean id="fpkt" class="com.nt.comp.Flipkart" autowire="autodetect"/>
```

"autowire" attribute possible values

---

byType
byName
constructor
autodetect (removed from spring 3.x)
default (default)
no

autowire="deafault" fallbacks to default autowire mode that is specified in
========== "default-autowire" attribute of <beans> tags..

<beans default-autowire="constructor" ... >

Enables constructor mode auto wiring on all
the spring beans of current spring bean cfg file

autowire="no" Disables auto wiring on certain spring bean cfg .. even though default autowiring is enabled

```
<beans   default-autowire="byType"  .....>
    ....
    ....
   <!-- Cft Target class  as spring bean -->
    <bean id="fpkt"  class="com.nt.comp.Flipkart"  autowire="no"  />

    <bean id="fpkt1"  class="com.nt.comp.Flipkart"  autowire="byName" />
```

The possible values of "default-autowire" attribute of <beans> tag are
byName
byType
constructor
no
default (default)

<beans default-autowire="no" .....>

Disables autowiring on all spring beans that are there in current spring bean cfg file

<beans default-autowire="default" ...>

In case of nested beans/inner beans the outer bean autowire mode will be applied where as in case of normal beans it is equal to default-autowire="no" value..

Can we enable autowring on spring bean without using autowire attribute?

Ans) yes ,, by specifying autowire mode in "default-autowire" attribute of <beans> tag.. (other than no, default values should specified)

Can we overide default autowire mode ?

Ans) yes ... use "autowire" attribute in every <bean> keeping otherthan "default" value

Making Spring bean autowire candidate

In order to make certain dependent spring beans not participating in autowiring .. we can disable them as autowire condidates by using autowire-candidate="false" ..This is one solution to solve ambguity problem that comes with byType,constructor mode of autowiring with out using primary="true"...

(Still primary="true" is the best solution to solve ambiguity problem)

```
<!-- cfg all dependent classes as spring beans -->
    <bean id="dtdc" class="com.nt.comp.DTDC" />
     <bean id="bDart" class="com.nt.comp.BlueDart" autowire-candidate="false" />
    <bean id="fFlight" class="com.nt.comp.FirstFlight" autowire-candidate="false" />

  <!-- Cft Target class as spring bean -->
  <bean id="fpkt" class="com.nt.comp.Flipkart" autowire="byType" />
```

```
<bean id="dtdc" class="com.nt.comp.DTDC" autowire-candidate="false" primary="true"/>
```

(Wrong combo becoz after making sprng bean as non autowire condidate there is no meaning of keeping primary="true")

The possible values for "autowire-candiate" attribute are

a)true
b)false
c) default (default)

autowire-candidate="true" makes spring bean to participate in autowiring

autowire-candidate="false" makes sprign bean not to participate in autowiring

autowire-candidate="default" :: fallbacks to default-autowire-candidates attribute of <beans> tag.. i.e if the current bean id is there in the list if bean ids that are specified in "default-autowire-condidates " attribute of <beans> tag then it will act as autowire candidate otherwise it will not participate in autowiring..

```
<beans default-autowire-candidates="dtdc,bDart" .... ..... >
```

if we do not specify this attribute in <beans>... then all spring beans are autowire candidates..

(No default values for this attribute)

What is the diff b/w autowire="no" and autowire-candidate="false"? (No default values for this attribute)

Ans) autowire="no" make IOC container not perform any autowiring based injections to current spring bean by considering it as target spring bean.

autowire-candidate="false" makes IOC container not to consider current spring bean as dependent to inject to other beans..

usecase:: if want suspend certain bean as dependent bean to participate in autwiring for temporary period then go for this autowire-candidate="false" option..

note:: The dependent bean which is disabled as autowire-candidate .. can be used as dependent through explicit wiring /manual wiring concept..

```
<bean id="bDart" class="com.nt.comp.BlueDart" autowire-candidate="false" />

<bean id="fpkt" class="com.nt.comp.Flipkart" autowire="byType" >
  <property name="courier" ref="bDart"/>
</bean>
```

What is happens , if we enable both autowiring and explicit wiring on the bean property of target class?

Ans) if both explicit wiring and autowiring are performing same setter injection or same constructor injection the explicit wiring takes place..

if one wiring performs setter injection and another wiring performs constructor injections then setter injection values will be taken as final values...

```
<!-- cfg all dependent classes as spring beans -->
  <bean id="dtdc" class="com.nt.comp.DTDC" />
   <bean id="bDart" class="com.nt.comp.BlueDart" />
  <bean id="fFlight" class="com.nt.comp.FirstFlight" primary="true" />

<!-- Cft Target class as spring bean -->
<bean id="fpkt" class="com.nt.comp.Flipkart" autowire="byType" >
   <constructor-arg ref="bDart"/>   explicit wiring          autowiring
</bean>
```

---

note: we can enable both autowiring and explicit wiring on different bean proeprties of spring bean class .. i.e one few propertes we can enable explicit wiring .. on few other properties we can enable autowiring..

Assingnments

Q)what happens if we place only private constructor in spring bean class?

A) Still IOC container creates Spring Bean class object by accessing private constructor of spring bean class using reflection api .. Internally calls setAccssible(true) method on java.lang.reflect.Constructor class object that reprsents private constructor.

Q) can we take spring bean class as private class?

Ans) In java itself we can not take outer classes as private classes.. So
In spring also normal spring bean classes can not be taken as private class.

```
private class Test{ //wrong..
}
```

<bean id="t" class="pkg.Test"/> (error)

In java we can take inner class as private class .. So we cfg that inner class as spring bean specifying outer class name.. then the spring bean can be private class..

```
public class Test{

  private static class ABC{
  }
}
```

<bean id="t1" class="pkg.Test.ABC"/> (valid)

(valid)

IOC container does load classes from .java files.. It Load the classes from .class files..

Reflection API Code getting access to Private Constructors

```
package com.nt.test;

import java.lang.reflect.Constructor;
import com.nt.beans.TestBean;

public class PrivateConstructorAccessTest {

	public static void main(String[] args) {
		Class c=null;
		Constructor cons[]=null;
		TestBean tb1=null,tb2=null;
		try {
			//Load the class
			c=Class.forName("com.nt.beans.TestBean");
			//get Access to all the cosntructors of the class..
			cons=c.getDeclaredConstructors();
			//create objects
			cons[0].setAccessible(true);
			tb1=(TestBean)cons[0].newInstance();
			System.out.println(tb1);
			System.out.println("-------------------------------");
			cons[1].setAccessible(true);
			tb2=(TestBean)cons[1].newInstance(10,20);
			System.out.println(tb2);
		}//try
		catch(Exception e) {
			e.printStackTrace();
		}

	}

}
```

IOC container can create spring bean class object

a) Using O-param constructor
[if spring bean class is cfg with no injection or only with setter injection]

b) Using parameterized cosntructor
[if spring bean calss is cfg by enabling constructor injection]

c) Using static factory method
[if we configure "factory-method" attribute in <bean> tag]

d) Using instance factory method
[if we configure "factory-bean", "factory-method" attributes in <bean> tag]

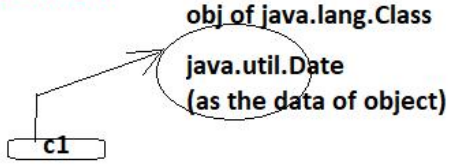
note:: if we enable factory method bean instantation .. we can specify the factory method names using "factory-method" attribute.. of <bean> tag .. and we can pass argument values to factory methods using <constructor-arg> tags..

in applicationContext.xml

eg:::
```
<bean id="c1" class="java.lang.Class" factory-method="forName">
    <constructor-arg value="java.util.Date"/>
</bean>
```

here <constructor-arg> is used to pass arg value to forName(-) method call.

internal code is
Class c1=Class.forName("java.util.Date");



Internal Cache of IOC cotainer

key	value
c1	java.lang.Class obj ref

Class c1=new Class("java.util.Date");
(Wrong guess of internal code becoz java.lang.Class is having only one private 0--param constructor.. So it can not be instantiated using new operator out side of its calss)

In client app
```
============
Class   c=factory.getBean("c1", Class.class);
```

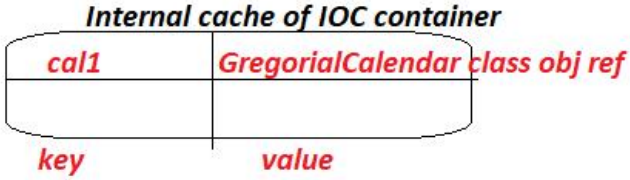
Q) Can u configure abstract class /interface as the spring bean in the spring bean cfg file?

Ans) Possible .. but we must enable static factory-method bean Instantation..In this situtation IOC container does not create object for the configured abstract class or interface .. It will create object for the sub class /impl class ...

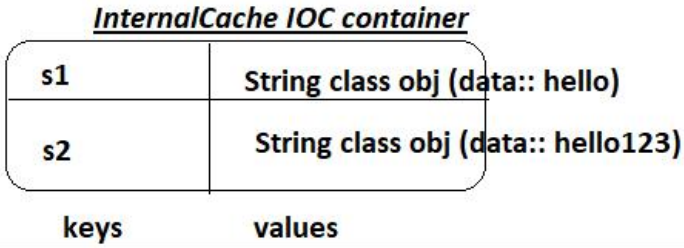
Q) Can u configure abstract class /interface as the spring bean in the spring bean cfg file?

Ans) Possible .. but we must enable static factory-method bean Instantation..In this situtation IOC container does not create object for the configured abstract class or interface .. It will create object for the sub class /impl class ...

```
<!-- Static factory bean instanation giving related class object -->
    <bean id="cal1" class="java.util.Calendar" factory-method="getInstance"/>
```



```
<!-- Instance factotry method bean instantiation giving same class object -->
    <bean id="s1" class="java.lang.String" >
      <constructor-arg value="hello"/> <!-- for construcotr injection -->
    </bean>
    <bean id="s2" factory-bean="s1" factory-method="concat">
       <constructor-arg value="123"/> <!-- for concat(-) method arg values -->
    </bean>
```



```
<!-- Instance factotry method bean instantiation giving unrelated class object -->
    <bean id="sb" class="java.lang.StringBuffer" >
      <constructor-arg value="hello how are u?"/> <!-- for construcotr injection -->
    </bean>
    <bean id="s3" factory-bean="sb" factory-method="substring">
       <constructor-arg value="0"/>
       <constructor-arg value="5"/>
    </bean>
```

note:: In Instance factory method bean instatiation specifying "class" attribute haivng spring bean calss name is optional ...

Singleton class / singleton java class

==========

=> The java class that allows us to crete only one object.. in the entire execution of the App is called singleton java class...

=> It is GOF Pattern..

=>pre-defined singleton classes :: java.awt.Desktop , java.lang.Runtime

Problem:: creating multiple objects for java class when class is not having state or class having read only state(final) or sharable state is waste of memory and cpu time..

Solution :: In the above situations create only one object for java class and use it for multiple times i.e we need restrict java class allows the programmers to create only one object.. by making it as singleton java class..

implementation ::: While developing singleton java class close all the doors of creating object for java class from outside of the class but open only one door (static factory method) where u can check for object is available or not before creating and returning object..

```
//Singleton class with minimum standards
public class Printer{
 private  static  Printer  INSTANCE;

 privat e Printer(){
   //no body
  }

 //static factory method
 public static Printer  getInstance(){
   if(INSTANCE==null)
        INSTANCE=new Printer();
   return INSTANCE;
  }
 //b. method
 public void  printData(String msg){
    S.o.p(msg
  }
}
```

a) Take class as public class to make it visible inside and outside of current package.

b) Add private only constructors in the java class to stop outsiders to create object by using new operator

c) Take private static reference variable as the memeber variable of the class to hold/refer the single object that will be created , So we can we can use this ref variable in static factory method as criteria to create and return new obj ref or to return existring object ref

d)public Static factory method , that checks the object availablility .. if avaliable returns that object ref otherwise creates new object

We can break above singleton class using

a) multi-threaded env..
b) reflection api
c) using deserialziation
d) using cloning
e) Using Custom ClassLoaders

we can solve
all these problems
using different techniques..

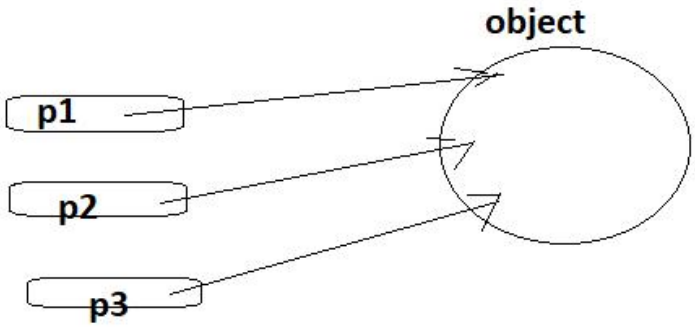
---

note:: By Developing singleton class as Enum .. we can solve max of the above problems.. implicitly

Enums are internally classes..

```
// Enum based singleton
public enum Printer{
   INSTANCE      //constant of current Enum type

   public void printData(String msg){
     S.o.p(msg);
   }
}
```

```
Printer p1=Printer.INSTANCE;
Printer p2=Printer.INSTANCE;
Printer p3=Printer.INSTANCE;
```



spring core -basic
spring core -advanced
spring jdbc
spring aop | pre-req: core-basi
spring TxMgmt | pre-req:core b
spring ORM | pre-req: corebasic+ hiberr (3-4)
spring data | pre-req: corebasic+ hibe
spring web mvc | pre-req: corebasic+
spring security
spring Batch
spring mail (spring context/JEE)
spring social
spring oauth

4 modes of spring programming
a)Xml driven cfgs
b) Annotation driven cfgs
c) 100% code driven cfgs
d) spring boot driven cfgs

Q)When should we take class as singleton class ?

Ans1) if class is not having any state (No member variables declaration)

```
eg:: public class Arithmetic{
       public int sum(int x,int y){
         return x+y;
       }
     }
```

(bad)

⇩

Creating multiple objs for a class with no state is bad practice

```
public class Arithmetic{
  private static Arithmetic INSTANCE;
  private Arithmetic(){ }
  //factory method
  public static Arithmetic getInstnace(){
      if(INSTANCE==null)
        INSTNACE=new Arithmetic();
      return INSTANCE;
    }
  //b.method
  public int sum(int x,int y){
     return x+y;
  }
}
```

(good)

```
public enum Arithmetic{
    INSTANCE;
  //b.method
  public int sum(int x,int y){
      return x+y;
  }
}
```

(good)

1. Dynamic Servlet Registration @ 9:00 AM (IST) on 16th August
   (pre-requisite :: servlet)

Ans2) when class is read only state (final variables)

```
public class Circle{
  private float final PI=3.14f;

  public float calcArea(float radius){
     return PI*radius*radius;
   }
}
```

(Bad)

creating multiple objects of a class having fixed state is bad practice..

```
public class Circle{
  private final float PI=3.14f;
  private static Circle
        INSTANCE=new Circle(); //eager instantiation
  private Circle(){                    (not recomanded)
   }
  //factory method
  public static Circle getInstance(){
      return INSTANCE;
  }
  // b.method
  public float calcArea(float radius){
     return PI*radius*radius;
   }

}
```

(Good)

```
public enum Circle{
   INSTANCE;
   private final float PI=3.14f;
   //b.method
   public float calcArea(float radius){
      return PI*radius*radius;
    }

}
```

(Good)

(Good)

Ans3) When class is having sharable state (non-final) across the multiple classes other class of the App/Project in Synchronized env.. (eg:: Cache/buffer)

⇩

(To avoid multi threading issues)

Sample code ::: we will discuss in AOP clases..

IOC Container can create and manage the objects of spring bean class having diff scopes. they are

spring 1.x	spring 2.x/3.x/4.x		spring 5.x	
singleton (default)	singleton (default)		singleton(default)	
prototype	prototype		prototype	
	request		request	
	session	(only in web applications)	session	(only in web applicaitons)
	globalSession (removed from 3.x)		application	
			websocket	

=>use "scope" attribute of <bean> tag to specify spring bean scope

Spring ORM (for 6pm batch students)
pre-requisites:: Spring core , hibernate
only today :: u join by 6.45 pm
tommorrows onwords join by 6pm

scope="singleton"
==========

=>It is default scope , if no scope is specified
=> IOC container creates only one object for spring bean class and reuses that object
=> IOC container does not make spring bean class as singleton java class.. But it creates only one object , keeps that object in Internal Cache of IOC container and uses that object for mutliple times i.e returns that one object ref by collecting from Internal cache for multiple factory.getBean(-) method calls..

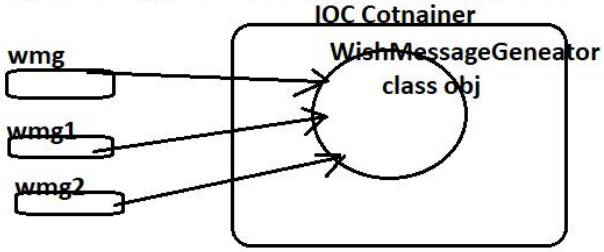
```
<bean id="wmg"  class="pkg.WishMessageGenerator" scope="singleton"/>
```

Internal Cache of IOC container

wmg	WishMessageGenerator class obj ref

In clientapp
==========

```
WishMessageGenerator wmg=factory.getBean("wmg",WishMessageGenerator.class);
WishMessageGenerator wmg1=factory.getBean("wmg",WishMessageGenerator.class);
WishMessageGenerator wmg2=factory.getBean("wmg",WishMessageGenerator.class);
```



note:: IOC container creates only object for each singleton scope sprign bean class cfg..

applicationContext.xml

```
<bean id="wmg" class="com.nt.beans.WishMessageGenerator" scope="singleton">

<bean id="wmg1" class="com.nt.beans.WishMessageGenerator" scope="singleton" >
```

(This proves that the "singleton" scope does not make spring bean class as singleton class..)

Internal Cache of IOC container

wmg	WishMessageGenerator class obj ref
wmg1	WishMessageGenerator class obj ref

note:: With in a IOC container , the bean ids must be unique .. but same class we can cfg as multiple spring beans...with diff bean ids..

scope="prototype"

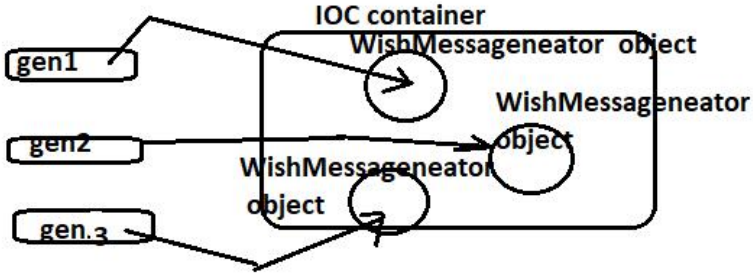
=> IOC container creates new Object for Spring bean class for every factory.getBean(-) method call

=> IOC container does not keep this scope spring bean class objs in the "Internal Cache of IOC container"

eg:: <bean id="wmg" class="pkg.WishMessageGenerator" scope="prototype"/>

Client App

```
WishMessageGenerator gen1=factory.geTBean("wmg",WishMessageGenerator.class);
WishMessageGenerator gen2=factory.geTBean("wmg",WishMessageGenerator.class);
WishMessageGenerator gen3=factory.geTBean("wmg",WishMessageGenerator.class);
```



note::
if spring bean class not having state or having only read-only state or sharable state across the other classes then go for cfg that java class as singleton scope spring bean class.

use
When should we singleton scope and prototype scope in real time projects?

Ans) In Realtime layered Applications , we configure DataSource class , DAO classes , Service classes and Controller class as singleton scope spring beans. Reasons are

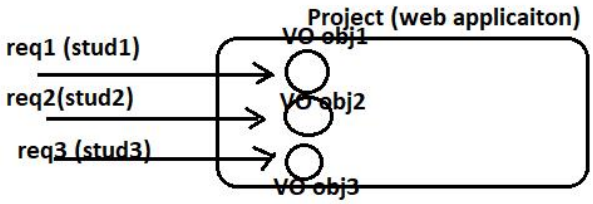
=>The State kept DataSource obj(jdbc properties, con pool properties) are sharable across the multiple DAO classes .. More every multiple DAOs want to use same DataSource object (indirectly same jdbc con pool) to interact with a DB s/w .. So we need to take DataSource class as singleton scope spring bean..

=> DAO class contains SQL queries at the top of class as final String variables values (readonly) and also DataSource object (having fixed state) So we can take DAO as singleton scope spring bean..

=>Service classes contain DAO class objects as state (indirectly fixed state) So we can take service classes as singleton scope spring beans.

=>Controller class is 1 per Project contains service class objs as the state(indirectly fixed state) So we can take take controller class as singlton scope spring bean.

=>We can take VO,BO ,DTO classes (java beans) as prototype scope spring beans , if we want to configure them as spring beans.. Becoz they need to hold multiple sets of inputs simultaenously..
(specically in web applications)



Q)What happens if we configure real singleton java class as spring bean having prototype scope?

Ans1) if we do not enable static factory method instantiation, the IOC container creates multiple objects by accessing private 0-param constructor for multiple factory.getBean(-) method method calls..

```
<bean id="p1" class="com.nt.sdp.Printer"  scope="prototype" >
```

Ans2) if we enable static factory method Bean Instantiation , then the IOC container creates only one object for multiple factory.getBean(-) method calls .

```
<!-- Singleton java class  as prototype scope spring bean -->
    <bean id="p1" class="com.nt.sdp.Printer"  scope="prototype"  factory-method="getInstance"/>
```

scope="request" *(for http protocol)*

==========

=>IOC container keeps Spring bean class object as request attribute of web applicaiton env.. i.e Spring bean class object will be created on 1 per request

note: Here spring bean class object will be maintained as request attribute

scope="session" *(http protocol)*

==========

=>IOC container keeps Spring bean class object as session attribute of web applicaiton env.. ie. spring bean class object will be created on 1 per each browser s/w of a client machine.

scope="application" *(http protocol)*

==========

IOC container keeps spring bean class object as ServletContext attribute i.e 1 springbean class object for entire web application..

scope="websocket" *(websocket protocol)*

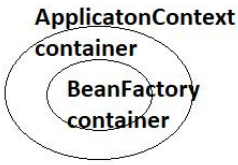
==========

IOC cotnainer keeps spring bean class object as required for websocket Programming..

note:: singleton, prototype scope can be used both in stanalone , web application env..

ApplicationContext container

==========



=>It is extension of Beanfactory Container

=> To create container , create object for java class that implements org.sf.context.ApplicationContext(I) (which is a sub interface of BeanFactory(I))..

=> The popular implementation classes for "ApplicaitonContext"(I) are

a) FileSystemXmlApplicationContext (In standalone Apps)

-->Creates "AC" container by locating spring bean cfg file from the specified path of file system

eg: ApplicationContext ctx=
new FileSystemXmlApplicationContext("src/main/java/com/nt/cfgs/applicationContext.xml");

b) ClassPathXmlApplicationContext (in standalone Apps)

->creates "AC" container by locating spring bean cfg file from jars and directories added to CLASSPATH or build path

Eclipse "src" --> default folder in classpsath
Eclipse src/main/java -> default folder in classpsath (Gradle/Maven)

ApplicationContext ctx=new ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");

c) XmlWebApplicationContext (In Spring mvc web applications)

-> creates "AC" container in web application env.. by taking <servlet logical name>-servlet.xml of WEB-INF folder as spring bean cfg file. (useful in spring MVC web applicaitons)

d) AnnotationConfigApplicationContext (In standalone Apps)

=>In 100% code driven, Spring boot driven Spring application developement,, we replace spring bean cfg file (xml file) with Configuration class (java class).

=>This class is useful to create IOC container by taking given Configuration class ..

ApplicaitonContext ctx=new AnnotationConfigApplicationContext(AppConfig.class);

Configuration calss

e) AnnotationConfigWebApplicationContext (In Spring mvc web applications)

=> given to create "AC" container by taking given Configuration class as input class in web application env...

and etc..

Addtional features of ApplicationContext container with respect to BeanFactoryContainer

==========

(a) Pre-instantiation of singleton scope beans
(b) Ability to work with place holders and Properties file (automatic recognization)
(c) support for I18n (Internationalization)
(d) Support for Event handling & Publishing
(e) Ability to stop/close Container
(f) Automatic registration of BeanPostProcessors
(g) Automatic registration of BeanFactoryPostProcessors
(h) Support for Annotation driven, 100%Code driven and Boot driven Spring Programming
and etc...

(a) Pre-instantiation of singleton scope beans

=>BeanFactory Container creates spring bean class object only when factory.getBean(-) methods are called .. not when IOC container is created.. So we can say BeanFactory container is performing lazy instantiations(object creations) and Injections.

=> The moment ApplicationContext container is created, all singleton scopes will be instantiated and injections takes place on the those beans irrespective ctx.getBean(-) methods will be called or not .. this is called pre-instantiation/eager instantiation of singleton scope beans..

=> if prototype scope bean is dependent to singleton scope bean.. then that Prototype scope bean will also be pre-instantiated..to support pre-instantiation and injection of singleton scope target bean..But it does not the scope of Dependent bean.

```
<!-- cfg all dependent classes as spring beans -->
   <bean id="dtdc" class="com.nt.beans.DTDC" scope="prototype"/>
   <bean id="bDart" class="com.nt.beans.BlueDart"/>
   <bean id="fFlight" class="com.nt.beans.FirstFlight" scope="prototyp

<!-- Cft Target class as spring bean -->
<bean id="fpkt" class="com.nt.beans.Flipkart">
   <property name="courier" ref="fFlight"/>
</bean>
```

ServletContainer perofrms lazy instantiation on Servlet comps by default.. To enable PreInstantation or eager instantiation on serlvet comps enable <load-on-startup> on servlet comps..

Q) Why Prototype scope beans are not participating in pre-instantiation process directly?

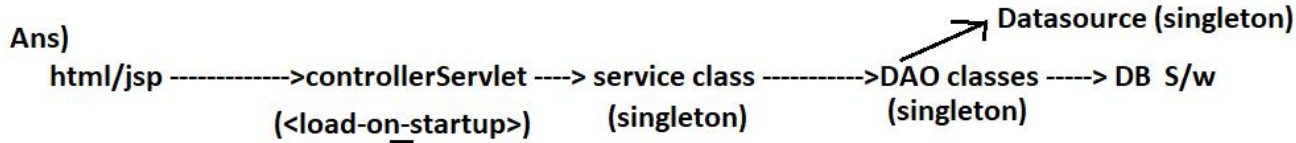
Ans) The object created prototype scope bean as part of pre-instantation process will always be wasted becoz for every factory.getBean(-)/ctx.getBean(-) method call the IOC container should create seperate new object for prototype scope bean...

note:: request, session, application , websocket scope beans also will not partcipate in pre-instantiation process.

note:: request, session, application , websocket scope beans also will not partcipate in pre-instantiation process.

Q) What is the advantage of pre-instantiation of spring beans with respect to real time Project?

Ans)



(ApplicationContext container is created in the init() of Controller Servlet comp)

Having the above setup, if we deploy web application

=> ControllerServlet class obj is pre-instantiated during the deployment becoz of <load-on-startup>

=> init(-) controller Serlvet excutes and creates ApplicationContext container

=> applicationContext container completes pre-instantation of all singleton scope DataSoruce, DAO classes, Service classes with neccessary injections

note:: now the first request given to Servlet comp directly participates in request processing..using already created object i.e does not waste time for creating objects and doing injections

---

=>if we want to disable pre-instantiation on singleton scope spring beans by Instructing IOC container then we can use lazy-init="true" of <bean> tag...

```
<bean id="dtdc" class="com.nt.beans.DTDC" scope="singleton" lazy-init=" true "/>
```

The possible values for "lazy-init" attribute are

a) true b) false c) default

lazy-init="true" --> enables lazy instantation on singleton scope spring bean

lazy-init="false" --> enables pre-instantation on singleton scope spring bean

lazy-init="default" --> fallbacks to "default-lazy-init" attribute of <beans> (default)

=>The possible values for "default-lazy-init" attribute of <beans> tag are

default-lazy-init="true" --> enables lazy instantationon all singleton scope spring beans current spring bean cfg file

default-lazy-init="false" --> enables pre-instantiation on all singleton scope spring beans current spring bean cfg file

default-lazy-init="default" --> if the singleton scope bean is inner bean/nested bean then it carries lazy instantation behaviour from its outer bean cfgs otherwise acts default-lazy-init="false" (default)

Q) we have 10 spring beans.. but i want see pre-instantation only on 6 spring beans , what should I do?

Ans) =>cfg 6 springbeans as singleton scope beans and 4 spring beans as prototype scope beans

=>Do not make prototype scope beans as dependent beans to singleton scope beans

=> do not enable lazy instantation directly or indirectly on singleton scope spring beans..

b) Ability to work with Properties file and place holders(${ <key>})

properties file
==========

It is the text file that maintains the entries in the form of key=value pairs.. It is very useful to pass info to Application from outside of the Application (softcoding)

info.properties

```
#personal info
nit.name=raja
nit.age=30
nit.addrs=hyd
```

jdbc properties (driver class name, url, db username, db pwd and etc..) will not be hardcoded inside the app.. they will be softcoded with the support of properties file...

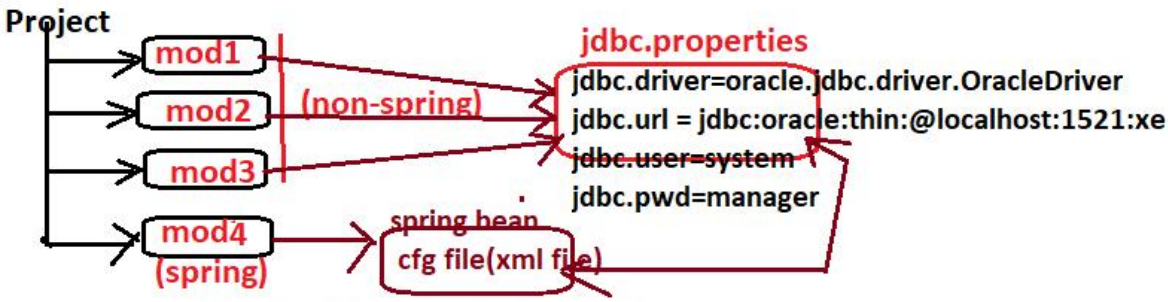
```
int a=10;   // hardcoding
int a=sc.nextInt(); // softcoding
```

collecting from enduser
using cmdline args,sys properties,
streams,properties file,xml file and etc..

Why should we encourage properties file in Spring Apps when it already supporting softcoding through xml file (spring bean cfg file)?

Ans1) Editing properies file is quite easy when compare to editing xml file as developer/enduser

Ans2) if multiple non-spring modules of project are getting jdbc proeprties from common properties file.. then the newly added module which is developed in spring should also get jdbc properties from same properties file..to support the centralization of properties file..



Ans3) while working with 100%code driven, spring boot driven Spring app development.. we must avoid xml file.. then we should gather inputs with the support of properties file.

Exmaple on Ans2 (Linking spring bean cfg file (xml file) with properties file)

==========

step1) keep MiniProject App ready..

step2) add Properties file having jdbc properties..

com/nt/commons/jdbc.properties

jdbc.properties

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

Makes the underlying AC container to recognize the placer holdes to collect their values from the given properties file

step3) Configure Properties file with spring bean file by using "PropertyPlaceHolderConfigurer" spcifying the name and location of properties file..

in applicationContext.xml

```
<bean id="pphc" class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
        <property name="location" value="com/nt/commons/jdbc.properties"/>
</bean>
```

step4) keep place holders (${ <key>}) in spring bean cfg file specifying keys of the properfies file to collect values from properties file..

in applicationContext.xml

```
<!-- Configiure DataSource -->
   <bean id="ds"    class="org.springframework.jdbc.datasource.DriverManagerDataSource">
         <property name="driverClassName"     value="${jdbc.driver}" />
         <property name="url"         value="${jdbc.url}" />
         <property name="username" value="${jdbc.user}" />
         <property name="password" value="${jdbc.pwd}" />
   </bean>

   <!-- cfg DAO class -->
   <bean id="oraCustDAO" class="com.nt.dao.OracleCustomerDAOImpl">
         <constructor-arg ref="ds" />
   </bean>

   <bean id="mysqlCustDAO" class="com.nt.dao.MysqlCustomerDAOImpl">
         <constructor-arg ref="ds" />
   </bean>
```

step5) Devleope the Client App using ApplicationContext container

step5) Devleope the Client App using ApplicationContext container

Using the above "PropertyPlaceholderConfigurer" setup.. we can collect and inject following values to Spring beans

(a) values from the configured properties file
(b) system properties values like os.name, java.vm.vendor and etc..
(c) Env.. variable values like path

```
<property name="osName" value="${os.name}"/>  //here os.name is system property name
<property name="path" value="${path}"/>  // here path is env.. variable name
```

note:: Instead of configuring "PropertyPlaceholderConfigurer" as spring bean , we can add the the following tag in spring bean cfg file..

```
<context:property-placeholder location="com/nt/commons/jdbc.properties" />
```

import context name space — *does not allow to get values from env.. variables..*

with

To work multiple properties files

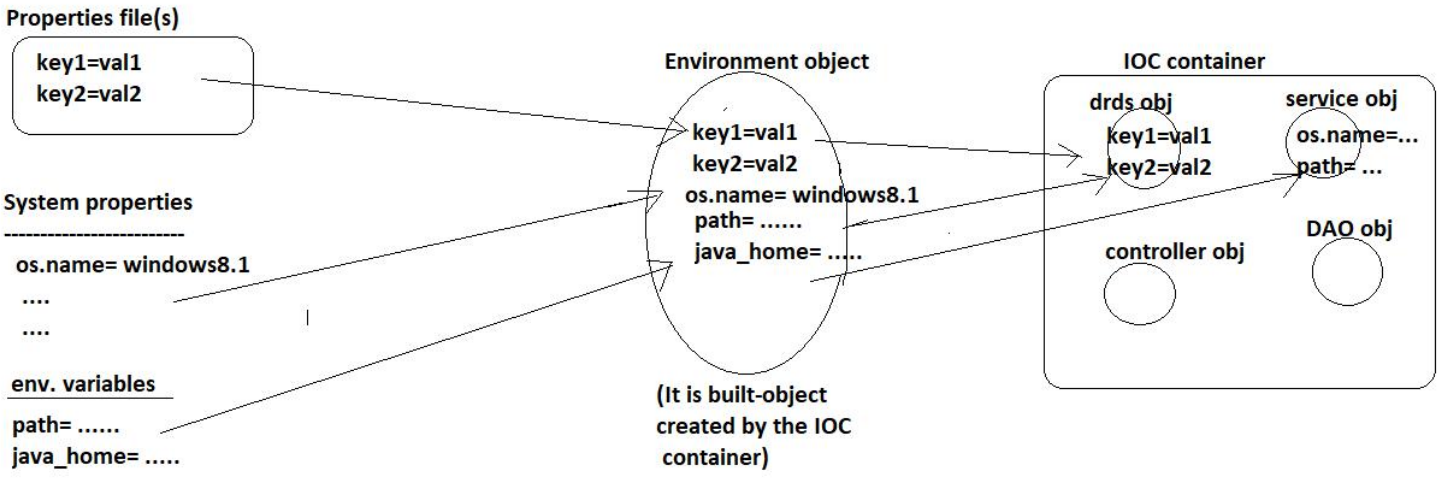
```
<context:property-placeholder location="com/nt/commons/jdbc1.properties,com/nt/commons/jdbc2.properties" />
```

(or)

```
<bean id="pphc" class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
    <!-- <property name="location" value="com/nt/commons/jdbc.properties"/> -->
    <property name="locations">
      <array>
        <value>com/nt/commons/jdbc1.properties</value>
        <value>com/nt/commons/jdbc2.properties</value>
      </array>
    </property>
</bean>
```

if multiple properties files are having same keys with different values then the lastly configured properties file data will override the remaining properties file data..

Internal flow



I18n (Internationalization)

=>Making our App working for different locales is called I18n..

=> Locale means language +country

en-US :: english as it speaks in US
en-BR : english as it speaks in Britan
fr-FR :: french as it speaks in france
fr-CA :: french as it speaks in Canda
de-DE :: german as it speaks in Germany
hi-IN :: hindi as it speaks in India
te-IN :: telugu as it speks in India

=> By enabling I18n on our App we can make our product or App giving service to multiple clients (different locale clients)

=> I18n is all about changing presentation logic as required for enduser .. and attracting different Locale enduser to operate the Application..

=>while working with I18n we need to take care

a) presentation labels (Using the support of multiple properties files)
b) number formats
c) currency symbols
d) Date,time formats
e) Indentation

and etc....

In jdk env.. we can use java.util.Locale , java.util.ResourceBundle clases to enabled I18n support.

To enable I18n for presentation lables

==========

a) Take multiple properties files on 1 locale basis including one base properties file
b) All properties files should have same keys and diff values collected from the google translators
c) base file name should reflect in all other properties file names...
d) All properties files extension should be .properties

App.properties (base file-En -- English)
App_fr_FR.properties (fr- french)
App_de_DE.properties (de --german)
App_hi_IN.properties (hi- hindi)

Java is based on unicode characterset i.e we can render the outputs of Java App using any language alphabets.. including indian languages...

java.util.Locale obj holds language and country where as ResourceBundle object reads and holds specific properties Locale specific file info based on "base name" and Locale obj data we supply..

Limitations with I18n in JSE/JEE env.. (Non-Spring Standalone , web application env...)

(a) if we enable I18n in java web application.. either we should create ResourceBundle object in every web comp or we should place either request scope or session scope based on the requirement.. i.e we must spend some time to analyze and decide the scope..

(b)if want to display messages collected from multiple properties files in a single web page then we need create multiple objects for ResourceBundle class.

```
//prepare Local object having language and country
      Locale locale1=new Locale("fr","FR");
//Create ResourceBundle object having Locale object  (Collects the properties file info into bundle
object based on given Locale obj data)
      ResourceBundle bundle1=ResourceBundle.getBundle("com/nt/commons/App",locale1);

  /prepare Local object having language and country
       Locale locale2=new Locale("hi","IN");
  //Create ResourceBundle object having Locale object  (Collects the properties file info into bundle object
  based on given Locale obj data)
       ResourceBundle bundle2=ResourceBundle.getBundle("com/nt/commons/App",locale2);
```

To overcome the above problems take the support spring based I18n

===

=>Spring's ApplicationContext container gives built-in support for I18n

=> we need to use ctx.getMessage() method to get message from properties file based on Locale object data that we pass..

=> we need cfg "ResourceBundleMessageSource" as spring bean having fixed bean id "messageSource" and specifying base file name..

```
IOCProj36-SpringI18n
    |---->src/main/java
            |---->com.nt.cfgs
                    |--->applicationContext.xml
            |-->com.nt.commons
                    |---->App.properties ,App_fr_FR.properties
                    |--->App_de_DE.properties , App_hi_IN.properties
            |--->com.nt.test
                    |---->SpringI18nTest.java
            |
            |--->build.gradle
```

(base file)

if matching locale specific properties file is not found for the given Locale object data.. then it fallbacks to base properties file..

In client App

```
String cap1=ctx.getMessage("btn1.cap", null, locale);
String cap2=ctx.getMessage("btn2.cap", null, locale);
String cap3=ctx.getMessage("btn3.cap", null, locale);
String cap4=ctx.getMessage("btn4.cap", null, locale);
System.out.println(cap1+"   "+cap2+"   "+cap3+"   "+cap4);
```

applicationContext.xml

fixed bean id

Internally uses ResourceBundle

```
<bean id="messageSource"   class="pkg.ResourceBundleMessageSource">
     <property name="basename" value="com/nt/commons/App"/>
 </bean>
```

What is the difference ctx.getBean() and ctx.getMessage() method?

Ans) ctx.getBean(-) gives spring bean class object based on given bean id ...
where as ctx.getMessage(-) method gives message from Locale specific properties file or Base properties file based on key and Locale object data that are supplied...

note:: ctx.getMessage() internally calls ctx.getBean(-) having fixed bean id "messageSource".

Q)Why should we give fixed bean id "messageSource" for ResourceBundleMEssageSource class cfg?

Ans) ctx.getMessage(-) method internally called ctx.getBean(-) with fixed bean id "messageSource" to get ResourceBundleMessageSource class object and to get basename property value from it..
So we should also give same bean id as fixed bean id while cfg "ResourceBundleMessageSource".

Each Message in properties file can have max of 5 args {0} ,{1} ... {4} we can supply these argument values from our Application.. as Stirng[] or Object[] values through ctx.getMesage(-,-,-) as show below

In App.properties
==========
btn1.cap=insert {0}

In Client app
==========

key or code in properties file

Message argument value {0} ... {4}

Default message

```
String cap1=ctx.getMessage("btn1.cap", new String[] {"student"} ,"msg1", locale);
```

gives insert student {0}

Locale object

Spring I18n Advnatages
==========

a) Using single object of "ResourceBundleMessageSource" we can get messages from either one locale spefic properfies file or from multiple locales specific properties files...
b) In Web applicaiton env.. we need not worry about keeping "ResourceBundleMessageSource obj in a scope.. By just creating object.the IOC Container uses in multiple places
c) There is a ability to pass default messages as fallback messages
d) Threre is a facility to pass argument values {0} .... {4} to messages...

Event Handling in spring

==========

=>Event is an action performed on the comp or object

eg:: clicking on the button , loading the page , opening window, creation of object, creating of container, closing of container and etc..

=>Executing some logic when event is raised is called event handling .. for this we need event handling methods supplied event Listeners..

=> Event Listener/Handler is a class that implements java.awt.EventListener(I) having Event handling logics in the event handling methods..

=> Servlet Listeners are also some kind of Event Listerners...

=> We do not call Event handling methods manually.. they will be called automatically when the event is raised..

note:: In Java every event is an object created by JVM internally.

=>For Event handling we need 4 details

a) source object -> eg: Button (c) obj

b) Event class -> eg: java.awt.ActionEvent (c) (when we click the Button)

c) Event Listener -> eg: java.awt.ActionListener (I)

d) Event handling method :: public void actionPerformed(ActionEvent ae){ ... }

=> Event Handling on ApplicationContext container is possible.. and helps us to know when the ApplicationContext container is started and ended/stopped/closed. Using this we can evaluate the performence of ApplicationContext container...

With respect to Spring AC container

Source object :: ApplicationContext object

Event class :: ApplicationEvent (c) [will be raised when container created and stopped/closed]

Event Listener :: ApplicationListener (I)

Event handling method :: public void onApplicationEvent(ApplicationEvent e) { }

(Executes when IOC container creates/started and stopped/closed)

note:: Event handling methods are generally callback method i.e we do not call them manually they will be called automatically...

steps for event handling

==========

step1) keep any ApplicationContext container based App ready

step2) Develop EventHandler/listener class implementing ApplicationListener(I) having logic in onApplicationEvent(-) method

step3) Cfg Listener class as spring bean in spring bean cfg file

step4) Run the Client App (make sure that ctx.close() is called)

```
package com.nt.listener;
import java.util.Date;
import org.springframework.context.ApplicationEvent;
import org.springframework.context.ApplicationListener;
public class IOCContainerMonitoringListener implements ApplicationListener {
    private long start,end;

    public IOCContainerMonitoringListener() {
            System.out.println("Listener:: 0-param constructor");
        }

        @Override
        public void onApplicationEvent(ApplicationEvent e) {
            if(e.toString().indexOf("ContextRefreshed")!=-1) {
                start=System.currentTimeMillis();
                System.out.println("IOC contaier is started at::"+new Date());
            }
            else if(e.toString().indexOf("ContextClosed")!=-1) {
                end=System.currentTimeMillis();
                System.out.println("IOC contaier is stopped/closed at::"+new Date());
                System.out.println("IOC container active duration ::"+(end-start)+" ms");
            }

        }

}
```

applicationContext.xml

```
<bean class="com.nt.listener.IOCContainerMonitoringListener" />
```

=>All Listener classes of any env.. directly or indirectly implements java.awt.EventListener(I).. Afer performing pre-instantation of singlescope beans and related injections all Listener classes that are cfg spring beans will also be instantiated irrespective their scopes..

What is the difference b/w BeanFactory container and ApplicationContext container?

BeanFactory	ApplicationContext
(a) Does not support pre-instantation of spring beans	(a) supports pre-instantiation of singleton scope beans
(b) No Direct Support for properties files and placeholders (${ ...})	(b) There is direct support
(c) No support for I18n	(c) supports
(d) Does not support Event Handling and Event Publication	(d) supports
(e) No support for closing/stopping Container	(e) we can stop/close Container (ctx.close()/ctx.stop())
(f) Does not support Anntoation based programming	(f) supports
(g) supports only Xml driven cfgs	(g) supports xml driven cfgs, annotation driven cfgs, 100%code cfgs /java Config cfgs and Spring boot cfgs
(h) No Automatic registration of BeanPostProcessors	(h) Suppports Automatic registration of BeanPostProcessors
(i) No Automatic registration of BeanFactoryPostProcessor	(i) supports Automatic registration of BeanFactoryPostProcessor
j) Uses bit less memory	j) Uses bit extra memory..
k) Can not be used in spring mvc based web applicaiton development	k) can be used...

When should we use BeanFactory Container and when should we ApplicationContext container?

Ans) Always prefer using ApplicationContext container in applications.. becoz its supports all features of spring... but if application is memory critical Applicaiton (1 or 2 extra kilo bytes are matters) like mobile Apps, Embedded System/MicroController Applicaitons and really not using ApplicationContext container features then go for BeanFactory Container otherwise always prefer using Application Container in all kinds of Applications including standalone ,web , enterpirse Applications...

P NameSpace and C NameSpace

==========

=>Xml schema namepaces is a library that contains set of xml tags..

=>every xml schema namespace is identified with its namespace uri/url.

=> To use xml schemaspace in our xml file we must import namespace uri/url in the xml file

Spring gives multiple built-in xml schema name spaces like

name sapce		name space uri
beans	---------->	*https://www.springframework.org/schema/beans/*
context	---------->	*https://www.springframework.org/schema/context/*
p	---------->	*https://www.springframework.org/schema/p/*
c	---------->	*https://www.springframework.org/schema/c/*

aop

tx

jee

and etc..

P namespace is given alternate to the lengthy <property> tag... to perform setter Injection cfgs..

c namespace is given alternate to the lengthy <constructor-arg> to perform constructor injection cfgs

```
public class Employee {
    private int eno;
    private  String ename;
    private  Date  dob;
    private  Dept  dept;
     //setters
       .....
     //toString()
       ...
}
```

```
public class Department {
    private int dno;
    private  String dname;
    private  Date  dos;
     //3 param constuctor  to consturctor injection
       .....
     //toString()
       ...
}
```

application.xml (old style)

```
==============
  <bean id="dob"  class="java.util.Date">
      <property name="year"  value="90"/>
      <property name="month"  value="11"/>
      <property name="date"  value="30"/>
  </bean>

   <bean id="dos"  class="java.util.Date">
      <property name="year"  value="100"/>
      <property name="month"  value="5"/>
      <property name="date"  value="20"/>
   </bean>

    <bean id="dept"  class="pkg.Departmement">
        <constructor-arg  value="5001"/>
         <constructor-arg  value="IT"/>
        <constructor-arg  ref="dos"/>
    </bean>

    <bean id="emp"  class="pkg.Employee">
       <property name="eno"  value="1001"/>
       <property name="ename"  value="rakesh"/>
       <property name="dob"  ref="dob"/>
       <property name="dept"  ref="dept"/>
     </bean>
<beans>
```

p namespace based syntax for setter Injection

```
<bean  p:<property>="<value>"   p:<propety>-ref="<beanid>" />
```

(name: p:<property>="<value>" — for simple property; name: p:<propety>-ref="<beanid>" — for object/ref type property)

c namespace based syntax for constructor Injection

```
<bean  c:<property>="<value>"  c:<property>-ref="<beanid>"/>
```

(name: c:<property>="<value>" — for simple property; name: c:<property>-ref="<beanid>" — for object type property)

Example

applicationContext.xml

```
<beans>
   <bean id="dob"  class="java.util.Date"  p:year="90" p:month="11" p:date="30"/>
   <bean id="dos"  class="java.util.Date"  p:year="100" p:month="5" p:date="20"/>
    <bean  id="dept"  class="pkg.Department"  c:dno="5001"  c:dname="IT"  c:dos-ref="dos"/>
    <bean id="emp"  class="pkg.Employee"  p:eno="1001"  p:ename="rakesh"  p:dob-ref="dob"  p:dept-ref="dept"/>
</beans>
```

int <variablename>; syntax

int a; =>example1

int b; => examle2

note:: Both Containers (BF, AC) supports P-namespace, c-namespace based programming..

Limitations of P and C namepaces

=>These namespace do not support inner beans cfgs..

=> these namespaces do not support collection/Array Injections

=>these namespaces do not support Collection merging

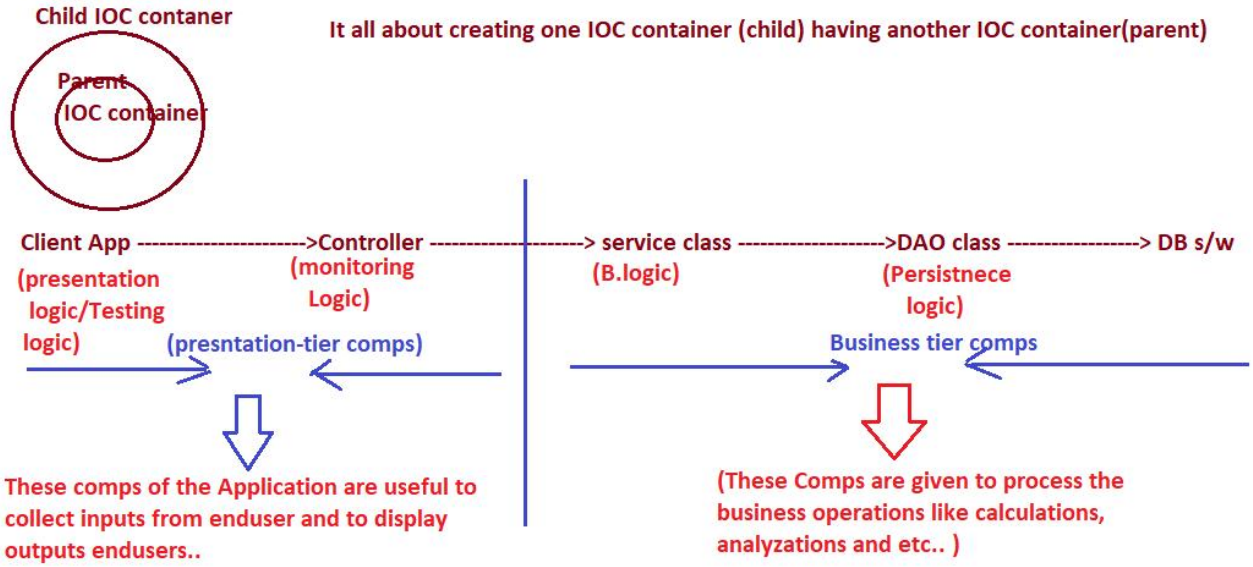
=>these namespaces do not support Null Injections

=>Does not allow to resolv constructor params by type,by index or by order..
(allows to resolve only by name)

=>Came lately when industry is moving towards Annotation driven programming..

note:: we can mixup both <property> tags Pname space and <constructor-arg> tags with c namespace in a single application..

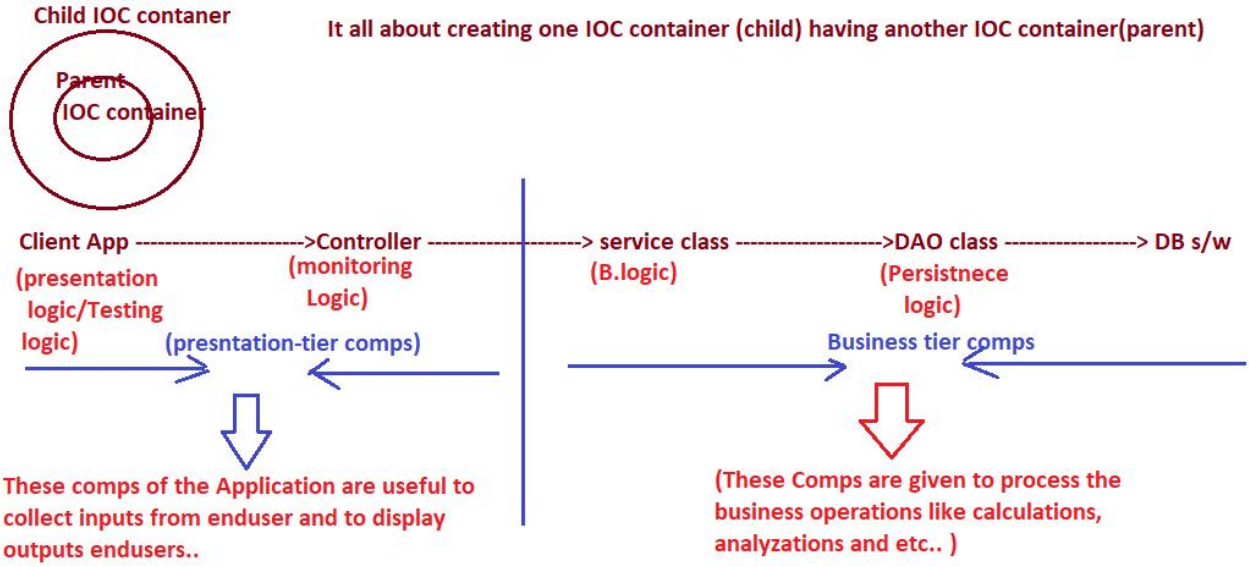
Nested BeanFactory / Nested ApplicationContext



note:: We should always think about developing presentation tier comps and business tier comps having loose coupling i.e the degree of dependency must be very less between both tiers... This can be achieved by taking two seperate IOC containers 1 for Presnetation tieer comps and another for business tier comps.. This helps us to enable ro disable spring supoprt in each tier comps irrenspective other tier comps....

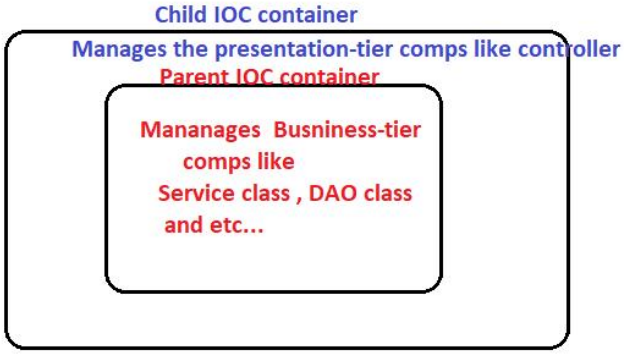
Nested BeanFactory / Nested ApplicationContext

==========



note:: We should always think about developing presentation tier comps and business tier comps having loose coupling i.e the degree of dependency must be very less between both tiers... This can be achieved by taking two seperate IOC containers 1 for Presnetation tieer comps and another for business tier comps.. This helps us to enable ro disable spring suport in each tier comps irrepspective other tier comps...

note:: Taking two independent IOC containers is not going to work out completly .. becoz we need to inject Service class object object to Controller class object i.e we need to keep Two Containers are parent and child IOC container... Since Parent IOC container bean can be injected to Child IOC container Bean.. and reverse is not possible So we need to take business-tier comps in parent IOC container and presentiation tier comps in child IOC container. This allows to inject Service class object to Controller class obj.



create
To nested Containers with ApplicationContext

==========

Cfg service ,DAO classes

```
//parent IOC container
ApplicationContext parentCtx=new ClassPathXmlApplicationContext("com/nt/cfgs/business-beans.xml");
//child IOC container
ApplicationContext childCtx=new ClassPathXmlApplicationContext("com/nt/cfgs/presentation-beans.xml",parentCtx);
```

Cfg Controller class

To create Nested BeanFactory

Cfg Controller class

To create Nested BeanFactory
==========

```
//parent Container
DefaultListableBeanFactory  parentFactory=new DefaultListableBeanFactory();
XmlBeanDefinitationReader pReader=new XmlBeanDefinitationReader(parentFactory);
pReader.loadXmlBeanDefinitations("com/nt/cfgs/parent-beans.xml");
//child Container
DefaultListableBeanFactory  childFactory=new DefaultListableBeanFactory(parentFactory);
XmlBeanDefinitationReader cReader=new XmlBeanDefinitationReader(childFactory);
cReader.loadXmlBeanDefinitations("com/nt/cfgs/child-beans.xml");
```

MiniProject#2 Discusssions
==========

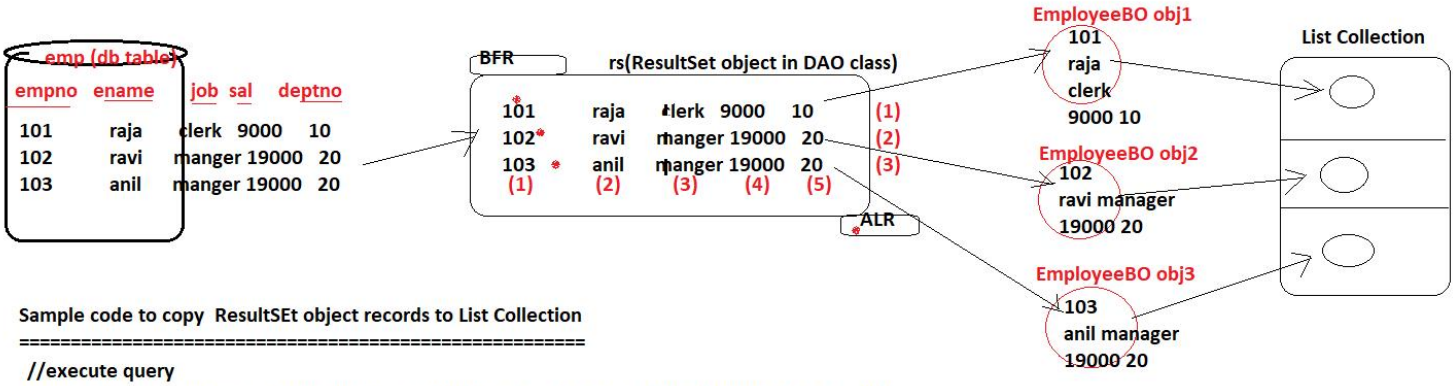
=> if DAO class executes select query and gives multiple Records into ResultSet object then sending that ResultSet object to Service class is bad practice becoz we should place jdbc code only in DAO class.. not in other layers.. To overcome this problem copy ResultSet object records to the objects of BO /Entity class and send them to Service class by keeping them in array/collection (collection is recomanded)..

After placing data into any Collection.. if we are performing only read opearations on that collecction then prefer working with non-synchonized collections for perfromence
eg:: ArrayList , HashMap and etc...

After placing data into any Colection , if we are looking perform both read and write oeprations simultaenosuly then prefer using synchronized collections for thread safety..
eg:: Vector , Hashtable and etc.

=> if data is key -value pairs , then go for map collection...
=> if want to preserve the insertion order and wants access elemennts using indexes then for List collection otherwise go for Set collection...

conclusion:: While generating reports by collecting data/records from Db table..we prefer using ArrayList to store those records as the object of BO class in the elements..



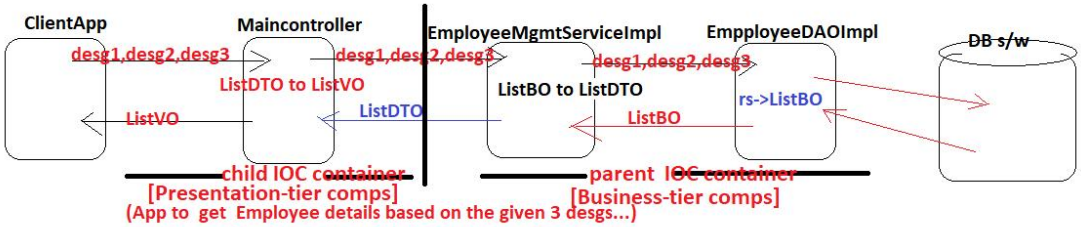
Sample code to copy ResultSEt object records to List Collection
==========

```
//execute query
PreparedStatement ps=con.prepareStatement("select empno,ename,job,sal,deptno from emp");
ResultSet rs=ps.executeQuery();

List<EmployeeBO> al=new ArrayList();

while(rs.next()){
  //copy each record of rs to each object of BO
  EmployeeBO bo=new EmployeeBO();
  bo.setEmpNo(rs.getInt(1));
   bo.setEname(rs.getString(2));
  bo.setJob(rs.getString(3));
  bo.setSalary(rs.getFloat(4));
   bo.setDeptNo(rs.getInt(5));
 //add each BO obj to List colelction
   al.add(bo);
}
```

```
public class EmployeeBO{
  private Integer empNo;
  private String ename;
  private  String job;
  private   Float sal;
  private   Integer deptno;
  //setters && getters
  ....
  ....
}
```

In Java beans like VO ,DTO, BO classes we prefer taking wrapper data types for properies .. not the primitive data types.. becoz primitive variables default values like 0, 0.0 and etc.. allocate memory .. where wrapper variable default values is null and null does not allocate memory..

while Persisting BO object data to Db table record.. the primtive variables defailt values like 0,0.0 and etc.. insertion will allocate memory in DB table.. where as the Wrapper variables default value null insertion does not allocate memory in Db table...



(App to get Employee details based on the given 3 desgs...)

Story board of Mini Project-2 (nested IOC container)

DAO --> 1 per db table
Service -->1 per module
controller --> 1 per project

oracle Db s/w
emp (db table)

DAO

EmployeeDAO.java

```
public interface EmployeeDAO{
   public List<EmployeeBO> getEmpsByDesgs(String desg1,String desg2,String desg3)throws Exception
}
```

EmployeeDAOImpl.java

```
public class EmployeeDAOImpl implements EmployeeDAO{
   private DataSource ds;
   public EmployeeDAOImpl(DataSource ds){
       this.ds=ds;
    }
   public List<EmployeeBO> getEmpsByDesgs(String desg1,String desg2,String desg3) throws Exception{
      //jdbc code to ResultSet having emp details
       ....
       ....
     //convert ResultSet object records into ListBO
       ...
       ...
     return listBO;
    }
}
```

(q) (p) (r)

Service class

EmployeeMgmtService.java

```
public interface EmployeeMgmtService{
   public List<EmployeeDTO> fetchEmpsByDesgs(String desg1,String desg2,String desg3) throws Exception;
}
```

EmployeeMgmtServiceImpl.java

```
public class EmployeeMgmtServiceImpl implements EmployeeMgmtService{
    private EmployeeDAO dao;
   public EmployeeMgmtSErviceImpl(EmployeeDAO dao){
       this.dao=dao;
    }
   public List<EmployeeDTO> fetchEmpsByDesgs(String desg1,String desg2,String desg3)throws Exception{
      //convert desgs into upper case (b.logic)
      desg1=desg1.toUpperCase();
      desg2=desg2.toUpperCase();
      desg3=desg3.toUpperCase();
      //use DAO
      List<EmployeeBO> listBO=dao.getEmpsByDesgs(desg1,desg2,desg3);
      //convert listBO to listDTO
        ....
       return listDTO;
    }
}
```

(n) (s) (o) (t)

note:: Taking only model/BO/Entity class passing it to DAO to service to controller to Client App is bad pracice.. becoz some times we need add or decrease or modify persisted data collected from DB before presenting to the enduser..

Controller
=========

```
public class  MainController{
  private EmployeeMgmtService service;

  public  MainController(EmployeeMgmtSErvice  service){
     this.service=service;
  }
  public  List<EmployeeVO>  gatherEmpsByDesgs(String desg1,
                                              String desg2,String desg3) throws Exception{
    //use Service
    List<EmployeeDTO> listDTO =service.fetchEmpsByDesgs(desg1,desg2,desg3);
    //convert  listDTO to listVO
      ....
    return listVO;
  }
}
```

(l) (m) (u) (v)

NestedIOCContainerTest.java
==========

```
public class NestedIOCContainerTest{
    public  static  viod  main(String args[]){
      //create parent IOC container
       ApplicationContext parentCtx=new ClasspathXmlapplicationContext("business-beans.xml");
      //create create Child IOC container
       ApplicationContext childCtx=
            new ClasspathXmlapplicationContext("presentation-beans.xml",new String[]{},parentCtx);
       //get Controller class object
       MainController controller=childCtx.getBean("controller",MainController.class);
       //invoke method
       try{
        List<EmployeeVO> listVO=controller.gatherEmpsByDesgs("CLERK","MANAGER","SALESMAN");
        sysout(listVO);
       }
       catch(Exception e){
        e.printStackTrace();
       }
    }//main
}//class
```

(a) (b) (c) checking well formed,valid ness and inMemomety MetaData (f) (g) checking well formed,valid ness and inMemomety MetaData (j) (m) (w) (x)

childCtx: InMemoryMetaData of presentation-beans.xml → parentCtx: InMememory MetaData of business-beans.xml

jars
====
spring-context-support-<ver>.jar
ojdbc8.jar
hikaricp-<version>.jar
lombok-<ver>.jar

business-beans.xml

```
<beans ...>
  <bean id="hkDs"  class="pkg.HikariDataSource">
       .....
       ....
  </bean>
  <bean id="empDAO"  class="pkg.EmployeeDAOImpl">
      <constructor-arg ref="hkDs"/>
  </bean>
  <bean id="empService" class="pkg.EmployeeMgmtServiceImpl">
       <constructor-arg ref="empDAO"/>
  </bean>
</beans>
```

(d) pre-instantation singleton scope beans

presentation-beans.xml

```
<beans ....>
    <bean id="controller"  class="pkg.MainController">
        <constructor-arg ref="empService"/>
    </bean>
</beans>
```

(h) pre-instantiation of singleton scope beans

(e) InternalCache of parentIOC container

hkDs	HikariDataSource obj ref
empDAO	EmployeeDAOImpl class obj ref
empService	EmployeeMgmtSErviceImpl obj ref

Internal Cache of childIOC container

controller	MainController object ref (k?) (i)

```
// https://mvnrepository.com/artifact/com.zaxxer/HikariCP
implementation group: 'com.zaxxer', name: 'HikariCP', version: '3.4.5'
// https://mvnrepository.com/artifact/org.springframework/spring-context-support
implementation group: 'org.springframework', name: 'spring-context-support', version: '5.2.8.RELEASE'
// https://mvnrepository.com/artifact/com.oracle.ojdbc/ojdbc8
implementation group: 'com.oracle.ojdbc', name: 'ojdbc8', version: '19.3.0.0'
// https://mvnrepository.com/artifact/org.projectlombok/lombok
implementation group: 'org.projectlombok', name: 'lombok', version: '1.18.12'
```

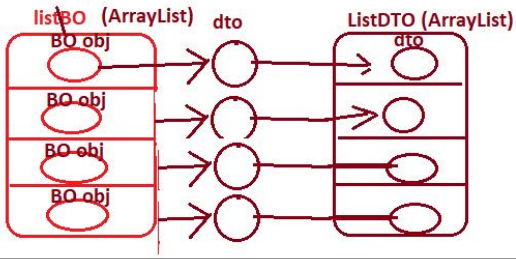
=>The PRocess of converting simple data type values to wrapper class objs automatically is called autoboxing and reverse is called auto unboxing (java 5 feature)

Q) What is excpetion rethrowing ? how does it will be used in Project Development?

=>Catching exception raised in try block beging from catch block and re throwing either same exception or other exception to caller method is called exception rethrowing.For this we need to use try/catch ,throws, throw statements togher.. we generally do this work DAO class methods while writing jdbc code...

note: "throws" declares the exception to be thrown as part method signature (like beware of dog)
"throw" actually create and throws new excpetion or rethrows same or new exception.

java8 fatures : (prepare by tommorow)
Lamda expression..
foreach(-) method
method reference



Java 7 and Before

1) Declaration interface

```
public inteface Arithmetic{
    public int sum(int x,int y);
 }
```

2) Impl class for Interface

```
public class Calculator implements Arithmetic{
    public int sum(int x,int y){
        return x+y;
    }
}
```

3) creating object for impl class

```
Arithemtic ar=new Calculator();
```

4) invokation of the method

```
S.o.p("result is "+ar.sum(10,20));
```

java 8 onwords

1) Declartion of interface with method

```
public inteface Arithmetic{
    public int sum(int x,int y);
 }
```

2) Implementation class + object creation :: Lamda expression

⇩ allows to write code in short form

syntax ::

```
<interface> <ref var name>=(params)-> {  <body> }
```

```
eg1:: Arithmetic ar=(int x,int y)->{ return x+y; }
eg2:: Arithmetic ar=(int x,int y)-> return x+y;  //{ } is optional for single line body
eg3:: Arithmetic ar=(x,y)-> return x+y;  // data types are optional for params
eg4:: Arithmetic ar=(a,b)-> return a+b;  // param names can be changed
```

note:: () are optional for single param

```
eg5:: Arithmetic ar=(a,b)-> a+b; // if no { }  is taken ,
                                 //return keyword optional
```

4) invocation of the method

```
S.o.p("result is "+ar.sum(10,20));
```

note:: LAMDA Expression can be applied only with Funcationl interfaces...

note:: LAMDA Expression can be applied only with Funcationl interfaces...

Functional interface

==========

=>the interface that contains single Abstract method (SAM) directly or indirectly is called Functional interface..

```
public interface Test{    // valid functional interface
   public  void m1();
 }
```

note:: funcational interface can have multiple default methods and static method definations..

@FuncationalInterface does not make interface as funcational interface.. It makes java compiler to check current interface is functional interface or not.

```
@FunctionalInterface
public interface Test{
   public  void m1();
 }
```

note:: functional interface should have SAM(single abstract method) directly or indirectly even after it is extending from other interfaces

```
public interface Demo{              //invalid funcational interface
   public void m1();
   public void m2(int x,int y);
}

public interface Sample extends  Test{   //  valid funcational  interface
   }

public interface  Sample1 extends  Test{   //invalid functional interface
    public void m2();
  }

 public interface  Sample2 extends  Test,Sample{   //valid functional interface
   }
```

3 for loops in java

==========

a) trandtional for loop (from java1)

b) enhance for loop /for each loop (from java5)

c) forEach() method (from java 8)

(Best performence) ⇩

=>can not be used on arrays

=>can be used only on collections

```
//copy listBO elements listDTO elements..
List<EmployeeDTO>  listDTO=new ArrayList();
List<EmployeeBO> listBO=dao.getAllEmpsByDesgs(desg1, desg2, desg3);
            //convert listBO to listDTO
            listBO.forEach(bo->{
                  //copy each bo to dto
                  EmployeeDTO dto=new EmployeeDTO();
                  BeanUtils.copyProperties(bo,dto);
                  dto.setSerialNo(listDTO.size()+1);
                  //add each dto listDTO
                  listDTO.add(dto);
            });
```

<ref > tag attributes
==========

<property>
<ref bean="...."/>
</property>

is equal to

<property ref="...."/>

<constructor-arg>
<ref bean="...."/>
</constructor-arg>

is equal to

<constructor-arg ref="...."/>

atrributes of <ref> tag are
(a) bean
(b) parent
(c) local (removed from spring 4.x onwords)

<ref bean="<dependent bean id>">

=>Searches given bean id based spring bean class cfg first in the current/local IOC container spring cfg file ,if not available then it searches in parent IOC cotainer spring bean cfg file...

<ref parent="<dependent bean id>">

Searches only in the parent IOC container spring bean cfg file...

<ref local="<dependent bean id>">

from (removed spring 4.x)

Searches only in the current/Local/child IOC container spring bean cfg file ...

Annotation driven Spring Programming in Core module
==========

Annotations support is introduced to spring from spring 2.0 added more annotations incrementely in later versions...

List of Annotations related spring core module

==========

Spring 2.0

@Configuration @Required @Repository @Order

Spring 2.5

@Autowired @Qualifier @Scope

Stereotype annotations:

@Component @Service @Controller @Repository

Spring 3.x

@Bean	*@ComponentScan*
@DependsOn	*@PropertySource*
@Lazy	*@PropertySources*
@Value	*@Primary and etc..*
@Import	
@ImportResource	

Spring4.x:

@Lookup and etc..

Spring5.x:

@Nullable
@NonNull
@NonNullApi
@NonNullApi
@NonNullFieldds and etc..

@Required

==========

=>While using any parameterized constructor for consturctor injection ,we must configure all params of that constructor for injection..otherwise exception will come.. This restriction is not there while working with setter injection... To bring such restriction on our choice bean properties through setter injection .. we need go for @Required annotation...

```
public class  PersonInfo{
  private  int pid;
  private String pname;
  private  String paddrs;
   @Required
   public  void  setPid(int pid){ this. pid=pid;}
    @Required
   public  void  setPname(String pname){ this.pname=pname;}
   public  void  setPaddrs(String paddrs){ this.paddrs=paddrs;}
   //toString()
     ....
 }
```

@Required is applicable at method level..

note:: pid ,pname properties must be cfg for setter injection otherwise exception will be thrown.

@Required is deprecated in spring 5.1 saying go for constructor injection in order to add restriction on injections..

=>The entire functionality of @Required annotation is placed in a readymade class called "RequiredAnnotationBeanPostProcessor" , So we should cfg this class as spring bean..

note:: Every BeanProrcessor will be activated automatically .. Once it is configured as spring bean.. in spring bean cfg file..

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

cfg Bean PostProcessor for every annotation seperately is complex process .. To overcome that Problem just place <context:annotation-config> in spring bean cfg file

Activates various annotations to be detected in bean classes: Spring's @Required and @Autowired, as well as JSR 250's @PostConstruct, @PreDestroy and @Resource (if available), JAX-WS's @WebServiceRef (if available), EJB 3's @EJB (if available), and JPA's @PersistenceContext and @PersistenceUnit (if available).

From spring 5.1 , this tag is not working for deprecated annotaitons like @Required.

=>we can not use @Required along lombok api generated setter methods..

@Autowired
==========

=> Performs byType,byName,constructor mode of autowiring (detecting the dependent beans dynamically with out using <property> ,<costructor-arg> tags).

=> Can be applied at field level(instance variables),
constructor level , setter method level and arbitary method level..
(any method)

=>Can not be used to inject values to simple properties.. can be used to inject values only to Object type/ref type bean properties.

```
public class  Flipkart{
   @Autowired
   private   Courier courier;

   public   String  shopping(String[] items, float [] prices){
       ...
       ...
    }
}
```

```
public class  DTDC implements Coureir{
    ...
    ...
}
```

applicationContext.xml
==========

```
<beans .... >

   <bean id="dtdc"  class="pkg.DTDC"/>

   <bean id="fpkt"  class="pkg.Flipkart"/>

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

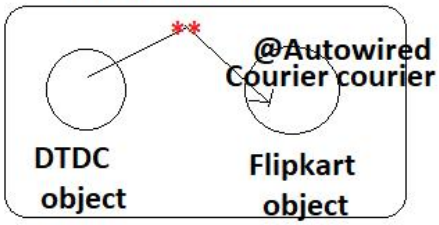
Activates certain annotaitons like @Required, @Autowired and etc..
by internally cfg their respective BeanPostProcessors as spring beans.

In xml driven spring programming Dependency injection is possible only through setter methods and costructors.. where as @Autowired based Injections the dependency injection is possible through directly using fields , setter methods , constructor and arbitary method.

note:: Using @Autowired at field level with out having setter method ,parameterized constructor is possible .. In that situation it will access private proeprty through reflection api and performs injection on that property .. by seeing @Autowired Annotation...

Internal workFlow :: IOC Container creation ---> loading of spring bean cfg file ---> checking well-formedness,validness --> Creating InMemory Metata of spring bean cfg file ---> Pre-instantiation of all singleton scope beans ---> Activation of BeanPostProcessors based <context:annotation-config> tag.. -->Uses reflection api and detects in @Autowired on the top of "private Courier courier" field in Flipkart class ---> get access to that property and searches Courier(I) type springbean cfg in the In memory metadata of spring bean cfg file --> finds "DTDC" cfg ,so takes DTDC class object and assings/injects to "courier" property --> keeps all spring beans in the internal cache of IOC container..

IOC container



internal cache of IOC container

dtdc	DTDC class obj ref
fpkt	Flipkart class obj ref

** :: this injection takes place becoz of <context:annotation-config> supplied/activated @Autowired related BeanPostProcessor.

@Autowred perform byType mode of autowiring by default..

if @Autowired does not finds its dependent then it throws org.springframework.beans.factory.NoSuchBeanDefinitionException: No qualifying bean of type 'com.nt.beans.Courier' available: expected at least 1 bean which qualifies as autowire candidate. Dependency annotations: {@org.springframework.beans.factory.annotation.Autowired(required=true)}

applicationContext.xml

```
<!-- cfg all dependent classes as spring beans -->
    <bean id="dtdc"  class="com.nt.beans.DTDC"/>

 <!-- Cft Target class  as spring bean -->
 <bean id="fpkt"  class="com.nt.beans.Flipkart"/>

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

default value "true"

```
@Autowired(required = true)
private  Courier courier;  //rule1 ,2
```

if "Courier(I)" type dependent bean cfg is not available in spring bean cfg file then "NoSuchBeanDefinitationException" will be raised

---

```
@Autowired(required=false)
private Courrier  courier;
```

if "Courier(I)" type dependent bean cfg is not available in spring bean cfg file then no Exception will be raised)

if ambiguity problem raises then it throws

org.springframework.beans.factory.UnsatisfiedDependencyException: Error creating bean with name 'fpkt': Unsatisfie
through field 'courier'; nested exception is org.springframework.beans.factory.NoUniqueBeanDefinitionException: No qua
'com.nt.beans.Courier' available: expected single matching bean but found 2: dtdc,bDart

How to resolve Ambugity Problem that comes while working @Autowired based Dependency Injection?

Ans1) Using Qualifier(-) with dependent bean id or using <qualifier> name of dependent bean

```
@Autowired
@Qualifier("bDart") (or)
@Qualifier("b1")
private Courier courier;
```

Best Solution...

Indirectly it is performing byName mode of autowiring..

In applicationContext.xml

```
<!-- cfg all dependent classes as spring beans -->
  <bean id="dtdc" class="com.nt.beans.DTDC">
    <qualifier value="d1"/>
  </bean>
  <bean id="bDart" class="com.nt.beans.BlueDart" >
    <qualifier value="b1"/>
  </bean>

<!-- Cft Target class as spring bean -->
<bean id="fpkt" class="com.nt.beans.Flipkart"/>
 <context:annotation-config/>
```

Ans2) By matching Target field/property name with dependent class bean id ...

In Flipkart.java

```
@Autowired
private Courier courier;
```

In applicationContext.xml

```
<!-- cfg all dependent classes as spring beans -->
    <bean id="dtdc" class="com.nt.beans.DTDC"/>
    <bean id="courier" class="com.nt.beans.BlueDart" />
```

Ans3) Using primary="true" of <bean> (in xml driven spring bean cfg) or @Primary (in annotation driven spring bean cfg) for one dependent bean..

in Flipkart.java

```
@Autowired
private Courier courier;
```

In applicationContext.xml

```
<!-- cfg all dependent classes as spring beans -->
    <bean id="dtdc" class="com.nt.beans.DTDC">
    <bean id="bDart" class="com.nt.beans.BlueDart" primary="true">
```

Ans4) By making only one dependent bean as "autowire candidate" in spring bean cfg file

in Flipkart.java

```
@Autowired
private Courier courier;
```

in applicationContext.xml

```
<!-- cfg all dependent classes as spring beans -->
    <bean id="dtdc" class="com.nt.beans.DTDC" autowire-candidate="false"/>
    <bean id="bDart" class="com.nt.beans.BlueDart" />
```

@Autowired can be applied at fiel level, constructor level (parameterized) , setter method level and arbitary method level.

At setter method level

==========

```
@Autowired
@Qualifier("bDart")
public void setCourier(Courier courier ) {
        System.out.println("Flipkart.setCourier(-)");
        this.courier=courier;
}
```

At paramterized constructor level

==========

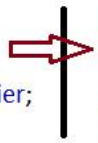
```
@Autowired
    public Flipkart(@Qualifier("dtdc") Courier courier) {
        this.courier=courier;
        System.out.println("Flipkart:: 1-param constructor");
    }
```

At arbitary method level

```
@Autowired
@Qualifier("bDart")
public  void  assign(Courier courier) {
        System.out.println("Flipkart.assign(-)");
        this.courier=courier;
}
```

At field level

```
@Autowired
@Qualifier("bDart")
private  Courier courier;
```



It is industry standard becoz we can work with lombok api ..and there is no need of any setter methods, paramaterized constructors, arbitarymethods supporting injection...

if we enable @Autwire on same property at multiple levels in which order the autowiring takes place?

Ans) field level --> parameter constructor level ---> setter/arbitary methord level (both will execute.. order can not be decided)

note:: setter method/ arbitary method level will come as final value..

imp obervations on @Autowired applied on the constructors

==========

a) applying @Autowired on 0-param constructor is meaning less

b) if multiple overloaded constructors having @Autowired(required=true) then exception will be raised

c) if multiple overloaded constructors having @Autowired(required=false) then the more param constructor will execute

d) if multiple overloaded constructors having @Autowired(required=false) on few constructor and @Autowired(required=true) on few other constructors then also exception will be raised.

e) if multiple overloaded constructors having @Autowired(required=false) with same no.of params then IOC container picksup the Constructor randomly..

note:: we can apply @Autowired on multiple fields ,setter methods , arbitary methods either having required=true /required=false .. But only constructor can have @Autowired with required=true. (refer the above points)

Stereo type annotations
==========
=>we have multiple with similar behaviour .. having minor differences.. So they are called
Stereo type annotations
@Component ---> To cfg java class as spring bean (with no specialities)
(To instantiate java class obj and to make it as sprign bean by IOC container)
@Service -----> @Component + also make as service class by giving Tx mgmt support
@Repository ---> @Component + also make as DAO class by Exception translation ability (SQLException to spring specific exceptions)
@Controller ----> @Component + also makes web controller class by having ability to take/process http requests
and etc..

To make IOC container going to different specified packages and their sub pkgs to search and recognize stereo type annotation classes as spring beans (by loading them and also by instantiating them) we need to place <context:component-scan packages="....." /> tag in spring bean cfg file...

multiple packages can be specified as coma seperated values...

=> All Stereo type Annotations must be applied at class level (Type level)

bean id / object name/ refence variable name

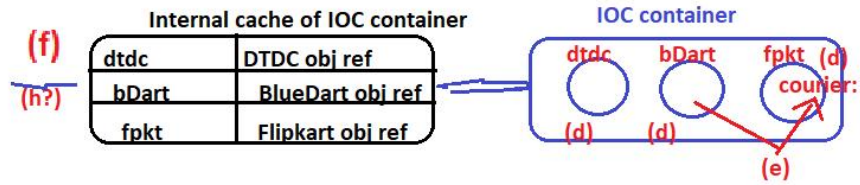
```
package com.nt.beans;
@Component("dtdc") //or @Component(value="dtdc")
public class DTDC implements Courier{
  ....
}
```

(d)

```
package com.nt.beans1;
@Component("fpkt")
public class Flipkart {
  @Autowired (e)
  private Courier courier;

  public Stirng shopping(String items[], float prices[]){
    ...
    ...
    return "..."; (l)
  }
}
```

(d) (k)

```
package com.nt.beans;
@Component("bDart") @Primary (d)
public class BlueDart implements Courier{
  ..
}
```



applicationContext.xml
==========

```
<beans .... >
  <context:component-scan base-Package="com.nt.beans,com.nt.beans1"/>
        (or)                                                    (c)
  <context:component-scan base-Package="com.nt"/>
                                        (Recomanded)
</beans>
```

import <beans> ,<context> namespace

note:: <context:componet-scan > tag internally maintain behaviour of <context:annotataion-config> ,So no need of placing explicitly..

ClientApp
=========

```
//create IOC container (a) (IOC container creation) (b) (Loading of xml file , checking
ApplicationContext ctx=new CPXAC("com/nt/cfgs/applicationContext.xml"); wellformness, validness-->
//get Bean (i)                                                creating InMemory Meta data of
Flipkart fpkt=ctx.getBean("fpkt",Flipkart.class); (g)        spring bean cfg file (xml file)
//invoke methods
S.o.p(fpkt.shopping(new String[]{"...","..."} , new Float[]{.. , ...} );
(m)          (j)
```

<context:component-scan>
==========

Scans the classpath for annotated components that will be auto-registered as Spring beans. By default, the Spring-provided @Component, @Repository, @Service, @Controller, @RestController, @ControllerAdvice, and @Configuration stereotypes will be detected. Note: This tag implies the effects of the 'annotation-config' tag, activating @Required, @Autowired, @PostConstruct, @PreDestroy, @Resource, @PersistenceContext and @PersistenceUnit annotations in the component classes,

List all stero type annotations
==========
@Component , @Service , @Controller, @RestController ,@ControllerAdvice,@Configuration
@Repository

note:: In xml cfgs driven spring programming the pre-instantation of singleton scope beans takes place in the order of their cfg in spring bean cfg file.. where as in Annotation driven cfgs.. the pre-instanitation takes Alphabetic order of Spring bean class names (With respect to each pakage)

In Annoataion driven env...the default bean id(Object name) for spring bean is
"<bean class name having first letter in lower case>"
eg:: if class name is "Flipkart" ---> default bean id :: "flipkart"
if class name is "EmployeeDAOImpl" ---> default bean id :: "employeeDAOImpl"

```
@Component
public class Flipkart{
  ...
}
```

In client app
==========

```
Flipkart fpkt=ctx.getBean("flipkart",Flipkart.class);  (or)
Flipkart  fpkt=ctx.getBean(Flipkart.class);
```

note:: if class is pre-defined class , we can not cfg it as spring bean using stereo type annotations becoz we can not edit source code any pre-defined class to add strero type annotations.. So we need to cfg that class as spring bean using <bean> tag in spring bean cfg file (xml)

```
@Component("fpkt")
public final class Flipkart {
      @Autowired
      private  Date date;
  ....
  ....
}
```

applicationContext.xml
==========

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

  <bean id="dt"   class="java.util.Date"/>
</beans>
```

Q)Can we do injection on static bean properties/fields/attributes/member variables using any mode of injection including @Auitowired?

Ans) No , we can not..

```
@Autowired    (fails)
private  static Date date;
```

INFO: Autowired annotation is not supported on static fields: private static java.util.Date com.nt.beans.Flipkart.date

@Lazy(true) --> Disables pre-instantation on singleton scope bean and enables lazy instantiation
@Lazy(false) --> continues pre-instantation on singleton scope bean ... default value is "true"

@Lazy(true) is equal to lazy-init="true" attribute of <bean> tag..

```
@Component("dhl")
@Lazy(true)
public final class DHL implements Courier {
  ...
}
```

@Scope
========
It give to specify the scope of Spring bean class object.. the default scope is singleton scope.. In standalone env.. we can work with only two scopes (singleton, prototype).. In web env.. we can use singleton,prototype,session,application,request,websocket scopes.. @Scope is equal to "scope" attribute of <bean> tag..

```
@Scope("prototype")
public class Flipkart{
  ...
  ...
}
```

---

while developing Layered Apps/Projects using Annotation driven cfgs we need to use the following Thum rule

a) Cfg pre-defined /third party supplied classes as spring beans using <bean> of spring bean cfg file
b) Cfg user-defined java classes as spring beans using the support of streo type annotations..or java config annotaitons and link them with spring bean cfg file using <context: component-scan>

note:: while working certain special features for which annotations are not available , we still need to go for xml driven cfgs even through spring bean classes user-defined classes. The special feature are

->inner beans, collection injection, collection merging , bean inheritence , factory method bean instantiation , method replacer and etc...

How can u say xml driven cfgs are not outdated , they are just limited in the utilization ?

a) Still we need web.xml for centain web application cfgs like welcome file, context params, security cfgs and etc..
b) Still we need hibernate cfg file as xml in hibernate programming (no alternate with annotations)
c) In Spring , for the following features we need to use xml driven cfgs...

->inner beans, collection injection, collection merging , bean inheritence , factory method bean instantiation , method replacer and etc...

d) In order to override cfgs done through annotations with out touching the source code, we need to xml files based configurations...

if inject values to simple properties (primitive data type,String type,wrapper datatype) in annotation driven env.. use @Value annotation..

```
@Value("30")
private  int age;
```

note:: we can get simple properties values from properties file by @Value as shown below..

In any bean class

```
@Value("${cust.age}")
private  int age;
```

jdbc.properties

```
cust.age=30
```

note:: Using @Value annotation we can not supply input values to other annotation like @Qualifier, we can just inject values to simple properties either by hardcoding or by getting them properties file...

note:: Through xml file or proerties file the String/text values injected to the simple properties will be converted to appropriate type internally by IOC container by taking the support "PropertyEditors"..

Java Config Annotations in Spring env..

==========

=>Working with spring supplied annotations makes our spring App code/classes as invasive code/clases (Tight coupling with spring api..).To overcome this problem we have java config annotations given by JSE,JEE modules..

=>The Java config Annotations will get specific behaviour based context/env.. where they are being used.. if there are used in spring env.. they give spring specific behaviour.. Similarly in hibernate they hibernate specific behaviour.. and etc.... This helps to make our code non-invasive code (Loosely coupled code with spring api)

eg:: @Named, @Inject , @Resource , @PreDestroy ,@PostConstruct and etc...

While developing annotation driven spring Apps we should use diff annotations having the following priority

=>Java config annotations (Very lesss)
=>Spring supplied Annotations
=> Third party supplied Annotations
=> Custom Annotations

⇨ Indursty does not care this recomanded becoz very lesss java config annotations are avaliable and as now there is not alternate framework for to move spring code to that framework by having non-invasive behaviour..

@Inject

========

=>Alternate to @Autowired having minor differences.. like no "required" attribute i.e depdent bean cfg is always mandatory..

=> To work this annotation we need to add javax.inject-<ver>.jar file (JEE supplied annotation) and we get that from mvnrepository.com

for build.gradle

==========

```
// https://mvnrepository.com/artifact/javax.inject/javax.inject
 implementation group: 'javax.inject', name: 'javax.inject', version: '1'
```

@Named

=========

It is having two angels of utilization

a) To cfg java class as spring bean (As alternate to @Component)

b) To resolve ambiguity problem (as alternate to @Qualifer)

=> To work this annotation we need to add javax.inject-<ver>.jar file (JEE supplied annotation) and we get that from mvnrepository.com

=> It JEE supplied java config annotation..

```
@Named("fpkt")                          To cfg java class as spring bean
public final class Flipkart {
      //property                        For autowiring /implicit injection
      @Inject
      @Named("dhl")                     To resolve ambigutity Problem
      private  Courier courier;  //rule1 ,2
      @Inject
      private   Date date;
        ...
        ...
 }
```

=>@Inject is applicable at Filed level ,constructor level, setter method level and arbitary method level

=> if @Named is applied at class level then it is for cfg java class as spring bean.. if it is applied at field/construcotr/method level to resolve ambguity problem.

@Resource

==========

=> It is JSE/Jdk supplied java config annotation to perform autowiring / implicit depedency injection..

=> It is same as @Autowired but can not be applied at Constructor level i.e can not perform constructor mode autowiring.

=> Since javax.annotation package is removed from jdk9.. we should change the setup to java8 in order to work this annotation..

```
@Resource(name="dhl")
private Courier courier;  //rule1 ,2
```

Here we can resolve ambiguity problem by using "name" attribute of @Resource tag itself .. No need of using seperate @Qualifier or @Named..

=>@Inject is JEE supplied Java config Annotation which should be used by adding javax.inject-<ver>.jar file.. but @Resource can be used directly with out adding any jar file (but upto java8)

=>@Resource is applicable at field level, setter method level and arbitary method level form autowiring.. but not at constuctor level..

note:: if want to use javax.annotation pkg from jdk 9 onwords we need to add the following jar file

javax.annotation-1.2.jar file

Its gradle info is

==========

```
// https://mvnrepository.com/artifact/javax.annotation/javax.annotation-api
compile group: 'javax.annotation', name: 'javax.annotation-api', version: '1.2'
```

What are the differences among @Autowired ,@Resource ,@Inject?

@Autowired	@Inject	@Resource
a) It is spring supplied annotation	a) It Is JEE supplied java config annoation	a) It is jdk supplied java config annotation (but removed from java9 onwords --we should add javax.annotation-1.2.jar to the build path in order use from java9)
b) we need to add spring jars	b) we need to add javax.inject-<ver>.jar	b) no jars are required upto java8 .. but javax.annotation-<ver>.jar is required from java9
c) Can perform byType,byName, constructor mode autowiring	c) same	c) can perform only byType and byName autowiring
d) applicable at fileld , method, constructor level	d) same	d) applicable at fileld, method for injections
e) To solve ambuiguity problem by matching name/bean id we use @Qualifier..	e) To solve ambuiguity problem by matching name/bean id we use @Named	e) To solve ambuiguity problem by matching name/bean id we use "name" attribute of @Resource tag

note:: Since we can not develop entire Spring App using java config annotations... because they are in limited numbers, it is recomanded to use @Autowired for autowirng operations.

Spring Bean life cycle (very very important)

==========

=> IOC container manages spring bean life cycle from birth to death (object creation to obj destruction)

=> Serlvet container manages Servlet comp life cycle from birth to death (object creation to obj destruction)

=> Container raises life cycle events while managing the life cycle

=> Servlet comp life cycle events raised by Servletcotnainer are

a)Instantiation event (raises when Servlet container creates our servlet class obj)

b) Request processing event (raises when Servlet container keeps our servlet comp ready to process the request)

c) Destruction event (raises when Servlet container about to destroy our servlet comp class obj)

ServletContainer calls 3 life cycle methods for 3 life cycle events .. by overriding these 3 life cycle methods in our servlet comp we can make Servlet container executing our logics. as part of life cycle management..

a) Instantiation event ----> public void init(ServletConfig cg)

b) Request processing event ----> public void service(ServletRequest req,ServletResponse res) throws SE,IOE

c) destruction event --> public void destroy() method..

=>IOC Container raises two life cycle events while managing spring bean life cycle and calls two life cycle methods to handle those events...

a) Instantiation event (raises when IOC container creates our spring bean class obj and injections are done)

b) Destruction event (raises when IOC container is about to destroy our spring bean class obj)

Servlet life cycle method name are fixed.. becoz every Servlet comp is tightly bound with Sevlet api (implementation javax.servlet.Sevlet(I) is mandatory) where as spring bean life cycle methods not fixed in defatult setup becoz spring beans non-invasive (not tightly coupled with spring api)

names

we can cfg spring bean life cycle in 3 approaches

a) Using Declarative Approach (xml cfgs)

b) Using Programatic Approach (java code - by implementing spring api interfaces)
(Here life cycle method names fixed)

c) Using Annotation Approach (@PostConstruct, @PreDestroy)

In (a),(c) approaches life cycle method names are user-defined.. they must be cfg explicitly.. otherwise they will not be recognized as life cycle methods...

note:: In the Init life cycle method (Instantation event life cycle method) of spring bean we place the following 3 logics

a) initializing left over properties/fields which are not participating in dependency injection

b) verifying wheather important properties are injected with coorrect values or not ??

c) Bean Posting Processing (correcting injected values)

note:: In the destroy life cycle method (destructione event life cycle method) of spring bean we place clean up logics like nullifing properties and releasing non-java resources...

if

a) Using Declarative Approach (xml cfgs)

=> Allows to develop spring bean class as non-invasive class
=> but we need to cfg life cycle methods explicitly in spring bean cfg by using "init-method" , "destroy-method" attributes of <bean> tag.
=>if the spring bean class is pre-defined class , then we need search for life cycle methods.. to cfgs them explicilty using the the above said attributes
=> The init life cycle method and destry life cycle method creation should follow rules

a) must be public
b) shoud have "void" as the return type
c) should not have params

```
public  void  <method-name>(){
    ....  //logic
    ...
}
```

Example
=======

```
// init life cycle method /custom life cycle method
    public  void myInit() {
        System.out.println("Voter.myInit()");
        dov=new Date();  // initializing left over properties
        boolean flag=false;
        if(name==null) {   // validation  logic
            System.out.println(" name must not be null");
            flag=true;
        }

        if(age<0)
            age=age*-1;  //correcting the values //Bean post processing

        if(age>100) {  //validation  logic
            System.out.println(" age must not >100");
            flag=true;
        }
        if(flag)
            throw new IllegalArgumentException(" invalid inputs");

    }


        // destroy life cycle method/custom destroy method
        public void myDestroy() {
            System.out.println("Voter.myDestroy()");
             //nullify bean properties
            name=null;
            age=0;
            dov=null;
        }
```

applicationContext.xml

```
<bean id="voter"  class="com.nt.beans.Voter"  init-method="myInit"  destroy-method="myDestroy">
      <!-- <property name="name" value="raja"/> -->
      <property name="age"  value="300"/>
 </bean>
```

Limitations of Declarative Approach of Spring bean life cycle

(a) we should remember and cfg life cycle methods .. otherwise no life cycle activaties takes place
(b) While cfg pre-defined, third party supplied java classes as spring beans we should search, identity life cycle methods in order to cfg them..

advantages

(a) we can develop spring bean as non-invasive class..

What is the diff b/w @Required/Cosntructor Injection and Init life cycle method ?

Ans) Using @Required and Constructor Injection we check wheather properties are participating in the injection or not but we can not check wheather valid values are injected or not. whrease by using custom init method we can do validation... bean post processisng(modification of injected data) and initliazing left over properties ..

note:: @Required is depreacted from spring 5.1 in support to constructor injection and init life cycle method..

note:: Only for Singleton scope beans.. the destroy life cycle method executes... not for other scopes.. becoz the other scope bean objects will not placed in the internal cache of IOC container..

note:: instead of cfg same init, destroy life cycle methods for multiple beans...we can cfg only for 1 time as default init, destroy life cycle methods in the <beans> tag as shown below..

```
<beans default-init-method="myInit"   default-destroy-method="myDestroy"  .... >
  <bean id="voter"  class="com.nt.beans.Voter"   scope="singleton">
        <property name="name" value="raja"/>
        <property name="age"  value="30"/>
    </bean>

    <bean id="voter1"  class="com.nt.beans.Voter"   scope="singleton"  >
        <property name="name" value="ravi1"/>
        <property name="age"  value="20"/>
    </bean>
</beans>
```

note ::In case of BeanFactory Container.. the destroy life cycle method will execute only when we call factory.destrotySingletons() method.. becoz there is not provision to stop or close BeanFactory cotainer..

Spring Bean life cycle in Programatic Approach

==========

=> We need to make our spring bean class implementing two interfaces

(a) IntializingBean (I)

|----> public void afterPropertiesSet() ---> alternate custom init method

(b) DisposibleBean (I)

|---> public void destroy() ---> alternate to custom destroy method

=> This apporach makes spring bean as invasive (tight coupling with spring api)

=> In this approache the life cycle methods will be executed automatically by seeing the implementation of interface on spring bean class.. i.e there is no need of configuring life cycle methods any where...

```
public class Voter  implements  InitializingBean,DisposableBean {
        private String name;
        private float age;
        private  Date  dov;

        public Voter() {
                System.out.println("Voter:: 0-param constructor");
        }

        public void setName(String name) {
                System.out.println("Voter.setName()");
                this.name = name;
        }
        public void setAge(float age) {
                System.out.println("Voter.setAge()");
                this.age = age;
        }
        public void setDov(Date dov) {
                System.out.println("Voter.setDov()");
                this.dov = dov;
        }

       //b.method
              public String   checkVotingElgibility() {
                     System.out.println("Voter.checkVotingElgibility()");
                     //b.logic
                     if(age>=18)
                            return  "Mr/Miss/Mrs."+name+ "  u  r  eglible for voting::"+age+"-->  on  ->"+dov;
                     else
                            return  "Mr/Miss/Mrs."+name+ "  u  r not  eglible for voting::"+age+"--> on->"+dov;
              }//method
```

in applicationContext.xml

==========

```
<bean id="voter"  class="com.nt.beans.Voter">
        <property name="name" value="ramesh"/>
        <property name="age" value="20"/>
    </bean>
```

```java
@Override
public void destroy() throws Exception {
        System.out.println("Voter.destroy()");
         //nullify bean properties
        name=null;
        age=0;
        dov=null;
}
```

```java
@Override
public void afterPropertiesSet() throws Exception {
  System.out.println("Voter.afterPropertiesSet()");
  dov=new Date();  // initializing left over properties
        boolean flag=false;
        if(name==null) {   // validation  logic
                System.out.println(" name must not be null");
                flag=true;
        }

        if(age<0)
                age=age*-1;  //correcting the values //Bean post processing

        if(age>100) {  //validation  logic
                System.out.println(" age must not >100");
                flag=true;
        }
        if(flag)
                throw new IllegalArgumentException(" invalid inputs");
}
```

```java
}//class
```

advantages programatic apporach

==========

(a) no need of life cycle methods.. they will be executed automatically not
(b) if any pre-defined class is implementing based this apporach , we need to search and cfg life cycle methods seperately..

Disadvantages

==========

(a) Makes spring bean as invasive ...

(b) we can not make third party supplied classes implementing these interfaces.

if i enable spring life cycle in both programatic and declarative apporach, which will be will be executed?

Ans) Both will execute... but first IOC container calls programatic approach methods later it calls declarative approach methods..

For instantation event --> a) InitializingBean's afterPropertiesSet()
b) custom init method

for Destruction event ---> a) DisposibleBean's destroy()

In DataSourceTransactionManager cass (spring supplied pre-defined class)

```
@Override
  public void afterPropertiesSet() {
        if (getDataSource() == null) {
              throw new IllegalArgumentException("Property 'dataSource' is required");
        }
  }
```

Annotations driven Spring bean life cycle

==========

=> we need to cfg custom init method having @PostConstruct annotation
=> we need to cfg custom destroy method having @PreDestroy annotation
=> Both these annotations are java config annotations.. (given jdk)
=> Makes our spring bean as non-invasive..
=> @PostConstruct and @PreDestroy are part of jdk api upto java 8... from java 9 onwords they are removed (in fact all java config annotations) from jdk to make the jdk s/w as the light weight s/w... So to addtional jar file to the build path..

```
// https://mvnrepository.com/artifact/javax.annotation/javax.annotation-api
implementation group: 'javax.annotation', name: 'javax.annotation-api', version: '1.3.2'
```

=>This approach can not be used while working with BeanFactory container..
=>HEre there no need of configuration life cycle methods.. any where.. Based on the annotations that are added the life cycle method will execute automatically.

```
// init life cycle method /custom life cycle method
@PostConstruct
public  void myInit() {
      System.out.println("Voter.myInit()");
      dov=new Date();  // initializing left over properties
      boolean flag=false;
      if(name==null) {   // validation  logic
            System.out.println(" name must not be null");
            flag=true;
      }

      if(age<0)
            age=age*-1;  //correcting the values //Bean post processing

      if(age>100) {  //validation  logic
            System.out.println(" age must not >100");
            flag=true;
      }
      if(flag)
            throw new IllegalArgumentException(" invalid inputs");

}
```

advantages
==========
(a) makes spring bean as non-invasive
(b) no need of cfg life cycle methods any where.. based on annotations they will be executed automatically

disadvantages

(a) we can not use this approach for third party supplied , pre-defined classes..

```
      // destroy life cycle method/custom destroy method
@PreDestroy
      public void myDestroy() {
            System.out.println("Voter.myDestroy()");
             //nullify bean properties
            name=null;
            age=0;
            dov=null;
      }
```

Q) A Sub class is cfg as spring bean, life cycle methods are not in sub class ,but available in super class .. will they execute as sub class life cycle method when the sub class is cfg as spring bean?

A) Yes..

The methods on which apply @PostConstruct and @PreDestroy annotations should have following signature ...

```
public   void  <method>(<no params>){
   ...
   ..
}
```

What happens if apply all 3 approaches of spring bean life on the spring bean class>

Ans) The init and destroy life cycle methods will execute in the following order
a) Annotation driven b) Programatic c) declarative

Conclusion on spring bean life cycle

==========

a) if the spring bean is user-defined class , then go for annotation driven approach for life cycle management
b) if the spring bean is spring supplied pre-defined class,then check for InitializingBean, DisposibleBean interfaces implementation...
if implemented automatically programatic approach continues.. if not implemented go for declarative approach
c) if spring bean is thrird party supplied class , then go for declarative approach..

How did u use spring bean life cycle in u r project?

Ans) in controller, service ,dAO classes we use init life cycle metho to check wheather imp bean properties are injected or not .. and destroy life cycle mdthod to nullify bean properties..

For example ::

In service class .. we use init life cycle method to check DAO objects are injected or not...
we use destroy life cycle method to nullify the injected DAO objects..

```
@PostConstruct
    public void myInit() {
        if(dao==null)
            throw new IllegalArgumentException("dao not injected");
    }

    @PreDestroy
    public void myDestroy() {
        dao=null;
    }
```

Aware Injection /Inteface Injection/ Contexual Dependency Lookup

==========

Diff modes of depdency injections

==========

- setter injection
- constructor injection
- Aware Injection
- LookupMethod Injection
- Method Replacer/Method Injection

when should we go dependency lookup and when should go for depedency injection (setter/Constructor Injection)?

=> if dependent obj is required only in one method of target class then go for dependency lookup.

eg:: Viechle(target class) ---->Engine(dependent)

|-->Engine is required only in the move() of target class not in other methods.. So go for dependency lookup.

|---> To implement it, create an extra IOC container in the specific one method of target class.. and call getBean(-,-) using that extra IOC container

=> if dependent bean obj is reuqired in multiple methods of target class then go for dependency injection (setter/constuctor injection)

|---> Cricketer(target) -----> Ball (dependent)

|--->Ball is required in multiple methods of Cricketer class like batting(), bowling(), fiedling(), wicketKeeping() and etc.. so go for dependecy injection

---

if we perform tradtional Dependecy lookup by Using Base Container and Addtional Container as ApplicationContext cotnainer then we need to enable lazy-init for singleton scope taget, dependent bean classes.. then unnecessary pre-instantation on singleton beans for multiple times will be avoided..

```
<!-- Dependent bean class -->
 <bean id="engg1"  class="com.nt.beans.Engine"  lazy-init="true"/>

 <!-- target bean class -->
 <bean id="vehicle"   class="com.nt.beans.Vehicle"  lazy-init="true">
    <!--  <constructor-arg  value="engg2"/> -->
      <constructor-arg> <idref bean="engg1"/>  </constructor-arg>
 </bean>
```

@Lazy annotation

Limitations of Tradtional dependency Lookup

(a) Taking extra IOC container in the specific method of target class is bad
(b) The Injected Dependent class bean id in target class is having more visibility in multiple methods of target class. (Actually it is required only in one method of target class)
(c) if the IOC container is ApplicationContext.. it forces to disable pre-instantiation of singleton by using lazy-init="true" or by using @Lazy

The spring beans that are annotated with Stereo type annotations or @Named Annotation will be instantiated using parameterized constructor only when @Autowired is placed on the top of paramterized cosntructor.. otherwise they will be instantiated using 0-param constructor.. so make sure that 0-param constructor is always available.

To overcome the above problems of tranditional depenency lookup we can use
==========

(a) Desing specific method taking Client Supplied Container object

In Vehicle.java

```
public void journey(String sourcePlace , String destPlace,
                    ApplicationContext ctx) {
  //get Dependent class object
   Engine engg=ctx.getBean(beanId,Engine.class);
   //use Dependent
    .....
    ....
}
```

In client App

```
Vehicle vehicle=ctx.getBean("vehicle",Vehicle.class);
vehicle.journey("hyd","warangal",ctx)
```

Here @Lazy or lazy-init="true" is not required

The big limitation here is designing business method with Spring Specific IOC container object.. This kills non-invasive behaviour of target class.

(b) Design the spefic method target class taking Client created IOC container object and Dependent bean class id.

In Vechicle.java
==========

```
public void journey(String sourcePlace , String destPlace,
                    ApplicationContext ctx,String beanId) {
  //get Dependent class object
  Engine engg=ctx.getBean(beanId,Engine.class);
   ....
   ...
}
```

In client App
==========

```
vehicle=ctx.getBean("vehicle",Vehicle.class);
vehicle.journey("hyd","warangal",ctx,"engg"); //version3
```

=>Here @Lazy or lazy-init="true" not required
=>No need of taking property in target bean class as instance variable to hold dependent bean id
=> This is bad pratice.. b.method and spring bean is becoming -invasive becoz Container object, bean id..

get Client created IOC Container to Taget Bean class using Aware Injection.. and use it specific method of taget bean class...

IOC container injects special values to spring bean based on the XxxAware Interface that is implemented by spring bean..

XxxAware interfaces are

BeanNameAware ---> To inject current spring id/name to spring bean itself
BeanFactoryAware --> To inject underlying BeanFactory object
ApplicationContextAware-> To inject underlying ApplicationContext object
and etc..

note::Aware injection is not for injecting simple values or dependent bean objects .. it is object injecting IOC container maintained special values..

note:: IOC container internally maintains some special values along with Spring bean objects like bean ids , BeanFactory object, ApplicationContext object and etc.. To inject these values go aware Injection

BeanNameAware (I)
|---> p v setBeanName(String beanId)
BeanFactoryAware (I)
|---> p v setBeanFactory(BeanFactory factory)
ApplicationContextAware (I)
|---> p v setApplicationContext(ApplicationContext ctx)

=>These are not setter methods of setter injection .. These are fixed methods declared in the interfaces..

=>setter method will be there for each property. For example , if the property name x .. then setter method will be setX(-) method

Example code Vechicle .java

```
public class Vehicle implements ApplicationContextAware{
    private String beanId;
    private ApplicationContext ctx;

    public Vehicle(String beanId) {
        this.beanId=beanId;
        System.out.println("Vehicle:: 1-param constructor");
    }

    @Override
    public void setApplicationContext(ApplicationContext ctx) throws BeansException {
        System.out.println("Vehicle.setApplicationContext(-)");
        this.ctx=ctx;
    }

    public void entertainment() {
        System.out.println("Vehicle is equipped with DVD Player for entertainment");
    }

    public void soundHorn() {
        System.out.println("Vehicle is equippled with skoda horn ");
    }

    public void fillFuel() {
        System.out.println("Vehicle is having fuel tank of 50 liters");
    }

    public void journey(String sourcePlace , String destPlace) {
        System.out.println("Vehicle.journey()");
        Engine engg=null;

        //get Dependent class obj (using looup operation)
        engg=ctx.getBean(beanId, Engine.class);
        //use dependent object
        engg.start();
        System.out.println("journey started at ::"+sourcePlace);
        System.out.println("journey is going on..... from "+sourcePlace+" to "+destPlace);
        engg.stop();
        System.out.println("jounery ended at ::"+destPlace);
    }
}
```

Aware Injection

Not setter method for setter injection
It is method declared in ApplicationContextAware(I)

(No extra container is taken here)

order of execution
==========
=> Bean Instantation (constuctor Injection)
=>setter injection
=>Aware Injection (based on XxxAware interfaces that are implemented)
=>@PostConstruct method (if available)
=> IntializingBean's afterPropertiesSet() (if avaiable)
=> custom init method (if configured)
.
.

}

=>Aware Injection is aslo called Interface injection becoz the injection is taking place only after implementing special XxxAware(I) on spring bean class.
=>Aware Injection is aslo called Contextual dependency lookup/Injection becoz the injection is taking place only after implementing special XxxAware(I) on spring bean class as the contract /context to be agreed /satisfied by spring bean class.

=>Servletcontainer injects ServletConconfig object to servlet class obj whose class implementing/ satisfying javax.servlet.Servlet(I) directly or indirectly.. So we can this process as Inteface injection/Contextual Dependency lookup

Limitations of AwareInjection based Depedency lookup
==========

a) makes the spring bean class as invasive (becoz we need to implement XxxAware (I))
b) The Injected BeanFactory/ApplicationContext object visible through out target class though it is required in one method
b) The Injected Dependent class bean id is visible through out target class though it is required in one method.

Advantages
==========

=>No need of taking Extra container in target class method
=> No need of enabling lazy init on spring beans.. (lazy-init=true or @Lazy not required)
=> No need of desiging b.method with technical inputs like taking Container object ,bean id as the args

Conclusion::: Go LMI (Lookup method to solve all problems) with out any side effects...

Lookup Method Injection (LMI)

==========

Problem:: if taget spring bean scope is singleton and dependent spring bean scope is prototype... some how dependent spring bean acts as singleton only..

AI -Aware Injection
DL -->dependency lookup

taget bean scope	dependent scope	resultant scope of dependent bean
singleton	singleton	singleton (OK)
singleton	prototype	singleton (not OK) ⇨ (use DL or AI+DL or LMI to solve this problem)
prototype	singleton	singletin (OK)
prototype	prototype	prototype (OK)

=> One ServletContainer/WebContainer takes all requests... but it uses multiple objects of RequestHandler to handle/process multiple requests.. if we design them as spring beans...

Servletcontainer/WebContainer <----------> RequestHandler
(target class) (singleton scope) (Dependent class)(prototype scope)

⇩

(if we use setter/ constructor injection to inject dependent object to target object.. some how prototype scope dependent bean acts as singleton scope bean)

Faculty <----------> Student
(target-singleton) (dependent-prototype)

Conclusion:: do not use setter ,constructor Injections in the following two situations

==========

(a) if Dependent object is required only in one method of target class
(b) if target spring bean scope is singleton and dependent spring bean scope prototype.

⇨ **(Here prefere using Traditional DL (or) AI +DL (or) LMI (best)**

```
<!-- Dependent bean cfg -->
  <bean id="handler"  class="com.nt.beans.RequestHandler"  scope="prototype"/>

 <!-- Target  bean cfg-->
 <bean id="container"  class="com.nt.beans.WebContainer"  scope="singleton">
    <property name="rh"  ref="handler"/>
 </bean>
```

To solve the above problems *refer IOCProj56-LMIProblem*

Approach1) Use Traditional lookup by taking an IOC container (extra Dependecy)

In WebContainer.java (target class)

```
public  void  processRequest(String data) {
        ApplicationContext ctx=null;
        RequestHandler handler=null;
        System.out.println("WebContainer is processing request with data::"+data+ "by giving it to handler");
        //create Extra IOC cotainer
        ctx=new ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");
        //get Dependent object using dependency lookp
        handler=ctx.getBean(beanId,RequestHandler.class);
        handler.handleRequest(data);
}
```

In applicationContext.xml file

```
<!-- Dependent bean cfg -->
  <bean id="handler"  class="com.nt.beans.RequestHandler"  scope="prototype"/>

 <!-- Target  bean cfg-->
 <bean id="container"  class="com.nt.beans.WebContainer"  scope="singleton"  lazy-init="true">
    <property name="beanId">
       <idref  bean="handler"/>
    </property>
 </bean>
```

refer IOCProj57-LMI-Solution1-TDL

Limitations of Approach1

a) taking an extra IOC container in specific method of target bean class is bit heavy and kills the performence..
b) the Injected bean id of Dependent class is visible in mulitple methods of target class unneccessarly.
c) singleton scope target bean will be pre-instantiated for multiple times .. becoz the extra IOC container (This can be solved by using lazy-init="true" or @Lazy)

Approach2 Aware Injection + Traditional dependency lookup with out taking extra IOC container

WebContainer.java

```
public class WebContainer  implements ApplicationContextAware {
    private String beanId;
    private ApplicationContext ctx;

    public void setBeanId(String beanId) {
        this.beanId = beanId;
    }

    @Override
    public void setApplicationContext(ApplicationContext ctx) throws BeansExceptio
        System.out.println("WebContainer.setApplicationContext(-)");
        this.ctx=ctx;
    }

    public WebContainer() {
        System.out.println("WebContainer:: 0-param constructor");
    }

    public   void  processRequest(String data) {
        RequestHandler handler=null;
        System.out.println("WebContainer is processing request with data::"+data+ "by giving it to handler");
        //get Dependent object using dependency lookp
        handler=ctx.getBean(beanId,RequestHandler.class);
        handler.handleRequest(data);
    }
}
```

applicationContext.xml

```
<!-- Dependent bean cfg -->
  <bean id="handler"  class="com.nt.beans.RequestHandler"  scope="prototype"/>

 <!-- Target  bean cfg-->
 <bean id="container"  class="com.nt.beans.WebContainer"  scope="singleton">
    <property name="beanId">
        <idref  bean="handler"/>
    </property>
 </bean>
```

LMI --- Lookup Method Injection Internal flow

==========

ResquestHandler.java (Dependent class)

==========

```
package com.nt.beans;
public class RequestHandler {
   private  static int count;
   public RequestHandler() {
       count++;
             System.out.println("RequestHandler:: 0-param consturctor ::::"+count);
       }                    (v)
   public  void handleRequest(String  data) {
       System.out.println("handling request with "+data +"using the object ::"+count);
   }
}
```

WebContainer.java (Target class)

```
public abstract class WebContainer  {
       public WebContainer() {
             System.out.println("WebContainer:: 0-param constructor ::");
       }
       public abstract   RequestHandler  createRequestHandler();
       public   void  processRequest(String data) {   (m)
             RequestHandler handler=null;
             System.out.println("WebContainer is processing request with data::"+data+ "by giving it to handler");
             //get Dependent object using dependency lookup code generated by the container...
        (t)  handler=createRequestHandler();   (n)
             //use dependent object..
             handler.handleRequest(data);  (u)
       }   (w)
}
```

applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans https://www.springframework.org/schema/beans/spring-beans-4.3.xsd
            http://www.springframework.org/schema/context https://www.springframework.org/schema/context/spring-context-4.3.xsd">

  <!-- Dependent bean cfg -->
   <bean id="handler"  class="com.nt.beans.RequestHandler"  scope="prototype"/>

 <!-- Target  bean cfg-->                                    (d)
 <bean id="container"  class="com.nt.beans.WebContainer"  scope="singleton">
       <lookup-method  name="createRequestHandler"   bean="handler"/>
 </bean>                                          (e)

</beans>
```

(d) becomes ready to perform pre-instantion singleton scope beans

InMemoryProxy class Genrated by CGLIB Libraries at runtime

(f)--InMemory Proxy class generation by seeing <lookup-method> tag..

```
pubic class   WebContainer$CGLIB$Proxy  extends  WebContainer  implements  ApplicationContextAware{
  private  ApplicationContext ctx;
   public  void setApplicationContext(ApplicationContext ctx){
      this.ctx=ctx;        (g)  (aware injection)
    }

    public  RequestHandler createRequestHandler(){     (o)

        //get Bean id from  the  bean attribute of   <lookup-method> tag   (handler)
           ....  .....
        //get Dependent class object through dependency lookup operation
 (r)     RequestHandler handler=ctx.getBean("handler",RequestHandler.class);  (p)
new obj) return handler;  (s)
     }
```

Internal cache of IOC container (h)

Limitations of Approach2

a) implementation of ApplicationContextAware makes target bean as invasive

b) the Injected bean id of Dependent class is visible in mulitple methods of target class unneccessarly.

c) The Injected ApplicationContext object through aware Injection is visible in all the methods of Target bean unneccessarly..

advantages
==========
(a) no need of taking extra IOC container ,So gives good performence
(b) no need of enabling lazy-init="true" or @Lazy on target bean class..

Approach3 Use Lookup Method Injection (perform dependency lookup but u do not do that.. let the IOC container do that work internally)
==========

step1) Take target spring bean class as abstract class having one abstract method whose return type is dependent bean class/type. (This method is called as lookup method becoz IOC container internally implements this method having dependency lookup to get and return a Dependent class obj in the IOC container generated In Memory sub class for target class)

In WebContainer.java (abstract class)

```
public abstract RequestHandler createRequestHandler();
```

step2) cfg the above method lookup method in spring bean cfg file....

```
<!-- Dependent bean cfg -->
  <bean id="handler" class="com.nt.beans.RequestHandler" scope="prototype"/>

 <!-- Target bean cfg-->
 <bean id="container" class="com.nt.beans.WebContainer" scope="singleton">
     <lookup-method name="createRequestHandler" bean="handler"/>
 </bean>
```

step3) use the dependent class bean object in target class b.method by getting that object through method call..

In WebContainer.java

```
public void processRequest(String data) {
        RequestHandler handler=null;
        System.out.println("WebContainer is processing request with data::"+data+ "by giving it to handler");
        //get Dependent class obj
       handler=createRequestHandler();
        handler.handleRequest(data);
    }
```

By seeing <lookup-method> tag under <bean> , the IOC container internally generates one InMemory sub class for cfg bean class (in our case for WebContainer class) in JVM memeory of RAM implemented the cfg method (in our case createRequestHandler method given in "name" attribute) having dependency lookup logic to get and return specified bean id based Dependent bean class obj (in our case it is "handler" given in "bean" attribute)

Advantages of Approach3
==========
a) Both and target and depedent bean classes are non-invasive
b) Container takes care of Dependency lookp by generating one InMemory class as sub class of target class..
c) No need of extra IOC container
d) No need of injecting Dependent class id..
d) No need of enabling lazy-init or @Lazy for Target bean class..
e) It is industry standard..

note:: we can use @Lookup as alternate <lookup-method> tag in annotation driven env...

```
//@Lookup   // alternate to <lookup-method>
 @Lookup("handler")
 public abstract   RequestHandler  createRequestHandler();
```

Client App

==========

```
public class LMISolutionTest {    (a)
    public static void main(String[] args) {
        ApplicationContext ctx=null;
        WebContainer wc=null;
        //create IOC container   (b)->IOC container creation        (c) creating InMemory MetaData
        ctx=new ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");
        //get target bean class object
(k)     wc=ctx.getBean("container",WebContainer.class);   (i)
        //invoke method
(x)     wc.processRequest("hello");   (l)
        System.out.println("----------------------");
        wc.processRequest("hai");
        System.out.println("----------------------");
        wc.processRequest("123");

        //close container
        ((AbstractApplicationContext) ctx).close();
    }

}
```

Internal cache of IOC container (h)

container	WebContainer$CGLIB$Proxy class obj (j?) (q?)

Can we configure abstract class as spring bean?

Ans) Basically not possible.. becoz instantiation(object creation) for abstract class is not possible.. but we can confiure it as spring bean by enabling static factory method bean instantiation or Lookup method injection.. In both cases IOC container does not create object for the cfg abstract class... It will create object for other class/generated sub class..

eg1:: `<bean id="cal" class="java.util.Calendar" factory-method="getInstance"/>`

Here IOC container gets object for GregorianCalendar by calling Calendar.getInstance() method makes that obj as spring bean ..

eg2:

```
<bean id="container" class="com.nt.beans.WebContainer" scope="singleton">
    <lookup-method name="createRequestHandler" bean="handler"/>
</bean>
```

Can we take the target class of LMI Application as an interface instead of abstract class?

Ans) possible.. but we need to make b.methods as default or static methods.. by having java version 8+ . This indirectly allows us to cfg java8+ interface as sprinbean.. but IOC container creates the impl class of interface and makes it as spring bean object ...

```
public interface WebContainer {

    RequestHandler createRequestHandler();

    default void processRequest(String data) {
        RequestHandler handler=null;
        System.out.println("WebContainer is processing request with data::"+data+ "by giving it to handler");
        //get Dependent object using dependency lookup code generated by the container...
        handler=createRequestHandler();
        //use dependent object..
        handler.handleRequest(data);
    }
}
```

In applIcatIonContext.xml

```
<!-- Dependent bean cfg -->
  <bean id="handler" class="com.nt.beans.RequestHandler" scope="prototype"/>

<!-- Target bean cfg-->
<bean id="container" class="com.nt.beans.WebContainer" scope="singleton">
      <lookup-method name="createRequestHandler" bean="handler"/>
</bean>
```

Interface

Here the proxy class comes as Impl class of the WebContainer(I)..

Can we cfg java interface as spring bean?

Ans) Not possible upto 7 ... but possible from java8 ..

note:: Here IOC container does not create object for interface... It actually creates object for impl class.. and makes it as spring bean

---

var --> java 10 (type inferen

Method Injection or Method Replacer

Env.. to run the App

(a) Dev Env.. (Programmers) | (Before release)

(b) Test Env.. (Tester +Programmers) |

(c) UAT env/ Pilot Env / Sanity Test (Client org -IT dept emps/testers) | (after release)

(d) Prod env.. (Project in Live --> endusers are using) |

UAT --->User Acceptence Test..

To modify the b.logic for tempoary period and to revert back to orginal logics we can use the following techniques (With out using Spring)

==========

(a) Comment the source of b.logic to write new logic and uncomment it to revert back to orginal logics

=> we need to touch soruce code of main class/target class/service directly...

=>Some times client org will not get access to soruce code becoz s/w company does not deliver source code to Client org.. (No decompiler is 100% accurate)

=>Any code u modify should go through testing.. regressively...

(b) Write sub class for service class/target class where b.logic is there and override b.methods in the sub class having new logics and use target class b.logic to revert back to orginal logics

=> No need of touching source code of target class/main class/ service class to go to new logics or to revert back to old logics

=> We need touch the source code other classes like controller or client App to chanage from main class/target class/service to sub class (for new logics) and vice-versa (for reverting old logics)

=>Some times client org will not get access to soruce code becoz s/w company does not deliver source code to Client org.. (No decompiler is 100% accurate)

=>Any code u modify should go through testing.. regressively...

Usecase 1::

Bank ==>LoanMela

standard roi :: 24%

offer roi :: 15% (offer period:4)

Bank ---- Loan mela

standardard intrest :: compound

offer intrest :: simple

Usecase2::

E-commerice web sites

regualar days :: standard prices

sale days /offer days :: upto 30% discount

Method replacer or method Injection

=>To overcome the above two problems..in spring we can use method replacer/method injection concept..

Steps to work with Method Replacer /Method Injection

==========

step1) develop target class/main class/service class having b.logic in methods

step2) cfg target class as spring bean in spring bean cfg file..

step3) Develop Method Replacer class by implementing MethodReplacer(I) keep new b.logic for b.method in reimplement(-,-,-) method..

step4) cfg the above Method Replacer class as spring bean in spring bean cfg file

step5) Link Method replacer with target bean cfg by placing <replaced-method> under <bean> tag..

org.springframework.beans.factory.support.MethodReplacer (I)

public Object reimplement(Object obj,Method method,Object[] args) throws Throwable

Object obj---> holds main classobject ref
Method method --> holds taget b.method details
Object[] args ---> holds target method b.method args value

These are given to be used in reimplement(-,-,-) while writing new logics

simple intrest amount formulae :: ptr/100

compound intrest amount formulae :: $p(1+r/100)^t - p$

```
<!-- target/main class cfg -->
  <bean id="bank" class="com.nt.target.BankLoanMgmt">
    <replaced-method name="calculateIntrestAmount" replacer="bankCIAR">
      <arg-type>float</arg-type>
      <arg-type>float</arg-type>
      <arg-type>float</arg-type>
    </replaced-method>
  </bean>

  <!-- Method Replacer cfg -->
  <bean id="bankCIAR" class="com.nt.replacer.BankLoanMgmt_CaculateIntrestAmountReplacer"/>
```

b.method signature

Refer IOCProj61-MethodReplacer

note:: As of now no annotation is given for method replacer as equalent to <replaced-method> tag.

Method replacer or method Injection and LMI are designed based proxy design pattern.. becoz they internally proxy class having new logics or addtional responsibilities...

While working with method injection or replacer

==========

(a) We can not take target class as final class .. becoz the generated proxy class/InMemory class comes as the sub class of the target class ... and final classes can not be sub classes... The generated exception is

```
Exception in thread "main"
org.springframework.beans.factory.BeanCreationException: Error creating
bean with name 'bank' defined in class path resource
[com/nt/cfgs/applicationContext.xml]: Instantiation of bean failed; nested
exception is java.lang.IllegalArgumentException: Cannot subclass final class
com.nt.target.BankLoanMgmt
```

(b) since final methods can not be overridden the sub classes... So we can not take target methods in target class (for the methods for which we want to write method replacer) as final methods...
(This will not give exception... but always target method will excute..and method replacer logics will not execute)

(c) Though we can write new logics for multiple target methods in single reimplement(-,-,-) of a Method Replacer class , it is recomanded to take seperate replacer class for every target method.

[if needed , it is recomanded to take seperate Method replacer for every target method/b.method]

(d) There is no annotation for for <replaced-method> tag, So while working with annotations..target class must be cfg using Xml <bean> in order to work with Method Replacer.. though the targt class is user-defined class..

(e) the <arg-type> tags under <replaced-method> are use-ful to cfg method replacer for only 1 form of target method .. even though there multiple overloded forms of target method...

(f) The underlying IOC container generates Proxy class /InMemory sub class for the target class only when <replaced-method> tag is present under <bean> tag that configures target spring bean class..

Internal flow of Method Replacer Example

==========

BankLoanMgmt.java

```
public  class BankLoanMgmt {
        public   float    calculateIntrestAmount(float pAmt,float rate,float time) {
                System.out.println("BankLoanMgmt.calculateIntrestAmount() ::: compound Intrest Amount");
                return  (float)(pAmt*Math.pow(1+rate/100, time))-pAmt;
        }
}
```

BankLoanMgmt_CaculateIntrestAmountReplacer .java

```
@Component("bankCIAR")     (d)
public class BankLoanMgmt_CaculateIntrestAmountReplacer implements MethodReplacer {

        @Override
        public Object reimplement(Object obj, Method method, Object[] args) throws Throwable {   (O)
                System.out.println("BankLoanMgmt_CaculateIntrestAmountReplacer.reimplement(-,-,-) :: simple Intrest amount");
                float pAmt=0.0f;
                float time=0.0f;
                float rate=0.0f;

                //get  target /orginal method arg values
                pAmt=(float)args[0];
                rate=(float)args[1];
                time=(float)args[2];
                //write new logics  (Simple Intrest amount)
   (P)          return pAmt * rate* time/100.0f;
        }

}
```

applicationContext.xml

```
<beans  .....>
  <!-- target/main class cfg -->
   <bean id="bank"  class="com.nt.target.BankLoanMgmt">          (d)
     <replaced-method  name="calculateIntrestAmount"  replacer="bankCIAR">
        <arg-type>float</arg-type>
(e)     <arg-type>float</arg-type>
(observes)  <arg-type>float</arg-type>
     </replaced-method>
   </bean>

   <context:component-scan base-package="com.nt.replacer"/>     (d)
</beans>
```

(d) pre-instantation of singleton scope beans..

IOC contaner generated InMemory Proxy class

```
(f)
public class   BankLoanMgmt$$CGLIB$$Proxy extends BankLoanMgmt  implementsn ApplicationContextAware{

  private ApplicationContext ctx;

    public void setApplicationContext(ApplicationContext ctx){
      this.ctx=ctx;       (g)    (aware Injection)
    }

    (L)
    public  float  calculateIntrestAmount(float pAmt,float rate,float time){
      //get Replacer bean id from <replaced-method> tag using xml api..  from InMemoryMetaData
       // of spring bean cfg file..  ( gets  bankCIAR )

       //get Method replacer class object... by using  the above Bean id..
(M)     MethodReplacer replacer=ctx.getBean("bankCIAR", MethodReplacer.class);
       //gets Target class  object
       BankLoanMgmt target=ctx.getBean("bank",BankLoanMgmt.class);
       //get target Method  info....
       Method method=target.getClass().getDeclaredMethod("calculateIntrestAmount");;  (reflection api)
       //get target method args..
        Object args[]=new Object[]{pAmt,rate,time};
       //invoke reimplement(-,-,-) on replacer object..    (N)
(Q)    float retVal=replacer.reimplement(target,method,args);

       return retVal;  (r)
    } //method

} //class
```

=>Normal classes related .java ,class files allocate memory in the harddisk.. where InMemory proxy classes source and compiled generated takes place in JVM memory of RAM .. So we can not see them.

Spring IOC container use CGLIB Libraries to generate the proxy classes... which are now part of spring jar/libraries from spring 3.x ... (In older versions .. we used supply CGLIB Libraries seperately)

CGLIB --->Code Generation Libraries...

Each object of java.lang.reflect.Method class holds info about one java method..

(i?)

Internal cache of IOC container (g)

bank	BankLoanMgmt$$CGLIB$$Proxy obj ref
banCIAR	BankLoanMgmt_CaculateIntrestAmountReplacer obj ref

(u)

MethodInjectionTest.java

```
public class MethodInjectionTest {
      public static void main(String[] args) {   (a)
            ApplicationContext ctx=null;
            BankLoanMgmt bank=null;
             //create IOC container          (b) Container creation      (c) InMemory Meata
            ctx=new ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");
            //get target class object
      (j)   bank=ctx.getBean("bank",BankLoanMgmt.class);   (h)
            System.out.println(bank.getClass()+"  " +bank.getClass().getSuperclass());
            //invoke methods
      (s)   System.out.println("Intrest Amount:::"+bank.calculateIntrestAmount(100000, 2, 12));   (k)

            //close container
            ((AbstractApplicationContext) ctx).close();    (t)
       }
}
```

Why the proxy class is comming as the sub class of Target class?

Ans) Since super reference variable can refer Sub class object... they made target class as the super class for Proxy class.. So that we can use target class ref variable to refer Runtime generated proxy class object... otherwise we can not get proxy class obj in our client Apps..

FactoryBean
==========

Target Bean -----> NormalBean (Dependent) :: Injected Bean to Target Bean :: Normal Bean obj
Target Bean ------> FactoryBean (dependent) :: Injected Bean to taget Bean is the FactoryBean created/gathered Resultant object

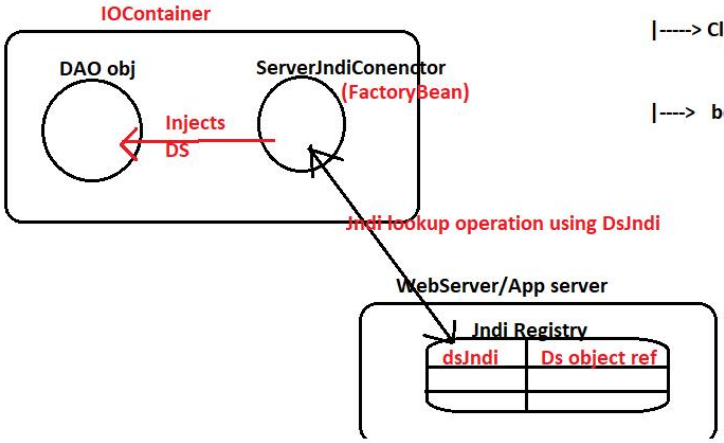
FactoryBean ----> It is a spring bean acting selfless bean.. i.e it never gives its obj , it always gives resultant object... (This object comes becoz certain logics execution)

DAO (target Bean) ---------> ServerJndiConnector (dependent)
|----> gets DS object from Server managed Jndi registry using jndi code
|---> DS is resultant object..

=>Here we should take ServerJndiConnector as FactoryBean , So though it is cfg is as Dependent Bean to DAO.. the DAO will not be injected with ServerJndiConnector object.. It will be injected with ServerJndiConnector supplied DS (DataSource object) (as resultant object) as Dependent object..

==>Java class acts FactoryBean only when it implements
org.springframework.beans.factory.FactoryBean<T> (I) (java 8 interface)
(upto spring 4 normal interface)
(from spring 5 java 8 interface)

|----> Object getObject() (Abstract method)
(place logic to gather/create resultant object)
|-----> Class<?> getObjectType (Abstract method)
(place logic return object class java.lang.Class having Resultant object class name)
|----> boolean isSingleton() (default method)
(specifies wheather the resultant object is singleton scope bean or not)
true --->singleton (scope)
false ---> prototye (scope)
(default impl of this method returns true always)



Schedule Remainder (Target Bean) <----------> DateFactoryBean (Dependent Bean)
|---> remainder()
Local
Here Date class obj will be injected..
Local
|--->Date class object (resultant object)

refer IOCProj63-FactoryBean-POC

POC example ::

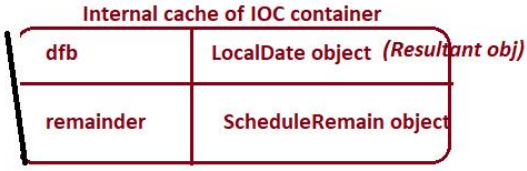
java.util.Date class most of cosntructors and methods are deprecated... So do not use that class.. and prefer iusing the following
a) java.util.Calendar
b) java.time.LocalDateTime
c) java.time.LocalDate
d) java.time.LoalTime
(from Java 8)

How FactoryBean class is giving Resulttant object as spring bean?

=> As Pre-instantation or Lazy instantiation , first the FactoryBean class object will be created , complets all injection process but before keeping FactoryBean class object in the internal cache of IOC container it checks Bean is normalBean or FactoryBean (Based FactoryBean(I) is implemented or not) if it is FactoryBean then it will not keep current FactoryBean obj in the internal cache.. More ever it calls getObject() method on the cureent FactotryBean object.. to get Resulttant object and keeps that Resulttant object in the IOC container internal cache having the bean id of FactoryBean ... as shown below..

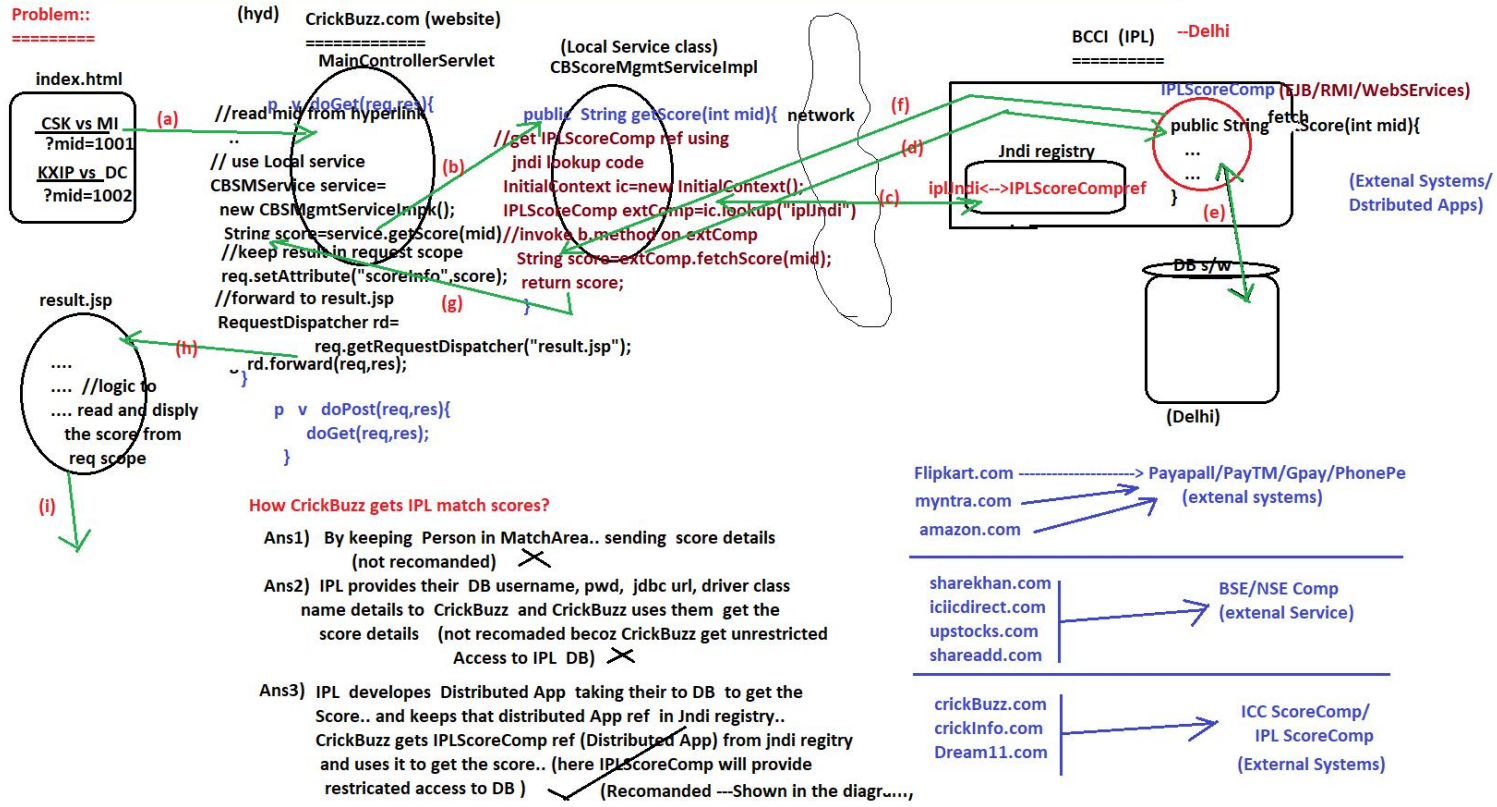
in applicationContext.xml

```
<!-- Dependent bean -->
  <bean id="dfb"  class="com.nt.beans.DateFactoryBean">
     <constructor-arg value="2020"/>
     <constructor-arg value="9"/>
     <constructor-arg  value="30"/>
  </bean>

  <!-- target bean -->
  <bean id="remainder"  class="com.nt.beans.ScheduleRemainder">
    <constructor-arg  ref="dfb"/>
  </bean>
```



note:: IOC container creates FactoryBean class object.. to get call getObject() and other methods on it.. but it will not be kept in internal cache of IOc cointainer..

```
package com.nt.beans;
import java.time.LocalDate;
import org.springframework.beans.factory.FactoryBean;
public class DateFactoryBean implements FactoryBean<LocalDate> {
        private  int year;
        private  int month;
        private int day;
        public DateFactoryBean(int year, int month, int day) {
          System.out.println("DateFactoryBean.DateFactoryBean()");
              this.year = year;
              this.month = month;
              this.day = day;
        }
        @Override
        public LocalDate getObject() throws Exception {
              System.out.println("DateFactoryBean.getObject()");
              return  LocalDate.of(year,month,day);
        }

        @Override
        public Class<?> getObjectType() {
              return LocalDate.class;    // returns the Object of java.lang.Class
having  LocalDate as class name
        }

        @Override
        public boolean isSingleton() {
              System.out.println("DateFactoryBean.isSingleton()");
          return false;   //decide this  based getObject() method logic...
                          //of getObject() method returs single object always then
go for true otherwise go for false..
        }
}
```



How CrickBuzz gets IPL match scores?

Ans1) By keeping Person in MatchArea.. sending score details (not recomanded) ✗

Ans2) IPL provides their DB username, pwd, jdbc url, driver class name details to CrickBuzz and CrickBuzz uses them get the score details (not recomaded becoz CrickBuzz get unrestricted Access to IPL DB) ✗

Ans3) IPL developes Distributed App taking their to DB to get the Score.. and keeps that distributed App ref in Jndi registry.. CrickBuzz gets IPLScoreComp ref (Distributed App) from jndi regitry and uses it to get the score.. (here IPLScoreComp will provide restricated access to DB) ✓ (Recomanded ---Shown in the diagr....,

Flipkart.com
myntra.com
amazon.com
----------> Payapall/PayTM/Gpay/PhonePe (extenal systems)

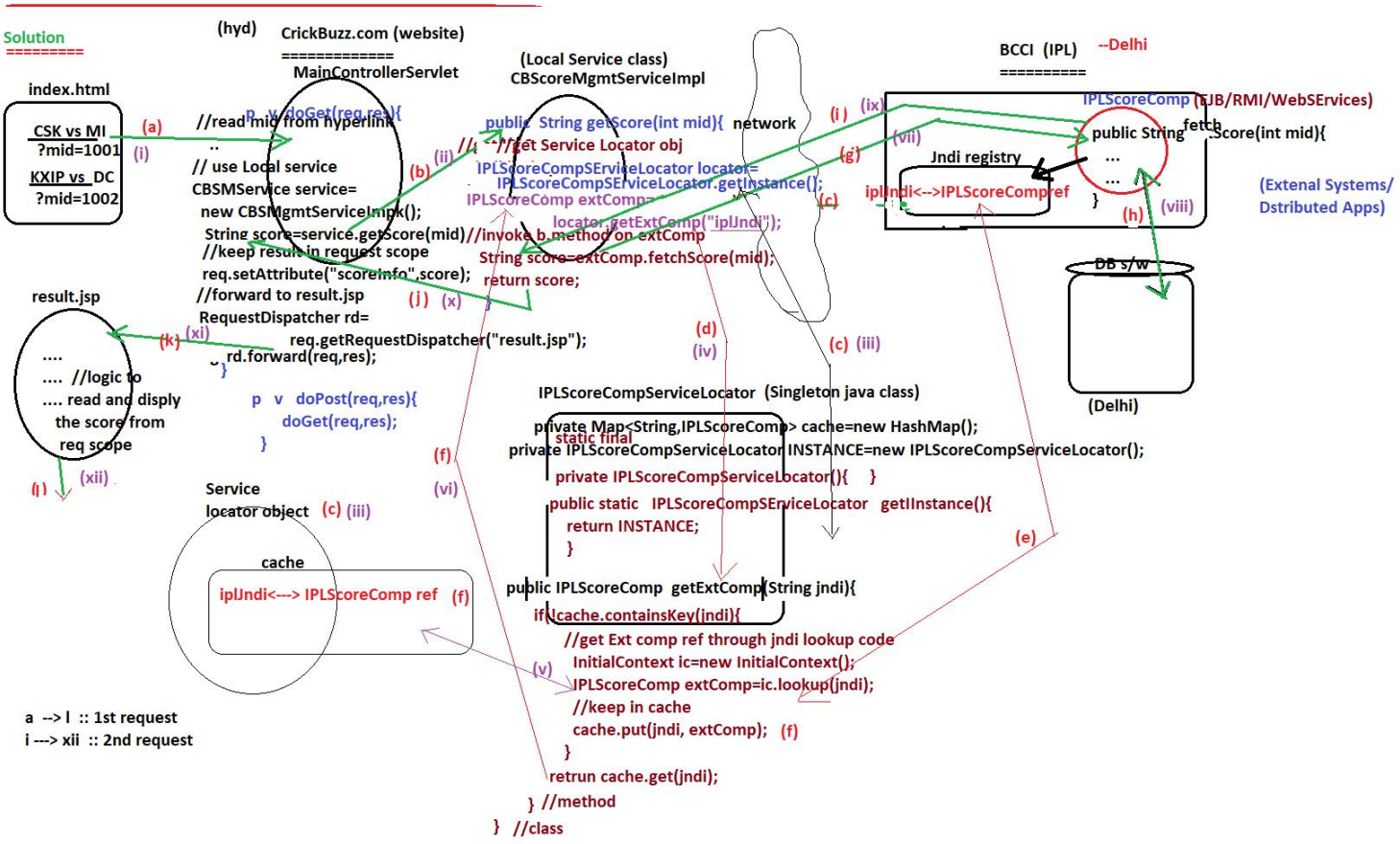
sharekhan.com
iciicdirect.com
upstocks.com
shareadd.com
→ BSE/NSE Comp (extenal Service)

crickBuzz.com
crickInfo.com
Dream11.com
→ ICC ScoreComp/ IPL ScoreComp (External Systems)

Writing Jndi lookup code in the Local Service class Client app (like CrickBuzz is having the following limitations)

(a) Jndi lookup code is not reusable across the multiple Service classes
(b) if Extenal Comp (IPLScoreComp) Technoloy or Location is chnaged we need to modify Jndi lookup code..
(c) if Jndi registry Technoloy or Location is chnaged we need to modify Jndi lookup code..
(d) Since Caching is not implemented .. there more n/w round trips b/w LocalService classes (CBuzz) and jndi registry to get same ExtenalComp ref for multiple times... (poor performence)

To overcome the above problems , take the support ServiceLocator DesignPattern.. Which says place only Jndi lookup code that is required to get Extenal Comp ref by talking with Jndi registry in a seperate class also maintain cache.. haivng extenal comp ref.. (This seperate class is called SErviceLocator)


Solution
(hyd)
CrickBuzz.com (website)
MainControllerServlet
(Local Service class)
CBScoreMgmtServiceImpl
BCCI (IPL) --Delhi
index.html
CSK vs MI
?mid=1001
KXIP vs DC
?mid=1002
(a)
(i)
//read mid from hyperlink
// use Local service
CBSMService service=
new CBSMgmtServiceImpk();
String score=service.getScore(mid)
//keep result in request scope
req.setAttribute("scoreInfo",score);
//forward to result.jsp
RequestDispatcher rd=
req.getRequestDispatcher("result.jsp");
rd.forward(req,res);
(b)
(ii)
public String getScore(int mid){
//get Service Locator obj
IPLScoreCompSErviceLocator locator=
IPLScoreCompSErviceLocator.getInstance();
IPLScoreComp extComp=
locator.getExtComp("iplJndi");
//invoke b.method on extComp
String score=extComp.fetchScore(mid);
return score;
}
network
(i) (ix)
(vii)
(g)
IPLScoreComp (EJB/RMI/WebSErvices)
public String fetchScore(int mid){
...
...
}
Jndi registry
iplJndi<-->IPLScoreComp ref
(h)
(viii)
(Extenal Systems/
Dstributed Apps)
DB s/w
(Delhi)
result.jsp
....
.... //logic to
.... read and disply
the score from
req scope
(k) (xi)
(j) (x)
(l) (xii)
p v doPost(req,res){
doGet(req,res);
}
(c) (iii)
(d)
(iv)
IPLScoreCompServiceLocator (Singleton java class)
private Map<String,IPLScoreComp> cache=new HashMap();
static finial
private IPLScoreCompServiceLocator INSTANCE=new IPLScoreCompServiceLocator();
private IPLScoreCompServiceLocator(){ }
public static IPLScoreCompSErviceLocator getIInstance(){
return INSTANCE;
}
public IPLScoreComp getExtComp(String jndi){
if(!cache.containsKey(jndi){
//get Ext comp ref through jndi lookup code
InitialContext ic=new InitialContext();
IPLScoreComp extComp=ic.lookup(jndi);
//keep in cache
cache.put(jndi, extComp); (f)
}
retrun cache.get(jndi);
} //method
} //class
(e)
(f)
(vi)
Service
locator object (c) (iii)
cache
iplJndi<---> IPLScoreComp ref (f)
(v)
a --> l :: 1st request
i ---> xii :: 2nd request

} //class

Advantages of ServiceLocator DesignPattern
==========

(a) The Jndi lookup code to get External comp ref becomes reusable across the multiple local service classses
(b) if external comp technology or location is modified,we just need to modify ServiceLocator code... not the multiple local service classes code.
(c) if Jndi registry technology or location is modified,we just need to modify ServiceLocator code... not the multiple local service classes code.
(d) Since the Service class Locator manages cache having External comp ref, the network round trips b/w Client App/project and External comp ref will be reduced drastically.

What is the difference Service class and Service Locator?

Ans) Service class contains b.logic .. where as Service locator container jndi lookup to get Extenal comp ref.. Infact Service class of client App/Client Project uses Service Loctor internally.. We generally develop Service class as normal java class... where we develop Service Locator as singleton class .. to maintain cache with external comp ref..

note:: In spring env.. Service class is normal Spring bean.. wehre as SEriceLocator is FactoryBean..

---

note::: if develop Client App/Project as Spring based Application we need to take Local Service class as target and SErviceLocator class as dependent spring bean.. But if we observe very carefully , LocalService class not intrested in ServiceLocator object .. It is actually intrested in the SErviceLocator supplied External comp ref (resultant object).. So we need to take SErviceLocator as Factorybean.. always..

Advantages of taking ServiceLocator as Factory Bean class in spring env
==========

(a) No need of developing it as Singleton java class.. just "singleton" scope is sufficient.
(b) No need to developing cache seperately.. having external comp ref.. becoz the Intenral Cache of IOC container itself maintains Extenalcomp ref..

Limitations
==========

(a) SErviceLocator becomes invasive .. becoz to make it factory bean.. we need to implement the Spring API supplied FactoryBean(I).

What is the diff b/w Local SErvice class and external comp/distributed App/External service ?

Ans) LocalService is an ordinary java class having b.logic of Client App / Client Project.. (like service class is Flipkart) where Extenal comp/service is a distributed App that can have different type remote or local cleent Apps ..accessing its services .. Extenal Comps/services will be developed by using Distributed Technologies like RMI,EJB,WebServices.. (these are like PhonePe, GPay , PayPall and etc...)

if Servlet,jsp based MVC Web application is using Spring env.. for devleoping Model layer service, DAO, ServiceLocator classes can u tell me where should we create IOC container that is required for Model Layer classes and how to get LocalService class obj to the controller servlet?

Ans) create IOC container in the init() method ControllerServlet Comp by enabling <load-on-startup> on Controller Servlet , close IOC container in the destroy() method and get Service class object to Controller Servlet in the Service(-,-)/doGet(-,-)/doPost(-,-) method by calling ctx.getBean(-,-) method.

[In the above setup , the Controller SErvlet comp pre-instantiation and Spring beans related pre-instanitation takes either during the deployment of web application or during the server startup)

<load-on-startup> enabled on the servlet comp makes Servlet container to create Servlet class obj during the deployment of web application.. nothing performs eager/pre-instantiation of sevlet comp..

Procedure to develop Gradle based web application in Eclipse IDE

step1) create the Gradle Project convert it to web application

To convert to web application

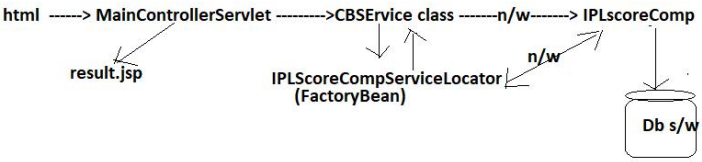
RTC on Project ---->Project facets ---> Dynamic web module ---> 4.0 -->Apply and Close.

=>Add web.xml file manually (File menu --->new ---> xml ---> file name: web.xml)
(This webcontent folder to the project)
to WEB-INF

step2) Add the jars /dependencies to build.gradle by adding the "war" plugin..

build.gradle

```
plugins {
  // Apply the java-library plugin to add support for Java Library
  id 'war'
}
repositories {
  jcenter()
}
dependencies {
   // https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api
  implementation group: 'javax.servlet', name: 'javax.servlet-api', version: '4.0.1'
  // https://mvnrepository.com/artifact/org.springframework/spring-context-support
  implementation group: 'org.springframework', name: 'spring-context-support', version: '5.2.9.RELEAS
}
```

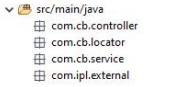


step3) add web.xml file Web Project..

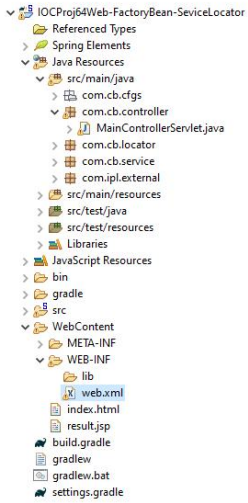
=>RTC on WEB-INF --> new ---> others ---> xml file ---> ... ...

https://www.youtube.com/c/NareshIT/search?query=maven%20by%20nataraj
(link for 5 videos)

step4) create the packates in src/main/java folder..



step5) develop the soruce code...



note:: It is always recomanded to develop Extenal Comp having interface , implementation class model .. So that interface can be give Local or remote client Projects to make them to recieve and hold extenal comp ref.

eg:: IPL People gives IPLScoreComp related interface to CrickBuzz client Project to make them to recieve and hold IPLScoreCompRef.

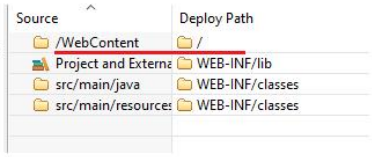
step6) cfg Tomcat server with Eclipse IDE

step6) cfg Tomcat server with Eclipse IDE

window menu --->preferences ---> server --> runtime env.. add----> select apache Tomcat9 --> choolse the tomcat installation folder (E:\Tomcat9.0) --->apply and ok..

step7) Perform deployment Assembly operation on the web Project.. to move webcontent folder to deployment..

RTC on Project --> properties --->Deployment assembly ---> add ----> folder ---> webcontent ---> .....



step8) run the Application

RTC on Project ----> Run as ---> run on server ----> select tomcat server ---> ....

note:: org.sf.jndi.JndiObjectFactoryBean is the pre-defined Factorybean that is given as ServiceLocator.. It is very useful to get Resulttant object from underlying Server Jndi registry based in jndi name that we provide.. (Especially useful to get DataSource object of serverManaged jdbc con pool)

in applicationContext.xml

```
<bean id="jofb"  class="pkg.JndiObjectFactoryBean">
   <property name="jndiName"  value="DsJndi'/>
</beans>
```

Get DataSource obj from jndi registry of undrlying SErver bbased on the given Jndi name.. and make it as spring bean having "jofb" as the bean id

```
<bean id="empDAO"  class="pkg.EmployeeDAOImpl">
      <constructor-arg ref="jofb"/>
</bean>
```

Injects DataSoruce obj to DAO class becoz "jofb" bean id is pointing to DataSource obj.

note:: Since the spirng Bean is becoming FactoryBean.. becoz FactoryBean(I) impl.. not becoz of any xml tag ... So there is not annotation for it..

Limitation of Developing ServiceLocator as FactoryBean
==========

(a) Makes Service Locator as Invasive Spring bean becoz of implementing spring specific FactoryBean(I)
(b) Writing logics by implementing 3 methods is quite complex
(c) The object created for FactoryBean class itself looks quite wasted object..

note:: To overcome the above problems.. it is recomanded to take SErviceLocator having static factory method bean Instantiation (For this no annotations are given .. we need to cfg java class as spring bean by keeping factory-method attribute in <bean> though spring bean is user-defined java class)

Example code
==========

```
----------IPLScoreCompServiceLocator.java----------
public interface IPLScoreCompServiceLocator {
    final static IIPLScoreComp extComp=new IPLScoreCompImpl();  //eager instantatiation..

        public static IIPLScoreComp getExtComp() {  //static factory method
            return extComp;
        }
}
```

In the following situations we need to go for xml cfgs
though spring beans are user-defined classes
a) inner beans
b) static/instance factory method bean instantiation
c) Collection injection
d) Collection merging
e) Bean Inheritence
f) Method Replacer/Method Injection
g) Bean Aliasing
and etc..

```
}
```

applicationContext.xml

```
<!-- Spring bean cfg by enabling static factory method bean instatiation -->
<bean id="locator"  class="com.cb.locator.IPLScoreCompServiceLocator"   factory-method="getExtComp"/>

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

To avoid deployment assembly operation while working Gradle based web applications and to make webContent folder participating in deployment process place the following code in build.gradle file

```
eclipse {
      wtp {
        component {
              resource sourcePath: "/WebContent", deployPath: "/"
            }
       }
 }
```

wtp :: eclipse web tools platform

Adavantages of developing SErviceLocator by enabling static factory method bean instantiation

(a) Service Locator acts non-invasive
(b) We can take service Locator as abstract class or java8 interface i.e not waste object will be created for ServiceLocator itself
(c) Code becomes simple.. (complexitity is less)
(d) There is no need of implementing Cache to hold extenal comp ref.. becoz IOC container internal cache itself will that work..

Can we cfg interface as Spring Bean?

Ans) Directly not possible , but we can make it possible in special situtations..
a) While Working with LMI(Lookup method) , we can take target class as java8 Interface having b.methods as default methods with
b) While woking static factory method Bean instantiation, we can take Java 8 interface with static method as shown above..

note:: In the situations .. we can also use abstract classes in the place java8 interfaces..

PostProcessing
==========

=>The logic that executes after creating bean class object and after completing all injections is called post processing..

=> We can do each spring bean object specific post processing by using customInit() or
** InitializingBean's aftterPropertiesSet() or @PostConstruct method.. (Bean Life cycle init method)
=> if multiple spring bean and their objects are looking common processisng logics then istread of writing inside every spring bean class..we can write only for 1 time out side of all spring beans.. by taking the support of Bean Post Processor (BPP) as the class that implements
org.springframework.beans.factory.config.BeanPostProssor (I) [Java8 Interface]

default Object postProcessBeforeInitialization(Object bean,
String beanName) throws BeansException
This method will execute before spring bean init life cycle methods execution

default Object postProcessAfterInitialization(Object bean,
String beanName) throws BeansException
This method will execute after spring bean init life cycle methods execution.

order of execution
==========

=>Spring Bean Instantiation (constructor Injection)
=>setter injection
=> Aware Injection
=> postProcessBeforeInitialization(-,-) (BPP)
=> Spring bean init life cycle method(s) **
=> postProcessAfterInitialization(-,-) (BPP)

conclusion:: if spring bean is not having init life cycle methods having postprocessing logic then we can place post processing logics in any one method of BPP class.. otherwise it is good to place in postProcessAfterInitialization(-,-) method

and other operations.. like FactoryBean or LMI or Method Injectio or staitc/instance factory method Injection ,....

CustomerBO
EmployeeBO
StudentBO

having DOJ Property of java.util.Date
⇩
(we want to have system date)

All the 3 classes cfg as spring beans.

note:: if spring bean init life cycle method to initialize doj with system date.. we need to write in every BO class seperately.. (totally 3 times) ..
By using BeanPostProcessor , the same thing can be done.only for 1 time but it will be applied for all the 3 beans..

=>@Required,@Autowired and etc.. basic annotations perform their funcationalities on spring beans... by taking the support of BeanPostProcessors.. internally..

org.springframework.beans.factory.annotation.RequiredAnnotationBeanPostProcessor
org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor

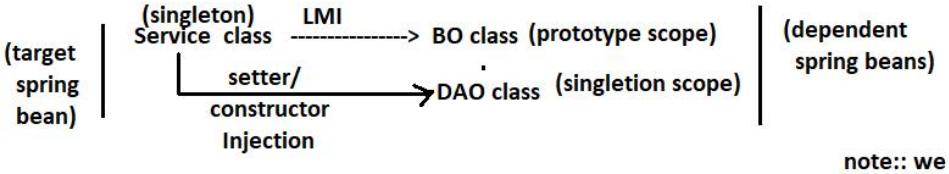
Example Application using LookupMethod Injection (LMI) and BeanPostProcessor (BPP)

==========

html/jsp form ----------> Servlet comp (controller) ----------DTO----------> Service class ----------BO----------> DAO class ------>DB s/w

(as needed) (1 per project) (1 per module) (1 per db table)

if want to take Service , DAO, BO classes as spring beans we generally take SErvice ,DAO classes as singleton scope spring beans and BO class prototype scope spring beans becoz BO class obj should hold every request data seperately .. Here Service class is target having singleton scope and BO class dependent class having prototype scope .. For this we need go for LMI. Another combination is Service class is target class and DAO class is Dependent class (both having singleton scope), For this we can go for Setter or constructor injection..

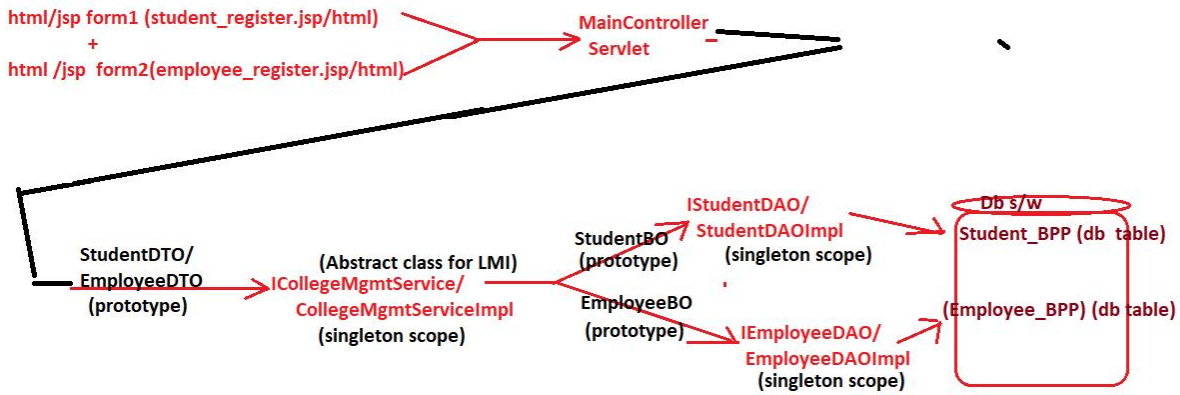


=> DTO is required in Servlet comp. which can not be cfg as Spring Bean.. So Servlet comp can use one of the following two approaches to get objects of DTO

note:: we can apply Bean PostProcessor on java classes only when they are cfg as spring beans.. here BO classes taken as spring beans to apply BeanPostProcessor on them..

(a) create DTO class object by new operator in service(-,-) /doXxx(-,-) methods on 1 per request basis

(b) cfg DTO class as prototype scope spring bean.. and get it service(-,-)/doXxx(-,-) method by calling ctx.getBean(-,-) method.. (Traditional Dependency lookup) (Best)



Student_register.html/jsp
|-->sno,sname,sadd,course,doj details
employee_register.html/jsp
|-->eno,ename,company,salary,doj details

note:: doj will be initialized dynamically
using BPP.. i.e it will not be collected
from enduser.

These are cfg as spring beans having prototype scope	StudnetDTO EmployeeDTO	doj (common property)
	StudnetBO EmployeeBO	doj (commin property)

Instead of initializing all DTOs DOJ property and all BOs doj property with system date seperately in every DTO or BO class .. Better to Initialize only 1 time by taking the support of single BeanPostProcessor..

note:: controllServlet gets DTO classes objs by using traditional depedency lookup
Service class gets BO classes objs by using LMI
SErvice class gets DAO classes objs by setter/constructor innjection

=>The BeanPostPrcoessor class executes for every object created for the spring bean class.. if the the spring bean scope is singleton then it executes only for 1 time.. if the spring bean scope is prototype then it executes for every object created for that spring bean.

=>Ordindary Interface upto spring 4.x
=>java 8 Interface with default methods from spring 5.x

org.springframework.beans.factory.config.BeanPostProcessor (I)
|---> default Object postProcessBeforeInitialization(Object bean,
String beanName) throws BeansException
|
|-->default Object postProcessAfterInitialization(Object bean,
String beanName) throws BeansException

```
@Component
public class BeanDOJPostProcessor implements BeanPostProcessor {
	public BeanDOJPostProcessor() {
		System.out.println("BeanDOJPostProcessor:0-param constructor");
	}
	@Override
	public Object postProcessBeforeInitialization(Object bean, String beanName) throws BeansException {
		System.out.println("BeanDOJPostProcessor.postProcessBeforeInitialization(-,-)");
		if(bean instanceof BaseBean) {
			((BaseBean) bean).setDoj(LocalDateTime.now());  // Initializing doj property with  sys date and time
			System.out.println(bean);
		}
	  return bean;
	}//method
}//class
```

refer :: IOCProj67Anno-Web-RegistrrationApp-LMI-BPP
for complete Application on LMI + BPP

note:: if spring is having init life cycle method(s) then it is better to write BPP logics in postProcessAfterInitialization(-,-) method otherwise we can write in any method

note:: instanceOf is a java operator to check wheather given object referencetype is certain class or not

note:: BeanPostprocessor class object will be created during IOC container creation..automcatically based on it BeanPostProcessor(I) implementation irrespective of its scope... (Same thing is applicable for BeanFactoryPostProcessor and EventListeners)

BeanFactoryPostProcessor

==========

=>After creating InMemoryMetaData of spring bean cfg file and before performing pre-instantation of singleton scope beans..if we want execute some logic to change /set data in InMemoryMEtaData of spring bean cfg file we need go for BeanFactoryPostprocessor.

note:: PropertyPlaceHodlerConfigurer or <context:property-placeholder >are internally BeanFactoryPostProcessor to recognize place holders(${....}) in the InMemory MetaData of spring bean cfg file and to replace them with the data collected from the properties file or system properties or env.. variables.. while working with @Value annotation the placeholder recognized and replaced by using the support of PropertyplaceholderConfigurer/Support class (BeanFactoryPostProcessor)

```
@Value("${<key>}")
private int age;
```

=>In ApplicationContext container BeanFactoyProcessor will be recognized and applied automatically once it cfg as spring bean.. In BeanFactory Container we must register it explictly..

Can u we Properties file and place holders while working with BeanFactory IOC container?

Ans) Directly not possible becoz BeanFactoryContainer does not register/recognize PropertyPlaceHolderConfigurer/Support as BeanFactoryProcessor directly.. We should go for explicit registration..

Example Application
==========

a) keep any application ready that using properties file and place holders (${...})

b) cfg PropertyPlaceholderConfigurer in the spring bean cfg file specifying the name and location of properties file(s).

```
<bean id="pphc" class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
<!--    <property name="location" value="com/nt/commons/jdbc.properties"/>
 -->
        <property name="locations">
            <array>
              <value>com/nt/commons/jdbc1.properties</value>
              <value>com/nt/commons/jdbc2.properties</value>
            </array>
          </property>
        </bean>
```

refer IOCProj68
==========

c) Take BeanFactorycontainer in client App and explicity register Container with PropertyPlaceholderConfigurer..

```
DefaultListableBeanFactory     factory = new DefaultListableBeanFactory();
XmlBeanDefinitationReader      reader = new XmlBeanDefinitionReader(factory);
reader.loadBeanDefinitions("com/nt/cfgs/applicationContext.xml");
//get PropertyPlaceHolderConfigurer
pphc=factory.getBean("pphc",PropertyPlaceholderConfigurer.class);
// register with Container
pphc.postProcessBeanFactory(factory);
```

PropertyEditors

==========

=>It is useful to convert configured values to as required for the bean properties ...i.e talks about auto convertion of values..

eg:: <property name="age" value="30"/>

<property name="avg" value="45.66f"/>

IOC container internally uses PropertyEditors concept to convert xml file/properties file supplied String inputs to appropriate type as required for bean properties.

=>Multiple Built-in PropertyEditors are already registered with BOTH IOC cotainers ... (BF ,AC Containers)

=> Every PropertyEditor is a java class that implements java.beans.PropertyEditor (I) directlry or indirectly..

ByteArrayPropertyEditor
CharacterEditor
CharArrayPropertyEditor
CharsetEditor
ClassArrayEditor
ClassEditor
CurrencyEditor
CustomBooleanEditor
CustomCollectionEditor
CustomDateEditor
CustomMapEditor
CustomNumberEditor
FileEditor
InputSourceEditor
InputStreamEditor
LocaleEditor
PathEditor
PatternEditor
PropertiesEditor
ReaderEditor
ResourceBundleEditor
StringArrayPropertyEditor
StringTrimmerEditor
TimeZoneEditor
URIEditor
URLEditor
UUIDEditor
ZoneIdEditor

Builtin-Properteditors belonging to org.sf.beans.propertyeditors package.

note:: we can develop and register Custom PropertyEditors with IOC containers...

Example on built-in proeprty editors

==========

```
public class PersonInfo {
	private long adharNo;
	private String pname;
	private  String[] addresses;
	private  float  salary;
	private  File  photoPath;
	private Currency  countryCurrency;
	private   Date  dob;
	private  Class  javaClass;
	private  Class[] javaClasses;
	private   InputStream inputFile;
	private URL  fbUrl;
	private  Locale   currentLocale;
	private  TimeZone  timezone;
	private  Properties props;
	//setters
```

Spring bean cfg file
==========

```
<beans .....>
    <bean id="pInfo"  class="com.nt.beans.PersonInfo">
    <property name="adharNo"  value="45678997666"/>
    <property name="pname"  value="ramesh"/>
    <property name="salary"  value="67000.55f"/>
    <property name="addresses"  value="hyd,vizag"/>
    <property name="dob"  value="11/23/1990"/>    <!--  MM/dd/yyyy -->
    <property name="countryCurrency"  value="INR"/>  <!-- Converts currency code into  java.util.Currency object -->
    <property name="photoPath" value="E://files//photo.jpg"/>  <!-- converts given location into java.io.File class object -->
     <property name="currentLocale"  value="hi-IN"/>  <!-- converts given string locale to  java.util.Locale object -->
     <property name="javaClass"  value="java.lang.System"/> <!-- converts given class name to java.lang.Clas object -->
     <property name="javaClasses" value="java.lang.System,java.lang.String"/>
      <property name="inputFile"  value="classpath:/photo.jpg"/> <!-- converts given String content url as the file location represented by InputStream obj --
     <property name="fbUrl"  value="http://facebook.com/userId=raja"/> <!-- converts given String content to java.util.URL object -->
     <property name="props"  value="name=raja,age=30,addrs=hyd"/>
     <property name="timezone" value="Asia/Calcutta"/>  <!-- converts given String to java.util.TimeZone object -->
  </bean>
</beans>
```

Custom PropertyEditor
==========

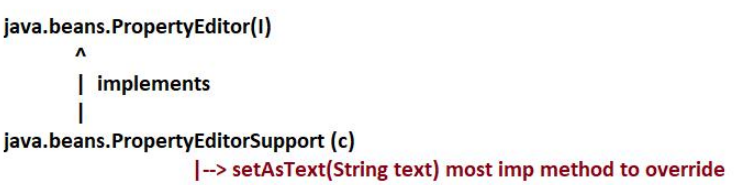
=> We can develop java class Custom PropertyEditor by making that class implementing java.beans.PropertyEditor (I)

=> java.beans.PropertyEditorSupport(c) is the impl class of PropertyEditor(I) having null method definitations.. So we create CustomPropertyEditor extending PropertyEditorSupport and we can override only those methods in which we are intrested in..

CustomPropertyEditor class implementing java.beans.PropetyEditor(I)
|--->should implement all the 12 methods..

CustomPropertyEditor class extending from java.beans.PropetyEditorSupport(C)
|--->can override only required methods..

java class implemeting Servlet(I)
|-->should implement 5 methods
java class extending from GenericServlet
|-->should implement only 1 method
java class extending from HttpServlet
|-->can override its choice methods



Example showing need of Custom PropertyEditor
==========

Dependent class
==========

```
public class  LoanAmtDetails{
  private float pAmt;
  private  float rate;
  private  float time;
  //setters methods  & getter methods
    ....
    ....
}
```

target class
==========

```
public class  LoanIntrestAmtCalculator{
  private   LoanAmtDetails  details;
 public void setDetails(LoanAmtDetails details){
     this.details=details;
  }
 public  float   calcIntrestAmt(){
   return  (details.getPAmt()* details.getRate()* details.getTime())/100.0f;
  }
}
```

applicationContext.xml (Actual code we should write)
==========

```
<beans ........>

  <bean id="laDetails"  class="pkg.LoanAmtDetails">
     <property name="pAmt" value="100000"/>
     <property name="rate" value="2"/>
     <property name="time" value="10"/>
  </bean>

  <bean id="laiCalculator" class="pkg.LoanAmtIntrestCalculator">
      <property name="details"  ref="laDetails"/>
  </bean>
</bean>
```

applicationContext.xml (i want like this)
==========

```
<beans ....>

   <bean id="laiCalculator"  class="pkg.  LoanAmtIntrestCalculator ">
       <property name="details"  value="1000000,2,10"/>
    </bean>
</beans>
```

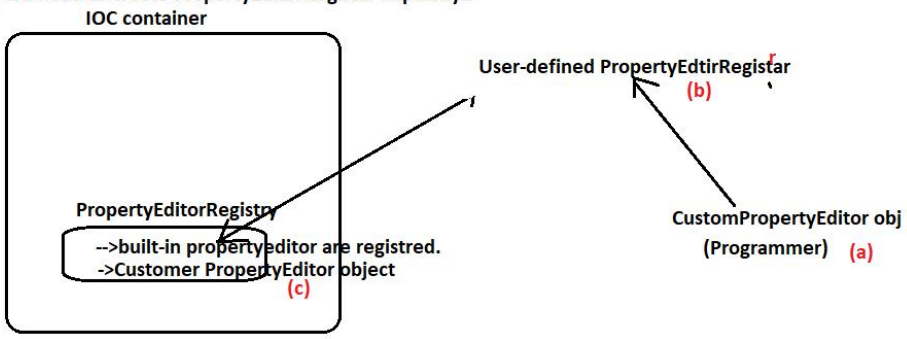
Since "details" is Object type bean property we can not inject simple/String value to it.. To make it possible we need to develope and configure CustomPropertyEditor to convert comma seperated String values to LoanAmtDetails object..

A spring bean class can have 4 types of bean properties
- a) Simple type (primitive type, String type ,wrapper type)
- b) Object/Reference type
- c) Array type
- d) Collection type

```
String s="hello1234";
 String s1=s.subString(0,4);  //hell
                              3,9
```

=>Every Customer PropertyEditor must registerd IOC container (for both BF,AC container)

=>IOC container maitains PropertyEditorRegistry where Custome PropertyEditors can be registered.. To get Access to this registry we need to create PropertyEditorRegistar explicitly..



we can get create our own PropertyEditorRegistrar by making our class implemeting
import org.springframework.beans.PropertyEditorRegistrar(I) and this gives access PropertyEditorRegistry
So that we can register any no.of customer PropertyEditors by speciyfing their property types for whom they should be applied

```
//nested inner class /static inner class
    private static class  CustomRegistrar  implements PropertyEditorRegistrar{

        @Override
        public void registerCustomEditors(PropertyEditorRegistry registry) {
            System.out.println("PropertyEditorSupport.CustomRegistrar.registerCustomEditors()");
          registry.registerCustomEditor(LoanAmtDetails.class,new LoanAmtDetailsEditor());  // (property type, CustomP

        } //method
    }//inner class
```

E class obj)

After creating Custom PRopertyEditorRegistrar we need to link/add that registrar to IOC container(BF) as shown below

```
//add CustomPropertyEditorRegistrar  to IOC container
        factory.addPropertyEditorRegistrar(new CustomRegistrar());
```

we have 4 types of inner classes
==========

a) Normal inner class (Use it when its logics are required in multiple non-static methods of outer class)
b) Nested/static inner class (Use it when its logics are required in multiple static methods of outer class)
c) Local inner class (Use it when its logics are required only in specific method definitation of outer class)
d) Anonymous inner class (Use it when its logics are required only in specific method call of outer class)

=>factory.addPropertyEditorRegistrar(new CustomRegistrar()); -->This method takes the given CustomPropertyEditorRegistrar object and calls registerCustomerEditors(-) having PropertyEditorRegistry obj as the argument value.. becoz of registry.registerCustomEditor(LoanAmtDetails.class,new LoanAmtDetailsEditor()); method the Customer PropertyEditor (LoanAmtDetailsEditor class obj will be registered with PropetyEditorRegistry againt the Property of type LoanAmtDetails).

note:: This Registered Custom PropertyEditor (LoanAmtDetailsPropertyEditor objectt) will be used by IOC Container (BeanFactory) container when it is performing Lazy Instantation of target bean class (LoanAmtIntrestCalculator) for method call factory.getBean("laiCalculator",LoanAmtIntrestCalculator.class);

Anonymous inner class based Logic to register CustomPropertyEditor with BF container

```
factory.addPropertyEditorRegistrar(new PropertyEditorRegistrar() {
      @Override
      public void registerCustomEditors(PropertyEditorRegistry registry) {
            registry.registerCustomEditor(LoanAmtDetails.class, new LoanAmtDetailsEditor());
       } //method
      }// anonymous inner class
);
```

```
public void registerCustomEditor(Class<?>
requiredType, Class<? extends PropertyEditor>
propertyEditorClass) {
      //logic
}
```

W. r.t above code

(a) One Anonymous (nameless) inner class is created implementing PropertyEditorRegistrar(I)

(b) In that Anonymous inner class registerCustomEditor(-) is method implemented having logic to register custom PropertyEdito

(c) Object of the anonymous inner class created using new operator and passed it as the argument value of factory.addPropertyEditorRegistrar(-) method.

=> Since PropertyEditorRegistrar(I) is java 8 functional interface (interface with 1 method decl) we can use LAMDA Expression based Anonymous inner class code.. as shown below.

```
factory.addPropertyEditorRegistrar(registry->{
                registry.registerCustomEditor(LoanAmtDetails.class, new LoanAmtDetailsEditor());
        });
```

=>While working with ApplicationContext container .. there is no provision to call addPropertyEditorRegistrar(-) on its object.. becoz this method is not basically available in BeanFactory(I), ApplicationContext(I) interfaces.. This method is avaliable in ConfiruableBeanFactory(I) and implemented by DefaultListableBeanFactory class.. and not implemented by any ApplicationContext Container classes... So we use other approach for ApplicationContext container.. that is working with a ready made BeanFactoryPostProcessor called "CustomerEditorConfigurer".. as shown below..

in applicationContext.xml

```
<bean class="org.springframework.beans.factory.config.CustomEditorConfigurer">
    <property name="customEditors">
      <map>
        <entry key="com.nt.beans.LoanAmtDetails" value="com.nt.editors.LoanAmtDetailsEditor"/>
      </map>
    </property>
  </bean>
```

->pre-defined BeanFactoryPostProcessor

property type

Custom ProperyEditor

note:: ApplicationContext container automatically registers the BeanFactoryProcessors right after creating InMemoryMetaData of spring bean cfg file and before performing pre-instanatation singleton scope beans.. In this process the above CustomEditorConfigurer internally creates One PRopertyEditorRegistrar to access to PropertyEditorRegistry and registers the given CustomPropertyEditor against given property type

---

=>Since we can activate /use BeanFactoryPostProcessor in BeanFactoryContainer by registering them explicilty we can use the above "CustomEditorConfigurer" in BeanFactory container also as shown below

applicationContext.xml

```
<bean class="org.springframework.beans.factory.config.CustomEditorConfigurer">
    <property name="customEditors">
      <map>
        <entry key="com.nt.beans.LoanAmtDetails" value="com.nt.editors.LoanAmtDetailsEditor"/>
      </map>
    </property>

  </bean>
```

In Client App

```
//get CustomEditorConfigure object
CustomEditorConfigurer cec=factory.getBean(CustomEditorConfigurer.class);
//register cec with BeanFactory
  cec.postProcessBeanFactory(factory);
```

100 % code Driven Spring App development / Java Config Approach of Spring App Development

Advantages

=>Xml based cfgs can be avoided in maximun cases
=>improves the readability
=>Debugging becomes easy
=> Foundation to learn Spring Boot

4 approaches of spring app development

a) xm driven cfgs
b) annotation driven cfgs
c) 100% code /java config drivern
d) spring boot app development

Thumb rule::

=>cfg user-defined classes as spring beans using strereo type annotations
and link them with Configuration class (alternate to spring bean cfg file (xml file) using @ComponetScan

note:: Java class that is annotated with @Configuration automatically becomes Configuration class

=> Cfg pre-defined classes as spring beans using @Bean Methods (method that is annotated with @Bean)
of @Configuration class.. (1 method for 1 object of bean class)

=> Use AnnotationConfigApplication Context class to create IOc container having @Configuration class as the input class name

WishMessageGenerator (target class) ----------> LocalDateTime (dependent class)
(use-defined class -->@Component) (pre-define class --> @Bean meethod)

Sample Configuration class

```
@Configuration
@ComponentScan(basePackages="<pkg name(s)")
public class AppConfig{

 @Bean(name="<beanId>")
 public  <class/interface> <method>()  {
   .....  //logic to create object and to set data
   .....
  return obj of class /impl class
  }

 }
```

The Object returned by this method becomes spring bean having the given bean id

note: @Bean methods of @Configuration class will be called automatically as part of IOC container singlescope beans pre-instantiation

Why we can not use stereo type annotations to cfgs pre-defined classes as spring beans?

Ans) We can not open the soruce code of pre-defined class to add stereo type annotations.. on the top of class .. So go for @Bean methods of @ Configuration class

since most of the methods in java.uti.Date class are deprecated , it is recomanded to use either java.util.Calendar class or java8 date ,time api
->LocalTime
->LocalDate
->LocalDateTime

note:: we can not @Bean Methods as overloaded methods in the @Configuration class..

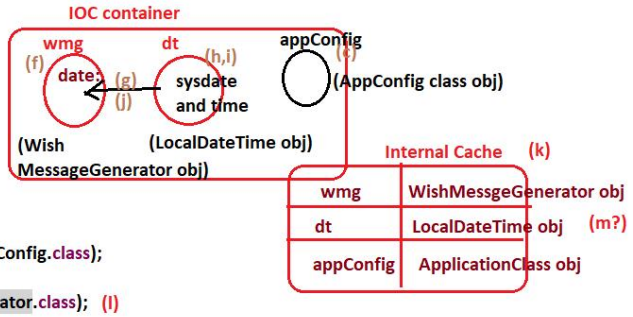
IOC container creation

```
ApplicationContext ctx=new AnnotationConfigApplicationContext(AppConfig.class);
                                                              (Configuration class)
```

(Configuration class)

```
//Target class
package com.nt.beans;  (e)
@Component("wmg")   (f) -object creation
public class WishMessageGenerator{
 @Autowired   (g) (looks for dependent)
 private LocalDateTime date;
                 (j) (Injection compited)
 .....
 ....
 }
```

```
AppConfig.java
==============
@Configuration
@ComponentScan(basePackages = "com.nt.beans")   //alternate <context:componet-scan> tag   (d)
public class AppConfig {      (c) --> configuration  class instantation

    @Bean(name="dt")
    public  LocalDateTime  createSysDateTime() {   (h)
        System.out.println("AppConfig.createSysDateTime()");
        return LocalDateTime.now();  //static factory method returning its own class object
    }                    (i)

}
```




```
TestClient.java
=============
public class TestClient {
                    (a)
    public static void main(String[] args) {
        ApplicationContext ctx=null;
        WishMessageGenerator generator=null;
        //create IOC container
        ctx=new AnnotationConfigApplicationContext(AppConfig.class);   (b)
        //get Target Bean class obj
(n)     generator=ctx.getBean("wmg",WishMessageGenerator.class);   (l)
        //invoke method
        System.out.println("result is ::"+generator.generateMessage("raja"));
                                                                    (o)
        //close container
        ((AbstractApplicationContext) ctx).close();

    }//main
}//class
```

Internal Cache (k)

wmg	WishMessgeGenerator obj
dt	LocalDateTime obj (m?)
appConfig	ApplicationClass obj

note: every @Configuration class is internally a spring bean
becoz @Configuration internally contains @Component

singleton scope
note:: 100% Code driven Approach first pre-instantiates user-defined spring beans that are refered through @ComponentScan and later pre-instantiates @Bean methods related singleton scope spring beans..

Approach	default bean id	example
xml driven cfgs	<fully qualified classname>#<n>	<bean class="com.nt.beans WishMessageGeneator"> --> default bean id is com.nt.beans.WishMessageGeneator#0
Annotations cfgs	bean class name having first letter in lower case	@Component puiblic class WishMessageGenerator{ ... } | default bean id :: wishMessageGenerator
@Bean methods	method name	@Bean public LocalDateTime createSysDateTime() { .... } | default bean id :: createSysDateTime

@Streo type annotations can not be applied at method level... similarly @Bean can not be applied at class level

while working with following features of spring core module.. there are no annotations So use xml file driven cfgs and link that xml file with @Configuration class using @ImporResource Annoations..
the features are 1. bean inheritence 2. collection merging 3.inner beans 4. method replacer
4.Factory method bean instantations 5.collection Injection and etc..

Example
========

applicationContext.xml
======================

```
<beans ....>
 <!-- target/main class cfg -->
  <bean id="bank"  class="com.nt.target.BankLoanMgmt">
    <replaced-method  name="calculateIntrestAmount"  replacer="bankCIAR">
       <arg-type>float</arg-type>
       <arg-type>float</arg-type>
       <arg-type>float</arg-type>
    </replaced-method>
  </bean>
```

AppConfig.java

optional

```
@Configuration
@ImportResource("classpath:com/nt/cfgs/applicationContext.xml")
@ComponentScan(basePackages = "com.nt.replacer")
public class AppConfig {

}
```

While developing Layered Applications.. insteading of confuring all DAO classes, service classes, AOP classes and Controller class as spring beans in single Spring bean cfg file (xml file) .. It recomanded to take multiple xml files and link them to single xml file ..

persinstence-beans.xml ---->DAOs ,DS cfgs
service-beans.xml -------> service classes
aop-beans.xml ------> aop classes
controller-beans.xml ---> controller class cfg

link with

applicationContext.xml

```
<beans .....................>
   <import resource="persistence-beans.xml"/>
    <import resource="service-beans.xml"/>
    <import resource="aop-beans.xml"/>
     <import resource="controller-beans.xml"/>
</beans>
```

In 100p Code driven Configurations
==========

```
@Configuration
PersistenceConfig   class  ---->DAOs , Ds  cfgs
@Configuration
ServiceConfig   class  ----> service classes  cfgs
@Configuration
AOPConfig   class  ----> AOP  classes  cfgs
```

Link them with single Configuration class

```
@Configuration
@Import(value={PersistenceConfig.class, SErviceConfig.class,AOPConfig.class}
AppConfig   class      ---> Main cfg class
```

=>By taking multiple spring bean cfg files or multiple Configuration classes as shown above .. we can decrease the possiblity getting conflicts in the team env.. while working with code Repository/Management tools like GIT/SVN and etc..

What is the diff b/w @Import and @ImportResource?

Ans) @ImportResource given to link spring bean cfg file(xml file) with @Configuration class
@Import given to link Helper configuration classes with Main @Configuration class.

```
@Configuration
@Import(value={PersistenceConfig.class, SErviceConfig.class,AOPConfig.class
- AppConfig class ---> Main cfg class -
```

```
@Configuration
@ImportResource("com/nt/cfgs/applicationContext.xml")
public class AppConfig { ....}
```

=> In every IOC container creation one built-in object will be maintained that "Environment" object having system propertery values (like os.name and etc..) and given properties files values and profiles Info
=>This Enviroment object can be injected to our spring beans to read and use its data... by submitting key to get value..

```
@Configuration
@ComponentScan(basePackages = "com.nt.dao")
@PropertySource(value = {"com/nt/commons/jdbc.properties",
                         "com/nt/commons/jdbc1.properties"})
public class PersistenceConfig {
    @Autowired
    private Environment env;

    @Bean(name="hkDs")
    public  DataSource createDS() {
        HikariDataSource  hkDs=null;
        hkDs=new HikariDataSource();
        hkDs.setDriverClassName(env.getRequiredProperty("jdbc.driver"));
        hkDs.setJdbcUrl(env.getRequiredProperty("jdbc.url"));      key in properties file
        hkDs.setUsername(env.getRequiredProperty("jdbc.user"));
        hkDs.setPassword(env.getRequiredProperty("jdbc.pwd"));
        hkDs.setMinimumIdle(Integer.parseInt(env.getRequiredProperty("pool.minsize")));
        hkDs.setMaximumPoolSize(Integer.parseInt(env.getRequiredProperty("pool.maxsize")));
        System.out.println("system property ::"+env.getProperty("os.name"));
        return hkDs;                                                    system property name
    }
```

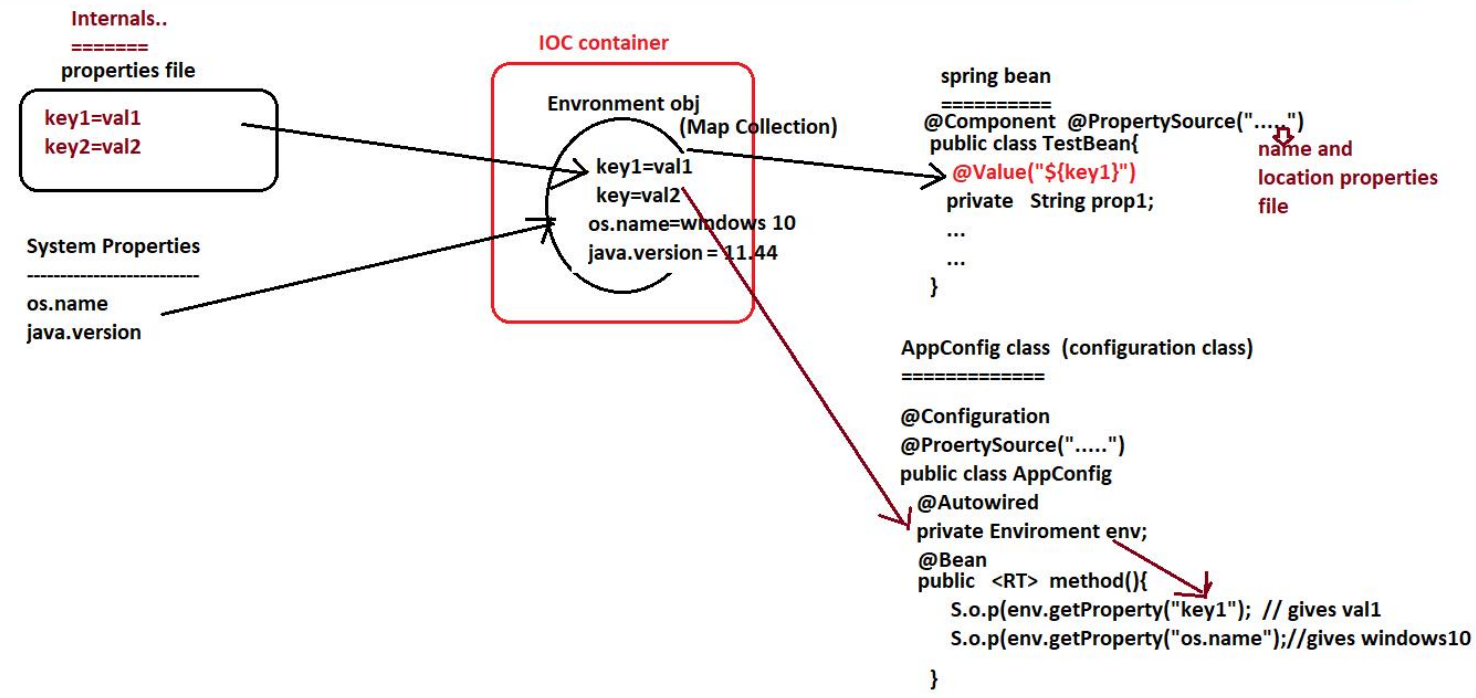
jdbc properties (com/nt/commons)

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

dbc1.properties (com/nt/commons)

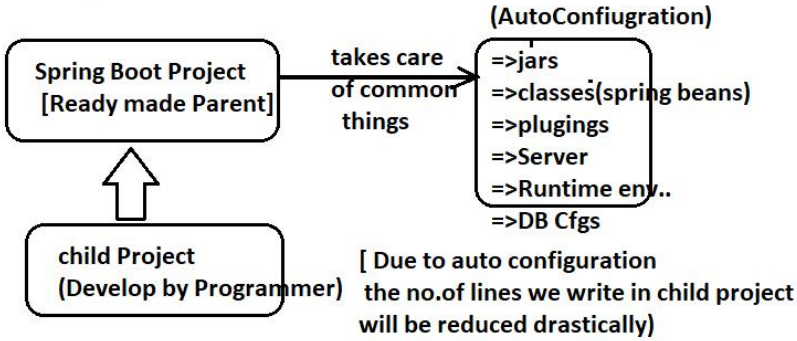
```
jdbc.pwd=manager
pool.minsize=10
pool.maxsize=100
```

==>Refer IOCProj73


Internals..
=======
properties file
key1=val1
key2=val2
System Properties
os.name
java.version
IOC container
Envronment obj
(Map Collection)
key1=val1
key=val2
os.name=windows 10
java.version = 11.44
spring bean
@Component @PropertySource("....")
public class TestBean{
@Value("${key1}")
private String prop1;
name and
location properties
file
AppConfig class (configuration class)
@Configuration
@ProertySource(".....")
public class AppConfig
@Autowired
private Enviroment env;
@Bean
public <RT> method(){
S.o.p(env.getProperty("key1"); // gives val1
S.o.p(env.getProperty("os.name");//gives windows10

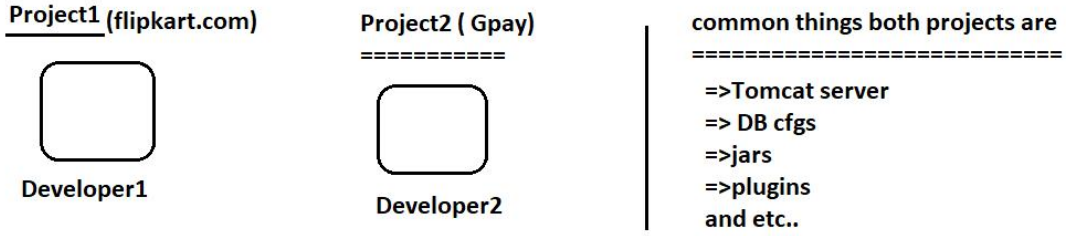
Spring Boot
==========

=>Spring boot is a spring Project that act as parent project to Programmer developed Child Project taking care common things application /Project development through a concept called AutoConfiguration..



What is AutoConfiguration?
==========

Spring Boot providing jars , classes(spring beans), Servers ,Runtime env.. plugins , DB cfgs and etc.. based on the "spring boot starters" that we add to Child Project is called AutoConfiguration



of

=>if both Projects are developed by using spring f/w .... Programmers only should take care these common things..

=>if both Projects are developed by using spring Boot (extension of Spring) ...

Programmers need not to take take of these common things.. becoz spring boot will generate them dynamically based on the "spring boot starters" that we added to the Project.

(Ready made partial code)

DB Connectivity Using spring

(Ready made partial code)

DB Connectivity Using spring
==========

```
@Configuration
@ComponentScan(basePackage="com.nt.dao")
public class PErsistenceConfig{     #Programmer
  @Bean
 public   DataSoruce createDs(){
     HikariDataSource ds=new HikariDataSource();
     ds.setDriverClassName("....");
      ds.setUrl("....");
        ....
        ...
    return ds;
   }
}
```

#Programmer
→ jars in classpath
- spring-context-<ver>. jar
- spring-jdbc-<ver>.jar
- hikaricp-<ver>.jar
- ojdbc8.jar

DAO class #Programmer
==========

```
@Repository("empDAO")
public class  EmployeeDAOImpl implements  EmployeeDAO{
   @Autowired
   private DataSource ds;

   public  int insert(EmployeeBO bo){
     //use ds here
      ...
      ..
   }
```

DB Connectivity Using Spring Boot
==========

=> just add "Spring-boot-starter-jdbc" to the Project to take care Autoconfiguration → gives
- =>spring jars (springcontext, spring jdbc and its dependent jars)
- =>spring beans (DS, JdbcTemplate and etc..)
- =>hikaricp jars

application.properties (Programmer)
==========

```
spring.datasource.driver-class-name= .....
spring.datasource.url= .....
spring.datasource.username= .....
spring.datasource.password= .....
```

DAO class #Programmer
==========

```
@Repository("empDAO")
public class  EmployeeDAOImpl implements  EmployeeDAO{
   @Autowired
   private DataSource ds;

   public  int insert(EmployeeBO bo){
     //use ds here
      ...
      ..
   }
```

working with

Thumb rule while working with spring boot
==========

=>cfg user-defined classes as spring beans using stereo type annotations
=>Cfg pre-defined classes as spring beans using @Bean methods in configuration class only if they are not coming through autoconfiguration based on spring starters that we have added
=> Give instructions to Autoconfiguration process using application.properties/yml file

yaml ->yet another markup language (or)
yaml -> YAML Ain't Markup Language
(Another approach of writing properties file nothing key=value pairs)

These are ready made spring boot starters and we need them either in build.gradle (Gradle) (or) pom.xml (maven)
==========

spring-boot-starter-<*> ---> naming convetion

eg:: spring-boot-starter-jdbc
spring-boot-starter-mail
spring-boot-starter-aop
spring-boot-starter-web
spring-boot-starter-data-jpa
and etc..

URL for starters
==========
https://docs.spring.io/spring-boot/docs/1.3.8.RELEASE/reference/html/using-boot-build-systems.html

in build.gradle
==========
```
// https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-jdbc
implementation group: 'org.springframework.boot', name: 'spring-boot-starter-jdbc', version: '2.3.4.RELEASE'
```

in pom.xml
==========
```
<!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-jdbc -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-jdbc</artifactId>
  <version>2.3.4.RELEASE</version>
</dependency>
```

=>Old projects (small,medium and large Scale) are in spring (Now there are in Maintence mode/Enhancement mode)
=>New Projects (small,medium Scale) are in spring f/w
=>New Projects (Larget Scale) are in spring Boot
=>Migration Projects (Spring to Spring Boot)

=>working with java technologies (like JDBC,SErvlet,jsp and etc..) :: Washing clothes manually
=>working with spring f/w :: semi-automated washing machine
=>working with spring boot :: fully automated washing machine

Spring Boot = spring f/w -- Xml files + AutoConfiguration

There multiple JVM based languages
==========
=> Java , kotlin, scala, go, groovy and etc..

=>Spring Boot gives single useful annotations by combining related multiple annoations
@SpringBootApplication = @Configuration + @ComponentScan + @EnableAutoconfiguration

Spring Boot= spring f/w - xml cfgs + AutoConfiguration + EmbeddedServers +Embedded DB + ...

we can develop spring boot Apps in multiple ways
==========
(a) using CLI (Command Line Interface)
(b) Using start.spring.io website
(c) Using Eclipse IDE having STS Plugin + Gradle/Maven support (best)

Every SpringBoot App main class/runner class(the class that is having main(-) method) must be annotated with @SpringBootApplication.

@SpringBootApplication contains
|--->@ComponentScan :: To recognize java classes/spring beans
and configuration classes automatically that are there in
the same package or sub packages of main class package..

|--->@Configuration ::To make main class/
runner class/starter class
as Configuration class

|--->@EnableAutoConfiguration :: To provide
commons things like jars, pre-defined
classes as spring beans, plugings, servers,
runtime and etc.. automatically based on
spring boot starters that are added.

com.nt (root package) --> Main class /runner /starter class
|--><MainAppclass>.java having @SpringBootApplication + main
|--->com.nt.dao
|---> all dao classes having @Repository
|--->com.nt.service
|--->all service clases having @Service
|--->com.nt.controller
|---> controller class having @Controller
|--->com.nt.beans
|---> helper classes having @Component
|--->com.nt.config
|--->PErsinstenceConfig ,SErviceConfig
and etc.. having @Configuration with @Bean methods

---

org.springframework.boot.
SpringApplication.run(-,-)
|---> Bootstraps the spring Appliation by creating multiple objs
internally like ApplicationContext object (IOC container) and etc..
|--->refreshes the ApplicationContext obj(IOC container) by loading
and pre-instantiating all singleton scope beans..
|-->returns the internally created ApplicationContext obj (IOC
container). So that we can use it to call ctx.getBean(-) methods

Spring Boot= spring f/w - xml cfgs + AutoConfiguration + EmbeddedServers +Embedded DB + ...

we can develop spring boot Apps in multiple ways
==========
(a) using CLI (Command Line Interface)
(b) Using start.spring.io website
(c) Using Eclipse IDE having STS Plugin + Gradle/Maven support (best)

Every SpringBoot App main class/runner class(the class that is having main(-) method) must be annotated with @SpringBootApplication.

@SpringBootApplication contains
|--->@ComponentScan :: To recognize java classes/spring beans and configuration classes automatically that are there in the same package or sub packages of main class package..

|--->@Configuration ::To make main class/ runner class/starter class as Configuration class

|--->@EnableAutoConfiguration :: To provide commons things like jars, pre-defined classes as spring beans, plugings, servers, runtime and etc.. automatically based on spring boot starters that are added.

com.nt (root package)
|--><MainAppclass>.java having @SpringBootApplication + main <-- Main class /runner /starter class
|--->com.nt.dao
|---> all dao classes having @Repository
|--->com.nt.service
|--->all service clases having @Service
|--->com.nt.controller
|---> controller class having @Controller
|--->com.nt.beans
|---> helper classes having @Component
|--->com.nt.config
|--->PErsinstenceConfig ,SErviceConfig and etc.. having @Configuration with @Bean methods

---

org.springframework.boot.
SpringApplication.run(-,-)
|---> Bootstraps the spring Appliation by creating multiple objs internally like ApplicationContext object (IOC container) and etc..
|--->refreshes the ApplicationContext obj(IOC container) by loading and pre-instantiating all singleton scope beans..
|-->returns the internally created ApplicationContext obj (IOC container) So that we can use it to call ctx.getBean(-) methods..

=>Bassed on the starters that we have added to the buildpath /classpath the spring boot performs auto configuration by giving certain pre-defined classes as spring beans and other operations . In this process if we want to provide instructions to Spring boot /starters the we can give use application.properties(main/java/resources).
It is part of Spring boot ecosystem .. i.e we need not cofigure it seperately.

if we add spring-boot-starter-jdbc to buildpath using pom.xml or build.gradle then
we get the following pre-defined classes throguh Autoconfiguration including their jar files
a) HikariDatasource
b) JdbcTemplate,NamedParameterJdbcTemplate
c) DataSourceTransactionManager

pom.xml
==========

```
<!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-
starter-jdbc -->
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-jdbc</artifactId>
   <version>2.3.4.RELEASE</version>
</dependency>
```

build.gradle
==========

```
// https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-jdbc
implementation group: 'org.springframework.boot', name: 'spring-boot-starter-jdbc',
version: '2.3.4.RELEASE'
```

MiniProject using Spring boot
==========

=>Create spring starter giving giving all basic details and selecting the following starters
a) jdbc api b) mysql connector/j c) oracle driver d) lombok api

boot
for spring application.properties list::
https://docs.spring.io/spring-boot/docs/current/reference/html/appendix-application-properties.html#data-properties

In application.properties
==========

```
#datasoruce cfg
#spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriv
#spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe
#spring.datasource.username=system
#spring.datasource.password=manager
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql:///ntsp713db
spring.datasource.username=root
spring.datasource.password=root
```

In DAO classes (Two classes)
==========

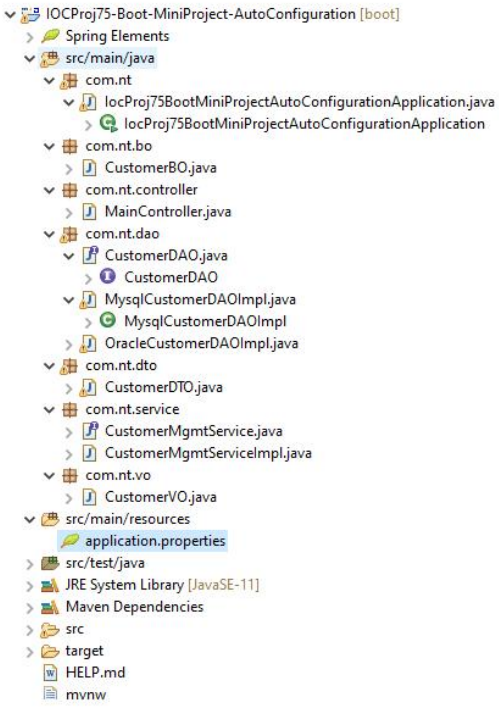
```
@Autowired
private DataSource ds;
```

note:: here we do not confiure HikariDataSource class using @Bean method any where becoz it is coming as spring bean through autoConfiguration

In client App or starter class
==========

In client App or starter class
==========

```
@SpringBootApplication
public class IocProj75BootMiniProjectAutoConfigurationApplication {

    public static void main(String[] args) {
        ApplicationContext ctx=null;
        MainController controller=null;
        Scanner sc=null;
        CustomerVO vo=null;
        String name=null,cadd=null,pamt,rate,time;

        // read inputs
        sc = new Scanner(System.in);
        System.out.println("enter Customername :: ");
        name = sc.next();
        System.out.println("Enter customer Addrs::");
        cadd = sc.next();
        System.out.println("Enter Principle  amount::");
        pamt = sc.next();
        System.out.println("Enter rate of intrest::");
        rate = sc.next();
        System.out.println("Enter time   ::");
        time = sc.next();
        // Store inputs in VO class object
        vo = new CustomerVO();
        vo.setCname(name);
        vo.setCadd(cadd);
        vo.setpAmt(pamt);
        vo.setRate(rate);
        vo.setTime(time);
        //get IOC container
                ctx=SpringApplication.run(IocProj75BootMiniProjectAutoConfigurationApplication.class, args);
                //get Controller class obj
                controller=ctx.getBean("controller",MainController.class);
        //invoke b.method
        try {
          System.out.println(controller.processCustomer(vo));
        }
        catch(Exception e) {
              System.out.println("Internal Problem.");
              e.printStackTrace();
        }

        //close container
        ((ConfigurableApplicationContext) ctx).close();
    }//main
}//class
```



=>Spring boot 2.x uses two dataSources as part of AutoConfiguration hierarchy

a) Hikaricp

2) ApacheDBcp2 (if hikari cp jars not there in build path)

How to work with Apache DBCp2 in spring boot application?

Ans) make sure that hikarcp jars execluded from build path with respect spring -boot -starter- jdbc and and add apache DBCp2 jar file to build path by adding following entries to pom.xml file

```
<dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-jdbc</artifactId>
            <exclusions>
                <exclusion>
                    <groupId>com.zaxxer</groupId>
                    <artifactId>HikariCP</artifactId>
                </exclusion>
            </exclusions>
        </dependency>

<!-- https://mvnrepository.com/artifact/org.apache.commons/commons-dbcp2 -->
<dependency>
  <groupId>org.apache.commons</groupId>
  <artifactId>commons-dbcp2</artifactId>
  <version>2.8.0</version>
</dependency>
```

note:: if both are there , it will take hikaricp

To break the default DataSource algorithm of spring boot and to configure u r choice dataSource class as default Datasoruce class of autoConfiguration then specify that datasoucer class name in application.properties also add relavent jar file to build path..

in application.properties

```
# To change default DataSource type of Autoconfiguration
spring.datasource.type=com.mchange.v2.c3p0.ComboPooledDataSource
```

in pom.xml

```
<dependency>
  <groupId>com.mchange</groupId>
  <artifactId>c3p0</artifactId>
  <version>0.9.5.5</version>
</dependency>
```

=>To findout the list of pre-defined classes that comes as spring beans through Autoconfiguration , we can use xxxxAutoConfiguration class names of different packages from
spring-boot-autoconfigure-<version>.jar

```
org.springframework.boot.autoconfigure.jdbc
  DataSourceAutoConfiguration.class
  DataSourceBeanCreationFailureAnalyzer.class
  DataSourceConfiguration.class
  DataSourceInitializationConfiguration.class
  DataSourceInitializer.class
  DataSourceInitializerInvoker.class
  DataSourceInitializerPostProcessor.class
  DataSourceJmxConfiguration.class
  DataSourceProperties.class
  DataSourceSchemaCreatedEvent.class
  DataSourceTransactionManagerAutoConfiguration.class
  EmbeddedDataSourceConfiguration.class
  HikariDriverConfigurationFailureAnalyzer.class
  JdbcOperationsDependsOnPostProcessor.class
  JdbcProperties.class
  JdbcTemplateAutoConfiguration.class
  JdbcTemplateConfiguration.class
  JndiDataSourceAutoConfiguration.class
  NamedParameterJdbcOperationsDependsOnPostProcessor.class
  NamedParameterJdbcTemplateConfiguration.class
  XADataSourceAutoConfiguration.class
```

spring-boot-starter-jdbc related classes that comes as spring beans through autoconfiguration

note:: To make certains clases of any spring boot starter , not coming through AutoConfiguration we need to use "execlude" param of @SpringBootApplicaiton annotation.

```
@SpringBootApplication
    (exclude = {JdbcTemplateAutoConfiguration.class,DataSourceTransactionManagerAutoConfiguration.class})
```

To disable spring boot banner
==========================

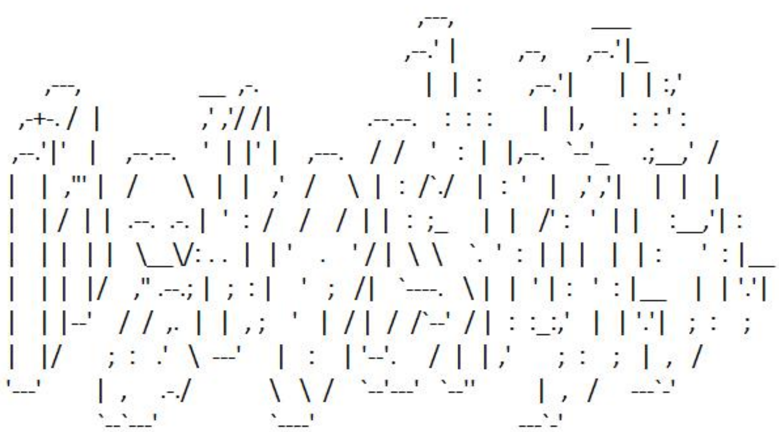
```
# To disable spring boot banner
spring.main.banner-mode=off
                        console  (default)
                        log   -->writes to log file
```

To Add custom startup banner to spring boot application
===

step1) get custom banner content from online
https://devops.datenkollektiv.de/banner.txt/index.html

step2) copy and paste the above banner content to a txt file ..

```
src/main/java
    |--->com/nt/banner/mybanner.txt
```



step3) enable spring boot banner and specify the the above file location... in application properties

```
#To enable spring boot banner
spring.main.banner-mode=console
#custom banner location
spring.banner.location=classpath:/com/nt/banner/mybanner.txt
```

Another approach of bootstapping spring boot application from the main class/starter class

In the main(-) method

```
SpringApplication app=new SpringApplication();
 app.setBannerMode(org.springframework.boot.Banner.Mode.CONSOLE);
 ctx=app.run(IocProj75BootMiniProjectAutoConfigurationApplication.class, args);
```

The spring-boot-starter-parent that we add to our pom.xml file will inherit spring boot parent project to our spring boot project (child project) using mave inheritence ..gives mutiple dependencies , maven plugings and congfiguration properites to child project (our project) from parent boot project..

=>add following line in our project (child project) pom.xml in order to inherit from spring boot parent project available in maven central repository

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.3.4.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>
```

The spring-boot-starter-parent project is a special starter project – that provides default configurations for our application and a complete dependency tree to quickly build our Spring Boot project.
It also provides default configuration for Maven plugins such as maven-failsafe-plugin, maven-jar-plugin, maven-surefire-plugin, maven-war-plugin.
Beyond that, it also inherits dependency management from spring-boot-dependencies which is the parent to the spring-boot-starter-parent.

For more info refer :: https://www.baeldung.com/spring-boot-starter-parent