Oct 04 Realtime DI

3 Types of JAva beans
==========
a) VO class (Value Object class)
b) DTO class (Data Transfer Object class)
c) BO class (Business object class)

VO class ==> carries inputs or outputs
Generally contains all its properties as string properties
DTO class ==> carries data with in projects across the multiple layers
and alos carries data across the multiple projects
BO class/Domain class/Entity class/Model class ==> represents persistable data (to be saved/inserted)
or persistent data (retrieved/collected from db table)

Standalone Layered App
==========

Client App --VO class--> Controller class --DTO class--> service class --BO class--> DAO class --SQL query inputs--> Db s/w

if one class of one layer wants to pass more than 3 values to another of another layer or same layer we do not desing methods having more than 3 params.. we design methods having java bean type parameters.

=>Client App passes multiple values to controller class as VO class object
=>Controller class passes multiple values to Service class as DTO class object
=>Service class passes multiple values to DAO class as BO class object.
=> One java Project sends multiple vlaues to another Java Project (Flikart to Paypal , flipart to Gpay and etc..) in the form of DTO class object.

Web based Layered Application
==========

html/jsp form in browser --req obj data--> Controller Servlet --DTO class or VO class obj--> Service class --BO class obj--> DAO --SQL query inputs--> DB s/w

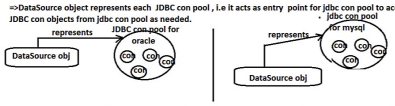
---

DataSource and JDBC con pool
==========

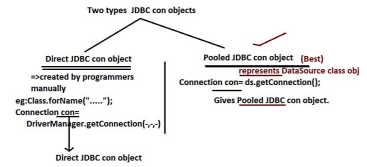
JDBC con pool
==========
=> It is a factory that contains set of readily avalable jdbc con objects before actually being used.
=>the jdbc con pool advantages are
i) resuability of jdbc con objs
ii) with minimum no.of jdbc con objects , we can make max client Apps/requets talking to DB s/w.
iii) creation of jdbc con obj in jdbc con pool,managing jdbc con object and destroying jdbc con object if needed will be taken care by jdbc con pool.. not the programmer.

=> all JDBC con objects in JDBC con pool represents the connectivity with same Db s/w..
eg1: jdbc con pool for oracle means , all the jdbc con objs in that jdbc con pool represents the connectivity with same oracle Db s/w
eg2: jdbc con pool for mysql means , all the jdbc con objs in that jdbc con pool represents the connectivity with same mysql Db s/w

=>DataSource object represents each JDBC con pool , i.e it acts as entry point for jdbc con pool to access all the JDBC con objects from jdbc con pool as needed.



=>DataSource obj means .. It is the object of java class implementing javax.sql.DataSource(I)

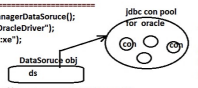


=> By creating object for DataSoruce class (class implementing DataSoruce(I)) and filling with JDBC driver details and db details (jdbc properties) .. we can make dataSoruce creating JDBC con pool having given details based jdbc con objs.. and that jdbc con pool will be represented by DataSource object.

=>Different classes implmenting DataSource (I) (popular classes)
=> org.sf.jdbc.datasoruce.DriverManagerDataSource (given by spring api)
=> <pkg>.HikariDataSource (given by Hikari CP)
=> <pkg> . BasicDataSource (given by Apache DBCP)
=> <pkg> . ComboPooledDataSource (Given by C3PO)
and etc..
=>Hikari CP(Best),Apache DBCP, C3P0 and etc.. third party supplied JDBC con pool libaries.

creating spring supplied DataSource object representing JDBC con pool
==========

```
DriverManagerDataSoruce ds=new DriverManagerDataSoruce();
ds.setDriverClassName("oracle.jdbc.driver.OracleDriver");
ds.setUrl("jdbc:oracle:thin:@localhost:1521:xe");
ds.setUsername("system");
ds.setPassword("manager");
```



=>DriverManagerDataSoruce intially creates only 1 one con object.. as need is increased it becomes ready to create more jdbc con objects.

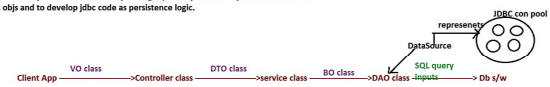
//To get one pooled jdbc con obj
Connection con=ds.getConnection();

In spring env.. we can make IOC container creating java class objects and performing injections as show below even for dataSoruce object.

In applicationContext.xml

```
<beans ....>
 <bean id="drds" class="org.sf.jdbc.datasoruce.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>

</beans>
```

=>We need to Inject the above DataSource obj to DAO class , So the methods of DAO class can use the injected DataSource object to get pooled jdbc con object to create other jdbc objs and to develop jdbc code as persistence logic.



=>Total 4 injections are required in this Layered App
a) jdbc properites to DS (DataSoruce)
b) DS(DataSource) to DAO
c) DAO to Service
d) service to Controller

```
//DAO class
==========

public class StudentDAO {
    //HAS- property   (interface type)
    private  DataSource  ds;
        //for constructor Injection
    public  StudentDAO(DataSource ds){
      this.ds=ds;
    }

     public  int insertStudent(StudentBO bo){
      try{
       //get Pooled JDBC con object from JDBC con pool
       // using DataSource obj
       Connectiion con=ds.getConnection();
       ....
       .... //jdbc code to insert the record
                by collecting  values to  StudentBO class obj
        ...
       }
      catch(SQLExcception se){
         se.printStackTrace();
       }
      return 0/1;
     }//method
}//class
```

improved applicationContext.xml
==========

```
<beans ....>
 <bean id="drds" class="org.sf.jdbc.datasoruce.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="stDAO"  class="<pkg>.StudentDAO">
    <constructor-arg  name="ds"   ref="drds"/>
 </bean>

    ....
    ....
    ...
</beans>
```

Exception propagation in Layered Apps

Oct 5 Realtime DI Story Board

Layered App
==========
Client App ------------>controller ----------> service ------------> DAO ----------------> DB s/w

DAO class
============
public int insert(StudentBO bo){
try{
.....
..... //jdbc code
....
}
catch(SQLException se){
...
}
}

If use try with catch blocks or try with resoruce in the layers of layered App .. the raised exception will be suppressed or eaten by catch of block that layer itself.. Actually the exception/problem must be propagated other layers until it reaches to Enduser throguh Client App.

=> Enduser is giving inputs to Client App..So results or error(exception) should be given to enduser throguh Client app.. For that we need to go exception progapagation with the support exception rethrowing concept using "throw" and "throws" key words.

DAO class
============
public int insert(StudentBO bo) throws Exception
try{
.....
..... //jdbc code
....
}
catch(SQLException se){
se.printStrackTrace();
throw se; //re throwing the exception
}
}

Exception propagation

=>In Layered Apps .. no layer should supress the exception by catching the exception.. every layer should rethrow the raised exception to its next layer until it reaches to Enduser through Client App .. w.r.t our App DAO , Service, Controller should propagate the excpetion to thier next layer.. where as Client App should catch and handle exception to display exception related message to enduser as non-technical guidign message.
(Basic Concept is Exception propagation.. In the Exception rethrowing concept will be used)

Q) How did propagate/Pass exception raised DAO class to Client App or enduser

=>DAO passes to Service class , Service class passes to Controller , controller class pass to Client app and The Client App catches and handles exception to display non-technical messages related to raised exception to enduser.

Layered App
==========
Client App ------------>controller ----------> service ------------> DAO ----------------> DB s/w

Exception handling using to display exception related messages to enduser.

Exception propagation to pass exception to next layer

Story board of Layered Application
==========
=>Registering Employee after calculating his gorssSalary and net Salary details
grossSalary :: basica salary + additions (pf+ta+da+hra+LTA+ and etc...)
netSalary :: Gross Salary - deductions (IT , PT and etc..)

Gross Salary = basic salary + 40% of basicSalary
Net salary= Grosss Salary - 20% basic salary

oracle Db s/w
layered_spring_employee

eno(pk) (n)	ename (vc2)	desg(vc2)	basicSalary(f)	grossSalary(f)	netSalary(f)
1	raja	devloper	100000	140000	98000

3 java beans clases
-> EmployeeDTO.java (DTO class) (To pass data from controller to service)
=>EmployeeVO.java (VO class) (To pass data from Client to Controller)
=> EmployeeBO.java (BO class) (To pass data from Service to DAO)

DAO interface and DAO Impl class
==========

IEmployeeDAO.java (DAO interface)
package com.nt.dao;
public interface IEmployeeDAO {
public int insertEmployee(EmployeeBO bo)throws Exception;
}

EmployeeDAOImpl.java (DAOImpl class)
package com.nt.dao;
public class EmployeeDAOImpl implements IEmployeeDAO{
//HAS-A property
private DataSource ds;

public EmployeeDAOImpl(DataSource ds){
this.ds=ds;
}

(28)
public int insertEmployee(EmployeeBO bo)throws Exception{
try{
//get pooled JDBC connection
Conenction con=ds.getConnection();
....
.... //jdbc code to insert record by collecting values
... // from BO class obj
}
catch(SQLException se){
se.printStackTrace();
throw se;
}
catch(Exception e){
e.printStackTrace();
throw e;
}
finally{
....
... //to close JDBC objs
...
}
return 0/1;
}//method
}//class

(29) real

applicationContext.xml

7 to 20 takes place in the InMemory MetaData of applicationContext.xml file

<beans .....> (167)
<bean id="drds" class="org.sf.jdbc.datasource.DriverManagerDataSource">
.... //jdbc properties injection (17) setter injetion of jdbc properties
...
</bean>
<bean id="empDAO" class="com.nt.dao.EmployeeDAOImpl"> (137)
(18) <constructor-arg name="ds" ref="drds"/> (14)
</bean>
<bean id="empService" class="com.nt.service.EmployeeMgmtSreviceImpl"> (107)
(19) <constructor-arg name="dao" ref="empDAO"/> (11)
</bean>
<bean id="controller" class="com.nt.controller.MainController"> (77)
(20) <constructor-arg name="service" ref="empService"/> (8)
</bean>
</beans>

Internal cache of IOC container (67) (97) (127) (157)

drds	DriverManagerDataSource obj ref
empDAO	EmployeeDAOImpl class obj ref
empService	EmployeeServiceImpl class obj ref
controller	MainController class obj ref

(21)

Service Interface and Its Impl class
==========

IEmployeeMgmtService.java (service interace)
package com.nt.service;
public interface IEmployeeMgmtService{
public String registerEmployee(EmployeeDTO dto)throws Exception
}

EmployeeMgmtServiceImpl.java (service impl class)
package com.nt.service;
public final class EmployeeMgmtServiceImpl implements IEmployeeMgmtService{
//HAS-A property
private IEmployeeDAO dao;

public EmployeeMgmtServiceImpl(IEmployeeDAO dao){
this.dao=dao;
}

(26)
public String registerEmployee(EmployeeDTO dto)throws Exception{
//collect inputs from DTO and generate grossSalary,netSalary
.....
.....
//prepare EmployeeBO class obj having persitable data (ename,desg,basicSalary,grossSalary,netSalary)
.....
.....
//use DAO (27)
(31) int count=dao.insertEmployee(bo);
if(count==0)
return "Employee registration failed";
else (32)
return "Employee registered successfully";
} //method
}//class

Controller class
==========

package com.nt.controller;
public class MainController {
//HAS-A property
private IEmployeeMgmtService service;

public MainController(IEmployeeMgmtService service){
this.service=service;
}

(24)
public String processEmployee(EmployeeVO vo)throws Exception{
//convert EmployeeVO class obj to EmployeeDTO class obj
.....
//use service (25)
String result=service.registerEmployee(dto);
return result;
}//method
}//class

(1) running theApp

Client App
==========
package com.nt.test;

public class RealtimeDITest{
p s main(String args[]){ (2)

//create IOC container
..... (3) Containe create
..... having InMempory MEtaData of spring bean cfg file
//read inputs as stirng values
and store them into EmployeeVO class obj
...
(4) ...
...

controller
empService
empDAO
drds

//get Controller class obj from IOC container
(22) MainController controller=factory.getBean("controller",MainController.class);
//invoke b.method (5)
try{
(33) (23)
String msg=controller.processEmployee(vo);
System.out.println(msg); (34)
}
catch(Exception e){
e.prinStackTrace();
S.o.p("Internal problem");
}
}//main
}//class

jars /dependencies
=>spring context support
=> ojdbc6/7/8.jar
=> spring jdbc (for DriverManagerDataSource)

Procedure to develope Realtime DI /Realtime StrategyDP Layered App or MiniProject [Oct 07 First App Develop Realtime DI]

step1) Use SQL develop create Db table in oracle Db s/w

RealTimeDI_ Layered_spring_employee

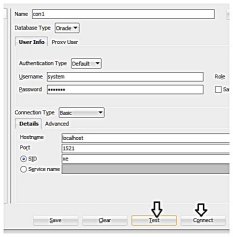
- **|-->eno (pk) (n)**
- **|-->ename (vc2)**
- **|-->desg (vc2)**
- **|-->basicSalary (float)**
- **|-->grossSalary (float)**
- **|-->netSalary (float)**

GUI DB tools make DB operations easy

eg:: SQL developer for oracle
TOAD for oracle
mysql workbench
ToAD for mysql and etc..

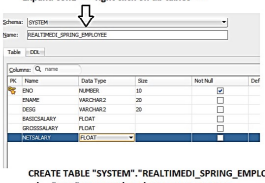
note:: SQL developer is seperate installation by dowloading from
https://www.oracle.com/tools/downloads/sqldev-downloads.html

a) create connection using SQL developer



b) create the above db table

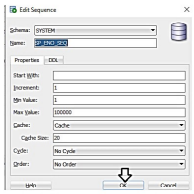
Expand con1 ---> right click on db tables --->




```
CREATE TABLE "SYSTEM"."REALTIMEDI_SPRING_EMPLOYEE"
(    "ENO" NUMBER(10,0) NOT NULL ENABLE,
     "ENAME" VARCHAR2(20 BYTE),
     "DESG" VARCHAR2(20 BYTE),
     "BASICSALARY" FLOAT(126),
     "GROSSSALARY" FLOAT(126),
     "NETSALARY" FLOAT(126),
      CONSTRAINT "REALTIMEDI_SPRING_EMPLOYEE_PK" PRIMARY KEY ("ENO"));
```

step2) create SEQuence in oracle Db s/w using sQL Query to generate ENO column values dynamically..

=> Right click on sequences -->new --->



=>Generated SQL query is

```
CREATE SEQUENCE  "SYSTEM"."SP_ENO_SEQ"
MINVALUE 1 MAXVALUE 100000 INCREMENT BY 1 START WITH 1 CACHE 20 NOORDER  NOCYCLE ;
```

step3) create Maven Project add the following dependencies in pom.xml file

spring context support
spring jdbc
ojdbc6/7/8

In pom.xml file

```
<!-- https://mvnrepository.com/artifact/org.springframework/spring-context-support -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context-support</artifactId>
    <version>5.3.10</version>
</dependency>
    <!-- https://mvnrepository.com/artifact/com.oracle.ojdbc/ojdbc8 -->
<dependency>
    <groupId>com.oracle.ojdbc</groupId>
    <artifactId>ojdbc8</artifactId>
    <version>19.3.0.0</version>
</dependency>
    <!-- https://mvnrepository.com/artifact/org.springframework/spring-jdbc -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>5.3.10</version>
</dependency>
```

step4) create the following packages in src/main/java folder

step5) Develop VO ,DTO ,BO classes

=> While taking properties in java bean classes ... prefer taking wrapper types becoz they hold "null" values representing no values .. So towards insertion to DB and retrieving values from DB for report generation becomes easy.. where as the simple data type variables like int,float, long and etc.. contain 0,0.0f , 0.0 as the defalues .. some these values insertion or display may change the meaning the of Data..

EmployeeDTO.java , EmployeeVO.java , EmployeeBO.java

For Carrying Inputs

Contains persitable data (to inserted)

step6) Develop DAO Interface and DAO Impl class performing exception rethrowing and exception propagation

IEmployeeDAO.java (DAO Interface)
EmloyeeDAOImpl.java (DAO Impl class)

step7) Develope Service Interface and Service Impl class

IEmployeeMgtmService.java (Service Interface)
EmployeeMgmtServiceImpl.java (Service Impl class)

step8) Develop the Controller class

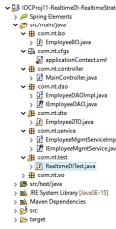
MainController.java

step9) DEvelop Spring Bean cfg file
applicationContext.xml

- **|--> cfg DriverManagerDataSource, EmployeeDAOImpl , EmployeeMgmtServiceImpl,MainControlelr class as spring beans**
- **|--->inject DataSoruce to DAO , DAO to service , SErvice to Controller classes**

step10) Develop the Client App

RealtimeDITest.java
(Client App)



Assigment ::

a) Customer registration after calculating billamount and providing discount (AP ,MP, Bihar)

b) Customer registration after calculating simple intrest amount for the given pAmt, rate, time (TS ,MH, UP)

b) Student registration after calculating total, avg, result for the given m1,m2,m3 (ODS, WB, PNG,KR, TN,RAJ, rest of india and forieners)

MySQL Db s/w
==========

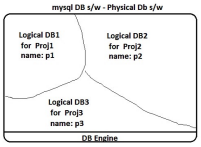
Oct 08 Realtime DI-MySql

=>Another flavour mysql Db s/w is Maria DB

type :: Db s/w
version : 8.x
vendor :: Devx/ Sun Ms /oracle corp
open source
To download mysql Db s/w https://www.mysql.com/downloads/
Gives built-in GUI Db tool called mysql workbench
default admin username : root
default admin password : root (should be chosen during the installation)
allows to create Logical DBs .. default logical dbs are "world", "sakila"

Logical DB is a Logial partiation of physical DB s/w ..Generaly it will created on 1 per Project basis

mysql DB s/w - Physical Db s/w



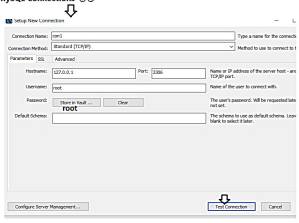
=>Every Project related
db tables , pl/sQL procedures
functions, view ,sequences and etc..
will be created .. in that project related
Logical DB.

In oracle DB s/w , every logical DB is identified with its SID (service id).. In Expression edition Oracle DB s/w
the default Logical Db name is "xe" .. In enterprise edition Oracle DB s/w the default Logical Db name is ORCL

Procedure to create logical Db in mysql DB s/w having db table using mysql work bench

step1) launch mysql work bench and create the the connection with mysql Db s/w..
lauch mysql work bench --->

MySQL Connections ⊕ ⇘



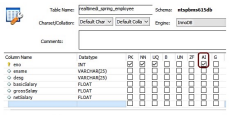
step2) create Logical Db in mysql DB s/w

(create logical DB)

logical Db name :: ntspbms615db ---next -->finish

step3) create db table in the above logical DB

expland our logical DB --->
right click tables --->



apply --->ok

=>Oracle supports sequences
=>mysql does not support sequences.. as alternate we can make
the PK colum as Identity column having auto increment constaint.
(Initial value max val + 1 .. next values previous val +1)

```
CREATE TABLE `ntspbms615db`.`realtimedi_spring_employee` (
  `eno` INT NOT NULL AUTO_INCREMENT,
  `ename` VARCHAR(25) NULL,
  `desg` VARCHAR(25) NULL,
  `basicSalary` FLOAT NULL,
  `grossSalary` FLOAT NULL,
  `netSalary` FLOAT NULL,
  PRIMARY KEY (`eno`),
  UNIQUE INDEX `eno_UNIQUE` (`eno` ASC) VISIBLE);
```

=>The type4 mechanism based jdbc driver for mysql Db s/w is called Connector/J JDBC driver and its details are

driver class name :: com.mysql.cj.jdbc.Driver
jdbc url :: jdbc:mysql:///<logicalDB> (for Local mysql DB s/w)
jdbc:mysql://<hostname>:<portno>/<LogicalDB> (for remote mysql DB s/w)
of mysql Db s/w
Jar file :: mysql-connector-java-<version>.jar (supports auto loading of jdbc driver)

```
<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>8.0.26</version>
</dependency>
```

Making our Mini Project working for mysql Db s/w prividing loose coupling to change from
oracle to mysql and mysql to Oracle Db s/w

step1) keep the Mini Project ready

step2) add mysql Connector/J JDBC driver dependency to pom.xml

```
<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>8.0.26</version>
</dependency>
```

step3) make sure that db table in mysql Db s/w is avalable
refer above

step4) Develope seperate DAOImpl class having persistence logic related mysql Db s/w


```
package com.nt.dao;

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

import javax.sql.DataSource;  //ctl+shift+o :: To import pkgs

import com.nt.bo.EmployeeBO;

public class EmployeeMySQLDAOImpl implements IEmployeeDAO {
    private static final String  EMP_INSERT_QUERY="INSERT INTO
REALTIMEDI_SPRING_EMPLOYEE(ENAME,DESG,BASICSALARY,GROSSSALARY,NETSALARY) VALUES(?,?,?,?,?)";
    //HAS-A Property
    private  DataSource ds;

    //Alt+shift+s, o
    public EmployeeMySQLDAOImpl(DataSource ds) {
        System.out.println("EmployeeMysqlDAOImpl: 1- param constructor");
        this.ds = ds;
    }


    @Override
    public int insertEmployee(EmployeeBO bo) throws Exception {
        System.out.println("EmployeeMysqlDAOImpl.insertEmployee()");
        int result=0;
        try(Connection con=ds.getConnection();
                PreparedStatement ps=con.prepareStatement(EMP_INSERT_QUERY);
                ){
            //set values to query params
            ps.setString(1,bo.getEname());
            ps.setString(2, bo.getDesg());
            ps.setFloat(3, bo.getBasicSalary());
            ps.setFloat(4, bo.getGrossSalary());
            ps.setFloat(5, bo.getNetSalary());
            //execute the Query
            result=ps.executeUpdate();
        }//try
        catch(SQLException se) {
            se.printStackTrace();
            throw se; //exception rethrowing
        }
        catch(Exception e) {
            e.printStackTrace();
            throw e; //exception rethrowing
        }

    }
```

step5) cfg DriverManagerDataSoruce and DAOImpl class related to mysql Db s/w.
in applicationContext.xml

```
<!-- DataSource cfg -->
<bean id="drdsOra" 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="drdsMysql" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
  <property name="driverClassName" value="com.mysql.cj.jdbc.Driver"/>
  <property name="url" value="jdbc:mysql:///ntspbms615db"/>
  <property name="username" value="root"/>
  <property name="password" value="root"/>
</bean>

<!-- DAO class cfg -->
<bean id="empOraDAO" class="com.nt.dao.EmployeeOracleDAOImpl">
  <constructor-arg name="ds" ref="drdsOra"/>
</bean>
<bean id="empMySqlDAO" class="com.nt.dao.EmployeeMySQLDAOImpl">
  <constructor-arg name="ds" ref="drdsMysql"/>
</bean>
```

step6) In service Impl cfg ..change DAO impl class bean id accordingly

```
<!-- service class cfg -->
<bean id="empService" class="com.nt.service.EmployeeMgmtServiceImpl">
  <!-- <constructor-arg name="dao" ref="empOraDAO"/> -->
  <constructor-arg name="dao" ref="empMySqlDAO"/>
</bean>
```

candidate key column
==========
=> The column db table whose values can be used to identify and access the records is called candidate key column.
Generally this colum does not allow duplicates and does not allow null values.

Person_Info (db table)

|---> aadharNo (ckc)
|---> name
|---> addrs
|----> passportNo (ckc)
|--->voterId (ckc)
|--->bankAcno (ckc)
|--->email Id (ckc)

CKC---Candidate key column

=>Primary key, Foreign key, Unique key, Not Null key and etc.. physical keys which can applied pratically on db table cols

candiate key , natural key, surrogate key and etc. logical keys

Natural key column
==========
=>The candiate key column Whose values are having business meaning and will be changed becoz of the outside world business polocies is called natual key column.. These column values expected from Endusers i.e they can not auto generated by App or DB s/w..

eg:: adharNo, voterId , bankAcno , email Id ,passportNo, aadharNo and etc..

Surrogate key column
==========
=> The candidate key column whose values are generated by underlying DB s/w or underlying App with out having any business meaning of outside world is called Surrogate key column

eg:: sequence value generated by oracle Db s/w
eg:: Auto icrment value given by mysql Db s/w
eg: Generators generated values in Hiberante programming (sequence, hilo ,seqhilo ,increment and etc..)

Limitations of taking Natural key column as PK column
==========
a) Values are very lengthy , so they need more memory to store
b) values are having business meaning i.e they will be changed becoz of the outside business polocies .. So change in natural key colum value that is taken PK column effects dependent db tables (FK colums) and relavent java classes and their dependent classes .. this change is going be very costly...
c) These values are expected from enduser ..if enduser does not supply these values .. record insertion process will be failed.

note:: We should not prefer taking Natural key columns as PK colums i.e taking aadharno, voterId , passport no and etc.. as PK column is total bad pratice..

Advantages of taking Surrogate key column as PK column
==========
a) values are shorter values .. So they do not need much memory
b) values are generated by underlying Db s/w or Application dynamically with out having any business meaning. So any change in outside business policies these values will not be distrbuted.
c) These values are not expected from endusers.. So they will not raise any problem towards record insertion

eg:: generated student no (pk col value) using sequence of oracle
eg:: generated student no (pk col value) using identity col of mysql (applying autoincrement constraint)

Singleton java class
==========
Tommorrow :: 12:15 noon Spring 7:30 batch

=>The java class that allows us to create only one object in any situation is called singleton java class

=>Instead of creating muliple objects for java class having no data or fixed data or sharable data .. just create object and use it for multiple times .. In that situation it better to take java class as singleton java class

pre-defined singleton classes in java api
a) java.lang.Runtime (Becoz one java app execution contains only 1 runtime)
b) java.awt.Desktop (Becoz one computer contains only 1 desktop)
c) org.apache.log4j.Logger (Every project wants to have single Logging mechanism for all classes)
and etc..

=> Though java class allows to create more objects .. if some one is just creating only object for that class then that class is not called singleton java class..
eg:: Servlet comp class is not singleton java class .. becoz servlet comp class not having restruction on more objects creation.. the Servlet container just creating only one object and using it.

=>For real singleton java class should returns already created object when we attempt to create more than object.

Developing Singleton java class with minimum standards
==========
a) Take only one private 0-param cosntructor (To stop object creation out side the class using new Operatior)
b) Take static factory method to check wheather object is already ready created or not.. if created return existing object ref .. if not created , then create new object and return its reference.
c) Take private static variable of type current singleton class to hold the single object that will be created.. same variable will be used in static factory method to check wheather single object already created or not..

Oct 10 Singleton Java class and Factory method Bean instantiation

Singleton java class

=>The java class that allows us to create only one object in any situation is called singleton java class

=>Instead of creating mulitple objects for java class having no data or fixed data or sharable data .. just create object and use it for multiple times .. In that situation It better to take java class as singleton java class

pre-defined singleton classes in java api
- a) java.lang.Runtime (Becoz one java app execution contains only 1 runtime)
- b) java.awt.Desktop (Becoz one computer contains only 1 desktop)
- c) org.apache.log4j.Logger (Every project wants to have single Logging mechanism for all classes)

and etc..

=> Though java class allows to create more objects .. If some one is just creating only object for that class then that class is not called singleton java class..
eg:: Servlet comp class is not singleton java class .. becoz servlet comp class not having restruction on more objects creation.. the Servlet container just creating only one object and using it.

=>For real singleton java class should returns already created object when we attempt to create more than object.

Developing Singleton java class with minimum standards

a) Take only one private 0-param cosntructor (To stop object creation out side the class using new Operatior)
b) Take static factory method to check wheather object is already ready created or not.. If created return existing object ref .. If not created , then create new object and return its reference.
c) Take private static variable of type current singleton class to hold the single object that will be created.. same variable will be used in static factory method to check wheather single object already created or not..

```
//Printer.java (Singleton java class with minimum standards)
package com.nt.ston;

public class Printer {
      //static  variable  to hold reference of the single object
   private static Printer INSTANCE;
      private Printer() {
            System.out.println("printer:: 0-param constructor (private)");
      }

      //static factory method
      public  static  Printer getInstance() {
            //singleton logic
            if(INSTANCE==null)
                  INSTANCE=new Printer();

            return INSTANCE;

      }

      //b.method
      public  void print(String msg) {
            System.out.println(msg);
      }
}
```

Client App

```
package com.nt.test;

import com.nt.ston.Printer;

public class SingletonTest {

      public static void main(String[] args) {
            Printer p1=Printer.getInstance();
            Printer p2=Printer.getInstance();
            System.out.println(p1.hashCode()+"  "+p2.hashCode());
            System.out.println("-------------------------");
            System.out.println("p1==p2"+(p1==p2));


      }

}
```

output

```
==========
Printer:: 0-param constructor (private)
359023572  359023572
-------------------------
p1==p2true
```

=> IOC container can perform spring bean class instantiation using

a) 0-param consturctor (If spring bean is cfg with no injections or with only setter injections)
b) parameterized consturctor (if spring bean is cfg with constictor injection)
c)static factory method (if we specify "factory-method" attribute in <bean> tag)
d)instance factory method (if we specify "factory-method","factory-bean" attributes in <bean> tag)

=>The method that is capable returning its own class or related class or unrelated class obj is called factory method

Factory methods are two types
- a)static factory method
- b) instance factory method

examples for static factory methods

```
String s1=String.valueOf(10);                 | static factory method
Thread t=Thead.currentThread();               | returning its own class obj
Class c=Class.forName("java.util.Date");      |

Calendar cal=Calendar.getInstance();          | static factory method
                                              | returning relevant class obj
                                              |  (Calenar.getInstance() returns GregorianCalendar class obj
                                              |                          whose  super class is Calendar (AC))

Properties props=System.getProperties();      | static factory method
Connection con=DriverManager.getConnection(-,-,-) | returning unrelated class
                                              | object.
```

usecases :: In the impl of Factory Pattern , Singleton java class pattern
If class is having only private constructors , then we use static factory method of that class to get the object being frm outside the class

examples of instance factory methods — usecases:: To create new objects using existing object and its data.

```
String s1=new String("hello");
String s2=s1.concat("123"); //gives  hello123
-------------------------------------------------
String s1=new String("hello how are u");
String s2=s1.substring(0,5);   //gives hello
-------------------------------------------------
StringBuffer sb=new StringBuffer("hello ,how are u?");
String s1=sb.substring(0,5);
-------------------------------------------------
Ddate  d=factory.getBean("d1",Date.class);
-------------------------------------------------
Statement st=con.createStatement();
```

//Instance factory returning thier own class objects

instance factory method returning unrelated class obj becoz String, StringBuffer are not in inheritence relationship

---

```
<?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-4.1.xsd">

<!-- static factory method returning its own class object -->
<bean id="c1" class="java.lang.Class" factory-method="forName"> <!-- calls Class.forName("java.util.Date") method -->
   <constructor-arg value="java.util.Date"/> <!-- not for constructor injection.. It is for passing factory method arg value -->
</bean>

<!-- static factory method returning relevant class object -->
<bean id="cal"  class="java.util.Calendar" factory-method="getInstance"/> <!-- calls Calendar.getInstance() -->

<!-- static factory method returning unrelated class object -->
<bean id="props"  class="java.lang.System" factory-method="getProperties"/> <!-- calls System.getProperties() -->

<!-- instance factoty method returning its own class object-->
<bean id="s1" class="java.lang.String">
   <constructor-arg value="hello"/> <!-- for constructor injection -->
</bean>
<bean id="s2"  factory-bean="s1" factory-method="concat"> <!-- calls  s1.concat("123") method -->
   <constructor-arg  value="123"/> <!-- To pass factory method arg value -->
</bean>

<!-- instance factoty method returning unrelated class object-->
<bean id="sb" class="java.lang.StringBuffer">
   <constructor-arg value="hello,how are u"/> <!-- for constructor injection -->
</bean>
<bean id="s3"  factory-bean="sb" factory-method="substring"> <!-- calls sb.substring(0,5) -->
      <constructor-arg value="0"/>
      <constructor-arg value="5"/>
</bean>



</beans>
```

```
package com.nt.test;

import java.util.Calendar;
import java.util.Properties;

import org.springframework.beans.factory.support.DefaultListableBeanFactory;
import org.springframework.beans.factory.xml.XmlBeanDefinitionReader;

public class FactoryMehod8eanInstantiationTest {

      public static void main(String[] args) {
            //create IOC container
            DefaultListableBeanFactory factory=new DefaultListableBeanFactory();
            XmlBeanDefinitionReader reader=new XmlBeanDefinitionReader(factory);
            reader.loadBeanDefinitions("com/nt/cfgs/applicationContext.xml");
            //get Spring bean class obj
            Class clazz=factory.getBean("c1",Class.class);
            System.out.println("clazz obj  class name::"+clazz.getClass()+" clazz obj data"+clazz.toString());
            System.out.println("-------------------------------------");
            Calendar cal=factory.getBean("cal",Calendar.class);
            System.out.println("cal obj  class name::"+cal.getClass()+" cal obj data"+cal.toString());
            System.out.println("-------------------------------------");
            Properties props=factory.getBean("props",Properties.class);
            System.out.println("props obj  class name::"+props.getClass()+" props obj data"+props.toString());
         System.out.println("=========================================");
            String s2=factory.getBean("s2",String.class);
            System.out.println("s2 obj class name::"+s2.getClass()+"  s2 obj data ::"+s2);
            System.out.println("-------------------------------------");
            String s3=factory.getBean("s3",String.class);
            System.out.println("s3 obj class name::"+s3.getClass()+"  s3 obj data ::"+s3);

      }

}
```

output

```
==========
clazz obj  class name::class java.lang.Class clazz obj dataclass java.util.Date
-------------------------------------
cal obj  class name::class java.util.GregorianCalendar cal obj datajava.util.GregorianCalendar[time=1633853895568,areFieldsSet=true,areAllFieldsSet=true,lenient=true,zone=sun.util.calendar.ZoneInfo[id="Asia/Calcutta",offset=19800000,dstSavings=
0,useDaylight=false,transitions=7,lastRule=null],firstDayOfWeek=1,minimalDaysInFirstWeek=1,ERA=1,YEAR=2021,MONTH=9,WEEK_OF_YEAR=42,WEEK_OF_MONTH=3,DAY_OF_MONTH=10,DAY_OF_YEAR=283,DAY_OF_WEEK=1,DAY_OF_WEEK_IN_MONTH=2,AM_PM=1,HOUR=
1,HOUR_OF_DAY=13,MINUTE=48,SECOND=15,MILLISECOND=568,ZONE_OFFSET=19800000,DST_OFFSET=0]

props obj  class name::class java.util.Properties props obj data{java.specification.version=16, sun.cpu.isalist=amd64, sun.jnu.encoding=Cp1252, java.class.path=G:\Worskpaces\Spring\NTSPBMS615\JOCProj13-FactoryMethod8eanInstantiation\target\classes;C:\Users\Nareshit\.m2
\repository\org\springframework\spring-context-support\5.3.10\spring-context-support-5.3.10.jar;C:\Users\Nareshit\.m2\repository\org\springframework\spring-beans\5.3.10\spring-beans-5.3.10.jar;C:\Users\Nareshit\.m2\repository\org\springframework\spring-context\5.3.10
\spring-context-5.3.10.jar;C:\Users\Nareshit\.m2\repository\org\springframework\spring-aop\5.3.10\spring-aop-5.3.10.jar;C:\Users\Nareshit\.m2\repository\org\springframework\spring-expression\5.3.10\spring-expression-5.3.10.jar;C:\Users\Nareshit\.m2\repository\org
\springframework\spring-core\5.3.10\spring-core-5.3.10.jar;C:\Users\Nareshit\.m2\repository\org\springframework\spring-jcl\5.3.10\spring-jcl-5.3.10.jar, java.vm.vendor=Oracle Corporation, sun.arch.data.model=64, user.variant=, java.vendor.url=https://java.oracle.com/,
user.timezone=Asia/Calcutta, java.vm.specification.version=16, os.name=Windows 10, sun.java.launcher=SUN_STANDARD, user.country=US, sun.boot.library.path=G:\Softwares\Eclipse\eclipse-jee-2021-06-R-win32-x86_64\eclipse\plugins
\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_16.0.1.v20210528-1205\jre\bin, sun.java.command=com.nt.test.FactoryMehod8eanInstantiationTest, jdk.debug=release, sun.cpu.endian=little, user.home=C:\Users\Nareshit, user.language=en,
java.specification.vendor=Oracle Corporation, java.version.date=2021-04-20, java.home=G:\Softwares\Eclipse\eclipse-jee-2021-06-R-win32-x86_64\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_16.0.1.v20210528-1205\jre, file.separator=\,
java.vm.compressedOopsMode=32-bit, line.separator=
, java.vm.specification.vendor=Oracle Corporation, java.specification.name=Java Platform API Specification, user.script=, sun.management.compiler=HotSpot 64-Bit Tiered Compilers, java.runtime.version=16.0.1+9-24, user.name=Nareshit, path.separator=;, os.version=10.0,
java.runtime.name=OpenJDK Runtime Environment, file.encoding=UTF-8, java.vm.name=OpenJDK 64-Bit Server VM, java.vendor.url.bug=https://bugreport.java.com/bugreport/, java.io.tmpdir=C:\WINDOWS\TEMP\, java.version=16.0.1, user.dir=G:\Worskpaces\Spring
\NTSPBMS615\JOCProj13-FactoryMethod8eanInstantiation, os.arch=amd64, java.vm.specification.name=Java Virtual Machine Specification, sun.os.patch.level=, java.library.path=G:\Softwares\Eclipse\eclipse-jee-2021-06-R-win32-x86_64\eclipse\plugins
\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_16.0.1.v20210528-1205\jre\bin;C:\WINDOWS\Sun\Java\bin;C:\WINDOWS\System32;C:\WINDOWS;G:/Softwares/Eclipse/eclipse-jee-2021-06-R-win32-x86_
64/eclipse//plugins/org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_16.0.1.v20210528-1205/jre/bin/server;G:/Softwares/Eclipse/eclipse-jee-2021-06-R-win32-x86_64/eclipse//plugins/org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_
16.0.1.v20210528-1205/jre/bin;C:\Program Files\Java\jdk1.8.0_31\bin;C:\Program Files\MongoDB\Server\4.4\bin;.;;G:\Softwares\Eclipse\eclipse-jee-2021-06-R-win32-x86_64\eclipse;;, java.vm.info=mixed mode, java.vendor=Oracle Corporation, java.vm.version=16.0.1+9-24,
sun.io.unicode.encoding=UnicodeLittle, java.class.version=60.0}
==========
s2 obj class name::class java.lang.String  s2 obj data ::hello123
-------------------------------------
s3 obj class name::class java.lang.String  s3 obj data ::hello
```

Spring bean scopes
==========

Oct 11 Spring Bean Scopes

IOC container not only creates the spring bean class objs.. it also can keep them in different scopes.

scopes in spring 1.x
---
singleton (default)
prototype

scopes in spring 2.x
---
singleton (default)
prototype
reqeust
session
globalSession
(portlet env..)

scopes in spring 3.x
---
singleton (default)
prototype
reqeust
session
globalSession (deprecated)

scopes in spring 4.x
---
singleton (default)
prototype
reqeust
session
websocket

scopes in spring 5.x (latest)
---
singleton (default)
prototype — can be used in both standalone, web applications
reqeust
session
application
websocket — only in web applications

=> To specify spring bean scope in xml driven cfgs use "scope" attribut of <bean> tag
=>To specify spring bean scope in annotation driven cfgs use @Scope annotation.

scope="singleton"
==========

=>The IOC container creates single object for given spring bean class with respect to spring bean id and also keeps that object in the internal cache of IOC container for reusability.

=> Uses same spring bean calss object ref across the multiple calls factory.getBean(-,-) method having same spring bean id becoz it collects Spring bean class obj ref from the internal cache of IOC container.

=>This is "default" scope if no scope is specified ..
=>This scope is no way related to singleton java class .. i.e it does not make spring bean class as singleton java class but it gives "singlton" behaviour by creting only one object for spring bean with respect to bean id and using that object ref across the muliple factory.getBean(-) method calls (having same bean id)

```
<bean id="wmg" class="com.nt.beans.WishMessageGenerator" scope="singleton">
  ....
</bean>
```

Internal cache of IOC container

wmg	WishMessageGenrator class obj ref

Code in client App
==========

```
//get Target spring bean class object
WishMessageGenerator generator1=factory.getBean("wmg",WishMessageGenerator.class);
WishMessageGenerator generator2=factory.getBean("wmg",WishMessageGenerator.class);

System.out.println("hashcodes=="+generator1.hashCode()+"  "+generator2.hashCode());  //gives same hashCode
System.out.println("generator1==generator2?"+(generator1==generator2));   //gives true
```

=> Becoz the spring bean class object ref kept in internal cache of IOC container .. we are getting "singleton" behaviour for spring bean whose scope="singleton".. But it is not going make spring bean class singleton java class.

What is the difference between signleton java class and singleton scope for spring beans?

Ans) Singleton java class means we add addtioal to normal java class to make that class allowing only one object creation in any given situation.

"singleton" scope for spring bean makes IOC container to create only one object for spring bean with repsect to bean id and to reuse that object ref across the multiple calls to factory.getBean(-) method calls having same bean id by keeping that object ref in the internal cache .

=>Singleton java ==>Restrict users to create only one object for java class
=>"Singleton"scope ===> Rrestricts IOC container to create only one object for spring bean with repsect to bean id.

Singleton java class ==> Class level restriction to have only one object (Sincere employee)
"singleton" scope spring bean ==> NoClass LEvel Restriction .. only IOC container level rstriction to create only one for given spring bean class with respect to id. (Sincere organization employee)

What happens ,if we configure same spring bean class having singleton scope for 2 times with two different bean ids?

Ans) Two objects will be created for spring bean class on 1 per each bean id and they will be in the internal cache of IOC container having bean ids as keys and bean class objs references as the values.

In applicationContext.xml

```
<bean id="wmg" class="com.nt.beans.WishMessageGenerator" >
  <property name="date" ref="dt"/>
</bean>

<bean id="wmg1" class="com.nt.beans.WishMessageGenerator" >
  <property name="date" ref="dt"/>
</bean>
```

WishMEssageGeneator obj(wmg)
WishMEssageGeneator obj(wmg1)

IOC Container 's internal cache (map)

keys	values
wmg1	WishMessageGenrator class obj ref1
wmg	WishMEssagGenator class obj ref2

note:: All spring beans cfg in spring cfg file must have unique bean ids..

What happens if cfg real singleton java as spring bean having singleton scope?
Does it continue to behave as singleton java class or not?

=>if that "singlton java class" cfg as spring bean only 1 time as having "singleton" scope then that "singleton java class " behaviour continues becoz the IOC container keeps spring bean class obj in the internal cache of IOC container and uses that object across the multiple factory.getBean(-) method calls with same bean ids.

applicationContext.xml — *real singleton java class*

```
<bean id="p1" class="com.nt.ston.Printer"/>
```

Client App

```
//get Spring bean class obj
Printer p1=factory.getBean("p1",Printer.class);
Printer p2=factory.getBean("p1",Printer.class);
System.out.println(p1.hashCode()+"  "+p2.hashCode());  //gives same hashCode
```

Problem::
=> if that java "singleton java class" is cfg for multiple times as spring beans having different bean ids with the scope singleton then "singleton java class" behaviour will be broke becoz IOC container uses reflection api to access private constructor of sigleton java class and creates multiple objects for singleton java class with respectve multiple bean ids of same spring bean class cfgs.

In applicationContext.xml

```
<bean id="p1" class="com.nt.ston.Printer"/>
<bean id="p" class="com.nt.ston.Printer"/>
```

In Client App

```
//get Spring bean class obj
Printer p1=factory.getBean("p1",Printer.class);
Printer p2=factory.getBean("p1",Printer.class);
System.out.println(p1.hashCode()+"  "+p2.hashCode());
System.out.println("====================");
//get Spring bean class obj
Printer p3=factory.getBean("p",Printer.class);
Printer p4=factory.getBean("p",Printer.class);
System.out.println(p3.hashCode()+"  "+p4.hashCode());
```

//Total two objects are created for Printer class and two objects will be created for two diffrent bean ids (singleton java class is broken)

Solution1 :: Make IOC container creating Singleton Java class obj using static factory-method bean instantiation

code

```
<bean id="p1" class="com.nt.ston.Printer" factory-method="getInstance"/>
<bean id="p" class="com.nt.ston.Printer" factory-method="getInstance"/>
```

static factory method bean instanitation is enabled

Internal cache of IOC container

p1	Printer class obj ref
p	Printer class obj ref

both are same obj references

Client App code

```
//get Spring bean class obj
Printer p1=factory.getBean("p1",Printer.class);
Printer p2=factory.getBean("p1",Printer.class);
System.out.println(p1.hashCode()+"  "+p2.hashCode());
System.out.println("====================");
//get Spring bean class obj
Printer p3=factory.getBean("p",Printer.class);
Printer p4=factory.getBean("p",Printer.class);
System.out.println(p3.hashCode()+"  "+p4.hashCode());
```

all of them give same Printer class obj ref so we can say singleton java class pattern is not broken

---

Solution2: Make Printer as perfect singleton java class (make Printer as reflection api protected Singleton Java class)

We will study this in DP class while developing perfect singleton java class by spending 10 day time)

Oct 12 Spring Bean Scopes

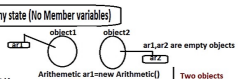
When should we java class as singlton java class or spring bean as singleton scope spring beans?

a) If class is not having any state (No Member variables)
b) If class is having olny read only state (final variables)
c) If calss is having sharable state across multiple other classes (cache implementations)

a) If class is not having any state (No Member variables)

Problem

```
public class Airthmetic{

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



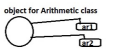

```
Arithmetic ar1=new Arithmetic()
ar1.sum(10,20);
Arithmetic ar2=new Arithmetic()
ar2.sum(20,30);
```

Two objects with no state are created

=>creating multiple objects for same java class with out state is waste of memory and cpu time , Make the class singleton java class or singlton scope spring bean to minimize the memory consumtion and cpu time

Solution1::

```
public class Arithmetic{
   private static  Arithmetic INSTANCE;

   private  Arithmetic(){   }
   public  Arithmetic getInstance(){
      if(INSTANCE==null)
         INSTANCE=new Arithmetic();
      return  INSTANCE;
   }

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

}
```




```
Arithmetic ar1=Arithmetic.getInstance();
ar1.sum(10,20);
Arithmetic ar2=Arithmetic.getInstance();
ar2.sum(20,40);
```

Only one object is created but used for two times

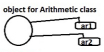
Solution2:

applicationContext.xml

```
<bean id="ar" class="pkg.Arithmetic"  scope="singleton"/>
```

"Normal class with singleton scope"

note:: To get singleton behaviour in spring ,we need not to develope java class "singleton" java class. we just need to cfg that java class as "singleton" scope spring bean as shown above.



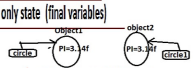
In client app

```
Arithmetic ar=factory.getBean("ar",Arithmetic.class);
ar.sum(10,20);

Arithmetic ar=factory.getBean("ar",Arithmetic.class);
ar.sum(10,20);
```

---

b) If class is having olny read only state (final variables)

Problem: Normal java class



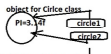

```
public class Circle{
   private float PI=3.14f;

   public  float calcArea(float radius){
      return  PI*radius*radius;
   }

}
```

```
Circle circle=new Circle();
circle.calcArea(10.0f);

Circle circle1=new Circle();
circle1.calcArea(20.0f);
```

mulitple Creating objects with fixed state is waste of memory and wast of CPU time

Solution1: Singleton java class

```
public class Circle{
  private  final float PI=3.14f;
  private static  Circle INSTANCE;

  private Circle(){
  }
  public static  Circle getInstance(){
       if(INSTANCE==null)
          INSTANCE=new Circle();
      return  INSTANCE;
    }

    public  float calcArea(float radius){
      return  PI*radius*radius;
    }
}
```



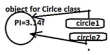

```
Circle circle1 =Circle.getInstance();
circle1.calcArea(10.0f);

Circle circle2 =Circle.getInstance();
circle2.calcArea(20.0f);
```

Here only object will be created and that will be used for multiple times

Solution2::

in applicaitonContext.xml

```
<bean id="circle" class="pkg.Circle" scope="singleton"/>
```



In client App

```
Circle circle1= factory.getBean("circle",Circle.class);
circle1.calcArea(10.0f);
Circle circle2= factory.getBean("circle",Circle.class);
circle2.calcArea(20.0f);
```

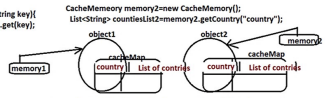
---

c) if calss is having sharable state across multiple other classes (cache implementations)

Problem::

```
public class  CacheMemory{
  private   Map<String,Object> cacheMap=null;
                                (cache)
 pubic CacheMemory(){
  cacheMap=new HashMap();
    .....
    ..... //jdbc code get Countries  from DB table
    ....  and put them in cacheMap object
 }

  public  List<String> getCountries(String key){
     return  (List<Srring>) cacheMap.get(key);
    }
  }
```


```
CacheMemeory memory1=new CacheMemory();
List<String> countiesList1=memory1.getCountry("country");

CacheMemeory memory2=new CacheMemory();
List<String> countiesList2=memory2.getCountry("country");
```



=>creating multiple objects having same sharable data is waste of memory and waste of CPU time.

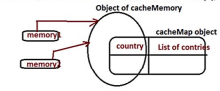
Solution1:: Develop CacheMemory class singleton java class

```
public class CacheMemory{
  private static  CacheMemory INSTANCE;

  private   Map<String,Object> cacheMap=null;
                                (cache)
private . CacheMemory(){
   cacheMap=new HashMap();
     .....
     ..... //jdbc code get Countries  from DB table
     ....  and put them in cacheMap object
 }
public static  CacheMemory  getInstance(){
  if (INSTANCE==null)
     INSTANCE=new CacheMemory();
  return INSTANCE;
 }

  public  List<String> getCountries(String key){
     return  (List<Srring>) cacheMap.get(key);
    }
  }
```

List<String> countiesList1=memory1.getCountry("cour

Client app

```
CacheMemory memory1 =CacheMemory.getInstance();
List<String> countiesList1=memory1.getCountry("country");

CacheMemory memory2 =CacheMemory.getInstance();
List<String> countiesList2=memory2.getCountry("country");
```

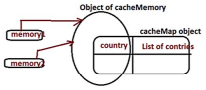


Solution2:: make Cache Memory calss as spring bean having singleton scope

in applicationContext.xml

```
<bean id="memory" class="pkg.CacheMemory" scope="singleton"/>
```

```
CacheMemory memor1=factory.getBean("memory",CacheMemory.class);
List<String> countriesList=memory1.getCountries("country");
CacheMemory memor2=factory.getBean("memory",CacheMemory.class);
List<String> countriesList=memory2.getCountries["country");
```



singleton scope spring beans in layred Apps

=> DataSource classes , DAO classes , service classes and controller class of spring layered app genrallay contains effectively read only i.e almost fixed state/read only state .. So it is recomanded to take them singleton scope spring beans.

note: if u know how to use singleton scope of spring bean effectively ..then there is no need of implementing "singleton class" Design pattern in spring projects..

====> generics ==========
https://www.youtube.com/watch?v=Wb3HSRj1PAY
========reflection api ==========
https://www.youtube.com/watch?v=elFmT_d_OYE&t=198s

===== java annotation fundamentals ==========
https://www.youtube.com/watch?v=2t22AjtbyEY&t=25s
https://www.youtube.com/watch?v=-bGWDg91vR4&t=41s

===== lombok api==========
https://www.youtube.com/watch?v=4hlnLkwgxig&list=RDCMUC4o8Fdpv3g_AjgShAeIvqpA&index=1
https://www.youtube.com/watch?v=e1_9uPJ31Lw&list=RDCMUC4o8Fdpv3g_AjgShAeIvqpA&index=2

==========JAva Bean vedios ==========
https://www.youtube.com/watch?v=Io2TcIOcF2Y
https://www.youtube.com/watch?v=GOvak6awDrI&t=161s

==========maven vedios==========
https://www.youtube.com/watch?v=2ZWK87ws2tM
https://www.youtube.com/watch?v=g8tyXC0fCd8
https://www.youtube.com/watch?v=OpIE9qRhXI8
https://www.youtube.com/watch?v=WccmV8_MeC8

next class is on oct 19 th (tuesday)

spring bean scopes in spring 5.x Oct 19 Spring Bean Scopes
==========

singleton (default)
prototype
request
session
application
websocket

scope="prototype"
==========
=> makes IOC container to create new object for spring bean for every factory.getBean(-) method call
=> This scope object of spring bean will not be kept in the internal cache of IOC container ..So there is not reusability spring bean class object..
=>Every factory.getBean(-)method call returns new object created for the spring bean.

```
<bean id="wmg"  class="com.nt.beans.WishMessageGenerator" scope="prototype"/>
```

Client app
==========

```
WishMessageGenerator generator1=factory.getBean("wmg",WishMessageGenrator.class);
WishMessageGenerator generator2=factory.getBean("wmg",WishMessageGenrator.class);
```

gives two different objs for WishMessageGenrator class

```
System.out.println(generator1.hashCode()+"  "+generator2.hashCode()); //gives two diffent hashCodes
System.out.println(generator1==generator2);  // gives false.
```

=> The scope "prototype" is inspired from the design pattern "prototype" which says create new objects based on existing objects using cloning process.. but the scope "prototype" does not use any cloning process to create new objects.. in fact it creates every object normally using reflection api.

=>When should we use prototype scope spring beans?
=> if the state of spring bean class obj is changing time to time
=>if the state of spring bean class obj is changing request to request in web application
=> If we need multple objects for same spring bean class at time with different states
then we can go protptype scopes..

usecase: In spring based web applications... we prefere taking prototype scope for VO,DTO,BO classes if think to configure these classes as spring beans in layered apps.. where as the DAO,Datasoruce, service and controller classes will be taken as singlton scope spring beans..

if 300 user are suing spring based layered web application.. then 300 customers/users details must be stored and passed at at time acrosss the multiple layerd for that we need multiple objects for VO,DTO,BO classes with different states .. For that prototype scope is good for them.

=>Generally the DAO, DataSoruce, Service, controller classes contain effectively read only/final state..So we take them as singleton scope spring beans.. where as the VO, DTO, BO classes maintain changing states time to time or request to request So go for prototye scope.

Exceptional case1::
=> if the object is created through cloning process or DeSerialization process then constructor will not be executed.
Exceptional case2::
=> if the object for the sub class of abstract class is created the abstract class constructor executes but that does not mean object created for abstract class.

What happens if configure "real singleton java class" as prototype scope spring bean?

Ans) The singleton java class behaviour will be broken becoz the IOC container internally uses reflection api to private constructor and to instanitation singleton java class as many times as required.

applicationContext.xml

```
<bean id="p1" class="com.nt.ston.Printer" scope="prototype" />
```

In client App
==========

```
//get Spring bean class obj
Printer p1=factory.getBean("p1",Printer.class);
Printer p2=factory.getBean("p1",Printer.class);
System.out.println(p1.hashCode()+"  "+p2.hashCode());  // gives two different hashCodes
System.out.println("====================");
```

Solution1:: Enable static factory method -bean instantiation on singleton java class that is cfg as spring bean

```
<bean id="p1" class="com.nt.ston.Printer" scope="prototype"  factory-method="getInstance"/>
```

In client App
==========

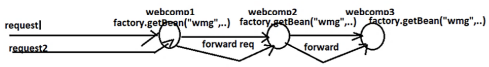
```
//get Spring bean class obj
Printer p1=factory.getBean("p1",Printer.class);
Printer p2=factory.getBean("p1",Printer.class);
System.out.println(p1.hashCode()+"  "+p2.hashCode());  // gives  same  hashCodes
System.out.println("====================");
```

Solution2: Develope Singlelton java class as perfect /strict singleton java class by providing protectionreflect api based object creation accessing private constructor
in
(will study DP classes)

Q) Though spring bean scope is taken as prototype .. How can we make spring bean beahving like "singleton" scope spring bean?
Ans) Take spring bean as "singleton Java class" and enable static factory method bean instantation.
referer solution1.

---

scope="request"
==========
=> Can be used only in web applications
=> IOC container keeps spring bean class obj in request scope i.e places spring bean class obj as request attribute having bean id as the atrtibute name and spring bean class obj as the attribute (req.setAttribute(-,-))
=> This scope specific to each request i.e across the same spring bean class object will be used through through out in different web comps that are processing a request.



if "wmg" bean id scope is "request" then all 3 times same object will be given for request1 and another same object will be given for 3 times for request2 | total two objects

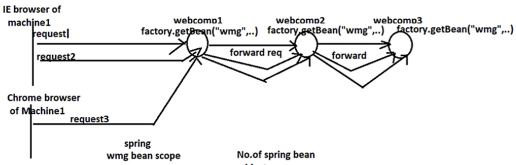
if "wmg" bean id scope is "prototype" then all 3 times 3 diffrent objects will be given for both request and request2 | total 6 objs

if "wmg" bean id scope is "singleton-" then all times same object will be for all the reuqests | total 1 object

note:: Generally The java bean class object that holds form data in spring based web application will have request scope compare to prototype scope.

scope="session"
==========
=> can be used only in web applications..
=> IOC container keeps spring bean class object in session scope by making session attribute.. i.e takes it as bean id as session attribute name and spring bean class obj as session attribute value.
=> Generaly the java bean that holds Login credentials (username, password,country and etc...) will be maintained as session scope spring bean..
=> session scope means specific to each browser s/w of a each client machine

scope="application"
==========
=> can be used only in web applications..
=>IOC container creates spring bean class object in applicationscope by making it as application attribute/ servlet context attribute taking spring bean id as the application attribute name and spring bean class obj as attribute value.
=> For the Entire web applicaton only one object of spring bean will be created
=> if spring bean class obj is carring global data of web application .. like request count and user's count days count and etc.. then go for application scope

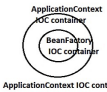


spring wmg bean scope	No.of spring bean objects
"singleton"	1
"prototype"	9
"request"	3
"session"	2
"application"	1

What is the differnccе b/w "singleton" and "application" scopes?
Ans) "singletom" scope can be used in both "standalone" , "web applications".. where as "application scope" can be used only in web application.
"singleton" scope uses the Internal cache of IOC container to manage spring bean class obj.
"application" scope uses the Servletcontext obj/application object to manage spring bean class obj.

ApplicationContext container Oct 20 ApplicationContext container

=>It is another IOC container that built on the top of BeanFactory container
=> It is having every feature of BeanFactory cotnainer and also provides addtional features... they are
=>working with Properties files and place holders
=> support for Internationalization (I18n)
=> support for pre-instantiation on singleton scope spring beans
=> Ability to stop or close or refresh IOC container
=> Support for annotation driven cfgs, 100% code driven cfgs, spring boot programming
=> Support for BeanPostProcessors and BeanFactoryPostProcessors
=>Support for Event handling and event publicatications
and etc...



ApplicationContext IOC container= BeanFactory IOC container ++

=>To Create ApplicationContext container .. we must instantiation the class that implements org.springframework.context.ApplicationContext(I) and this is sub interface of org.springframework.beans.factory.BeanFactoryBeanFactory(I).

org.springframework.beans.factory.BeanFactory(I)
↑ extends
org.springframework.beans.factory.ListableBeanFactory(I)
↑ extends
org.springframework.context.ApplicationContext(I)

Popular Implementation classes of ApplicationContext(I)

a) **FileSystemXmlApplicationContext**
=>creates ApplicationContext container in standalone env.. by taking given spring cfg file from specified path of file system
=> we can pass either relative path or absolute path of spring bean cfg file

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

b) **ClassPathXmlApplicationContext**
=>Creates the ApplicationContext container in standalone env.. by taking the given spring bean cfg file from jars and directories added to the CLASSPATH env.. variable.
(build PATH)

```
ClassPathXmlApplicationContext ctx=new ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");
```

by default src/main/java folder every maven project resides in CLASSPATH/BuildPath

c) **XmlWebApplicationContext**
=>creates ApplicationContext (AC) container in java web application by taking <servlet logical name>-servlet.xml of WEB-INF folder as the spring bean cfg file.

eg:: XmlWebApplicationContext ctx=new XmlWebApplicationContext();

if the controller servlet logical name given in <servlet-name> tag of web.xml file is "abc" then it takes abc-servlet.xml of WEB-INF folder as the spring bean cfg file name.

=>while developing spring Apps in 100% code driven mode and also in spring boot mode we do not take xml file as spring bean cfg file rather we need to take java class as "spring bean configuration calss".. In that siutations we need to use following classes to create ApplicationContext container.

d) **AnnotationConfigApplicationContext**
=>creates ApplicationContext (AC) container in standalone env.. by taking given java class as the configuration class.

```
AnnotationConfigApplicationContext ctx=new AnnotationConfigApplicationContext(AppConfig.class);
```

Recomand name.. any name can be taken
java class acting as the spring bean configuration class.

e)**AnnotationConfigWebApplicationContext**
=>creates ApplicationContext (AC) container in web application (spring MVC App) by taking given java class name as the spring bean configuration class name.

```
AnnotationConfigWebApplicationContext ctx=new AnnotationConfigWebApplicationContext(AppConfig.class);
```

Imposed Client App1

```
//StrategyDPTest.java
package com.nt.test;

import org.springframework.beans.factory.support.DefaultListableBeanFactory;
import org.springframework.beans.factory.xml.XmlBeanDefinitionReader;
import org.springframework.context.support.FileSystemXmlApplicationContext;

import com.nt.comp.Flipkart;

public class StrategyDPTest {

    public static void main(String[] args) {
        FileSystemXmlApplicationContext ctx=new FileSystemXmlApplicationContext("src/main/java/com/nt/cfgs/applIcationContext.xml");
        //get Target class obj
        Flipkart fpkt=ctx.getBean("fpkt",Flipkart.class);
        //invoke the b.method
        String resultMsg=fpkt.shopping(new String[] {"shirt","trouser","glasses"},
                                       new float[] {5679.60f,5557.55f,4545.77f} );
        System.out.println(resultMsg);

        //close container
        ctx.close();

    }//main
}//class
```

Improved Client App2

```
//StrategyDPTest.java
package com.nt.test;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.nt.comp.Flipkart;

public class StrategyDPTest1 {

    public static void main(String[] args) {
        ClassPathXmlApplicationContext ctx=new ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");
        //get Target class obj
        Flipkart fpkt=ctx.getBean("fpkt",Flipkart.class);
        //invoke the b.method
        String resultMsg=fpkt.shopping(new String[] {"shirt","trouser","glasses"},
                                       new float[] {5679.60f,5557.55f,4545.77f} );
        System.out.println(resultMsg);

        //close container
        ctx.close();

    }//main
}//class
```

By default src/main/java folder is placed buildpath/classpath of the Project

---

Improved Client App3

```
//StrategyDPTest.java
package com.nt.test;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.nt.comp.Flipkart;

public class StrategyDPTest1 {

    public static void main(String[] args) {
        //ClassPathXmlApplicationContext ctx=new ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");
        ClassPathXmlApplicationContext ctx=new ClassPathXmlApplicationContext("applicationContext.xml");
        //get Target class obj
        Flipkart fpkt=ctx.getBean("fpkt",Flipkart.class);
        //invoke the b.method
        String resultMsg=fpkt.shopping(new String[] {"shirt","trouser","glasses"},
                                       new float[] {5679.60f,5557.55f,4545.77f} );
        System.out.println(resultMsg);

        //close container
        ctx.close();

    }//main
}//class
```

For this we must make src/main/java/com/nt/cfgs folder as the source folder by adding the classpath /build path

while working with xml driven cfgs, annotations +xml driven cfgs based spring app development in stanalone env.. we need to prefer using "ClassPathXmlApplicationContext" class.

Feature1 of ApplicationContext Container

=> support for pre-instantiation on singleton scope spring beans

=> The BeanFactory container does not create spring bean class object until factory.getBean(-) method is called for any scope spring bean.. i.e it will create spring bean class obj only when need is there..that to demand.. So it is called lazy instantiation.

=> The moment ApplicationContext container created.. all the singleton scope spring beans will be instantiated and injections will be done on those beans irrespective wheather those spring bean objects will be used or not. this called eager instantiation or pre-instanitation of spring beans..

=> If the target singleton scope spring bean is having protototype scope spring bean as the dependent spring bean.. the dependent spring bean also participates in pre-instanation with out changing its scope to support injections on singleton scope spring bean.

applicationContext.xml

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

  <!-- cfg dependent classes -->
  <bean id="dtdc" class="com.nt.comp.DTDC"/>
  <bean id="dhl" class="com.nt.comp.DHL" />
  <bean id="bDart" class="com.nt.comp.BlueDart" scope="prototype"/>

  <!-- cfg Target class -->
  <bean id="fpkt" class="com.nt.comp.Flipkart" scope="singleton">
    <property name="courier" ref="bDart"/>
  </bean>
```

Here all 4 spring beans participates in pre-instantiation if the container is "AC" container. Even BlueDart spring bean participates in pre-instalation in order to make it ready to complete injection activities on target Flipkar class obj who participates pre-instantiation..

Q) How to make prortotype scope spring bean participating in pre-instantiation?
a) Make it as the dependent spring bean to the target singleton scope spring bean in "AC" IOC container.

Q) Why there no is pre-instantiation for other than singleton scope spring beans?

Ans) The other than singleton scope spring beans objects will not be placed in the internal cache of IOC container and will not be reused .. So creating objects for those spring beans through pre-instatiation process makes the created objects as the waste objects.. So no pre-instantiation enabled for other than singleton scope spring beans.

Oct 22 ApplicationContext container

How to disable Pre-instantiation on singleton scope spring beans?

Ans) By using lazy-init="true" we can disable pre-instantiation on singleton scope spring bean

```
<bean id="dtdc" class="com.nt.comp.DTDC" lazy-init="true"/>
```

Q)If lazy-init enabled singleton scope bean is dependent to singleton scope target spring bean class then what happens?

Ans) The lazy-init enabled singleton scope spring bean also participates in pre-instantation in support to the pre-instantiation of singleton scope target spring bean.

in applicationContext.xml

```
<bean id="dtdc" class="com.nt.comp.DTDC" lazy-init="true"/>

<!-- cfg Target class -->
<bean id="fpkt" class="com.nt.comp.Flipkart" scope="singleton">
  <property name="courier" ref="dtdc"/>
</bean>
```

Q) we have 10 spring beans cfg in spring bean cfg file.. how to enable pre-instantiation only 5 spring beans?

case1:: =>Take 5 spring beans as the singleton scope spring beans
=> Another 5 spring beans as the prototype scope spring beans
=> make sure that prototype spring beans are not taken as the depedent spring beans to singletion scope spring beans

case2:: =>Take 5 spring beans as the singleton scope spring beans
=> Another 5 spring beans as the singleton scope spring beans by enabling lazy-init
=> make sure that lazy init enabled spring beans are not taken as the depedent spring beans to singletion scope spring beans

What is the use of the pre-instantiation of singleton spring beans in real projects?

web application with BeanFactory container and not enabling <load-on-startup> on controller servlet

request1 to web application :: =>servlet loading , servlet instantiation + initialization
=> spring beans loading ,spring beans instanitation, injections on spring beans
=> request processing operation.

other than request1 to web application :: request processing

1st request is taking more time compare to other than 1st request to words processing and generating generating becoz <load-on-startup> on servlet is not enabled + ApplicationContextContainer is not taken performing pre-instantiation of singleton scope spring beans.

web application with ApplicationContext container and enabling <load-on-startup> on controller servlet

During the deployment of the web application or during the server startup
=> Servlet loading +servlet instantiation + servlet intialization
=> IOC Container creation(AC) , singleton scope spring beans loading , instantiation and injections takes place

note:: in web application we create IOC container in controller servlet comp.

request1 to web application :: Directly request processing

other than request1 to web application :: directly request processing

request1 and other than request1 are taking same time to proces the request.

Features2 :: Ability to work with Properties file and place holders

properties file

=>The text file that maintains the entries in the form key=value pairs is called properties file

info .properties

```
# personal info
per.name=raja
per.age=30
per.addrs=hyd
 keys      values
```

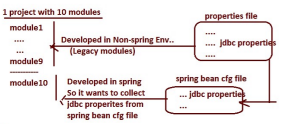
```
==================
Sunday 10:30 am
spring boot Logging
==================
========== Workshop Link =========
Meeting ID 928 0375 4982
Passcode 112233
```

in our Java Apps to keep jdbc properties we take the support of properties file .. it avoid hardcoding of jdbc properties allows us to modify jdbc properties with out touching the source code of the application.

jdbc properties
- jdbc driver class name
- jdbc url
- db username
- db password

Q) In Spring Apps we cfg DataSource injecting jdbc properties in spring bean cfg file.. there it self (xml file) we are avoiding harcoding of jdbc properties what is the need of taking seperate properties file in spring Apps.

Ans1)



problem:

if module1 to module9 code is gathering jdbc properties from properties file and 10th is gathering same properties from spring bean cfg file then any change in jdbc properties we need to modify in two places.. To get rid these problems use the the following solution as shown in the diagaram..

solution ::

make module10 which is developed in spring gathering jdbc properites from properties file through spring bena cfg file.

Ans2) **While developing spring apps using 100% code driven cfgs and in spring boot mode we can not take the xml file as spring bean cfg file.. there we need to use properties file only.**

Adding properties file support to MiniProject having jdbc properties

step1) keep u r mini project app ready

step2) add jdbc.properties file in com.nt.commons pkg of src/main/java folder having jdbc properties of certain Db s/w of ur choice

jdbc.properties

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

step3) cfg jdbc properties file in spring bean cfg file using <context:property-placeholder > tag

in applicationContext.xml ***(import context name space usisng namespaces tab)***

```
<context:property-placeholder location="com/nt/commons/jdbc.properties"/>
```

step4) place the key of properties file as place holders while injected jdbc properties to DataSource configuration

```
<!-- DataSource cfg -->
<bean id="drds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
  <property name="driverClassName" value="${jdbc.driver}"/>
  <property name="url" value="${jdbc.url}"/>
  <property name="username" value="${db.user}"/>
  <property name="password" value="${db.pwd}"/>
</bean>
```

place holders having keys of the properties whose values will be replaced with the real values collected from the properties file

step5) do changes in spring bean cfg file according to the changes done in DataSource cfg

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 http://www.springframework.org/schema/beans/spring-beans.xsd
            http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.3.xsd">

  <context:property-placeholder location="com/nt/commons/jdbc.properties"/>

  <!-- DataSource cfg -->
  <bean id="drds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName" value="${jdbc.driver}"/>
    <property name="url" value="${jdbc.url}"/>
    <property name="username" value="${db.user}"/>
    <property name="password" value="${db.pwd}"/>
  </bean>

  <!-- DAO class cfg -->
  <bean id="empOraDAO" class="com.nt.dao.EmployeeOracleDAOImpl">
    <constructor-arg name="ds" ref="drds"/>
  </bean>
  <bean id="empMySqlDAO" class="com.nt.dao.EmployeeMySQLDAOImpl">
    <constructor-arg name="ds" ref="drds"/>
  </bean>

  <!-- service class cfg -->
  <bean id="empService" class="com.nt.service.EmployeeMgmtServiceImpl">
    <constructor-arg name="dao" ref="empOraDAO"/>
    <!-- <constructor-arg name="dao" ref="empMySqlDAO"/> -->
  </bean>

  <!-- controller cfg -->
  <bean id="controller" class="com.nt.controller.MainController">
    <constructor-arg name="service" ref="empService"/>
  </bean>

</beans>
```

This tag internallay locates given properties file and also replaces the place holders of inMemoryMetaData of spring bean cfg with the real values collected from proerties file

Since the jdbc proeprties are dynamically coming from properties file at runtime , there is no need of taking multiple DataSource cfg for multiple Db s/w.. just take one DataSource cfg change jdbc proeprties time to time to the change Db s/w

step5) change BeanFactory container to ApplicationContext container in the Client App

In Client App

```
//create IOC container
        ClassPathXmlApplicationContext ctx=new ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");
        // get SpringBean class obj ref
        MainController controller=ctx.getBean("controller",MainController.class);
```

Oct 23 ApplicationContext Vs BeanFactory container

What is the difference b/w BeanFactory Container and ApplicationContext container?

BeanFactory	ApplicationContext
(a) Does not support pre-instantiation of singleton scope spring beans	(a) supports
(b) Performs only Lazy instatiation of spring beans	(b) supports both Lazy instantitation and pre/eager instantiation of spring beans as needed
(c) Does not have ablility to stop/close/ refresh IOC container	(c) allows to perform stop/close/refresh operations on the IOC container
(d) No direct support for properties file and recognizing the place holders	(d) There is direct support for properties file and recognizing the place holder ${<key>}
(e) No support for Internationalization(I18n)	(e) supports Internationalization (I18n)
(f) No support for event handling and event publication	(f) supports Event handling and event publication
(g) No direct support for annotation driven cfgs.. So can not be used 100%Code driven and spring boot mode spring apps development	(g) Supports annotation drvien cfgs ..So can be used in 100% code driven cfgs and also in spring boot programming
(h) Bit Light weight and takes less memory	(h) Bit heavy weight and needs more memory compare to BeanFactory container
(i) No Automatic registration of BeanPostProcessors and BeanFactoryProcessors	i) Supports Automatic registration of BeanPostProcessors and BeanFactoryPostProcessors
j) suitable for spring utilization done in ebmdded systems, mobile Apps	j) Suitable for spring based standalone Apps, web applications, dstriubted Apps, enterprise Apps

When should I use BeanFactory IOC Container and when should I use ApplicationContext IOC container?

Ans) if u r not using ApplicationContext IOC container in u r Project .. u should have proper reason for it .. otherwise prefer using ApplicationContext Container..

=> if spring is used in the Embedded System projects , Mobile Apps , AI Apps which are Memory Sensitive Apps where 1 or 2 kbs also matters ..then prefere using BeanFactory container that to if ur not using addtional features of "ApplicationContainer" in the spring code.

=> if spring is used to develop standalone Apps (core) , web applications(web mvc), enterprise Apps (all modules) , distributed Apps (spring rest) then prefer using ApplicationContext container.

Realtime spring Projects :: 99.99% uses ApplicationConext Container

---

<context:property-placeholder location="com/nt/commons/jdbc.properties"/> is not only given to cfg the names and locations of 1 or more properties and also makes IOC container to recognize place holders ${....} of spring bean cfg file and spring bean classes code to inject values collected from given properties files , system properties and env.. variables based on keys we placed placed holders.. while working system properties , env.. variable the keys in place holders are fixed like os.name ,path and etc..

Code in spring bean

```
public class EmployeeMgmtServiceImpl implements IEmployeeMgmtService {
    //HAS -A property
    private IEmployeeDAO dao;
    private String osName;
    private String path;

    public void setOsName(String osName) {
        System.out.println("EmployeeMgmtServiceImpl.setOsName()");
        this.osName=osName;
    }

    public void setPath(String path) {
        System.out.println("EmployeeMgmtServiceImpl.setPath()");
        this.path = path;
    }

    ...
    ...
    }
```

code in spring bean cfg file (xml file)

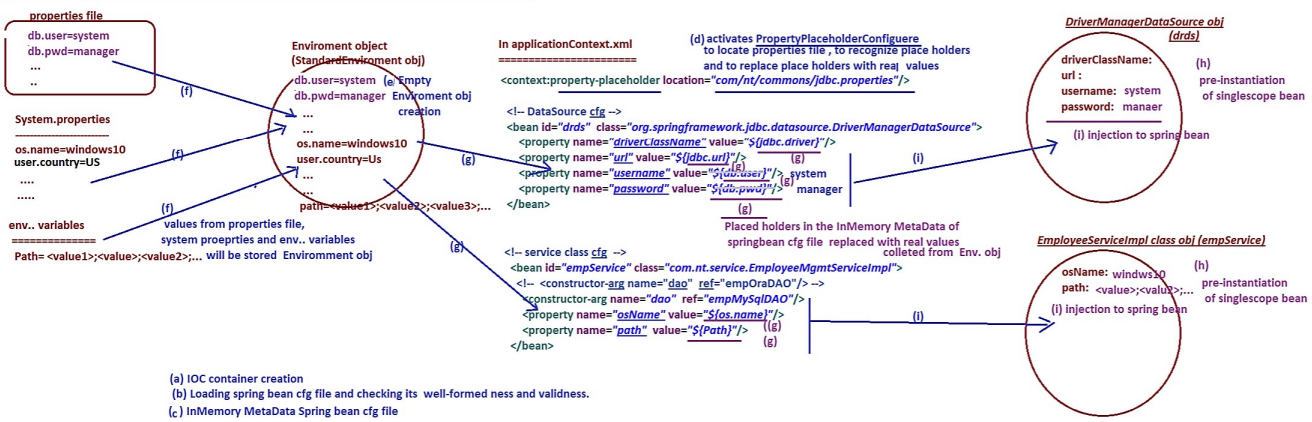
```
<!-- service class cfg -->
 <bean id="empService" class="com.nt.service.EmployeeMgmtServiceImpl">
  <!-- <constructor-arg name="dao" ref="empOraDAO"/> -->
   <constructor-arg name="dao" ref="empMySqlDAO"/>
    <property name="osName" value="${os.name}"/>
    <property name="path" value="${Path}"/>
 </bean>
```

To get all system Properties

```
System.out.println("==================");
        System.out.println(System.getProperties());
```

{java.specification.version=16, sun.cpu.isalist=amd64, sun.jnu.encoding=Cp1252, java.class.path=E:\classcontent\git\NTSPBMS615Repo\IOCProj16-RealtimeDI-propertiesfile\target\classes;C:\Users\Nareshit\.m2\repository\org\springframework\spring-context-support\5.3.10\spring-context-support-5.3.10.jar;C:\Users\Nareshit\.m2\repository\org\springframework\spring-beans\5.3.10\spring-beans-5.3.10.jar;C:\Users\Nareshit\.m2\repository\org\springframework\spring-context\5.3.10\spring-context-5.3.10.jar;C:\Users\Nareshit\.m2\repository\org\springframework\spring-aop\5.3.10\spring-aop-5.3.10.jar;C:\Users\Nareshit\.m2\repository\org\springframework\spring-expression\5.3.10\spring-expression-5.3.10.jar;C:\Users\Nareshit\.m2\repository\org\springframework\spring-core\5.3.10\spring-core-5.3.10.jar;C:\Users\Nareshit\.m2\repository\org\springframework\spring-jcl\5.3.10\spring-jcl-5.3.10.jar;C:\Users\Nareshit\.m2\repository\com\oracle\ojdbc\ojdbc8\19.3.0.0\ojdbc8-19.3.0.0.jar;C:\Users\Nareshit\.m2\repository\com\oracle\ojdbc\ucp\19.3.0.0\ucp-19.3.0.0.jar;C:\Users\Nareshit\.m2\repository\com\oracle\ojdbc\oraclepki\19.3.0.0\oraclepki-19.3.0.0.jar;C:\Users\Nareshit\.m2\repository\com\oracle\ojdbc\osdt_cert\19.3.0.0\osdt_cert-19.3.0.0.jar;C:\Users\Nareshit\.m2\repository\com\oracle\ojdbc\osdt_core\19.3.0.0\osdt_core-19.3.0.0.jar;C:\Users\Nareshit\.m2\repository\com\oracle\ojdbc\simplefan\19.3.0.0\simplefan-19.3.0.0.jar;C:\Users\Nareshit\.m2\repository\com\oracle\ojdbc\ons\19.3.0.0\ons-19.3.0.0.jar;C:\Users\Nareshit\.m2\repository\org\springframework\spring-jdbc\5.3.10\spring-jdbc-5.3.10.jar;C:\Users\Nareshit\.m2\repository\org\springframework\spring-tx\5.3.10\spring-tx-5.3.10.jar;C:\Users\Nareshit\.m2\repository\mysql\mysql-connector-java\8.0.26\mysql-connector-java-8.0.26.jar;C:\Users\Nareshit\.m2\repository\com\google\protobuf\protobuf-java\3.11.4\protobuf-java-3.11.4.jar, java.vm.vendor=Oracle Corporation, sun.arch.data.model=64, user.variant=, java.vendor.url=https://java.oracle.com/, user.timezone=Asia/Calcutta, java.vm.specification.version=16, os.name=Windows 10, sun.java.launcher=SUN_STANDARD, user.country=US, sun.boot.library.path=G:\Softwares\Eclipse\eclipse-jee-2021-06-R-win32-x86_64\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_16.0.1.v20210528-1205\jre\bin, sun.java.command=com.nt.test.RealtimeDITest, jdk.debug=release, sun.cpu.endian=little, user.home=C:\Users\Nareshit, user.language=en, java.specification.vendor=Oracle Corporation, java.version.date=2021-04-20, java.home=G:\Softwares\Eclipse\eclipse-jee-2021-06-R-win32-x86_64\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_16.0.1.v20210528-1205\jre, file.separator=\, java.vm.compressedOopsMode=32-bit, line.separator=
, java.vm.specification.vendor=Oracle Corporation, java.specification.name=Java Platform API Specification, user.script=, sun.management.compiler=HotSpot 64-Bit Tiered Compilers, java.runtime.version=16.0.1+9-24, user.name=Nareshit, path.separator=;, os.version=10.0, java.runtime.name=OpenJDK Runtime Environment, file.encoding=UTF-8, java.vm.name=OpenJDK 64-Bit Server VM, java.vendor.url.bug=https://bugreport.java.com/bugreport/, java.io.tmpdir=C:\WINDOWS\TEMP\, java.version=16.0.1, user.dir=E:\classcontent\git\NTSPBMS615Repo\IOCProj16-RealtimeDI-propertiesfile, os.arch=amd64, java.vm.specification.name=Java Virtual Machine Specification, sun.os.patch.level=, java.library.path=G:\Softwares\Eclipse\eclipse-jee-2021-06-R-win32-x86_64\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_16.0.1.v20210528-1205\jre\bin;C:\WINDOWS\Sun\Java\bin;C:\WINDOWS\system32;C:\WINDOWS;G:/Softwares/Eclipse/eclipse-jee-2021-06-R-win32-x86_64/eclipse//plugins/org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_16.0.1.v20210528-1205/jre/bin/server;G:/Softwares/Eclipse/eclipse-jee-2021-06-R-win32-x86_64/eclipse//plugins/org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_16.0.1.v20210528-1205/jre/bin;C:\Program Files\Java\jdk1.8.0_31\bin;C:\Program Files\MongoDB\Server\4.4\bin;.;G:\Softwares\Eclipse\eclipse-jee-2021-06-R-win32-x86_64\eclipse;;., java.vm.info=mixed mode, java.vendor=Oracle Corporation, java.vm.version=16.0.1+9-24, sun.io.unicode.encoding=UnicodeLittle, java.class.version=60.0}

How system properties , env variables values and properties file(s) values are Injected spring bean through place holders



(a) IOC container creation
(b) Loading spring bean cfg file and checking its well-formed ness and validness.
(c) InMemory MetaData Spring bean cfg file

The <context:property-placeholder> tag gets values to Enviroment obj from different places in the following order

a) from the specified properites files
b) from System properties
c) from env.. variables.

Oct 25 Spring Core Annotations

Annotation driven spring programming
==========

Different approaches of developing spring Apps
- a) Using xml driven cfgs
- b) Using Annotation driven cfgs (good in old maintainance projects, small scale spring projects)
- c) Using 100% code driven cfgs
- d) Using spring boot (good in latest projects , medium scale, large scale projects and also in Micro services projects)

Different types of annotations
==========

a) Annotations guiding the compiler
==========

=>These annotations will not be recorded to .class file .. they just give guidence to compiler to verify the code.

eg: @Override , @SupressWarnings

b) Annotation for API documentation
==========

=>These Annotations are given to geneerate api documention while working "javadoc" tool and documentation comments (/** ..... */)

@author , @since , @see , @param, @returns and etc..

c) Annotations for Code marking /code MetaData (For code about code activities)
==========

=>These annotations marks java code giving special indentity or behaviour .. Very use in code configurations required for Containers or frameworks

@WebServlet --> marks/configures the java class as Servlet comp
@WebFilter —>marks/configures the java class as Servlet filter comp
@Entity —>marks/configures the java class as an Entity class
@Component —>marks/configures the java class as spring bean class and etc..

d) Annotations for code marking and for InMemory Proxy code Generation
==========

=> These annotation like are like (c) type annotations and also contains the addtional behaviour of generating InMemory Proxy class code to provide addtional logics..

eg: @WebService , @EJB , @Lookup ande tc..

@WebService placed on java class marks java class as SOAP web service comp but real logic that in required to make the code as web service comp will be generated as the InMemory proxy class code as the sub class for current class.

```
@WebService
public class TestService{

  public String bm1(){
    ,....
    ...
  }
}
```

```
//Dynamically Generated InMemory Proxy class

public Test123567FFGService extends TestService{

   .....
   ..... //contains code to make the logics
   ...    webservice comp logics.
\
```

=>Annotations support in spring added from spring 2.0 incremently
i.e each version they have added more annotations to use

List of Annotations ========== related spring core module

Spring 2.0

@Configuration
@Required (deprecated from spring 5.x)
@Repository
@Order

Spring 2.5

@Autowired
@Qualifier
@Scope

Stereotype annotations:

@Component
@Service
@Controller
@Repository

=>these annotation are only for core module programming imp annotations and no way related spring boot programming exclusive annotations and other spring modules.

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..

JAva config annotations (Given by JSE,JEE modules So Every technology ,framework can provide functionality as per their need. These annotations fully support non-invasive programming)
==========

@PreDestroy
@Post Construct
@Resource
@Inject
@Named and etc..

//The funcaitonalties for these annotation will be given by underlying container/server/framework /technology. i.e if these annotation used in servlet programming the servletContainer provides the functionality.. and if these annotations are used in Spring programming spring IOC container provides the funcationality

---

BeanPostProcessor
==========

=> It is given to perform addtional and common activities on multiple spring beans once spring beans instantiation and injected (spring beans are ready to use).

Student (spring bean1)
|-->sno,sname,course, doj
Emmpoyee (spring bean2)
|-->eno,ename,desg,doj
Customer (spring bean3)
|-->cid,cname, billAamt,doj

=>Instead of intializing "doj" in all 3 spring beans with system date and seperately inside spring bean classes It recomanded to take one common BeanPostProcessor having same logic becoz BeanPostProcessor executes for all the bean having common and additional post processing logic.

=>For every Spring supplied/supported annotations there is Oone BeanPostProcessor provoding Annotation related functionalify
For @Required that is org.springframework.beans.factory.annotation.RequiredAnnotationBeanPostProcessor
For @Autowired that is org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor and etc..

=>All BeanPostProcessors are the classes implementing BeanPostProcessor(I) directly or indirectly.
=>We can develop CustomBeanPostProcessor for our spring beans related common post processing logics.. but generally we work with lots pre-defined BeanPostProcessors to get Annotations funcationalaties.

=> In ApplicationContext Container once the BeanPostProcessor is cfg as spring bean ..It will be recognized by IOC container automatically and keep the BeanPostProcessor ready before any pre-instanitation takes place..
Once the Injections on spring beans are over then BeanPostProcessors comes into picture to complete their job. like assinging doj with system date and time .. providing funcationality for annotations and etc..

@Autowired
========== also

Dependency injection is called as Bean wiring

Bean wiring of two types
==========

a) Explicit wiring
==>Here we need to <property> or <constructor-arg> tags explcitly for dependency injection cfgs (So far we did this)

b) Auto wiring /Implicit wiring
==>Here we do not use <property> or <costructor-arg> tags .. we make IOC container to detect the dependent spring beans dyanmically and to inject to target spring bean class objects/propeties based on the type or name of properties

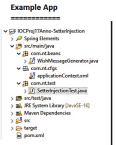
To enable autowiring in xml cfgs use "autowire" attribute <bean> tag .. In annotation driven cfgs use @Autowired annotation.

@Autowired can perform following injections by detecting dependent automatically
a) setter Injection b) consturctor injection c) Filed Injection d) Orbitary method injection (method with any name)

(@Autowired supports 4 types of injections where xml cfgs related <property>, <constructor-arg> tags support only 2 types of injections)

=>@Autowired annotation can be applied at field level, method level, constructor level .

Example App
==========




```
//WishMessageGenerator.java
package com.nt.beans;

import java.util.Date;

import org.springframework.beans.factory.annotation.Autowired;


public class WishMessageGenerator {
      //HAS-A property (supporting composition)
      @Autowired    // Field level injection
      private Date date;

      public WishMessageGenerator() {
            System.out.println("WishMessageGenerator:: 0-param constructor");
      }

      //B.method
      public String generateMessage(String user) {
            System.out.println("WishMessageGenerator.generateMessage():: date="+date);
            //get current hour of the day
            int hour=date.getHours(); // 24 hours format (0 to 23)
            //generate 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 Night::"+user;
      }//method
}//class
```

spring bean cfg file (applicationContext.xml)
==========

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

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

 <!-- target class cfg -->
 <bean id="wmg" class="com.nt.beans.WishMessageGenerator"/>


 <!-- AutowiredBeanPostProcessor cfg -->
 <bean class="org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor"/>

</beans>
```

Cleint App
==========

```
package com.nt.test;

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

import com.nt.beans.WishMessageGenerator;

public class SetterInjectionTest {

      public static void main(String[] args) {
            //create IOC container
            ClassPathXmlApplicationContext ctx=new ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");
            //get Bean spring bean class obj
            WishMessageGenerator generator=ctx.getBean("wmg",WishMessageGenerator.class);
            //invoke the b.method
            String result=generator.generateMessage("raja");
            System.out.println("Wish Message is ::"+result);

            //close the IOC container
            ctx.close();

      }//main
}//class
```

Oct 26 Spring Core Annotations-@Autowired

Flow of execution for annotation driven cfgs based spring App

Example App


```
//WishMessageGenerator.java
package com.nt.beans;

import java.util.Date;

import org.springframework.beans.factory.annotation.Autowired;


public class WishMessageGenerator {
      //HAS-A property (supporting composition)
      @Autowired   // Field level injection
      private Date date;

      public WishMessageGenerator() {
            System.out.println("WishMessageGenerator:: 0-param constructor");
      }

      //B.method                (17)
      public String generateMessage(String user) {
            System.out.println("WishMessageGenerator.generateMessage():: date="+date);
            //get current hour of the day
            int hour=date.getHours(); // 24 hours format (0 to 23)
            //generate wish messsage
            if(hour<12)
                  return "Good Morning::"+user;
            else if(hour<16)
                  return "Good AfterNoon::"+user;
            else if(hour<20)
                  return "Good Evening::"+user;   (18)
            else
                  return "Good Night::"+user;
      }//method
}//class
```

(5) finds @Autowired here takes the property type java.util.Date search spring bean for the same type

spring bean cfg file (applicationContext.xml)

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

<!-- Dependent class cfg -->
  <bean id="dt" class="java.util.Date" />        (6) pre-insatnitation of singleton scope bean

<!-- target class cfg -->
  <bean id="wmg" class="com.nt.beans.WishMessageGenerator"/>   (7) pre-instanitation of singleton scope bean

<!-- AutowiredBeanPostProcessor cfg -->        (8) This BeanProcessor searches for @Autwired annotation in all spring bean using reflection api
  <bean class="org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor"/>
                                                (5) keeps BeanPostProcessor class object ready
</beans>
```

Cleint App

```
package com.nt.test;

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

import com.nt.beans.WishMessageGenerator;
                                                (1) Run the Application
public class SetterInjectionTest {
                                                (2)
      public static void main(String[] args) {
            //create IOC container              (3) IOC container creation
            ClassPathXmlApplicationContext ctx=new ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");
(15)        //get Bean spring bean class obj    (13)
            WishMessageGenerator generator=ctx.getBean("wmg",WishMessageGenerator.class);
            //invoke the b.method               (16)
(19)        String result=generator.generateMessage("raja");
            System.out.println("Wish Message is ::"+result);
                                                (20)
            //close the IOC container
            ctx.close();  (21)    Closes the container by destroying all the spring bean objs placed in Internal cache of IOC container
      }//main
}//class   (22) Destroys the IOC container itself
```

(10) finds here

(9) java.util.Date class obj (dt) system date and time...

(11) injection

WishMEssageGenerator class obj(wmg) (#7) date::

Internal cache of IOC container

dt	java.util.Date class obj ref
wmg (147)	WishMesssageGenerator class obj ref

(12) (keeps spring bean class obj refs in the internal cache of IOC container)

(4) InMemory Meta-Data of spring bean cfg file

Limitations with Autowiring

finds

a) There is possiblity of getting ambiguity problem if IOC container multiple dependent beans to inject to target spring bean class obj (this can be solved using different techniques)

b) Kills the readability of code little bit

c) The BeanPostProcessors starts searching in all spring beans for their respective Annotation avalability to provide functionality ,this may effect performance little bit.

d) Autowiring is possible only on Object type Bean properties.

```
public class WishMessageGenerator {
      //HAS-A property (supporting composition)
      @Autowired(required = true)
      private Date date;
```

(recomanded)

```
@Autowired(required=true)      (default value is true)
```

=>The dependent spring bean cfg must availalble to complete the Autowing process otherwise exception will be raised.

```
Exception in thread "main" org.springframework.beans.factory.UnsatisfiedDependencyException: Error creating bean with name
'wmg': Unsatisfied dependency expressed through field 'date'; nested exception is
org.springframework.beans.factory.NoSuchBeanDefinitionException: No qualifying bean of type 'java.util.Date' available:
expected at least 1 bean which qualifies as autowire candidate. Dependency annotations:
{@org.springframework.beans.factory.annotation.Autowired(required=true)}
```

```
public class WishMessageGenerator {
      //HAS-A property (supporting composition)
      @Autowired(required =false)
      private Date date;
```

```
@Autowired(required=false)
```

=>Though Dependent spring bean cfg is not avaiable.. there is no problem.. if simply ignores injection proess with out any throwing any exception.. i,e target class property contains null value.

Ambiguity Problem with Autowiring

=>sometimes the IOC container finds multiple matching dependent spring beans having same type to inject to Taget class object bean property .. that situation is called Ambiguity Problem

sample code

```
//target class
public class WishMessageGenerator {
      //HAS-A property (supporting composition)
      @Autowired
      private Date date;

      .....
      ....
}
```

gets ambiguity with two matching dependent spring beans

```
org.springframework.beans.factory.
NoUniqueBeanDefinitionException: No qualifying bean of type
'java.util.Date' available: expected single matching bean
but found 2: dt,dt1
```

applicationContext.xml

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

<!-- Dependent class cfg -->
  <bean id="dt" class="java.util.Date" />
  <bean id="dt1" class="java.util.Date">
    <property name="year" value="110"/>
  </bean>

<!-- target class cfg -->
<bean id="wmg" class="com.nt.beans.WishMessageGenerator"/>

<!-- AutowiredBeanPostProcessor cfg -->
<bean class="org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor"/>

</beans>
```

Solutions for Ambiguity Problem

1) By making one of multiple dependent spring beans as the primary spring bean using primary="true" in <bean> tag(in xml cfgs) or @Primary (in annotation cfgs)

```
public class WishMessageGenerator {
      //HAS-A property (supporting composition)
      @Autowired
      private Date date;

      ....
      ....
}
```

applicationContext.xml

```
<!-- Dependent class cfg -->
  <bean id="dt" class="java.util.Date" primary="true" />
  <bean id="dt1" class="java.util.Date" >
    <property name="year" value="110"/>
  </bean>

<!-- target class cfg -->
<bean id="wmg" class="com.nt.beans.WishMessageGenerator"/>

<!-- AutowiredBeanPostProcessor cfg -->
<bean class="org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor"/>

</beans>
```

2) By matching Taget spring class property name with one of the matching dependent spring beans id

```
//target spring bean class
public class WishMessageGenerator {
      //HAS-A property (supporting composition)
      @Autowired
      private Date date;

      ....
      ....
      }
```

matching

applicationContext.xml

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

<!-- Dependent class cfg -->
  <bean id="dt" class="java.util.Date" />
  <bean id="date" class="java.util.Date" >
    <property name="year" value="110"/>
  </bean>

<!-- target class cfg -->
<bean id="wmg" class="com.nt.beans.WishMessageGenerator"/>

<!-- AutowiredBeanPostProcessor cfg -->
<bean class="org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor"/>

</beans>
```

3) By Using @Qualifier annotation specifying the one of the multiple matching beans ids. (Recomanded)

```
public class WishMessageGenerator {
      //HAS-A property (supporting composition)
      @Autowired
      @Qualifier("dt1")
      private Date date;

      ....
      ...
}
```

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 http://www.springframework.org/schema/beans/spring-beans.xsd
            http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.3.xsd">

<!-- Dependent class cfg -->
  <bean id="dt" class="java.util.Date"/>
  <bean id="dt1" class="java.util.Date" >
    <property name="year" value="110"/>
  </bean>

<!-- target class cfg -->
<bean id="wmg" class="com.nt.beans.WishMessageGenerator"/>
```

aadd context name space to spring bean cfg using namespace tab.

```
<!-- AutowiredBeanPostProcessor cfg -->
<context:annotation-config/>
</beans>
```

Single tag that detects and activates multiple annotations related BeanPostProcessor automtically during IOC container startup ...so need of configuring each annotation related BPP class(BeanPostProcessor class)

The above activates the following and thier supporting annotation's BPP classes

@Required , @Autowired , @PostConstruct , @PreDestroy , @EJB , @Resource , @Inject and etc..

if we all use all the above 3 techniques at a time on same spring bean property of target spring bean where each technique pointing seperate dependent then what will be finally injected depedent spring bean class obj?

Ans) @Qualifer

```
public class WishMessageGenerator {
      //HAS-A property (supporting composition)
      @Autowired
      @Qualifier("dt")
      private Date date;
        ....
        ...
}
```

dt bean-id based Date class obj will be injected.

```
<?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
http://www.springframework.org/schema/beans/spring-beans.xsd
            http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.3.xsd">

<!-- Dependent class cfg -->
<bean id="date" class="java.util.Date" >
    <property name="year" value="115"/>
  </bean>
  <bean id="dt" class="java.util.Date" />
  <bean id="dt1" class="java.util.Date"  primary="true">
    <property name="year" value="110"/>
  </bean>
```

=>By default IOC container performs "ByType" mode of Auto wiring (detecting dependent based on the type of bean property)
=>when enable primary="true" then also By type mode autowiring takes place

=>While working property name matching with dependent bean id and @Qualifier based Autowring it will go for "ByName" mode of Autowiring.

```
<!-- target class cfg -->
<bean id="wmg" class="com.nt.beans.WishMessageGenerator"/>

<!-- AutowiredBeanPostProcessor cfg -->
<context:annotation-config/>
</beans>
```

Oct 27 Spring Core Annotations-@Autowired

=>If we apply @Autowired on the top parameterized constructor then it performs constructor mode autowiring..

```
public class WishMessageGenerator {
    private Date date;

    @Autowired
    public WishMessageGenerator(@Qualifier("dt1")Date date) {
        this.date=date;
        System.out.println("\WishMessageGenerator:: 1-param constructor ::date::"+date);
    }
   ....
   ....
}
```

In this process if we get any ambiguity problem, we can solve that problem by using same old 3 solutions (primary="true" or match property name with bean id or @Qualifier(-)). Since @Qualifier(-) can not be not applied at consturctor level it must be applied at constructor parameter level.

=>At Max only one no.of parmaeterized constructor can have @Autowired annotation otherwise exception will be raised.

```
public class WishMessageGenerator {

    private Date date;

    @Autowired
    public WishMessageGenerator(@Qualifier("dt1")Date date) {
        this.date=date;
        System.out.println("\WishMessageGenerator:: 1-param constructor ::date::"+date);
    }

    @Autowired
    public WishMessageGenerator(@Qualifier("dt1")Date date,String value) {
        this.date=date;
        System.out.println("\WishMessageGenerator:: 2-param constructor ::date::"+date);
    }
    ...
    ....
    ....
    }
```

raises the exception ::

Exception in thread "main" org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'wmg': invalid autowire-marked constructor: public com.nt.beans.WishMessageGenerator(java.util.Date,java.lang.String). Found constructor with 'required' Autowired annotation already: public com.nt.beans.WishMessageGenerator(java.util.Date)

=> If @Autowired Annotation is applied on the top of setter method then it performs setter injection mode Auto wiring. Here also solve the ambiguity problems by using 1 of the 3 solutions.

```
public class WishMessageGenerator {
    private Date date;

    @Autowired
    @Qualifier("dt1")
    public void setDate(Date date) {
        System.out.println("\WishMessageGenerator.setDate(-)");
        this.date=date;
    }
    ....
    ....
}
```

=>If @Autowired is applied on single param arbitary method then it performs arbitary method mode Autowirine .. Here also we can solve ambiguity Problems usine same old 3 solutions..

```
// In target class
==============
public class WishMessageGenerator {
    private Date date;

    @Autowired
    @Qualifier("dt")
    public void assign(Date date) { // This method signature must match with setter method signature
        System.out.println("\WishMessageGenerator.assign()");
        this.date=date;
    }
    ....
    .....
    ....
    }
```

Field

note:: Industry prefers using Level Autowiring alot becoz minimum code to write.

If we enable all 4 four modes autowiring on the same property with different dependent objs , can u tell me what is the final dependent object that will be injected?

Ans) Since orbitary method excutes at the end of other onjections related execution .. we can say orbitary method injected value through autowiring will be taken as final value.

Order of autowiring mode execution
==========
a) Constructor Mode autowiring
b) Field mode autowiring
c) setter injection mode autowiring
d) orbitary method mode autowiring

```
//WishMessageGenerator.java
package com.nt.beans;

import java.util.Date;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;


public class WishMessageGenerator {
    //HAS-A property (supporting composition)
    @Autowired
    @Qualifier("dt")
    private Date date;          (b)

    @Autowired
    @Qualifier("dt3")
    public void assign(Date date) { // This method signature must match with setter method signature
(d)     System.out.println("\WishMessageGenerator.assign()");
        this.date=date;
    }

    @Autowired
    @Qualifier("dt2")
    public void setDate(Date date) {
        System.out.println("\WishMessageGenerator.setDate(-)");
        this.date=date;       (c)
    }

    @Autowired
    public WishMessageGenerator(@Qualifier("dt1")Date date) {
(a)     this.date=date;
        System.out.println("\WishMessageGenerator:: 1-param constructor ::date::"+date);
    }




    //B.method
    public String generateMessage(String user) {
        System.out.println("\WishMessageGenerator.generateMessage():: date="+date);
        //get current hour of the day
        int hourrdate.getHours(); // 24 hours format (0 to 23)
        //generate 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 Night::"+user;
    }//method
}//class
```

Default Bean ids
==========
=> If u do not provide bean id for spring bean cfg the IOC container takes default bean id internally..

In case of xml cfgs the default bean id is :: <fully qualified class name>
eg:: <bean class="com.nt.beans.WishMessageGenerator">
default bean id :: com.nt.beans.WishMessageGenerator (or) com.nt.beans.WishMessageGenerator#0

If we cfg same class for multiple times as multiple spring beans with out bean ids then the default bean ids are <fully qualified class name>#<n> (<n> series is 0,1,2,3,4.....)

applicationContext.xml

```
<bean class="com.nt.beans.WishMessageGenerator">   default bean id is :: com.nt.beans.WishMessageGenerator#0
<bean class="com.nt.beans.WishMessageGenerator">   default bean id is :: com.nt.beans.WishMessageGenerator#1

<bean class="com.nt.beans.WishMessageGenerator">   default bean id is :: com.nt.beans.WishMessageGenerator#2
```

note:: In applicationContext container we get info about spring beans using

```
System.out.println("spring beans count"+ctx.getBeanDefinitionCount());
System.out.println("bean ids :"+Arrays.toString(ctx.getBeanDefinitionNames()));
```

Stereotype annotations
==========

=>These multiple annotations that are having similar behaviour, So these called stereotype annotations
=>These annotations are given to cfg java classes as spring beans and to make IOC container to create objects for spring bean classes using given bean ids as the object names or default bean ids as the object names.

@Component :: Configures java class as spring bean
@Service : Configures java class as spring bean cum service class
(supports TxMgmt, ordering ,filtering and etc..)
@Repository:: configures java class as spring bean cum Repository or DAO class
(Supports Persistence operations)
@Controller :: configures java class as spring bean cum WebController class
(Captable of talking and processing
and etc.... http requests)

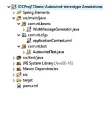
note:: @Service , @Reppository , @Controller are the extentions annotations of @Component

note:: To make IOC container scanning and detecting classes with stereo type annotations by searching in different packages and to make them as spring beans we need to specify the packages to search using <context:component-scan base-package="....."> tag.

This tag automatically brings the effects of <context:annotation-config> tag.

Thumb rule while working annotation driven cfgs based spring App
==========

=>Cfg pre-defined classes as spring beans using <bean> tags
=>Cfg user-defined classes as spring beans using stereo type annotations and link them with spring bean cfg file using <context:component-scan> by specifying their packages.



```
//Target class
==========
package com.nt.beans;

import java.util.Date;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Component;


@Component("wmg")
public class WishMessageGenerator {
    //HAS-A property (supporting composition)
    @Autowired
    private Date date;

  public WishMessageGenerator() {
      System.out.println("\WishMessageGenerator:: 0-param constructor");
  }

    //B.method
    public String generateMessage(String user) {
        System.out.println("\WishMessageGenerator.generateMessage():: date="+date);
        //get current hour of the day
        int hour=date.getHours(); // 24 hours format (0 to 23)
        //generate 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 Night::"+user;
    }//method
}//class
```

```
//spring bean cfg file
==========
<?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
http://www.springframework.org/schema/beans/spring-beans.xsd
                           http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.3.xsd">

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

    <!-- enable Component scanning -->
    <context:component-scan base-package="com.nt.beans"/>
</beans>
```

Client App
==========

```
package com.nt.test;

import java.util.Arrays;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.nt.beans.WishMessageGenerator;

public class AutowiredTest {

    public static void main(String[] args) {
        //create IOC container
        ClassPathXmlApplicationContext ctx=new ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");
                //get Bean spring bean class obj
                WishMessageGenerator generator=ctx.getBean("wmg",WishMessageGenerator.class);


                //invoke the b.method
                String result=generator.generateMessage("raja");
                System.out.println("Wish Message is ::"+result);


        //close the IOC container
        ctx.close();

    }//main
}//class
```

Oct 29 Spring Core Annotations-@Streo type Annotations

```
//Target class
==========
package com.nt.beans;

import java.util.Date;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Component;


@Component("wmg")  (8)   (finds only class with streo type
public class WishMessageGenerator {                annotation having singleton
    //HAS-A property (supporting composition)       scope)
    @Autowired     (11)
    private Date date;

    public WishMessageGenerator() {
        System.out.println("WishMessageGenerator:: 0-param constructor");
    }

    //B.method
    public  String generateMessage(String user) {
        System.out.println("WishMessageGenerator.generateMessage():: date="+date);
        //get current hour of the day
        int hourdate.getHours(); // 24 hours format (0 to 23)
        //generate 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 Night::"+user;
    }//method
}//class
```

9) pre-instantiation of singleton scope WishMessageGenerator

(12) Searches for spring bean of class name "java.util.Date" not finds in basepackage "com.nt.beans" finds spring in spring bean cfg file

```
//spring bean cfg file
=====================
<?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
http://www.springframework.org/schema/beans/spring-beans.xsd
                http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-
context-4.3.xsd">

  <!-- dependent class cfg  -->
   <bean  id="dt"  class="java.util.Date" />   (13) instantiation

        <!-- enable Component scanning -->
  </beans>   <context:component-scan base-package="com.nt.beans" />   (6) scanning of given packages and their sub packages

(5) --> From there takes <context-annotatin-config> tag
    instantiates  BeanPostProcessor classes of  @Required,
    @Autowired and etc... annotations (4 to 5 annotations)
```

WishMessageGenerator class obj (wmg) (9) Date date;

(14) Injection takes place bcoz of @Autowired

java.util.Date class obj (dt) (813) system date

10) Activation of @Autowired annotion BeanPostProcessor and uses reflection to search that annotation in spring bean class

Internal Cache of IOC container (14)

wmg	WishMessageGenerator class obj ref
dt	java.util.Date class obj ref

```
Client App
==========
package com.nt.test;

import java.util.Arrays;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.nt.beans.WishMessageGenerator;

public class AutowiredTest {                    (1) Run the Application

    public static void main(String[] args) {   (2)
        //create IOC container                  (3) IOC container creation
        ClassPathXmlApplicationContext ctx=new ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");
                                                //get Bean spring bean class obj                    toMeteData of  (4)
        WishMessageGenerator generator=ctx.getBean("wmg",WishMessageGenerator.class);   spring bean cfg file


        //invoke the b.method
        String result=generator.generateMessage("raja");
        System.out.println("Wish Message is ::"+result);


        //close the IOC container
        ctx.close();

    }//main
}//class
```

specify
@Scope(.) can be used to spring bean scope (default is singleton)

```
@Component("wmg")
@Scope("prototype")
public class WishMessageGenerator{
  ....
  ...
}
```

@Lazy --> is given to disable pre-instantiation on specific singleton scope spring bean

```
@Component("wmg")
@Scope("singleton")
@Lazy(true)
public class WishMessageGenerator {
    //HAS-A property (supporting composition)
    @Autowired
    private Date date;
  ....
  ...
}
```

=>Annotation driven cfgs the default bean id for the spring bean that is cfg streotype annotations is just class name taking first letter in lower case

```
@Component
public class WishMessageGenerator {
    //HAS-A property (supporting composition)
    @Autowired                          //default bean id is "wishMessageGenerator",
    private Date date;
  ....
  ...
}
```

code in Client App

```
//get Bean spring bean class obj
WishMessageGenerator generator=
    ctx.getBean("wishMessageGenerator",WishMessageGenerator.class);
```

=>In xml driven cfgs we can take same class as two spring beans with two different bean ids but same is not possible while working with streo type annotations based spring beans cfgs.

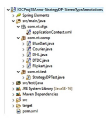
In xml driven cfgs

```
<bean id="wmg1" class="com.nt.beans.WishMessageGenerator">
  ...
</bean>

<bean id="wmg2" class="com.nt.beans.WishMessageGenerator">
  ...
</bean>
```

annotation driven cfgs

```
com.nt.beans
   |---->WishMessageGenerator.java

    @Component
    public class WishMessageGenerator {
        //HAS-A property (supporting composition)
        @Autowired
        private Date date;
      ....
      ...
    }
```

While working user-defined spring bean classes as the dependent classes that are configured using stereo type annotation, if any ambiguity Problem is raised then we need to use @Primary Annotation (or) @Qualifier or by matching one Dependent class bean id with target class property name



```
//DTDC.java (Dependent class1)
package com.nt.comp;

import org.springframework.context.annotation.Primary;
import org.springframework.stereotype.Component;

@Component("dtdc")
//@Component("courier")   //Soluton 3 for  Ambiguity Problem
//@Primary  //Solution1 for  Ambiguity Problem
public final class DTDC implements Courier {

    public DTDC() {
        System.out.println("DTDC:: 0-param consturctor");
    }

    @Override
    public String deliver(int oid) {
        return oid+" order id order  is delivered by DTDC";
    }

}
```

```
//DHL.java (Dependent class1)
package com.nt.comp;

import org.springframework.stereotype.Component;

@Component("dhl")
public final class DHL implements Courier {

    public DHL() {
        System.out.println("DHL:: 0-param consturctor");
    }

    @Override
    public String deliver(int oid) {
        return oid+" order id order is delivered by DHL";
    }

}
```

```
//DTDC.java (Dependent class1)
package com.nt.comp;

import org.springframework.context.annotation.Primary;
import org.springframework.stereotype.Component;

@Component("dtdc")
//@Component("courier")
//@Primary
public final class DTDC implements Courier {

    public DTDC() {
        System.out.println("DTDC:: 0-param consturctor");
    }

    @Override
    public String deliver(int oid) {
        return oid+" order id order  is delivered by DTDC";
    }

}
```

```
//Flipkart.java (target class)
package com.nt.comp;

import java.util.Arrays;
import java.util.Random;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Component;

@Component("fpkt")
public final class Flipkart {
    //HAS-A  property of type interface
    @Autowired
    @Qualifier("dtdc")    //Soluton 3 for  Ambiguity Problem
    private Courier courier;

    public Flipkart() {
        System.out.println("Flipkart:: 0-param constructor");
    }


    public  String shopping(String items[], float prices[]) {
        //caculate billAmt  (b.logic)
        float billAmt=0.0f;
        for(int i=0;i<items.length;i++)
            billAmt=billAmt+prices[i];
        //generate order id
        int oid=new Random().nextInt(100000);
        //ask courier for shipping
        String status=courier.deliver(oid);
        String finalMsg=Arrays.toString(items)+"are purchsed with prices"+Arrays.toString(prices)+". The geneated billAmount is::"+billAmt;

        return finalMsg+" ::"+status;
    }


}
```

```
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
http://www.springframework.org/schema/beans/spring-beans.xsd
                http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-
context-4.3.xsd">

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

</beans>
```

```
//StrategyDPTest.java
package com.nt.test;

import org.springframework.beans.factory.support.DefaultListableBeanFactory;
import org.springframework.beans.factory.xml.XmlBeanDefinitionReader;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.nt.comp.Flipkart;

public class StrategyDPTest {

    public static void main(String[] args) {
        //create IOC container
        ClassPathXmlApplicationContext ctx=new ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");
        //get Target class obj
        Flipkart fpkt=ctx.getBean("fpkt",Flipkart.class);
        //invoke the b.method
        String resultMsg=fpkt.shopping(new String[] {"shirt","trouser","glasses"},
                                       new float[] {5679.60f,5557.55f,4545.77f} );

        System.out.println(resultMsg);


    }//main
}//class
```

note:: We can cfg property file annotation driven cfgs using @PropertySource or using <context:property-placeholder /> both are same

=>Both are capable recognizing place holders ${<key>} that are placed in xml file, spring beans and replace them with the real values gathered from property file or system properties or env.. variables.

example

info.properties (com/nt/commons)

```
per.name=raja
per.age=30
```

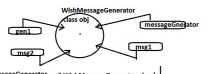
```
DTDC.java
=========
//DTDC.java (Dependent class1)
package com.nt.comp;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.PropertySource;
import org.springframework.stereotype.Component;
@Component("dtdc")
//@Component("courier")
//@Primary
@PropertySource("com/nt/commons/info.properties")
public final class DTDC implements Courier {
    @Value("${per.name}")
    private String name;                    keys of properties file
    @Value("${per.age}")
    private  int age;
    @Value("${os.name}")                    system property key
    private String os;
    @Value("${Path}")                       path env.. variable key
    private  String path;

    public DTDC() {
        System.out.println("DTDC:: 0-param consturctor");
    }

    @Override
    public String deliver(int oid) {
        return oid+" order id order  is delivered by DTDC";
    }

    @Override
    public String toString() {
        return "DTDC [name=" + name + ", age=" + age + ", os=" + os + ", path=" + path + "]";
    }


}
```

Oct 30 Spring Core Annotations-@Value and Layered App Annotations

=>To provide alias names to bean ids use <alias> tag.

In applicationContext.xml

```
<bean id="messageGenerator" class="com.nt.beans.WishMessageGenerator"/>

<alias name="messageGenerator" alias="msg1"/>
<alias name="messageGenerator" alias="msg2"/>
<alias name="msg1" alias="gen1"/>
```



In Client App

```
WishMessageGenerator generator=ctx.getBean("messageGenerator",WishMessageGenerator.class);
WishMessageGenerator generator1=ctx.getBean("msg1",WishMessageGenerator.class);
WishMessageGenerator generator2=ctx.getBean("msg2",WishMessageGenerator.class);
WishMessageGenerator generator2=ctx.getBean("gen1",WishMessageGenerator.class);
```

All these things will give same object reference

Internal cache of IOC container

keys	values
messageGenerator	WishMessageGenerator class obj ref
gen1	WishMessageGenerator class obj ref
msg1	WishMessageGenerator class obj ref
msg2	WishMessageGenerator class obj ref

=>Alias names are useful to provide short name for the lengthy bean id and also useful to achieve loose coupling in annotaion driven cfgs.

=>The prooperties file cfg using @PropertySource can be used only in user-defined spring bean classes.. where as the property file that is cfg in spring bean cfg file using <context:property-placeholder > can be used in pre-defined spring bean classes and also user-defined spring bean classes.

How to achieve loose coupling In annotation driven spring beans cfgs?

Am1) While working with xml+ annotations drvien cfgs we can use two solutions
- solution1:: @Qualifier(-) with <alias> tag based bean aliasing ✓
- Solution2: spring profiles

Am2) While working with 100% code driven cfgs, spring boot programming we can use 1 solution that is spring profiles

Am3) while working with xml driven cfgs use two solutions
- solution1 :: spring cfg file itself ✓
- solution2 :: spring profiles

---

solution1:: @Qualifier(-) with <alias> tag based bean aliasing ✓

step1) take properties file having dependent spring bean id

info.properties (com/nt/commons)

```
choose.courier=bDart
```

step2) cfg properties file in spring bean cfg file and provide fixed alias names by getting dependent spring bean id from properties through place holder.

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
http://www.springframework.org/schema/beans/spring-beans.xsd
            http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.3.xsd">

  <context:component-scan base-package="com.nt"/>
  <context:property-placeholder location="com/nt/commons/info.properties"/>

   <alias name="${choose.courier}" alias="logistic"/>

</beans>
```

step3) In target spring bean class use @Qualifier annotation passing the fixed alias name of Dependent spring bean id

```
@Component("fpkt")
public final class Flipkart {
     //HAS -A  property of type interface
     @Autowired
     //@Qualifier("@tds") // hardcoding of dependent bean id is bad pratice
     //@Qualifier(@Value("${choose.courier}")) // Will not work becoz @Qualifier does allow  @Value
     /*@Value("${choose.courier}")
     private String  cld;
     @Qualifier(cld)  we can not pass bean id as the variable name */
     @Qualifier("logistic")
     private Courier courier;

     public Flipkart() {
          System.out.println("Flipkart:: 0-param constructor");
     }

   public  String shopping(String items[], float prices[]) {
          System.out.println(courier);
          //caculate billAmt  (b.logic)
          float billAmt=0.0f;
          for(int i=0;i<items.length;++i)
               billAmt=billAmt+prices[i];
          //generate order id
          int oid=new Random().nextInt(100000);
          //use courier for shipping
          String status=courier.deliver(oid);
          String finalMsg=Arrays.toString(items)+"are purchsed with prices"+Arrays.toString(prices)+". The geneated billAmount is::"+billAmt;

          return finalMsg+" :::"+status;
     }
}
```

not

fixed alias name collected from properties file

Thumb Rule:-

in XML approach:- configure predefined and userdefined spring bean classes with bean tag.

in Annotation + XML approach:- configure predefined spring bean classes with bean tag and user defined classes with stereo type annotations and link them with <context:component-scan> tag having the base packages of spirng beans

---

Converting MiniProject into xml+ annotation driven cfgs spring app

DAO --->@Repository (@Component+ capable converting underling persistence technology exceptions to spring style exceptions)
Service class ---> @Service (@Component +capable of Transaction Management)
Controller class --->@Controller (@Component + capable of taking http requests)

Q) can i configure Service , controller, DAO classes using only @Component annotations
Ans) Yes ... but not recomanded.. becoz we will loosing the extra benifits given by @Repository ,@Service,@Controller annotations

step1) Cfg DAO ,Service ,Controller classes using streo type annotaions and injectes dependents to them using @Autowired

OracleEmployeeDAO.java

```
@Repository("oraEmpDAO")
public class EmployeeOracleDAOImpl implements IEmployeeDAO {
      private static final String  EMP_INSERT_QUERY="INSERT INTO
                                   REALTIMEDI_SPRING_EMPLOYEE VALUES(SP_ENO_SEQ.NEXTVAL,?,?,?,?,?)";

     //HAS-A Property
     @Autowired
     private  DataSource ds;

   ....
   ....
}
```

EmployeeMySqlDAO.java

```
package com.nt.dao;

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

import javax.sql.DataSource;  //ctl+shift+o  :: To import pkgs

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;

import com.nt.bo.EmployeeBO;
@Repository("mysqlEmpDAO")
public class EmployeeMySQLDAOImpl implements IEmployeeDAO {
      private static final String  EMP_INSERT_QUERY="INSERT INTO REALTIMEDI_SPRING
                                   _EMPLOYEE(ENAME,DEEG,BASICSALARY,GROSSSALARY,NETSALARY) VALUES(?,?,?,?,?)";
     //HAS-A Property
     @Autowired
     private  DataSource ds;
   .....
   .....
}
```

//serice class

```
@Service("empService")
public class EmployeeMgmtServiceImpl implements IEmployeeMgmtService {
     //HAS -A property
     @Autowired
     private IEmployeeDAO dao;

     ....
     ....
     }
```

Controller class

```
@Controller("controller")
public class MainController {
     //HAS -A property
     @Autowired
     private IEmployeeMgmtService service;
     ....
     ....
     ....
     }
```

step2) specify the base package of stereo type annotation classes in spribean cfg file using <context:component-scan> tag.. and also DataSource related prefined class using <bean> tag.

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
http://www.springframework.org/schema/beans/spring-beans.xsd
            http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-
context-4.3.xsd">

<context:property-placeholder location="com/nt/commons/jdbc.properties"/>

 <!  DataSource cfg  >
 <bean id="drds"  class="org.springframework.jdbc.datasource.DriverManagerDataSource">
   <property name="driverClassName" value="${jdbc.driver}"/>
   <property name="url" value="${jdbc.url}"/>
   <property name="username" value="${db.user}"/>
   <property name="password" value="${db.pwd}"/>
 </bean>

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


</beans>
```

step3) Solve ambiguity Problem on service class with respect to DAO classes using @Qualifier and <alias> tag

In jdbc.properites

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

# JDBC.DRIVER=COM.MYSQL.CJ.JDBC.DRIVER
# JDBC.URL=JDBC:MYSQL:///NTSPBMS615DB
# DB.USER=ROOT
# DB.PWD=ROOT

choose.dao=oraEmpDAO
```

In applicationContext.xml

```
<alias name="${choose.dao}"  alias="dbOption"/>
```

In service class

```
package com.nt.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Service;

import com.nt.bo.EmployeeBO;
import com.nt.dao.IEmployeeDAO;
import com.nt.dto.EmployeeDTO;

@Service("empService")
public class EmployeeMgmtServiceImpl implements IEmployeeMgmtService {
     //HAS -A property
     @Autowired
     @Qualifier("dbOption")
     private IEmployeeDAO dao;


}//class
```

Spring Boot Logging

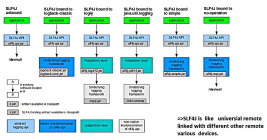
Oct 31 Spring Boot Logging

=>Logging is the process of keeping track of the application flow...
=>The log messages are useful for debugging , for fixing bugs and issues that are raised in the maintainance of the project..
=>Spring boot internally SLF4J for logging

SLF4J :: Simple Logging Facade for Java

=> SLF4J is logging framework that internally uses multiple other logging api/frameworks and provide unified env.. to work any logging api.

=>Various logging api are .. log4j , logback, commons logging , jdk logging , assertions and etc. (utils logging)



=>SLF4J is like universal remote linked with different other remote various devices.

=>Spring Boot's slf4j by default linked with commons-logging api..

=>Once we create Spring boot Project ... we get slf4j jars , log4j , logback and etc.. jar automatically ..
but by default this slf4j is linked commons-logging api.

Logger Levels (priority levels) in SLF4J (These are common for all logging apis that is supports)

TRACE < DEBUG < INFO< WARN < ERROR

=>SLF4J api acts facade or platform or base to work with multiple other logging apis

TRACE :: For those log messages that indiate flow of execution
(start of method ,end of method , start App, end of App)
DEBUG :: Also for flow of execution , but generally used for certain imp activity
(Transaction started , Traction ended , connection created, connection closed)
INFO :: When Most img activities are compltied or stated
(Server activities like container initialization, container closing , initialization important services like Adding filters , Dispatcher Servlet intialization and etc..)
WARN :: If Code executes deprecated apis or poor api which u r planning to replace lattors..
ERROR :: To write log messages from catch blocks indicating problems..

Levles
Logger to apply while retrieving the log message

OFF<TRACE < DEBUG < INFO < WARN < ERROR < ALL

note:: If logger level to retrive log message is TRACE then we get TRACE and above Level messages
note:: If logger level to retrive log message is INFO then we get INFO and above Level messages
note:: Logger Level to retrive Log Message is OFF then logging will be disabled.

note:: The internal code of spring boot App already having slf4j support based loggling activities with INFO Level as default Logger level to retrive log messages. To enable other levels of log messages we can place entries in application.properties file

application.properties

INFO=true (or) DEBUG=true (or) TRACE=true

note:: if u place ERROR=true , WARN=true in above place it still takes INFO=true OFF=true , ALL=true

How to enable Logging on user-defined java classes of spring boot App?

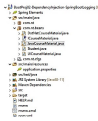
step1) Create Logger object in java classes using LoggerFactory.getLogger() method and use that logger obj in various places of class to write different levels of log messages as shown below.

Student.java

```
package com.nt.beans;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Component;


@Component("stud")
public final class Student {
    private Logger logger=LoggerFactory.getLogger(Student.class);
    @Autowired
    //@Qualifier("${course.choose}")
    @Qualifier("courseId")
    private ICourseMaterial material; // HAS-A property

    public void preparation(String examName) {
        logger.trace("start of method preparation() ");
        System.out.println("preparation started for "+examName);
        String courseConent =material.courseContent();
        logger.info("course content is gathered");
        double price =material.price();
        logger.info("course material price is gathered");
        logger.debug("dependent obj services are used");
        System.out.println("preparation is going on using"+courseConent+" material with price :"+price);
        System.out.println("preparation is completed for "+examName);
        logger.trace("end of method preparation() ");
    }

}
```

Java CourseMaterial.java

```
package com.nt.beans;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.context.annotation.Primary;
import org.springframework.stereotype.Component;

@Component("java")
public final class JavaCourseMaterial implements ICourseMaterial {
    private Logger logger=LoggerFactory.getLogger(JavaCourseMaterial.class);

    public JavaCourseMaterial() {
        logger.trace("JavaCourseMaterial :: 0-param constructor");
    }

    @Override
    public String courseContent() {
        logger.trace("JavaCourseMaterial :courseContent");
        return "1.oops ,2.Exception handling ,3. Collections and etc..";
    }

    @Override
    public double price() {
        logger.trace("JavaCourseMaterial.price()");
        return 400;
    }

}
```



Methods in logger obj to generate differnet levels of log messages
logger.debug("....");
logger.trace("....");
logger.info("....");
logger.warn("....");
logger.error("....");

step2) write following entries in application.properties

application.properties

INFO=true //To control spring boot's internal log messages
logging.level.com.nt.beans=TRACE //to control custom log messages of given package
logging.pattern.console=%30s %r [%t] %p %m %d %L %F %n
Format specifies to decide log message pattern

For All format specifiers refer :
https://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/PatternLayout.html
(name of log4j)

How enable logger level to retrive log messages on customer classes?
Ans) in application.properties we need to specify keys as shown below
logging.level.<pkg1>= <LEVEL>
logging.level.<pkg2>= <LEVEL>

example
logging.level.com.nt.dao=INFO
logging.level.com.nt.service=DE BUG
logging.level.com.nt.controller=TRACE

If multiple sub pkg classes are lookig take same logger level then we can use logging.level.com.nt=INFO

note: If this pkg based logger level is not specified then its INFO as default logger level for custom classes irrespective the logger level enabled for spring boot internal log messages

---

To use File as the Appender to record the log messages

In application.properties
INFO=true

logging.level.com.nt.beans=TRACE
logging.file.name=c:\\log.txt
logging.pattern.file=%c %r %t %p %m %d %n
logging.pattern.console=%30s %r [%t] %p %m %d %L %F %n

Integrating spring boot's SLF4J with log4j2

step1) add spring-boot-starter-log4j2 to pom.xml and exclude spring-boot-starter-logging from spring-boot-starter
The default commons-loggin api

In pom.xml

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-log4j2</artifactId>
</dependency>
```

collect from mvnrepository.com

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter</artifactId>
    <exclusions>
    <exclusion>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-logging</artifactId>
    </exclusion>
  </exclusions>
  </dependency>
```

do it from Dependency hierarchy tab

step2) Keep u r application ready with custom log messages as shown above

step3) add log4j2-spring.xml file in src/main/resources folder
(standard notation)

log4j2-spring.xml (For console and pattern layout)

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration >
  <Appenders>
    <Console name="cons" target="SYSTEM_OUT">
      <PatternLayout
          pattern="[%d{MM-dd HH:mm:ss.SSS}] [%p] [%c] - %m %F %L %n"/>
    </Console>
  </Appenders>
  <Loggers>
    <Root level="trace">
      <AppenderRef ref="cons"/>
    </Root>
  </Loggers>
</Configuration>
```

Logical name of appender cfg

---

for RollingOverFileAppender with Pattern Layout

```
<Configuration >
  <Appenders>
    <RollingFile
          name="rollingFile"
          fileName="E:/application_msg.log"
          filePattern="E:/application_msg.%i.log.zip"
          ignoreExceptions="false">
      <PatternLayout>
        <Pattern>%d{yyyy-MM-dd HH:mm:ss} %-5p %m %F %L %n</Pattern>
      </PatternLayout>
      <Policies>
        <SizeBasedTriggeringPolicy size="5kb" />
      </Policies>
      <DefaultRolloverStrategy max="3" />
    </RollingFile>
  </Appenders>
  <Loggers>
    <Root level="info">
      <AppenderRef ref="rollingFile"/>
    </Root>
  </Loggers>
</Configuration>
```

allows only 3 backup files.

once 5kb log messages are written to log file the backup files will be created as zip file
application_msg.log (main file)
application_msg1.log.zip (backup file1)
application_msg2.log.zip (backup file2)

For DailyRollingOverFileAppender

```
<Configuration >
  <Appenders>
    <RollingFile
    name="rollingFile"
    fileName="E:/app_msg.log"
    filePattern="E:/app_msg.%d{yyyy-MM-dd-hh-mm}.log.gz"
    ignoreExceptions="false">
    <PatternLayout>
      <Pattern>%d{yyyy-MM-dd HH:mm:ss} %-5p %m%n</Pattern>
    </PatternLayout>
    <Policies>
      <TimeBasedTriggeringPolicy filePattern="E:/app_msg.%d{yyyy-MM-dd-hh-mm}.log.gz" />
    </Policies>
    <DefaultRolloverStrategy max="5" />
  </RollingFile>
  </Appenders>
  <Loggers>
    <Root level="trace">
      <AppenderRef ref="rollingFile"/>
    </Root>
  </Loggers>
</Configuration>
```

Here are sample date/time patterns for rolling of files base on date/time.

DATE/TIME Pattern	Description	Intervale Attribute Example
%d{yyyy-MM-dd-hh-mm}.log.zip	Roll the log files every minutes	If interval=1, rollovers will occur every 1=0 minutes. E.g. : 2017-07-24-09-57.log.zip, 2017-07-24-09-58.log.zip, 2017-07-24-10-01.log.zip, 2017-07-24-10-02.log.zip etc..
%d{yyyy-MM-dd-HH}.log.zip	Roll the log files hourly	If interval=4, rollovers will occur every 4 hours. E.g. : 2017-07-24-09.log.zip, 2017-07-24-10.log.zip, 2017-07-24-11.log.zip etc..
%d{yyyy-MM-dd}.log.zip	Roll the log files daily	If interval=1, rollovers will occur every day. E.g. : 2017-07-24.log.zip, 2017-07-27.log.zip etc..