Course :: spring basics, spring boot, spring cloud and microservices **Aug 26 Intro 1**
pre-requisite :: core Java oops
duration :: 140 to 150 sessions
timings :: 7:30 pm to 9pm /9.15 pm Completely online course
weekly 6/7 classes

Faculty Details ::
name:: nataraj
FB Group name :: natarazjavaarena
FB group url :: https://www.facebook.com/groups/388095825162910
email id :: natarazjavaarena@gmail.com
admin details ::
Mr.Srikanth: +91 6302968665 (whats up only)

Course Fee :: 8`000/- with ReactJS or Angular or HB with JPA
5000 /- (with out free topic)

earlier days: ReactJS / Angular by sudhakr sharma sir.
1. spring course (90 sessions , fee : rs:3500)
2. spring boot and MicrosServices (80 session ,rs:4000)

Spring basics, spring boot , springcloud and microservices (spring stack Course)

free :: angular or reactjs or hiberante with JPA
free :: 15+ realtime java tools like Maven,Gradle, GIT, SVN,
log4j, slf4j, jenkins, docker , ...
free :: 10 + commons topics (resume preparation, interview tips,
Dos and do nots after joining in the s/w company,
AGILE-SCRUM, JIRA and etc..
free :: 7 to 8 Mini Projects.

free :: Job guarantee program by Naresh IT (u need to clear screening test)

Spring Modules covered (Spring stack)
==========
spring core
spring jdbc
spring aop Recorded vedios download course content from
spring TxMgmt https://www.facebook.com/groups/388095825162910/files
spring data jpa
spring data mongoDB
spring mvc
spring security
spring oauth
spring batch
spring rest
spring boot actuators
microservices
spring mail + dynamic content...
spring cloud
and etc...

Running batch time is :: 7 am to 8:30 /8:40 am

notes /materials
==========
=> old batch pdfs will be shared
=> old batch Worskpaces will be shared
=> u r batch running notes/material will be shared using FB group /Google classroom
=> u r batch Calss Room Apps , Mini Projects will be shared using GIT / GITHUB
=> Recorded session using www.youtube.com/nareshit

Who needs this course

Fresher :: spring basics ,spring boot (mandatory)
other advanced topics are optional (recomanded to show them added advantage)

=> 1+ , 2+ ,3+ 4+ ,5+ (complete spring stack course)

=>pkg for Freshers :: 4 to 6 lacs
=> pkg for 1+ ,2+ :: 5 to 7 lacs
=> pkg for 3+ , 4+ :: 6 to 12 lacs
=> pkg for 5+ :: 8 to 15 lacs

3+ years experience :: 16 lacs
(record package so far)

---

to

Do i need learn xml and webSErvices course seperately?

Ans) No ... not required becoz spring Rest is there web service implemenation

DO i need to learn Adv .java course (jdbc,servlet,jsp) ?

Ans) yes required ... we should have servlet,jsp knowlege before
it start spring MVC topic here.

Do i need to learn Hiberatne with JPA course ?

Ans) we can manage with out that course .. but learning is added advantages
=> spring data jpa internally uses hibernate... So learning hibernate makes
spring data jpp module learning process quite easy.

What else i should learn to become java full stack developer after this course?

Ans) spring stack course (current course)
+
angluar or reactjs (UI Technology)
+
oracle Db s/w
+
slenimum (Testing)
+ Design patterns
+
DevOps (Deployment)
+
AWS / Azzure /Google cloud

Java full stack developer pkg for 3+/4+/5+ :: 20 to 25 lacs.

Q) This is my first framwork course , is there any problem?
Ans) No Problem .. we will learn from 0% (right from what is framework?)

Q) can i learn adv.java course and framework course parellel?

Ans) yes u can

Q) My Core java is just started , can i sit in this course ?

Ans) No.

spring basics, spring boot, spring cloud and micro services

Aug 27 intro 2

(spring stack course or complete spring course or end to end learning of spring framework)

This course details with

spring basics
spring boot
(multiple module of spring will be discussed in spring boot mode)
|-->spring mail, spring mvc, spring security ,spring oauth ,spring data jpa, spring mongoDB and etc..)
spring Rest (For restfull web services)
spring batch
spring cloud
MicroServices

15+ tools
10 + common topics
7 to 8 Mini Projects..

free courses
->React JS or
-> Angular
-> Hibernate with JPA
(by nataraj)

by sudharkar sharma

For class room material :: https://www.facebook.com/groups/388095825162910/files
ur batch code :: NTSPBMS615

=>c,c++,java ,c# and etc.. programming languages (like raw materials)
=>JDBC, Sevlet,Jsp, EJB, JMS and etc.. are called Java Technoloies (like semi-finished products)
=> Spring,struts ,hibernate,JSF and etc.. called java frameworks.. (like fully finished products)

Can u explain the differences among programming languges, technologies and frameworks?

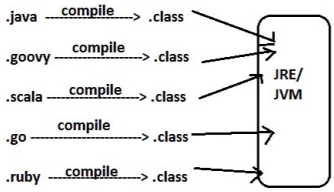
Programming language

syntax: rules of coding
semantics : structure of the code.

=>It is directly installable s/w acting as raw material providing basic features for developing s/w Applications.
=>It defines the syntaxes, semantic of programming by supplying compilers and interpeters.
=>Programming languages are base to create s/w technologies , tools, frameworks, Operating system,Db s/ws and etc.

eg: c,c++, java , java script ,c# , vc++ and etc..

JVM based languages
java , scala, groovy ,koltin, go ,ruby and etc..



Technologies

=>Technology is a s/w specification providing set of rules and guidelines in the form apis to create implementation softwares using one or another programming language.

=> Technology is not installable.. but the technology based implementation s/w is installable. Working with Technology based implementation s/w is nothing but working with technology.

eg:: jdbc , servlet, jsp , EJB,JMS , JAAS ,java Mail and etc..
Jsp :: java server pages
EJB :: Enterprise Java Beans
JMS : Java Messaging service
JAAS : Java Authentication and Authorization service

=>JDBC is a java technology that gives rules and guidlines in the form of jdbc api to create JDBC driver s/w using java programming language.
=>JDBC technology is not installable.. But jdbc technology based JDBC driver s/w is installable or arrangable .. Working withb JDBC Driver s/w (like ojdbc6.jar) is nothing but working with JDBC Technology.

API :: Application Programming interface

In "C" language API means set of functions given in the form of header files
In "C++" language API means set of functions,classes given in the form of header files
In "Java" language API means set of classes,interfaces,enum,annotations given in the form of packages

Upto java7

=> The Technology API package interfaces represent rules
=> The Technology API package abstract classes represent rules and guidelines
=> The Technology API package concrete classes represent guidelines

Normal java class for which object creation is possible is called concreate class.

From Java8 onwords

=>The Technology API Packages interfaces, abstract classes represent both rules and guidelines
=>The Technology API Packages classes represent guidelines

note:: Every technology does not given rules and guidelines in the form of english statements.. they given them as java api having packages with classes, interfaces and etc..

Two types of Technologies

1.Open Technologies

=>Here the technology rules and guidlines (apis) are open to all the vendor companies to create implementation s/ws

eg:: All JAva , JEE Technologies like JDBC,JNDI,EJB,SErvlet,Jsp, JMS and etc..

2.propritiery Technologies

=>Here the technology rules and guidlines (apis) are specific to vendor and only that vendor company is allowed to create implementation s/ws

eg: All MicroSoft technologies (asp.net, vb.net and etc..)

Frameworks Introduction

Technologies

=>Technology is a s/w specification providing set of rules and guidelines in the form apis to create implementation softwares using one or another programming language.

=> Technology is not installable.. but the technology based implementation s/w is installable. Working with Technology based implementation s/w is nothing but working with technology.

eg:: jdbc , servlet, jsp , EJB,JMS , JAAS ,java Mail and etc..
Jsp :: java server pages
EJB :: Enterprise Java Beans
JMS : Java Messaging service
JAAS : Java Authentication and Authorization service

=>JDBC is a java technology that gives rules and guidlines in the form of jdbc api to create JDBC driver s/w using java programming language.
=>JDBC technology is not installable.. But jdbc technology based JDBC driver s/w is installable or arrangable .. Working withb JDBC Driver s/w (like ojdbc6.jar) is nothing but working with JDBC Technology.

API :: Application Programming interface

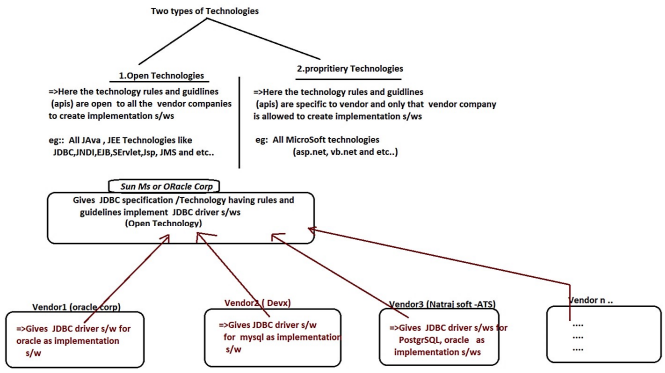
In "C" language API means set of functions given in the form of header files
In "C++" language API means set of functions,classes given in the form of header files
In "Java" language API means set of classes,interfaces,enum,annotations given in the form of packages

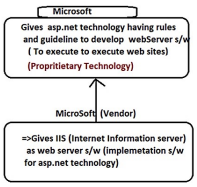
Upto Java7
==========
=> The Technology API package interfaces represent rules
=> The Technology API package abstract classes represent rules and guidelines
=> The Technology API package concrete classes represent guidelines

Normal java class for which object creation is possible is called concreate class.

From Java8 onwords
==========
=>The Technology API Packages interfaces, abstract classes represent both rules and guidelines
=>The Technology API Packages classes represent guidelines

note:: Every technology does not given rules and guidelines in the form of english statements.. they given them as java api having packages with classes, interfaces and etc..





=>Since all vendors are giving their implementation s/ws based on the common rules and guidelines software technology ..the way we work with implementation softwares of all the vendors is going to be much similar.
i.e if we know how to work with one implemetation s/w,we can use that knowledge to work with another implemetation s/w of same technology

=> All JDBC driver s/ws are given based on the common rules and guidlines of JDBC technology..So tha way we work with all the implementation s/ws (JDBC driver s/ws)is goining to be much similar.

Framework
=========
Def1:: Framework is installable s/w that is built on top of 1 or more technologies having capability to generate the common logics of the App/project dynamically ..and letting programmer to focus only on application specific logics devleopment.

Def2:: Framework is installable s/w that is built on top of 1 or more technologies providing abstraction on Technologies to simplify the Application/Project devleopment process.
hiding the implemetations/internals.

example of JAva framworks struts , JSF, spring , spring boot hibernate, Eclipse Link , Toplink, Sring Rest , Axis , Apache CFX and etc..
example of Java script frameworks /Libraries/Toolkits :: angular , reactJS and etc..
example of .net frameworks :: asp.net mvc ,aps.net core and etc..
example of php frameworks :: Drupal , Code Iginator , WordPress, Larvel and etc..
example of python frameworks :: DJango , flask and etc..
example of BigData framework :: Hadoop , Spark and etc..

=>While working with Technologies , makes the programmer to write both common logics and application specification logic (improves the burden on the programmer)
=>While working with Framework makes the programmer to write only the application specific logics becoz the the framework will take care of generating application specific logics devleopment.

JDBC Aplplication (Java Technology App)
==========
=>regster JDBC driver s/w (by loading jdbc driver class)
=> Establish the connection with DB s/w from Java App
=> create JDBC Statement object
Common logics

=> send and execute SQL queries in Db s/w (inputs) using Statement object
=> Gather SQL query results from Db s/w and process them using Statement object
application specific logics.

=> Exception handling
=> close connection with Db s/w
Common logics

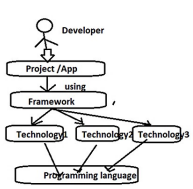
note: Here Programmer needs to take care of both commong logics and application specific logics
Common logics= biolerplate code (problem)

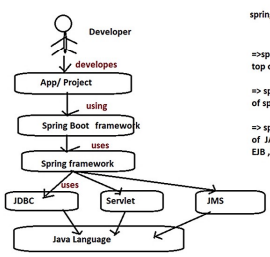
=>The code that repeates across the multiple parts of the project either with no changes or with minor changes is called biolerplate code (common logics) , i.e while working with technologies we have boilerplate code problem.

Spring JDBC App (Framework App) (note:: spring JDBC is part of spring framework)
==========
=>create JdbcTemplate class obj (pre-defined class given by spring framwork)
(Takes care of common logics internally by using technology)

=> send and execute SQL queries to Db s/w
=>Gather SQL query results and process the results
application specific logics

note ::Here programmer needs to take care of only application specific logic becoz the common logics will be generated by the framework dynamically .. i.e no boilerplate code problem.





.... techonologies

spring boot = spring ++

=>spring boot is framework that built on the top of spring framework

=> spring boot avoids boilter plate code of spring framework programming.

=> spring framework avoids boilter plate code of JAVA,JEE technologies like JDBC, servlet,jsp, EJB ,JMS and etc.. programming

Advantages with Frameworks
==========

Aug 31 Types of Frameworks

a) Simplifies application development by providing abstraction on one or more technologies

b) Avoids the boiler plate code by generating common logics of the application internally and dyanamically.

c) Improves the productivity becoz programmers just needs to develop only Application specific logics

d) Lots of framework are giving ready made apis to directly implement realtime scenarios or uses-cases.

e) Now a days frameworks have become industry defacto standard to develop medium scale and larget scale projects.

f) Some frameworks like spring boot are giving built-in servers , DB s/ws and etc.. So that we need not arrange them seperately in the devleopment mode and testing mode of the project..

(This feature is very useful while developing applications by taking s/w and hardware setup from cloud env.. like AWS,Google Cloud and etc.. on rental basis.

and etc..

Types of java frameworks (Based on the kind of Apps we can develop)
==========

a) ORM frameworks
b) Web application frameworks/MVC frameworks
c) JEE Frameworks/ Application frameworks
d) WebServices frameworks/ Distributed App Frameworks
e) Big Data frameworks (Not required for Java developer or full statck java developer. Required only for BigData Analyst or Data Scientist)

1) ORM Frameworks (ORM :: Object relational mapping)
==========

=> Provides abstraction on plain JDBC Technology and simplifies the Objects based DB s/w independent Persistence logic developement with out using any SQL queries.

=>The code that performs insert,update,delete ,select operations db s/w tables is called Persinstence logic (eg: jdbc code, hibernate code, eclipse link code and etc..)

=>JDBC code (persistence logic) is DB s/w dependent becoz it uses the Db s/w dependent SQL queries
=> ORM frameworks code (like hibernat code) is portable across the multiple Db s/ws becoz no SQL queries will be used and entire persistence logic will be develope using java objects.

eg::
hibernate ----------> from softTree /redhat (1)
Eclipse link ----------> from Eclipse (2)
OJB ----------> from apache OJB >> Object Java Beans
iBatis ----------> from apache (3)
Toplink ----------> from oracle corp
and etc..

2) Web application frameworks/ MVC frameworks
==========

=> provides abstration on servlet,jsp technologies and simplifies MVC architecture based web application devleopment..

MVC :: Model View Controller (Developing the web application as layered Application)

=>MVC architecture servlet,jsp web applicaiton devleopment (1000 Lines of code)
=>Spring MVC based MVC architectecture web application devleopment (500 lines of code)
=>Spring Boot MVC based MVC architectecture web application devleopment (200 lines of code)

Struts ----------> from apache
JSF ----------> from sun Ms /oracle corp (3) (JSF :: JAva Server Faces)
webwork -----> from Open symphony
ADF ----------> from oracle corp (ADF: Application Development framework)
Spring MVC ----> Interface 21 (team name : pivotal team) (2)
Spring boot MVC ----> Interface 21 (team name : pivotal team) (1)
and etc...

=>Spring framework /spring boot framework are having multiple modules like core , AOP, JDBC, Tx Mgmt , MVC, Rest, security , Batch ,mail and etc.. So we can spring MVC is module of spring framework and spring boot MVC is module of spring boot MVC framework

Fee details
==========

spring statck course + Free React JS or angular or hibernate with JPA :: 8000/-
only spring stack course :: 6000/-

only angular or react JS :: 5000/-
only hibernate with JPA :: 3000/-

Sep 01 Types of Frameworks

Types of java frameworks (Based on the kind of Apps we can develop)

==========

a) ORM frameworks
b) Web application frameworks/MVC frameworks
c) JEE Frameworks/ Application frameworks
d) WebServices frameworks/ Distributed App Frameworks
e) Big Data frameworks (Not required for Java developer or full statck java developer. Required only for BigData Analyst or Data Scientist)

JEE Frameworks/Application frameworks

=> provides abstraction on multiple Java ,JEE technologies and simplifies different Java ,JEE application devleopement process.

eg:: spring , spring boot

=> spring/spring frameworks are JEE frameworks that provides abstraction JAva technologies like JDBC,JNDI,RMI and et.. and also on JEE technologies like servlet,jsp,jms,ejb,java mail and etc.. to simplify all kinds of java ,jee applications development.

=>Using spring /spring boot we can devleop standalone Apps, web applications, distribited Apps and etc..

spring boot =spring++

WebServices framework / distributed Application framework

==========

Distributed App

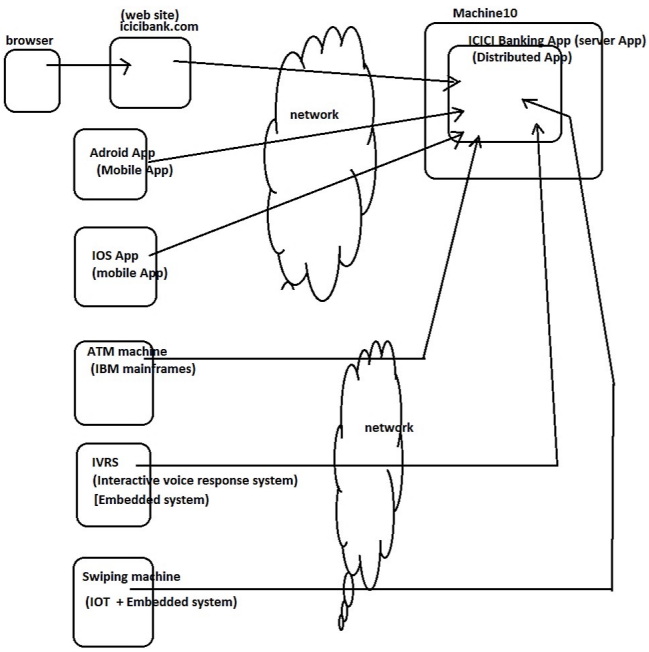
==========

=>The App that allows to have different types of Local or Remote clients to access and execute the logics is called Distributed Application.

eg:: Google Pay App , Phone Pay , PayPall (payment broker), Banking server Apps and etc.

note:: if Server App and client App are using same JVM then that client App is Local Client App called

note:: if Server App and client App are using two different JVMs of same computer or different computer then that client App is Remote Client App called

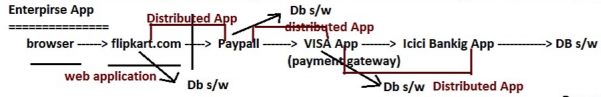


=>To develop distributed App we need to use webServices tehcnologies and frameworks

=>JaxWs , Jax-RPC and etc.. web service technologies
=>Spring Rest , Jersry , Axis , Apache cfx , Rest easy and etc..are web service framworks.

only web application or only destributed App or web application +distributed App is called an Enterprise App

Enterpirse App
==========



Payment gateways
VISA
Master
Mastrero
Rupay
and etc..

What is the diffence web application(website) and Distributed App?

a) It client -server App where Client is a browser and server is s/w App	a) It is Client -server App where both client and server Apps s/w Apps
b) allows only browser as the client	b) allows differen types of client Apps
c) It is Thin Client -Fat Server App	c) It is Fat client - Fat Server App
d) communication model request-response model	d) Communication model is method calls model
e) use servlet,jsp technologies to develop web sites in java	e) use jax-ws, jax-rpc ,jax-rs and etc. technologes to develope web service based distributed Applicaitons
f) use struts, JSF, spring MVC, spring boot MVC and etc.. f/w to devleop java web applications	g) use Spring Rest, Rest easy, jersy , Axis, Apache CFX and etc.. to develope web services based distributed Apps..

=>Standalone App is the App that is specific one to computer and allows only user at time to operate the App
eg:: class with main(-) , calculator App ,core Java Apps

=> The client -Server App where client is browser and server App is s/w App interacting with each other using http protocl in request-response model
eg: nareshit.com , flipkart.com

=> The client -Server App where both client and server Apps s/w Apps talking with each other through method calls sinvocation .. the Server app can have diffrent types of local or remote clients
eg:: UPI Apps, IRCTC App , BSE App, NSE App ,Weather Report App and etc..

Types of java frameworks (Based on the kind of Apps we can develop)

Sep 02 Types of Frameworks -Java Bean

a) ORM frameworks
b) Web application frameworks/MVC frameworks
c) JEE Frameworks/ Application frameworks
d) WebServices frameworks/ Distributed App Frameworks

part of spring/spring boot framework

e) Big Data frameworks (Not required for Java developer or full statck java developer.
Required only for BigData Analyst or Data Scientist)

BigData Frameworks (Required only for data scientist or BigData analyst)

=>The data that is beyond processing and storing capacity using the regular Db s/ws and computers is called Big Data (Generally talks PB,TB, XB,YB of data)

1024 bytes = 1kb
1024 kbs = 1 mb
1024 mbs = 1 gb
1024 gbs = 1 tb
1024 tbs – 1 pb
1024 pbs = 1 xb
1024 xbs = 1 yb

=> This store and process this kind of big data by using ordinary computers of LAN or cloud we need Bigdata frameworks

eg:: Hadoop (java) , spark (java)

=> spring ORM or spring data jpa module is buit on the top of hibernate f/w (ORM f/w)
=> spring MVC module is given as web application f/w
=> spring Rest module is given as webService f/w (Restfull websservices)
So we can spring or spring boot framework are JEE frameworks/ Application framworks/All rounder frameworks

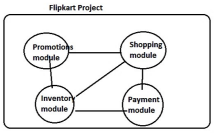
Using spring or spring boot we can develop two types Apps

a) Monolithic Architecture based Apps
b) MicroService Architecture based Apps

Monolithic Architecture

Each each service will be devleoped as each module and all the modules integrated module in a Project i.e we can not use each module as seperate service/project.

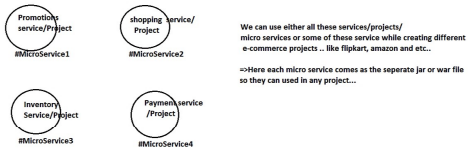
Flipkart Project



=>Here we need to use all modules to gather with in a project i.e we can not use each module as seperate independent service /project.
=> We get reusability or modurality with in project.. not acorss the multiple projects.

=>All modules togather will be released as single jar file /war file

MicroService Architecture Project



We can use either all these services/projects/ micro services or some of these service while creating different e-commerce projects .. like flipkart, amazon and etc..

=>Here each micro service comes as the seperate jar or war file so they can used in any project...

Can u explain JAva Bean class, POJO class, POJI, Bean class/Component class and spring bean?

Java Bean

=> It is a java class that is developed by following some standards
=> It is always used as helper class to pass multiple values as single object from one class to another class of same project or different projects.

The standards are

=>class must be publi class
=>Recomanded to implement java.io.Serialzable(I)
=> All member variables must be taken as private and non-static (bean properties)
=> Make sure that one Zero param constructor is going to come in the class directly or indirectly.
(given by developer) (compiler given)
=> Every bean property (non-static variable) should have setter method and one getter method (setter method is useful to set /modify data of bean property and getter method is useful to read data from bean property)

Example

```
public class CreditCardDetails implements java.io.Serializable{
  //bean properties
  private long cardNo;
  private String cardType;
  private String gateway;
  private int cvv;
  private Date expiryDate;

  //setter and getter methods
  public void setCardNo(Long cardNo){
    this.cardNo=cardNo;
  }
  public long getCardNo(){
    return cardNo;
  }
  ....
  .... //more setter and getter methods ( 4 + 4)
  .....
  .....
}
```

In Java Bean we do not place methods having b.logic.. becoz java bean is just data carrier.

problem:

```
public class StudentProgressReportService{

  public String generateResult(int sno,String sname, String addrs, int m1,int m2,int m3){
      ..... //calculate total, avg
      .... //generate result
      ....
    reuturn "pass"/"fail";
  }
}
```

Client App

```
public calss ClientApp{

  public static void main(String args[]){

    StudentProgressReportService servlce= new StudentProgressReportService();
      String result=service.generateResult(101,"raja","hyd",70,88,99);
      System.out.println(result);
    }
  }
```

Designing java method having more than 3 params is bad pratice , the reasons

a) we must remember the order of args while calling the method
b) Though we do not one or another arg value.. we still need to pass meaning full dummy value.
c) Remembering lengthy signature of the method is very complex.

Solution :: take Java Bean type parameter for java method

```
//Java Bean
public class Student{
  private int sno;
  private String sname;
  private String sadd;
  private int m1, m2,m3;
  //setter and getter methods
    ..... 6 setter , 6 getter methods
    .....
}
```

```
//serlvce class /Bussiness class
public class StudentProgressReportService{

  public String generateResult(Student st){
    .... //read data from "st" obj using
    .... //getter methods and calculate total,avg
    .... and generate reuslt;
    ....
    return "pass"/"fail"
  }
```

Client App

```
public calss ClientApp{

  public static void main(String args[]){

    StudentProgressReportService service= new StudentProgressReportService();

    Student st=new Student();
    st.setSno(101);
    st.setM1(90);
    st.setM2(30);
    st.setSname("raja");

    String result=service.generateResult(st);
      S.o.println(result);
  }
}
```

Advantages of working java bean

a) method signature becomes simple signature to remember
b) While setting data java bean class obj we need not to follow the order
c) the names of setter methods helps a alot while setting data to bean properties of java bean class obj
d) we can ignore to set one or two values which unknow us.

Sep 03 Types of Frameworks -POJO class, POJI, Spring Bean ,Bean class

POJO class
=========
=>Plain Old Java Object class
=> The ordinary java class with out specialities is called POJO class
=> The ordinary java class that compiled only using Jdk libraries is called POJO class
=> The Orindary java class that is not extending and implementing technology, framework apis classes and interfaces is called POJO class.

note: POJO class is not againt of inheritence or implementing interfaces .. It is aganist of inheriting and implementing technology,framework api classes and interfaces.

=> oops, exception handling, awt,swing, collection api , reflectin api, networking api , applets ,multithreading and etc..part java language.
=> Jdbc, servlet,jsp,jndi, jsp ,ejb and etc.. called java techologies
=> struts, spring, hibernate, spring rest and etc.. called java frameworks..

```
class Test{
  ...
  ..
}
POJO class

class Demo extends Test{
  ...
  ...
}
POJO class

class Demo extends Thread{
  ....
  ...
  ...
}
POJO class
(part of java lang api)

class Demo extends HttpServlet{
  ...
  ...
  ...
}
Not a POJO class
(given by servlet api)
```

```
class Demo implements java.rmi.Remote{
  ....
  ...
}
Not a POJO class
(part of RMI technology)

public class Demo1 implements java.sql.Connection{
  ....
  ....
}
not a POJO class
(part of JDBC technology)
```

```
class Test extends Test1{
  ....
  ....
}

class Test1 implements java.sql.Statement{
  ...
  ...
}

Test,Test1 are not POJO classes..
```

```
class Test extends Test1{
  ...
  ...
}

class Test1 implements java.io.Serializable{
  ....
  ...
  ...
}
(part of IO streams (java language))

Test,Test1 are POJO classes..
```

```
public class Demo{
  public void m1(){
    ....
    ... //hibernate API
  }
}
It is POJO class ..but can not be compiled just having jdk libraries in the class path (Exceptional case)

@Entity --> hibernate api annotation
class Demo{
  ...
  ..
}
It is POJO class ..but can not be compiled just having jdk libraries in the class path (Exceptional case)
```

=> POJO class need not follow java bean standards..
=>Every Java bean class is POJO class .. but every POJO class is not Java bean class
=> POJO classes can have b.methods with complex b.logics..

=>Spring supports POJO and POJI model programming i.e the main classes/resources of the spring projects /Apps can be taken as ordinary classes (POJO classes) and ordinary interfaces (POJI) .. with out having depedency with spring APIs .

POJI (Plain Old Java Interface)
====
=> An ordinary interface with out specialaties is called POJI
=> An ordinarly inteface that can be compiled only using jdk libraries is called POJI
=> The interface that is not extending from Technology , framework api interfaces is called is called POJI.

```
interface Test1{
  ...
  ...
}
It is POJI

inteface Test1 extends java.lang.Cloneable{
  ...
  ...
}
(java lang api)
It is POJI
```

```
Interface Test2 extends java.rmi.Remote{
  ....
  ...
  ...
}
(RMI Technology API)
It is not a POJI

interface Test3 extends java.sql.Connection{
  ....
  ...
}
Not a POJI
```

```
interaface Test4 extends java.lang.Runnable{
  ....
  ....
}
It is POJI

interface Test5 extends Test6{
  ...
}
interface Test6 {
  ...
}
"Test5","Test6" are POJIs
```

```
@Remote --> given EJB API
interface Test7{
  ...
  ..
}
not
=>POJI but can be compiled only having jkd libraries
(Exceptional case)
```

=> If project supprots POJO ,POJI model programming while devleoping main classes where b.logics and persistence logics.. then we can move those other frameworks having loose coupling classes to

note:: POJO classes , POJIs are very much specfic to programmer devleoped user-defiend classes.. we can not use these terminologies while refering technology ,framework apis supplied pre-defined classes..

Bean Class /Component class
=========
=>The java class that is havin state (member variables) and behaviour (mehtods) and state is used inside the behaviour having reusability is called Bean class /component class..
=> The methods of component/Bean class contains b.logic with reusability ..
=> This class can be developed as POJO class or not POJO class
=>This class not be devleoped as JAva Bean..
=> In real projects service class (contains b.logic) , DAO class (persistence logic), controller class (monitoring logic) are component /Bean classes

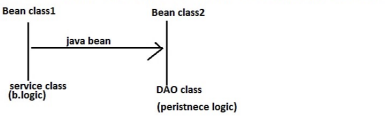
=>Servlet classes are component classes

example
==========

```
public class BankingService{
 //state
 private String branchName;
 private String ifscCode;
 private String location;
 private long mid;
 private long phoneNo;
```
member variables

```
 public String transferMoney(long srcAcno,
                             long destAcno,
                             double amount){
   .....
   ... //b.logic to transfer money
   ...    (state will be used in b.logic)
   }
}
```

=>Java bean class is different from Bean class /component..
=>Java bean is helper class .. acting as data carrier
=>Component /bean class is main class having main logics like b.logics, Persisntence logics and etc...
=>if needed u can use java bean as data carrier b/w two bea classes/component classes



note:: JAva Bean :: Java class acting data carrier having setter and getter methods
POJO class :: Ordinary java class supporting portability across the multiple java projects
POJI :: Ordinary java Interface supporting portability across the multiple java projects
Componet/Bean calss :: class having state , behaviour with reusability .. can be devleoped as POJO or as Non POJO clss.

What is the difference b/w tool and framework?

Spring Bean class
==========

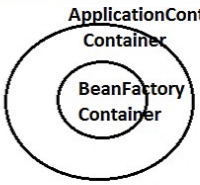
Sep 04 Spring Bean and Spring Core module

=>The java class whose object is created managed by spring Containers (part of spring framework) is called Spring bean i.e the whole life cycle spring bean class will taken care by spring containers.

=>A container is s/w program that manages whole life cycle of given resource/class i.e takes cares of all activities from birth to death (Object creation to object destruction).

=> Container is like an aquirioum taking care of fishes called comps/resources/classes.

=> Servletcontainer takes care of the servlet comps life cycle
=> Jsp Container takes care of the jsp comps life cycle
=> Spring Containers takes cares of spring beans life cycle.
note: spring containers no way related to servlet container and jsp container .In fact spring containers not alternate servlet container, jsp container.
=> Spring f/w given two built-in spring Containers
a) BeanFactory Container (Basic Container)
b) ApplicationContext container (advanced container)
ApplicationContext container =BeanFactory container ++



=> Spring bean class can be a user-defined class or pre-defined class or third party supplied java class but its life cycle management (object creation to object destruction) must be done by Spring containers.

=>Spring Bean class can be developed as POJO class or Non POJO class (POJO class is recomanded)
=>Spring Bean class can be a Java bean class or component /Bean class.

=> Generally , we will not take abstract classes ,interfaces as the spring beans becoz they can not be instantiated (object creation is not possible)

=> To make java class spring bean class we need to <bean> tag in xml file or @Component annotation.

=> We can give instructions/inputs to Springcontainers either using xml file(spring bean cfg file) or using annotations (best).

using xml cfgs

```
<bean id="bankService"  class="com.nt.beans.BankServiceMgmt"/>
```

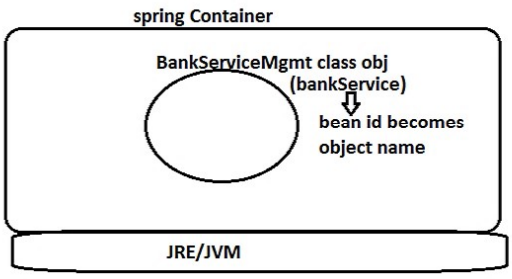
- id="bankService": Internally becomes spring bean class object name when spring container creates the object.
- class="com.nt.beans.BankServiceMgmt": Fully qualified java class to take as spring bean
  - pkg: com.nt.beans
  - class name: BankServiceMgmt

using xml cfgs (Giving details to Spring container using xml file and xml tags)

(or)

```
package com.nt.beans;

@Component("bankService")
public class BankServiceMgmt{
   ....
   ....
   ...
}
```

=> Annotation cfgs (Giving details to spring container using annotations)



=>Generally in core java standalone Apps we manually create object using "new" operator becoz we need to create less no.of objects as standalone App is speciific to computer and will be operated by 1 user at time..

=> While dealing web applicaitons(websites), distributed Apps, enterprise Apps .. we will be having lacs of users using multiple services simultaenously.. So huge no.of obejcts creation ,management , destruction is required continuosly 24/7 For this we take the support containers for automating the whole process.

Q) every JAva bean class is POJO class? (true)
Q) every POJO class is a Java bean class? (false)
Q) Spring Bean class must be a POJO class? (false)
Q) Spring Bean class can be a POJO class? (true)
Q) Thrird paty supplied java class can be taken as spring bean? (true)
Q) we create objects for spring bean classes? (false)
Q) spring containers takes all java classes of jdk library as spring bean calsses automatically? (false)
Q) Java bean can not be taken as spring bean? (false)
Q) The id of spring bean becomes spring bean class object intenrally (true)
Q) Every Spring bean class must component class ? (false)
Q) Component class can be taken as spring bean class ? (true)

Two types of frameworks based on mode of the Application development
==========

a) Invasive Framework
b) Non-Invasive Framework

Sep 06 Spring Core Module

a) Invasive Framework

=>Here the resources/classes framework based App devleopment are tightly coupled with Framework APIs i.e the classes of the App development should implement or should extend from Framework API interfaces or classees.
=> HEre the App classes developeed progammer for certain Framework .. should always be executed in the same framework and can not be moved to another frameworks.

eg: Struts Framework

=> Invasive frameworks do not support POJO and POJI model programming

b) Non-invasive Framework
==========

=>Here the resources/classes framework based App devleopment are loosely coupled with Framework APIs i.e the classes of the App development need not implement or extend from Framework API interfaces or classees.

=>HEre the resources/classes of the App develeopment will be developed as POJO classes i.e it support POJO/POJI model programming..

=>HEre the App classes develped from certain framework can be moved another frameworks dynamically becoz they are POJO classes

eg: Hibernate, Spring , spring boot, JSF and etc..

---

spring
=====
type :: Application framework or JEE framework
vendor :: Interface21 /pivotal team
open source framework
Creator :: Mr.Rod jhonson

Repository means :: small perminent storage place

spring libaries (jars) :: earlier they used give in form of zip file .. now they stopped giving..
asking to use Maven /Gradle dependencies mechanism (download jar files from internet repositories dynamically)

How an Open Source company survive financially?

Ans) a) By conducting corporate training on new technologies
b) By conducting certification exams
c) By conducting certification exams traning programs
d) By providing services to s/w companies if they struckup in the middle of the Project devleopment.
and etc..

Q) Is spring is alternate for EJB?

Ans) No, EJB is just distributed Technology to develop distributed Apps where as Spring is all-rounder or JEE framework which can be used to devlep standalone Apps,web applications, distributed Apps and etc..

Q) Is spring alternate to Struts ?

Ans) NO, Struts is just for web application development where as Spring is all-rounder or JEE framework which can be used to devlep standalone Apps,web applications, distributed Apps and etc..

Q) Is spring alternate for JEE Technologies?

Ans) No, Spring is a framework internally using JEE technologies to simplify the Application devlelopment.. i.e spring is altenrnate JEE technologies rather it compliments them by internally using them,

Q) Is spring boot is alternate spring ?

Ans) No, spring boot internally uses spring framework and simplifies spring Apps development.
note: spring boot extension spring to simplfiy spring Apps development.

note:: Boilerplate code (comnon logics) of JAva, JEE TEchologies programming will be avoided by spring frameworknote::
note:: Boilerplate code (comnon logics) of spring programming will bed avoided by spring boot framework

---

Spring/spring boot modules: core, aop, jee, Txmgmt, jndi,orm, jdbc, mvc, mail , jms and etc..
sprjng/spring boot extension modules :: security , batch, oauth , social , spring data jpa, spring mongo db ,sping cloud,spring rest and etc..

spring micro services concepts :: eureka server and client,
Load Balancer,
Flyway Database,
ZUUL API gateway,
WebClient VS FeginClient,
Spring config server & client,
circuit breaker,
Retry limiter pattern,
rate limiter pattern,
and et...

Why spring or spring boot framework libraries are given in the form modules?

Ans) Generally we do not or we can not use all spring/spring boot modules in one Project development.. we use different sets of modules in different projects.. So they releases spring /spring boot libraries (jar files) as modules.. So that we can add only thosse libraries of modules as per the project requirement.

Why the name spring is given for spring framework?

Ans) The creator liked spring season of his nepal trip ...so he named framework as spring boot

Spring core module
==========
=>Base module for other other modules
=>if this module is used alone ..we can develop only standalone Apps in spring
=> This modules gives Tow spring containers(BeanFactory ,AppicationContext) to perform spring beans life cycle management and Dependency Management

we developing spring Apps in 4 approaches
==========
approach1) Using Xml driven cfgs
approach2) Using Xml + Annotation driven cfgs ✓2
approach3) Using 100% code driven/Java Config cfgs
approach4) Using Spring Boot ✓1

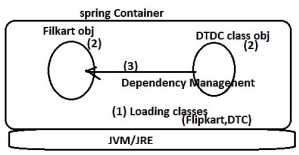
spring bean life cycle management
==========
=>if we given java class to Spring Containers .. it will care of total life cycle of that spring Bean like Loading bean class, creating object ,management object, destorying obejct.

Dependency Management
==========
=> Assigning/Arranging Dependency class object to target class object dynamically is called Dependency Management.

=>The spring bean/class that uses other spring services is called target class /target spring bean /main class
=>The spring bean/class that acts as helper class for other spring beans/classes is called Dependent class/helper class

=>Flipkart need DTDC services for courier activies So flipkart is target class ,DTDC is dependent class.
=>Vehicle needs Engine services for moving/driving activies So Vechile is target class ,Engine is dependent class.
=>Student needs CourseMaterial services for prepration activies So Student is target class ,CourseMaterial is dependent class.

=>if we given target and dependent class to spring containers.. it not only manages their life cycle.. it also assingns /arranges to Depedent class obj to target class obejct..So that target class can use dependent class services directly with out arranging it seperately..



(2) creating objects for spring bean clases

(3) assigning depenent class object to target class obj

(1)+(2)+(3) :: spring container performing spring bean life cycle management and Dependency management.

Dependency Mgmt

Spring core module

=>Base module for other other modules
=>if this module is used alone ..we can develop only standalone Apps in spring
=> This modules gives Tow spring containers(BeanFactory ,AppicationContext) to perform spring beans life cycle management and Dependency Management

spring bean life cycle management

=>if we given java class to Spring Containers .. it will care of total life cycle of that spring Bean like Loading bean class, creating object ,management object, destorying obeject.

we developing spring Apps in 4 approaches

approach1) Using Xml driven cfgs
approach2) Using Xml + Annotation driven cfgs ✓2
approach3) Using 100% code driven/Java Config cfgs
approach4) Using Spring Boot 1
(Approach 3+ AutoConfiguration)

(Based on the jar files added to the project certain pre-defined classes automatically become spring beans and also Dependency Management)

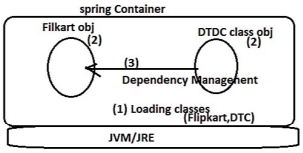
Dependency Management

=> Assigning/Arranging Dependency class object to target class object dynamically is called Dependency Management.

=>The spring bean/class that uses other spring services is called target class /target spring bean /main class
=>The spring bean/class that acts as helper class for other spring beans/classes is called Dependent class/helper class

=>Flipkart need DTDC services for courier activies So flipkart is target class ,DTDC is dependent class.
=>Vehicle needs Engine services for moving/driving activies So Vechile is target class ,Engine is dependent class.
=>Student needs CourseMaterial services for prepration activies So Student is target class ,CourseMaterial is dependent class.

=>if we given target and dependent class to spring containers.. it not only manages their life cycle.. it also assingns /arranges to Depedent class obj to target class obejct..So that target class can use dependent class services directly with out arranging it seperately..



(2) creating objects for spring bean clases
(3) assigning depenent class object to target class obj

(1)+(2)+(3) :: spring container performing spring bean life cycle management and Dependency management.

Dependency Management

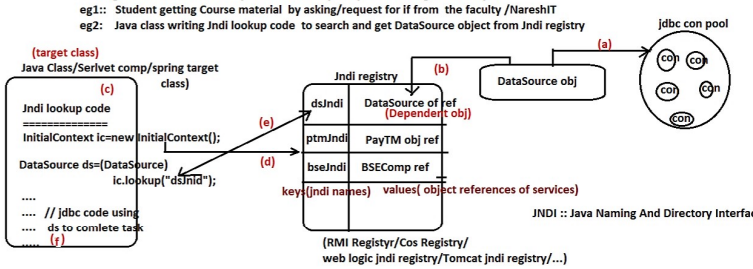
=> It is all about keeping Dependent class object ready for target class object ,So the target class can the use the services of dependent class while executing its own services.

Two types Dependency Management (is called as IOC :: Inversion of Control)

1.Dependency Lookup
2.Dependency Injection

1.Dependency Lookup

=> Here the target calss should spend time and should write some logic to search and get Dependent class obj from various resources/places.
=> target class searches for Dependent class obj and pulls it to target class object
eg1:: Student getting Course material by asking/request for if from the faculty /NareshIT
eg2: Java class writing Jndi lookup code to search and get DataSource object from Jndi registry



JNDI :: Java Naming And Directory Interface.

=>To provide global visibility and accessiblity to any java object we keep its reference in the Jndi registry
=> In Jndi registry we can keep only java objects references
=> The process searching and getting object references from Jndi registry is called Jndi lookup operation
eg:: Cos registry, RMI registry, weblogic registry and etc..
note:: InitialContext object establisht the connection b/w Java App and Jndi registry and provides base to perform lookup operation.

pros of Dependency Lookup

=> The target class can serach and get only required dependent class objects

cons

=>The target class should spend time and should write logic to search and gets Dependent class objs.

2.Dependency Injection

=> HEre the underlying container/server /framework/ JVM/JRE /another service dynamically assigns Dependent class object to target class obejct at runtime.

=>HEre the undelrying server/container/.... pushes dependent class obj to target class object

eg1:: Student getting Course material dynamically facutly the moment he joins the course
eg2:: The way JVM assings default values(dependent values) to object (like 0,0.0,null and etc..) the moment JVM creates the obejct of java class (target class)
eg3:: The way Servlet Container assigns Serlvetconfig object to Servlet class object that moment it creates Serlvet class obj

pros

=>Target class can directly use depedent class object and its services
=> Target class need not to spend time or need not write logics to search and get dependent class obj becoz the underlying container/server /.. takes cares of this work

cons

=> The underlying server/container/... may inject both neccessary and unnecessary objects.

=>spring containers support both Dependency lookup and Depedency Injection.. but realtime projects we use depedency injection.

Depedency Management/IOC in spring

1.Dependecny Lookup
2.Depedency Injection

a) setter Injection
b) constructor Injection
c) Aware Injection /Interface injection/Contexual Injection
d)Lookup Method Injection
e) Method Injection/Method Replacer
f) Field Injection (only in annotation driven cfgs)

What is the difference b/w IOC and Depenency Injection?

=>spring containers support both Dependency lookup and Depedency Injection.. but realtime projects we use depedency injection.

Spring Core Dependency Management types

Depedency Management/IOC in spring
==========

1.Dependecny Lookup

2.Depedency Injection
==========

a) setter Injection
b) constructor Injection
c) Aware Injection /Interface injection/Contexual Injection
d)Lookup Method Injection
e) Method Injection/Method Replacer
f) Field Injection (only in annotation driven cfgs)

What is the difference b/w IOC and Depenency Injection?

IOC : Inversion of Control

=>IOC is specification providing rules and guidelines to manage dependency between target and dependent classes .. It is like a Plan.

=>Dependency lookup and Dependency Injection are two implementation models of IOC /Dependency Management (It is like execution of a plan)

Spring Container are designed to perform Dependency Management /IOC .. So they are also called as IOC containers..

Why it is named as IOC ?
(or)
Why Dependency Management is named as IOC?

=>IOC means Inversion of Control i.e reversing the control/process that is actually happening..
=>Generally programmers hold control to assign /arrange depenedency class object to target class obj.. In IOC implementations that control/power is totally taken by the underlying server/container/framework/JVM.. i.e it is reverse/inversion of the regular process ,So it is named as IOC (given by Martin Flower)

=>Spring Containers/IOC container are light weight containers becoz
a) They can created in any kind java code by just instantiating one pre-defined java class given spring API (spring framework packages)
b) Spring Container/IOC container can be created directly on top of JVM i.e no need of having any heavy webServers(Tomcat) and Application servers(widlfy)

note: Spring containers are not alternate for servlet container, jsp Contaienrs
note:: In Xml driven cfgs based spring App development .. we need to pass spring bean cfg file(xml file) as input file while creating spring Cotnainer/IOC contaienrs..

Creating BeanFactory container in spring App
==========

=> By creating object for a class implemeting BeanFactory(I) .. we can create BeanFactory Container.. spring api provides multiple classes implementing this BeanFactory Interface.. One of them is "XmlBeanFactory" class.

org.springframework.beans.factory

Interface BeanFactory

All Known Subinterfaces:

ApplicationContext, AutowireCapableBeanFactory, ConfigurableApplicationContext, ConfigurableBeanFactory, ConfigurableListableBeanFactory, ConfigurableWebA... calBeanFactory, ListableBeanFactory, WebApplicationContext

Interface in org.springframework.context

All Known Implementing Classes:

AbstractApplicationContext, AbstractAutowireCapableBeanFactory, AbstractBeanFactory, AbstractRefreshableApplicationContext, AbstractRefreshableConfigApplicationContext, AbstractRefreshableWebApplicationContext, AbstractXmlApplicationContext, AnnotationConfigApplicationContext, AnnotationConfigWebApplicationContext, ClassPathXmlApplicationContext, DefaultListableBeanFactory, FileSystemXmlApplicationContext, GenericApplicationContext, GenericGroovyApplicationContext, GenericWebApplicationContext, GenericXmlApplicationContext, GroovyWebApplicationContext, ResourceAdapterApplicationContext, SimpleJndiBeanFactory, StaticApplicationContext, StaticListableBeanFactory, StaticWebApplicationContext, XmlBeanFactory, XmlWebApplicationContext

```
//Locate and hold Spring bean cfg file
FileSystemResource res=new FileSystemResource("...../applicationContext.xml");
   This class internally uses
     java.io.File class to locate
     hold
   and spring bean cfg name and location

//create BeanFactory IOC container
XmlBeanFactory factory=new XmlBeanFactory(res);
```

=>The above code can be placed in any kind of Java App to create BeanFactory contnainer

=>any <filename>.xml can be given as spring bean cfg file .but recomanded to take applicationContext.xml as the spring bean cfg file as the spring Container name "ApplicationContext".
=>spring bean cfg file contains the following cfgs (instructions) to Spring containers/IOC container

a)spring bean cfgs
b)depenency management cfg
c) spring bean life cycle cfgs
d) spring bean ids alia names cfgs
and etc..

if code is placed in standalone App

Standalone Java App
BeanFactory Container
applicationContext.xml
data..
JRE/JVM

(we use this model in spring's standalone Apps)

if code in placed in servlet comp of web application deployed Tomcat server

Tomcat server
Servlet Container
BeanFactory container
applicationContext.xml
data...
JRE/JVM

(we use this model in spring MVC, Spring Rest and MicroServices env..)

---

setter Injection -- Dependency Injection
==========

=> if the Spring Container uses setter method of target class to inject/assign Dependent class object to target class object then it is called setter Injection..
=>For this we need to given instructions to IOC container using <property> in springBean cfg file (applicationContet.xml) and we should also place setter method in target class.

com.nt.beans.WishMessageGenerator (target class) → java.util.Date (dependent class)

This class uses in the injected Date class obj having system date and time to get current hour of the day to generate wish message based on the current of the day.

WishMessageGenerator.java (target class)
==========

```
package com.nt.beans;
      import.java.util.*;
public class WishMessageGenerator{
      //HAS- A property (Composition)
     private Date date;

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

    //b.method
    public String generateMessage(String user){
     //get current of the day
       int hour=date.getHour(); // current hour of day in 24 hours format (0 to 23)

     if(hour<12)
       return "GM:"+user;
     else if(hour<16)
       return "GA:"+user;
     else if(hour<20)
        return "GE:"+user;
     else
       return "GN:"+user;
     }
}
```

applicationContext.xml (com/nt/cfgs)

```
<beans....>
    <!-- spring beans cfgs -->
   <bean id="dt"  class="java.util.Date"/>
          bean Id   fully qualified  class name
   <bean id="wmg"   class="com.nt.beans.WishMessageGenerator">
        <property name="date"  ref="dt"/>
   </bean>
              property name   dependent class bean id to injection depedent sprjng bean class obj to target clas sobj
</beans>
```

Dependent clss
for setter injection
target class

Client App (SetterInjectTest .java)
==========

```
package com.nt.test;
   ...  ...  //pkg imports

public class SetterInjection{
    p  s  v main(String args[]){
```

Spring Core Module 1st app development

setter Injection -- Dependency Injection
==========

=> if the Spring Container uses setter method of target class to inject/assign Dependent class object to target class object then it is called setter Injection..
=>For this we need to given instructions to IOC container using <property> in springBean cfg file (applicationContet.xml) and we should also place setter method in target class.

com.nt.beans.WishMessageGenerator (target class) → java.util.Date (dependent class)

This class uses in the injected Date class obj having system date and time to get current hour of the day to generate wish message based on the current of the day.

WishMessageGenerator.java (target class)
==========

```
package com.nt.beans;
    import.java.util.*;
public class WishMessageGenerator{
     //HAS- A property (Composition)
    private Date date;

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

    //b.method
    public String generateMessage(String user){
     //get current of the day
       int hour=date.getHour(); // current hour of day in 24 hours format (0 to 23)

     if(hour<12)
       return "GM:"+user;
     else if(hour<16)
       return "GA:"+user;
     else if(hour<20)
        return "GE:"+user;
     else
       return "GN:"+user;
      }

}
```

applicationContext.xml (com/nt/cfgs) (spring bean cfg file)

```
<beans....>
   <!-- spring beans cfgs --> (7)
   <bean id="dt"  class="java.util.Date"/>
          bean id        fully qualified  class name
                                                          Dependent clss
   <bean id="wmg"  class="com.nt.beans.WishMessageGenerator">   target class
(5)  <property name="date"  ref="dt"/>   for setter injection
   </bean>
          (8) property name   (6) dependent class bean id to injection depedent sprjng bean class obj to target clas sobj
</beans>
```

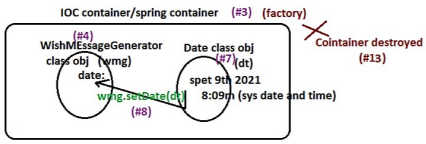
(4)

=>Spring Container /IOC container creates spring Bean class object with bean id (takes bean id as the object name /refereence variable name)..

=>Spring Cotnainer identities spring bean with its bean id if want to get springbean class from spring container and if want to inject one spring bean class object to another spring bean class object we need to use bean ids.

Client App (SetterInjectTest .java)
==========

```
package com.nt.test;                (0) Run the App
   ...  ...  //pkg imports
public class SetterInjection{        (1)
   p  s  v main(String args[]){

     //Locate and hold spring cfg  file name and location
      FileSystemResource res=new FileSystemResource(".../com/nt/cfgs/applicationContetx.xml");
(2)  //create IOC Cotainer
     XmlBeanFactory factory=new XmlBeanFactory(res);

     //Ask SpringContainer/IOC container to give Target spring bean class obj
(9)    Object obj=factory.getBean("wmg");   (3)

      //invoke b.method
      //   String result=obj.geneateMessage("Raja");
              (if Super class ref variable is pointing to sub class obj. then
              can not call direct methods of sub class .. So the above throws
              compile time error.. To make it possbile use downcasting/typecasting)
       //type casting
                                  (10)
       WishMessageGenerator generator=(WishMessageGenerator)obj;

        //invoke b.method
                                  (11)
           String result= generator.generateMssage("Raja");

        System.out.println("Wish MEssage is::"+result);   (12)

      }  (13)
  }
```



w.r.to code
==========

3) Client App ccall factory.getBean(-) method
4) IOC cintaienr searches for sprigng bean cfg whose gean bean id is "wmg" and loads class and creates the object
5 ,& 6) Based <property name="date" ref="dt"> IOC container becomes ready to perform setter injection "date" property
7) takes ref="dt" and seraches for spring bean class cfg with bean id is "dt" and finds java.util.Date class, So Loads and creates the object
8) Spring container creates calls wmg.setData(dt) method to complete setter injection
9) factory.getBean("wmg") returns WIshMssageGenerator class obj having Date class obj in it back to Client App and we are refering that object with java.lang. Object class ref variable.

Setup up required for 1s tApplication development
==========

a) Jdk any version (jdk1.8+) (jdk 16)
b) Eclipse IDE (2019+) (eclipse 2021-06)
|—>contains built-in maven support

download eclipse (as zip file)::
https://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/2021-06/R/eclipse-jee-2021-06-R-win32-x86_64.zip

---

Procedure to devlope spring First App (setter Injection using Eclipse IDE)
==========

step1) Lanuch eclipse IDE by choosisng workspace folder name and location. (G:\Worskpaces\Spring\NTSPBMS615)
The Folder where projects of eclipse IDE will be saved..

step2)

maven /gradle —>Build tools Simplifying all aspects project devleopment, executing ,Testing, release and etc..

=>Building the Project/App is nothing keeping the Project/App ready for execution /release.

maven/gradle features
==========

a) gives standard project directory strcutures (archetypes) for different types apps
b) Allows to donwload jar files/libraries/Dependencies dynamically to projects from Internet reposorities
c) Gives built-in decompilers to see the source code of api.
d) Allows to pack the code in different formats like jar file ,war file and etc..
e) Can run JunitTest cases and can generate Test report
f) if we add main jar files/libraries ..it automatically downloads dependent jars/libaries..
and etc..

maven /gradle -->Build tools Simplifying all aspects project devleopment,
executing ,Testing, release and etc..

=>Building the Project/App is nothing keeping the Project/App ready for execution /release.

Spring Setter Injection exaxmple App

maven/gradle features
=====================
a) gives standard project directory strcutures (archetypes) for different types apps
b) Allows to donwload jar files/libraries/Dependencies dynamically to projects from Internet reposorities
c) Gives built-in decompilers to see the source code of api.
d) Allows to pack the code in different formats like jar file ,war file and etc..
e) Can run JunitTest cases and can generate Test report
f) If we add main jar files/libraries ..it automatically downloads dependent jars/libaries..
and etc..

Setup up required for 1s tApplication development
==========
a) Jdk any version (jdk1.8+) (jdk 16)
b) Eclipse IDE (2019+) (eclipse 2021-06)
|--->contains built-in maven support

download eclipse (as zip file):: https://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/2021-06/R/eclipse-jee-2021-06-R-win32-x86_64.zip

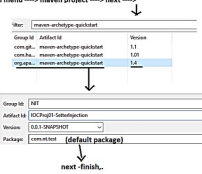
Procedure to devlope spring First App (setter injection using Eclipse IDE)
==========

step1) Lanuch eclipse IDE by choosisng workspace folder name and location. (G:\Worskpaces\Spring\NTSPBMS615)
The Folder where projects of eclipse IDE will be saved..

step2) create Maven Project by taking "maven-archetype-quickstart" as the archetype (Project Template for standalone App)

=>maven is providing 2000+ ready made archetypes as readymade Project templates .. the popular archetypes are
a) maven-archetype-quickstart ----> for standalone Apps
b) maven-archetype-webapp ------> for web applications/ Distributed Apps

File menu ----> maven project ----> next ---->



next -finish,.

In Maven Project/jar file/plugin are identified with 3 details

group id :: Company name
artifactid :: Project name/jar file name /plugin name
version :: Projection version/ jar file version/ plugin version

step3) chanage java version of the project to latest version
in pom.xml

```
16
<maven.compiler.source>1.7</maven.compiler.source>
<maven.compiler.target>1.7</maven.compiler.target>
16
```

POM : Project Object Model

Right click on project ---->maven --update.

step4) Add the following <depedency>tag in pom.xml file to download spring core module jar file to the classpath of project (build path)

=>maven dependencies means the libariries nothing but jar files

In <dependencies> section of pom.xml file
==========

```
<!-- https://mvnrepository.com/artifact/org.springframework/spring-context-support -->
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context-support</artifactId>
  <version>5.3.9</version>
</dependency>
```

save

To collect this tags go to mvnrepository.com ---> search for spring context support--> select version (5.3.9) ---> go to maven tab ---> copy xml code ---> paste in <dependencies> section of pom.xml

=>This code not only gvies spring-context-suppot-5.3.9.jar file and also gives its dependant jar files.

By default every Project these two jar files.. main jar file

dependent jar file

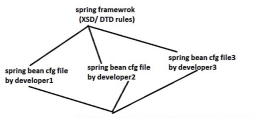
step5) Understand The directory stucture of the Project?

Programmer's testing on his own piece code is called unit testing. we can do that unit testing by using the tool called junit.

step6) Develope the packages having source code in "src/main/java" filder..

=>To make all spring pgogrammers developing spring bean cfg file by fllowing same set of rules and guidelines (i.e same tags and and attributes) we need to import DTD /XSD rules given by spring framework in spring bean cfg file..

DTD :: Document Type Definitation (old)
XSD :: Xml schema Definitation (new)



All these developers will devleop spring bean cfg files by using tags and attributes specified in XSD /DTD rules..

//WishMessageGenerator.java
==========

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

import java.util.Date;

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

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

	//setter method for setter Injection (Alt+shift+s , r -->select setDate(-) ---> ok)
	public void setDate(Date date) {
		System.out.println("WishMessageGenerator.setDate(-)");
		this.date = date;
	}

	//B.method
	public String generateMessage(String user) {
		System.out.println("WishMessageGenerator.generateMessage()");
		//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;
		else
			return "Good Night::"+user;
	}//method
}//class
```

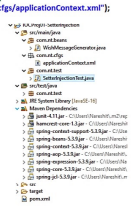
applicationContext.xml (right click on com.nt.cfgs -->new ---> file -->search xml file ---> xml from template --> ....

Collect it from Internet by searching for same spring 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"
  xsi:schemaLocation="
    http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

    <!-- spring bean cfgs -->
    <bean id="dt" class="java.util.Date"/>

    <bean id="wmg" class="com.nt.beans.WishMessageGenerator">
      <property name="date" ref="dt"/> <!-- Setter Injection -->
    </bean>

</beans>
```

SetterInjectionTest.java

```
package com.nt.test;

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

import com.nt.beans.WishMessageGenerator;

public class SetterInjectionTest {

	public static void main(String[] args) {
		//Hold spring bean cfg file name and location (ctrl+shift+o :: To import pkg)
		FileSystemResource res=new FileSystemResource("src/main/java/com/nt/cfgs/applicationContext.xml");
		//create IOC containe (BeanFactory container)
		XmlBeanFactory factory=new XmlBeanFactory(res);
		//get Target spring bean class object
		Object obj=factory.getBean("wmg");
		//typecasting
		WishMessageGenerator generator=(WishMessageGenerator)obj;
		//invoke the b.method
		String result=generator.generateMessage("raja");
		System.out.println("Wish Message is ::"+result);

	}//main
}//class
```



step7) Run the Client App

Go go Client App's .java file ---> run as---> Java App

(or)

ctrl +F11

Xml file /document Syntax rules are

Sep 13 First App internals

a) xml tags and attributes are case-sensitive
b) xml docs are strictly typed docs (rules must be followed)
c) every Open tag should have closing tag
d) Tags must be nested in proper order (In which order they are opened .. they must be closed in reverse order)
e) Attribute values must be quoted (must be there either in single quotes or double quotes)
f) The First tag in xml is called root tag and once is closed .. we should not place any other content in xml file.

What is main difference b/w HTML and XML?

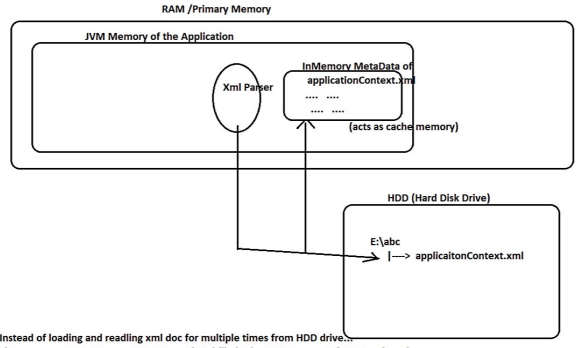
Ans) HTML is given to display to data/documents by applying styles where as XML is given to construct Data by describing it.
=>HTML is loosely typed .. where as XML is strictly typed.
=>HTML tags and attributes are pre-defined.. where XML tags and attributes are user-defined.

=> if Xml doc /file is statisfying its xml syntax rules then is called "well-formed" xml document.
=> if Xml doc /file is statisfying the imported DTD/XSD rules then is called "Valid-formed" xml document.

=> We can Xml Parser (software program) to check wheather given XML doc is well-formed or not , valid or not and also to read and process the xml documents.

Java based Xml parsers are
a) SAX parser (Simple API for xml processing)
b) DOM Parser (Document Object Model)
c) JDOM parser (Java Document Object Model)
d) DOM4J Parser (DOM for Java)
and etc..

=>java Xml parser loads the given XML doc, checks wellformed, valid or not , reads the document and prepares In Memory Metadata in the memory (JVM memory)where the current Application is of RAM is running..

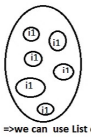


=> Instead of loading and readling xml doc for multiple times from HDD drive.. it is better to create InMemory MetaData of Xml file in the JVM memory of RAM and use its content for multiple times for better performence..

=> Spring Containers/IOC containers are having built-in Xml parser called SAX Parser.. So Spring containers also prepares InMemory MetaData of spring bean cfg file and uses that content for multiple times as discussed above.

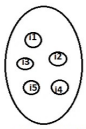
What is the difference b/w pool and cache?

Pool gives reusability of same items



=>we can use List collection to create our own pool

cache gives reusability of different items



=>we can use Map Collection to create our own cache ..

Problems with "new" operator

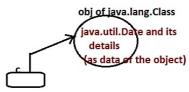
=> Test t=new Test();
=> "new" operator creates the object of java class at runtime.. but expects the presence of java class from compile time onwords i.e we can not use new operator to create object java class whose class name comes to our App dynamically at runtime either through xml file or properites file or cmd line args and etc.. For this we need to "newIsntance()" method of relfection API

note: spring container also uses "reflection API" newInstance() method to create spring bean class objs becoz spring cotnainer gets the class names of spring bean from spring bean cfg file dynamically at runtime.

Relfection API newInstance() method code

a) Load java class into App dynamically at runtime

```
Class c=Class.forName(args[0]);
```
assume the class name is "java.util.Date"



b) create the object of Loaded class using "c".

```
Object obj=c.newInstance();
```
creates the object for the Loaded class (java.util.Date) using 0-param constructor Internally..

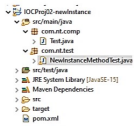
java.lang.Class is pre-defined class and forNAme(-) is static method of java.lang.Class

from Java 11 newInstance() of java.lang.Class is deprecated and recomading to use newIstance() of java.lang.reflect.Consturctor

```
System.out.println(obj.toString());
```

example App

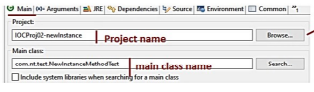
```
//Test.java
package com.nt.comp;

public class Test {
	private int a=10;
	private String b="hello";

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

	//alt+shift+s , s
	@Override
	public String toString() {
		return "Test [a=" + a + ", b=" + b + "]";
	}
}
```

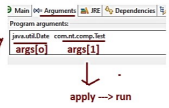



```
//NewInstanceMethodTest.java
package com.nt.test;

public class NewInstanceMethodTest {

	public static void main(String[] args) throws Exception{
		//Load classes
		Class c1=Class.forName(args[0]);
		Class c2=Class.forName(args[1]);

		//create the objects Loaded classes
		Object obj1=c1.newInstance();
		Object obj2=c2.newInstance();

		System.out.println(obj1.toString());
		System.out.println(obj2.toString());


	}

}
```

Passing cmdline arg for eclipse Java App

Right clik on client App --->Runs as ---> run configurrations ---->





apply ---> run

Working with newInstance() method of java.lang.reflect.Constructor class for dynamic object creation

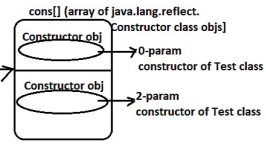
Sep 16 Reflection Api

=>This method can use either 0-param constructor or parameterized constuctor for object creation where as the newInstance() method of java.lang.Class can use only 0-param constructor for object creation.

sample code on newInstance() of java.lang.reflect.Constructor class

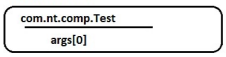
```
//Test.java
package com.nt.comp;

public class Test {
        private int a=10;
        private String b="hello";


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

        public Test(int a,String b) {
                System.out.println("Test:: 2-param constructor");
                this.a=a;
                this.b=b;
        }



        //alt+shift+s , s
        @Override
        public String toString() {
                return "Test [a=" + a + ", b=" + b + "]";
        }
}
```

```
// NewInstanceMethodTest1.java
package com.nt.test;

import java.lang.reflect.Constructor;

public class NewInstanceMethodTest1 {

        public static void main(String[] args) throws Exception{
                //Load classes
                Class c1=Class.forName(args[0]);
                //get All constructor of the Loaded class
                Constructor cons[]=c1.getDeclaredConstructors();
                //Dynamic object using 0-param constructor
                Object obj1=cons[0].newInstance();
                System.out.println(obj1);
                System.out.println("......................");
                //Dynamic object using 2-param constructor
                Object obj2=cons[1].newInstance(100,"india");
                System.out.println(obj2);
        }

}
```



[Elements in this array represents gives Test class constructors in order they are present in the source code "Test" class]

While running code ..Run As -->Run cfgs --->arggguments tab



-->apply ---> run.

First spring App Internal Flow (Complte flow end to End)

a) Enduser runs the Client App and man(-) method executes

b) //Hold spring bean cfg file name and location (ctrl+shift+o :: To import pkg)
FileSystemResource res=new FileSystemResource("src/main/java/com/nt/cfgs/applicationContext.xml");

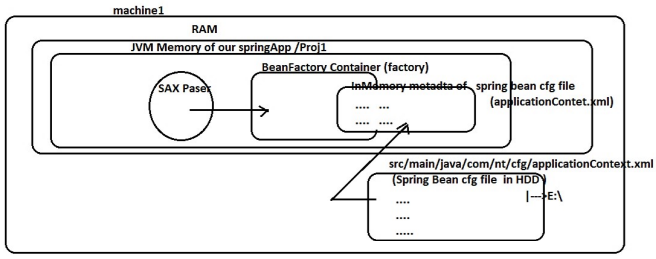
=>Internally uses java.io.File class to hold the name and location of given spring cfg file.. It does not actullay locate the given file.. it just holds name and location of the file.

c) //create IOC container (BeanFactory container)
XmlBeanFactory factory=new XmlBeanFactory(res);

these 3 operations are done by Xml parser of IOC container

i) takes given "res" object (FileSystemResource obj) .. to get the name and location spring bean cfg file
ii) Loads the spring bean cfg file (applicationContext.xml) and wheater it is well-formed or not and valid or not .. if not thorws exception. (checks)
iii) if the spring bean cfg file is well-formed and valid .. then InMemory Metadata of spring bean cfg file will be created in the JVM Memory of the RAM where the spring app is executing.

iv) create IOC container/spring container of type BeanFactory having the above created InMemory MetaData of spring bean cfg file and returns XmlBeanFactory object ref (factory) representing IOC container



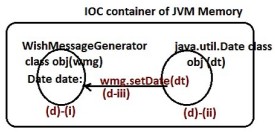
d) Object obj=factory.getBean("wmg");

i) factory.getBean("wmg") method called on IOC container object (factory) ,makes the IOC container to search "wmg" bean id spring bean class cfg in the InMemory MetaData of spring cfg file(applicationContext.xml file's InMemory MetaData) and finds com.nt.beans.WishMessageGenerator having setter injection cfg (becoz of <property>).. So its loads the class cretes the object using reflection api.

```
//Loading class
Class c1=Class.forName("com.nt.beans.WishMessageGenerator");
//get all constructors
Constructor cons[]=c1.getDelcaredConstructors();
//create dynamic object
  Object obj=cons[0].newInstance();
  WishMEssageGenerator wmg=(WisMessageGenerator)obj;
```

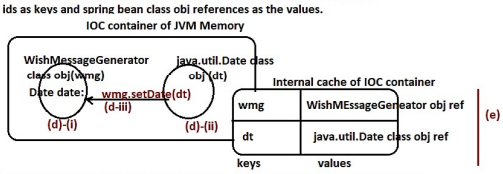
ii) Goes to <property> tag ref="dt" attribute and searches for the spring bean class cfg whose bean id is "dt" in the InMemory MetaData of applicationContext.xml and finds "java.util.Date" class cfg. Notices that no injects are configured ,So loads the class and creates the object using reflect api.

```
//Loading class
Class c2=Class.forName("java.util.Date");
//get all consturctors
  Constructor cons1[]=c2.getDeclaredConstructors();
// create dynamic object
   Object obj2=cons1[0].newInstance();
   java.util.Date dt=(java.util.Date) obj2;
```



iii) completes the setter Injection by calling wmg.setDate(dt) method based <property name="date" ref="dt"> tag of InMemory MEtadata of spring bean cfg file (setter Injection)
(target obj, dependent obj)

e) Spring Container /IOC container keeps spring bean calss objects in the Internal cache of IOC container for reusability having bean ids as keys and spring bean class obj references as the values.



f) Client App gets WishMEssageGeneator class obj ref as java.lang.Object class ref and it uses it for typcasating .

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

g) invocation of b.method getting the result and display the results..

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

h) End of the main(-) ..ends the app's eexeuction by vanising the objects .. In that process the IOC contianer(factory) and its spring bean objets (wmg,dt) , its InMemory MetaData and , Xml praser , Intenal cache and etc.. will also be vanished.

Sep 17 constructor Injection

constructor Injection

=>If the spring container/IOC container is using paramterized constructor to create spring bean class obj and also inject dependents then it called constructor injection.
=>For this we need to place <constructor-arg> tags under <bean> tag where we cfg java class as the spring bean.
=> if we place "n" <constructor-arg> tags under <bean> tag then then the spring container looks for n-param consturctor for consturctor injection ,For example if we place <constructor-arg> tags for 3 times then it looks for 3 param consturctor for spring Bean class instantation and injection.

=> spring container performs setter injection after creating spring bean class object , it seperately calls setter methhod for setter injection

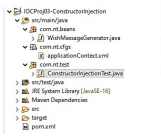
injection

=> spring container performs constructor while instantiating the spring bean class using parameterized constructor.

on

Example App constructor Injection

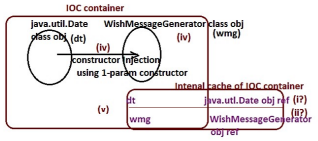
To disable block commenting in eclipse
window menu-->preferences --> java...code style..formatter..new....give name --> disable block commenting.




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

import java.util.Date;

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


      // for constructor injection
      public WishMessageGenerator(Date date) {
            System.out.println("WishMessageGenerator:1-param constructor");
            this.date = date;
      }


      //B.method
      public String generateMessage(String user) {
            System.out.println("WishMessageGenerator.generateMessage()");
            //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;
             else
                  return "Good Night::"+user;
      }//method
}//class
```

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">

      <!-- spring bean cfgs --> (ii?)
      <bean id="dt" class="java.util.Date"/> <!-- dependent class -->
                                  (i?)
      <bean id="wmg"  class="com.nt.beans.WishMessageGenerator"> <!-- target class -->
(i?)    <constructor-arg name="date" ref="dt"/> <!-- constructor injection -->
      </bean>
                                  (ii)

</beans>
```




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

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

import com.nt.beans.WishMessageGenerator;

public class ConstructorInjectionTest {

      public static void main(String[] args) {
            //Hold spring bean cfg file name and location  (ctrl+shift+o :: To import pkg)
            FileSystemResource res=new FileSystemResource("src/main/java/com/nt/cfgs/applicationContext.xml");
            //create IOC container (BeanFactory container)
            XmlBeanFactory factory=new XmlBeanFactory(res);
            //get Target spring bean class object
(vi)        Object obj=factory.getBean("wmg");  (i)
            //typecasting+
            WishMessageGenerator generator=(WishMessageGenerator)obj;
            //invoke the b.method
            String result=generator.generateMessage("raja");
            System.out.println("Wish Message is ::"+result);

      }//main
}//class
```

when factory.getBean("wmg") is called in Constructor Injection

i) Takes given bean id "wmg" , check in the internal cache of IOC container for Object ref availability .. since not not available it goes to InMemory MetaData of spring bean cfg file (applicationContext.xml) and finds "com.nt.beans.WishMEssageGenerator" class as spring bean and notices constructor injection cfg using <constructor-arg> tag.

note:: In setter Injection model, the IOC container first creates target class obj then creates Dependent class obj later assigns dependent class object to target class object by setter(-) on target class object
In constructor Injection model, the IOC container first creates depedent class obj and use that Dependent class obj as the constructor arg value while creating target class object.

ii) Since the enabled injection is constuctor Injection , it takes ref="dt" from <constructor-arg> tag and searches "dt" bean id spring bean class obj ref in the Internal Cache. since not available the IOC container searches in the InMemory MetaData of applicationContext.xml for spring bean cfg whose bean id "dt" and finds "java.util.Date" class with out having any injections cfg.

iii) IOC container loads "java.util.Date" class and creates the object for it using 0-param constructor and reflection API.

```
//Load the spring bean class
Class c1=Class.forName("java.util.Date");
//get all the constructors the loaded spring bean class
Constructor cons[]=c1.getDeclaredConstructors();
//Dynamic object creation
   Date dt=(Date)cons[0].newInstance();
```

iv) Completes constructor injection on target class by loading and instantiating target class (WishMessageGenerator) using 1-param constructor through reflection API as shown below

```
//Load class
Class c2=Class.forName("com.nt.beans.WishMessageGenerator");
//get All constructors of the loaded class
Constructor cons[]=c2.getDeclaredConstructors();
//Dynamic object creation.
  WishMessageGenerator generator=(WishMessageGenerator)cons[0].newInstance(dt);
                                                              Object creation and injection using 1-param
                                                              constructor of WishMEssageGenerator class
```

v) IOC container keeps both target and dependent class class obj in the Internal cache of IOC container taking 'bean ids as the keys and bean class obj refs as values..

vi) factory.getBean("wmg") returns "WishMessageGenerator" class obj ref back to Client App as java.lang.Object class ref.

---

if we perform both setter injection and construction injection cfgs on the same bean property of target class with different dependent values .. can u tell which will be taken as the final value?

Ans) since the setter method executes after constructor execution .. so the value injected by constructor Injection will be overridden with the value injected by the setter injection.

sample code
==========

in applicationContext.xml

```
<bean id="dt1"  class="java.util.Date">
      <property name="year" value="110"/> <!-- add 1900 to given year so it becoems 2010 -->
      <property name="month" value="4"/> <!-- 0 to 11-->
      <property name="date" value="20"/> <!-- 1 to 31-->
</bean>

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

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

note:: To inject one spring bean class object to bean property of another spring bean class use "ref" attribute specifying other spring bean id

```
<property name="date" ref="dt1"/>
```

To inject simple or string value to bean property of spring bean class use "value" attribute specifying hardcoded value

```
<property name="date" value="20"/>
```

=> we can cfg one java class as multiple spring beans with diffent bean ids (nothing diffent object names will be created)

```
<bean id="dt1"  class="java.util.Date">
      <property name="year" value="110"/> <!-- add 1900 to given year so it becoems 2010 -->
      <property name="month" value="4"/> <!-- 0 to 11-->
      <property name="date" value="20"/> <!-- 1 to 31-->
</bean>

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

=> IOC container identifies spring beans using its bean ids (object names/ reference variable names pointing spring bean class objects).So they must be unique with in a spring container.

=>The "name" attribute in <property> tag does not actually looks to take spring bean class property name.. It is actually the "xxx" word of setXxx(-) method that is taken for the setter Injection. Generally the xxx word setXxx(-) method matches with bean property name .. if not matched we need give "xxx" word in the name attribute of <property> tag.

Improving the Client App

spring bean class

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

    // for constructor injection
    public WishMessageGenerator(Date dat) {
        System.out.println("WishMessageGenerator:1-param constructor");
        this.date = dat;
    }

    public void setDate1(Date date) {
        System.out.println("WishMessageGenerator.setDate(-)");
        this.date = date;
    }
}
```

In applicationContext.xml

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

<bean id="wmg" class="com.nt.beans.WishMessageGenerator"> <!-- target class -->
    <property name="date" ref="dt"/> wrong
    <property name="date1" ref="dt "/> valid

</bean>
```

=> In the "name" attribute of <constructor-arg> we should give constructor param name .. not the spring bean class property nname.. generlly both names match.. if not match we should give param name not bean property name.

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

    // for constructor injection
    public WishMessageGenerator(Date dat) {
        System.out.println("WishMessageGenerator:1-param constructor");
        this.date = dat;
    }

    .....
}
```

applicationContext.xml

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

<bean id="wmg" class="com.nt.beans.WishMessageGenerator"> <!-- target class -->
  <constructor-arg name="date" ref="dt"/> //invalid
  <constructor-arg name="date1" ref="dt"/> <!-- constructor injection -->
</bean>
```

---

important statatents on spring bean instantiation done spring container

(a) if spring bean class is cfg with out any injections cfg then the IOC container creates that spring bean class object using 0-param constructor

(b) if spring bean class is cfg only having setter injections cfgs, then the IOC container creates that spring bean class object using 0-param constructor

(c) if spring bean class is cfg having constructor injection cfgs, then the IOC container creates that spring bean class object using parameterized constructor.

Q) Can I place only private constructor in spring bean class?

Ans) yes , we can place .. becoz spring container internally uses reflection api to get access to private constructor to perform spring bean instantiation and the necessary constructor injections. (Not a good pratice to keep private constructor in spring bean classes)

note:: we can not private setter methods for setter Iniection

Core Java sample Application to create the objects by accessing private constructor of java class outside of that java class

```
//Test.java
==========
/Test.java
package com.nt.comp;

public class Test {
    private int a=10;
    private String b="hello";


    private Test() {
        System.out.println("Test:: 0-param constructor");
    }

    private Test(int a,String b) {
        System.out.println("Test:: 2-param constructor");
        this.a=a;
        this.b=b;
    }


    //alt+shift+s , s
    @Override
    public String toString() {
        return "Test [a=" + a + ", b=" + b + "]";
    }
}
```

```
//NewInstanceMethodTest1 .java
package com.nt.test;

import java.lang.reflect.Constructor;

public class NewInstanceMethodTest1 {

    public static void main(String[] args) throws Exception{
        //Load classes
        Class c1=Class.forName(args[0]);
        //get All constructor of the Loaded class
        Constructor cons[]=c1.getDeclaredConstructors();
        //Dynamic object using 0-param constructor
        cons[0].setAccessible(true); //gives access to private constructor
        Object obj1=cons[0].newInstance();
        System.out.println(obj1);
        System.out.println(".....");
        //Dynamic object using 2-param constructor
        cons[1].setAccessible(true); //gives access to private constructor
        Object obj2=cons[1].newInstance(100,"india");
        System.out.println(obj2);
    }

}
```

---

we can use either FileSystemResource or ClassPathResource obj to locate and hold spring bean cfg file

What is the difference b/w both?

Ans) FileSystemResource obj makes the spring container to locate the given spring bean cfg file from the specified path of file system (all drives of computer managed by OS)..we can pass either relative path(good) or absolute path of spring bean cfg file

eg:: FileSystemResource res=new FileSystemResource("src/main/java/com/nt/cfgs/applicationContext.xml");
relative path w.r.t Project folder

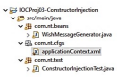
(or)

eg: FileSystemResource res=new FileSystemResource("G:\\Worskpaces\\Spring\\NTSPBMS615\\IOCProj03-ConstructorInjection\\src\\main\\java\\com\\nt\\cfgs\\applicationContext.xml");
absolute path (not a good pratice)

ClassPathResource obj makes the spring container to locate the given spring bean cfg file from the directories and jar files added to CLASSPATH or BuidPATH.

=>In MAVEN/Gradle Project "src/main/java" and "src/test/java" folders will be there in the build path /classpath by default.

eg:: ClassPathResource res=new ClassPathResource("com/nt/cfgs/applicationContext.xml");



(or)

eg: ClassPathResource res=new ClassPathResource("applicationContext.xml");

For this we need to make "com/nt/cfgs" folder as source folder to keep that folder in the buildpath of project.

right click on "com/nt/cfgs" folder ->builpath ->use as the source folder

after moving to classpath

=>In a running java app to hold class name , interface name, annotation name, enum name and their details we can use the object of java.lang.Class (c).

=> To hold numeric values in java app, we use numeric data type variables
=> To hold text values in java app, we use String class objects
=> To hold class name or interface name or annotation name or enum name and their details (metadata) we can take the support of java.lang.Class object. This object does not hold names as Stirng values..it holds them with out loosing the java meanings..

=>The easiest way to create the object of java.lang.Class holding any class/interface/enum/annotation name and details is ".class" property.

```
Class c1= java.util.Date.class;
Class c2=Demo.class;
Class c3=Test.class;
```

Built-in property of java class

object of java.lang.Class

Test class name and its details

c3

```
STring s1="Test";   //Here "Test" text content , so we can only String manipulations
Class c3=Test.class; // Here "Test" treated as java class ..So using c3 we can get all the details
                     "Test" class like super class name, constructor details, method details an etc.. and we use them
                     for different operations/invocations.
```

=>During the compilation of every java class ,the java compiler adds the default property called "class" as the public static property of type java.lang.Class ..

```
Test.java
=========
class Test{
}
```

javac Test.java

Compiler improvised code in Test class during the compilation

```
class Test extends java.lang.Object
  public static Class class;
  public Test(){
    super()
  }
```

gives Test.class file (physical file)

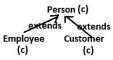
```
Class c1=Test.class;
    Class c2=java.util.Date.class;
    System.out.println("c1 obj data::"+c1);  //gives Test
    System.out.println("c2 obj data::"+c2);  //gives java.util.Date
```

=>Built-in properties in Java class are :: .class , .length
=>Built-in reference variables in Java App are :: this, super
=>Built-in Threads in Java App are :: main, gc
=>Built-in streams in Java App are :: System.in, System.out, System.err

Need of Generics
==========

Problem::
=========

Sep 21 Client App by class path



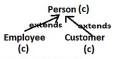

```
public   Person   getDetails(String type){
   if(type.equals("cust"))
      return new Customer();
   else if(type.equals("emp"))
      return new Employee();
   else
     return throw new IllegalArgumentException("Invalid PersonType");
  }
```

Testing code
==========

```
  Person per=getDetails("cust"); //valid
    //using "per" we can call only common of Person and class Customer class.. To call
   //  Direct methods of Customer class we need type casting..
     Customer cust=(Customer)per;
   //usign cust we can directly methods of "Customer"  class
        ....

     Person per=getDetails("emp"); //valid
      //using "per" we can call only common of Person and class Customer class.. To call
     //  Direct methods of Employee class we need type casting..
      Employee emp=(Employee)per;
     //usign emp we can directly methods of "Employee"  class
        ....
```

note:: when we do type casting there is possiblity of getting Type casting related exception that is ClassCast Exception. Code is not type safe ... To make code as type safe code take the support of Generics.

Solution using Genrics
==========




```
public  <T>  T  getDetails(String type, Class<T> clazz){
     if(type.equals("cust"))
       return   clazz.newInstance();
     else if(type.equals("emp"))
       return   clazz.newInstance();
     else
      return throw new IllegalArgumentException("Invalid PersonType");
   }
```

Here can any class bsed java.lang.Class obj

Testing code
==========

```
  Employee e=getDetails("emp",Employee.class);
```

Gives object of java.lang.Class having Employee class details as data

[Here no typecasting is required becoz of generics.. So we can say ClassCastException is avoided and we can say code is type safe code]

```
  Customer  cust=getDetails("cust",Customer.class);
```

More improved Code
==========

```
   public  <T extends Person>  T  getDetails(String type, Class<T> clazz){
        if(type.equals("cust"))
          return   clazz.newInstance();
        else if(type.equals("emp"))
          return   clazz.newInstance();
        else
         return throw new IllegalArgumentException("Invalid PersonType");

    }

  Employee e=getDetails("emp",Employee.class);
```

must be Person and its sub class based java.lang.Class obj

Gives object of java.lang.Class having Employee class details as data

[Here no typecasting is required becoz of generics.. So we can say ClassCastException is avoided and we can say code is type safe code]

```
  Customer  cust=getDetails("cust",Customer.class);
```

Old getBean(-) of BeanFactory (spring 1.0 onwords)

signature

```
public  Object getBean(String beanid)  throws BeansException
```

```
  eg::
    Object obj=factory.getBean("wmg");
     //using obj we can not call direct methods of "WishMessage" Generator class
     //for Go for Type casting
     WishMessageGenerator generator=(WishMessageGenerator)obj;
     //using "generator" we can call direct method WishMessageGenerator class.
       ...
       ..
```

note: Type casting required, So there is possbility of getting ClassCastExceptionException i.e code is not type safe.

Improvised getBean(-,-) method of BeanFactory (Using Generics from spring 2.5)
==========

Signature ::

```
  public  <T>  T  getBean(String bean id , Class<T> clazz)throws BeansException

  eg::
    WishMEssageGenerator generator=factory.getBean("wmg",WishMessageGenerator.class);
       //No Type casting required , So code becomes type safe code..
  eg::
    Date  d=factory.getBean("dt",Date.class);
       //No Type casting required , So code becomes type safe code..
```

passing WishMessageGenerator class as the the object of java.lang.Class

Improved factory.getBean(-,-) in the Client App
==========

```
package com.nt.test;

import java.util.Date;

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

import com.nt.beans.WishMessageGenerator;

public class ConstructorInjectionTest1 {

      public static void main(String[] args) {
              ClassPathResource res=new ClassPathResource("applicationContext.xml");
            //create IOC container (BeanFactory container)
            XmlBeanFactory factory=new XmlBeanFactory(res);
            //get Target spring bean class object
            WishMessageGenerator  generator=factory.getBean("wmg",WishMessageGenerator.class);
            //invoke the b.method
            String result=generator.generateMessage("raja");
            System.out.println("Wish Message is ::"+result);

      }//main
}//class
```

STS (Spring ToolSuite)
===

```
   =>STS is an IDE which exclusively given for  spring/spring Boot apps development.
   =>This IDE is based  Eclipse IDE
   => Instead of STS IDE .. we add  STS plugin to Eclipse ,So we can use
   Eclipse features and STS IDE features..
```

=>Plugin
==========

```
   => A plugin is patch s/w  that provides addtional features on the top of existing features.. in the
  software or  sofware  App.

            Eclipse having two type plugins
              a) Eclipse supplied plugins
                    => Can be added using  help menu => Install new software option
                    eg:: Gui builder, GlassFish Tools,  Jboss tools and etc..
              b) Third party plugins
                   =>Can be added using help menu=>Eclipse market place
                    eg::  STS plugin, sonar cube , jasper soft and etc..

   note:  From eclipse 2019  Gradle, maven , GIT, SVN and etc.. plugins are built-in plugins  in basic
   Eclipse Installation..

  STS plugin features
  ==========
  a) Allows to crete spring bean cfg file  by importing selected XSD names spaces
  b) helps to finish  spring cfg file develpment very fastly
  c) simplifies  spring boot App development by lots of starters..
  d) allows to create spring project and spring boot projects easilty..
  and etc..

  Process to STS Plugin
  ==========
     Help Menu ===>  Eclipse Market place ==> search STS --> go  --->
       selecct 3.9.18  version -->install --->confirm ---> next ----> accept terms conditions --> restart
     eclipse IDE
```

Limitations of creating BeanFactory IOC container using XmlBeanFactory class

Sep 22 DefaultListable BeanFactory & DesignPatterns

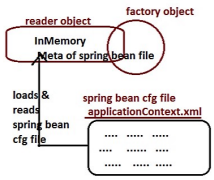
a)XmlBeanFactory class is Deprecated from the version spring 3.1
b)XmlBeanFactory does not allow to working with multiple spring bean cfg files at a time
c) We can not pass spring bean cfg file name and location as string values.. it is expecting them as Resource object (FileSystemResoruce or ClassPathResource obj)
d) The spring Beans given to XmlBeanFactory will be passed to DefaultListableBeanFactory class for registration .. So better to use DefaultListableBeanFactory as directly

```
java.lang.Object
    org.springframework.core.SimpleAliasRegistry
        org.springframework.beans.factory.support.DefaultSingletonBeanRegistry
            org.springframework.beans.factory.support.FactoryBeanRegistrySupport
                org.springframework.beans.factory.support.AbstractBeanFactory
                    org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory
                        org.springframework.beans.factory.support.DefaultListableBeanFactory
                            org.springframework.beans.factory.xml.XmlBeanFactory
```

To overcome the above problems .. prefer using DefaultLisableBeanFactory and XmlBeanDefinitationReader togather to create BeanFactory IOC container.

Code to create BeanFactory Container using DefaultListableBeanFactory and XmlBeanDefinitionReader

```
//create IOC container
DefaultListableBeanFactory factory=new DefaultListableBeanFactory();
XmlBeanDefinitionReader reader=new XmlBeanDefinitionReader(factory);
reader.loadBeanDefinitions("com/nt/cfgs1/applicationContext.xml");
```



Advantages of working DefaultListableBeanFactory

a) Not a Deprecated java class
b) Can take multiple xml files as the spring bean cfg files at a time (var args)
c)we can spring cfg file names and locations as string values. (No need of Resource objs)
d) The java classes become spring beans directly with out passing them to other classes..

DesignPatterns

=> These are set of rules that are given best solutions for reoccuring problems of application/project development.

=>Design patterns helps the developers to use programming languages, technolgoies and and frameworks more effectively in the Application development ..

=> Design Patterns are not part of Project designing .. they are part of Project Implementation (coding)

=>Desing Patterns acts best solutions for solving memory , perfomences , tight coupling related problems..

=> Every Object Oriented Programming language gives support to implement Design Patterns.

Different Types of Desing Patterns

a) GOF Pattterns (total 23 patterns)
=>Can be implemented in any oop language including java
=>Generally these are desing patterns related to standalone Apps devleopment

eg:: singleton class, factory pattern, abstract factory pattern, bridge pattern, strategy pattern and etc..

b) JEE Patterns (25 patterns)
=> Given by Sun Ms
=> useful to address the problems related to JAva Layred Application Development
=>Can be implemented only JAva ,JEE,java frameworks env..

eg:: DAO class , FrontController , ViewHelper ,Composite View and etc..

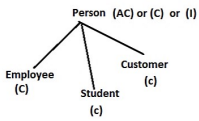
c) Micro services Patterns

=> Can be used while developing applications having micro services architectures
=> addresses problems related to MicroService architecture based application development.
eg: SAGA Pattern, SOLID principles pattern and etc..

Factory Pattern

=> Factory Pattern creates and returns one of several related related classes obj based on the data that is supplied by providing abstraction (hiding the details) on object creation process.

=> related classes means , the classes having common super class or commonly implementing interface.



if Factory pattern is not given .. The Client App fellow should know object creation process entirely.. sometimes it may be more complex becoz one object creation may need multiple other objects as dependents.

eg1:: Car Factory provides abstaction on Car Manifactring
eg2:: DriverManager.getConnection(-,-,-) provides abstraction on Jdbc con object creation based on arg values (jdbcurl, db user, dbpwd) details we supply)

```
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe",
                                           "system","manager");
```
gives oracle Db s/w connectivity related JDBC con obj

(or)

```
Connection con=DriverManager.getConnection("jdbc:mysql:///<logicalDB>","root","root");
```
gives mysql Db s/w connectivity related JDBC con obj

note:: All Jdbc statements that are given to create Statement obj,ResusltSet obj and etc.. are also based Factory Pattern.

```
Statement st=con.createStatement();
ResultSet rs=st.executeQuery("SELECT * FROM STUDENT");
```

eg3: spring containers (both BF ::BeanFactory and AC: ApplicationContext) are given based on factory Pattern.

```
Date dt=factory.getBean("dt",Date.class);

WishMessageGenerator generator=factory.getBean("wmg",WishMEssageGenerator.class);
```

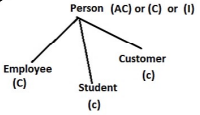
=>Factory pattern implementaton needs factory method .. The method that can returns either its own class object or related class obj or irrelavent class obj called factory method. The factory method used in Factory Pattern generally returns one of several related classes object based on the data /inputs the that are supplied.

The factory method of Factory pattern can static or non-static .. generally we take it is as static method having the common super class or common interface as the return type

```
public class PersonFactory{
    //Factory method having Factory pattern logic
    public static Person getPerson(String type){
        if(type.equalsIgnoreCase("emp"){
            return new Employee();
        else if(type.equalsIgnoreCase("cust"){
            return new Customer();
        else if(type.equalsIgnoreCase("stud"){
            return new Student();
        else
            throw new IllegalArguementExcpetion("invalid person type");
        }
    }
```

Common class or interface as the return type

Clients call method with out object



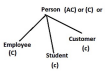
note:: Always assume Factory class and and ClientApps that usisng factory calsses are develped two different programmers..

Sep 23 Factory Pattern

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

=> Factory Pattern creates and returns one of several related related classes obj based on the data that is supplied by providing abstraction (hiding the details) on object creation process.

=> related classes means , the classes having common super class or commonly implementing interface.



If Factory pattern is not given .. The Client App fellow should know object creation process entirely.. sometimes it may more complex becoz one object creation may multiple other objects as dependents.

eg1:: Car Factory provides abstaction on Car Manifactring
eg2:: DriverManager.getConnection(-,-,-) provides abstraction on Jdbc con object creation based on arg values ([jdbcurl, db user, dbpwd]) details we supply)

```
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe",
                                           "system","manager");
```
gives oracle Db s/w connectivity related JDBC con obj

(or)

```
Connection con=DriverManager.getConnection("jdbc:mysql:///<logicalDB>","root","root");
```
gives mysql Db s/w connectivity related JDBC con obj

note:: All Jdbc statements that are given to create Statement obj,ResualtSet obj and etc.. are also based Factory Pattern.

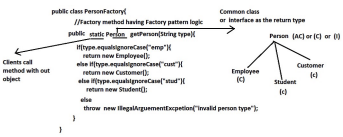
```
Statement st=con.createStatement();
ResultSet rs=st.executeQuery("SELECT * FROM STUDENT");
```

eg3: spring containers (both BF ::BeanFactory and AC: ApplicationContext) are given based on factory Pattern.

```
Date dt=factory.getBean("dt",Date.class);

WishMessageGenerator generator=factory.getBean("wmg",WishMEssageGenerator.class);
```

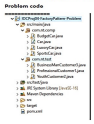
=>Factory pattern Implementaton needs factory method .. The method that can returns either its own class object or related class obj or irrelevant class obj called factory method. The factory method used in Factory Pattern generally returns one of several related classes object based on the data /inputs the that are supplied.

The factory method of Factory pattern can static or non-static .. generally we take It is as static method having the common super class or common interface as the return type



note:: Always assume Factory class and and ClientApps that using factory calsses are developed two different programmers..

Problem code
==========



Car.java
```
package com.nt.comp;

public abstract class Car {
  public abstract void drive();
}
```

```
//BudgetCar.java
package com.nt.comp;

public class BudgetCar extends Car {
  private String regNo;
     public BudgetCar(String regNo) {
          System.out.println("BudgetCar::1-param constructor");
          this.regNo=regNo;
     }

     @Override
     public void drive() {
       System.out.println("Driving Budget Car");

     }
```

```
//LuxoryCar.java
package com.nt.comp;

public class LuxoryCar extends Car {

     private String regNo;
     public LuxoryCar(String regNo) {
          System.out.println("LuxoryCar::1-param constructor");
          this.regNo=regNo;
     }

     @Override
     public void drive() {
       System.out.println("Driving Luxory Car");

     }

}
```

```
////SportsCar.java.java
package com.nt.comp;

public class SportsCar extends Car {
     private String regNo;

     public SportsCar(String regNo) {
          System.out.println("SportsCar::1-param constructor");
          this.regNo=regNo;
     }

     @Override
     public void drive() {
       System.out.println("Driving Sports Car");

     }
}
```

Client Apps

```
//BusinessManCustomer3 .java
package com.nt.test;

import com.nt.comp.BudgetCar;
import com.nt.comp.Car;
import com.nt.comp.LuxoryCar;
import com.nt.comp.SportsCar;

public class BusinessManCustomer3 {
     public static void main(String[] args) {
          Car car=new LuxoryCar("TS11 AB 5151");
          car.drive();

     }

}
```

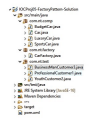
```
//ProfessionalCustomer1.java
package com.nt.test;

import com.nt.comp.BudgetCar;
import com.nt.comp.Car;

public class ProfessionalCustomer1 {
     public static void main(String[] args) {
          Car car=new BudgetCar("TS09EN 5656");
          car.drive();

     }

}
```

```
//YouthCustomer2.java
package com.nt.test;

import com.nt.comp.BudgetCar;
import com.nt.comp.Car;
import com.nt.comp.SportsCar;

public class YouthCustomer2 {
     public static void main(String[] args) {
          Car car=new SportsCar("TS10 RK 5656");
          car.drive();

     }
}
```

HEre the multiple client Apps should know following things
a) How to create objcts for different car classes
b) If need they should know how to create the required depeddent class objs
c) They should good command on the hierarchy of classes

=>To this solve this problem for clients provide then one factory class having factory pattern which takes care of object creation process.. by hiding details from clients..

Solution using Factory Pattern
==========



```
//CarFactory.java
package com.nt.factory;

import com.nt.comp.BudgetCar;
import com.nt.comp.Car;
import com.nt.comp.LuxoryCar;
import com.nt.comp.SportsCar;

public class CarFactory {

     //Factory having  factory pattern logic  (Logic to create and return one of several related classes obj)
     public static  Car  createCar(String type, String regVal) {
               if(type.equalsIgnoreCase("sports"))
                    return new SportsCar(regVal);
               else if(type.equalsIgnoreCase("budget"))
                    return new BudgetCar(regVal);
               else if(type.equalsIgnoreCase("luxury"))
                    return new LuxoryCar(regVal);
               else
                    throw new IllegalArgumentException("Invalid car Type");
     }
}
```

```
//YouthCustomer2.java
package com.nt.test;

import com.nt.comp.BudgetCar;
import com.nt.comp.Car;
import com.nt.comp.SportsCar;
import com.nt.factory.CarFactory;

public class YouthCustomer2 {
     public static void main(String[] args) {
          //get Car obj using factory
          Car car=CarFactory.createCar("sports", "TS08 EN 6688");
          car.drive();
          System.out.println("==================");

     }

}
```

```
//ProfessionalCustomer1.java
package com.nt.test;

import com.nt.comp.BudgetCar;
import com.nt.comp.Car;
import com.nt.factory.CarFactory;

public class ProfessionalCustomer1 {
     public static void main(String[] args) {
          //get Car obj using factory
          Car car=CarFactory.createCar("budget", "TS09 EN 5688");
          car.drive();
          System.out.println("==================");
     }

}
```

```
//BusinessManCustomer3 .java
package com.nt.test;

import com.nt.comp.BudgetCar;
import com.nt.comp.Car;
import com.nt.comp.LuxoryCar;
import com.nt.comp.SportsCar;
import com.nt.factory.CarFactory;

public class BusinessManCustomer3 {
     public static void main(String[] args) {
          //get Car obj using factory
          Car car=CarFactory.createCar("luxury",
"TS10 EN 5688");
          car.drive();
          System.out.println("==================");
     }

}
```

Strategy Pattern
==========

=>Stategy means planing

=> It is a GOF(Gang of Four) Pattern and can be implemented in any oop language
=> It has bocome very popular after the arrival spring framework .. but it is not spring framework pattern.
=> This patterns given set of rules/principles to designing classes having dependency as target and dependent classes.
=> This design pattern says develop the target and dependent classes of Dependency Management as loosely coupled interchangable parts.
(i.e we can change one dependent class to another dependent class with out effecting the the source code target class)

=>If the degree of dependancy is more b/w two comps/classes then they are called tightly coupled classes/comps
eg:: CPU Box and Console/DIsplay Unit
eg:: FAn and switch

=>If the degree of dependency is less b/w two comps/classes then they are called loosely coupled classes/comps
eg:: TV and Remote
eg: Laptop and Mouse

The principles /rules of strategy Pattern
==========

a) Favour Composition over inheritence
b) Code to Interfaces .. Never code to Concrete classes /abstract classes
c) Code must be open for extension and must be closed for modification.

a) Favour Composition over inheritence

Inheritence (IS-A relation)

```
class A{                    class B extends A{
  ....                        ----
   ..                          ...
}                           }
                    A
```
B is sub class of A class

Composition (HAS- Relation)

```
class A{                    class B{
  ...                           private A a=new A();
  ...}                            ...
                                }
```
B has got A class object.

=> If class1 wants to use entire state and behaviour of class2 then keep them in inheritance relationship.
=> If class1 wants to use specific state and behaviour of class2 then keep them in composition relationship.

Limitations of inheritance (while using it in deepndency management)
==========

(i) Few languages do not support inheriting from multiple classes (including java)
(ii) Code becomes fragile (Easy breakable)
(iii) Testing of code becomes very complex.

(i) Few languages do not support inheriting from multiple classes (including java)

problem ::

```
class A{        class B{        class C extends A,B{
  ...             ....            ....
}               }                 ---
                                }
```
Not allowed in most of the OOP languages.

=>Do not look at Interfaces in the angle of inheritence..becoz they does not contain any thing exclusively to inherit.. Always look at them in the angel of polymorphism.

Solution::

```
class A{        class B{        class C{
  ...             ....            private A a=new A();
}               }                 private B b=new B();
                                }
```
Can work with multiple other classes objects using composition.

Strategy Pattern
================

Sep 24 Stategy Pattern

=>Stategy means planing

=> It is a GOF(Gang of Four) Pattern and can be implemented in any oop language
=> It has become very popular after the arrival spring framework .. but it is not spring framework pattern.
=> This patterns given set of rules/principles to designing classes having dependency as target and dependent classes.
=> This design pattern says develop the target and dependent classes of Dependency Management as loosely copuled interchangable parts.
(i.e we can change one dependent class to another dependent class with out effecting the the source code target class)

=>if the degree of dependency is more b/w two comps/classes then they are called tightly coupled classes/comps
eg:: CPU Box and Console/Display Unit
eg:: FAn and switch

=>if the degree of dependency is less b/w two comps/classes then they are called loosely coupled classes/comps
eg:: TV and Remote
eg: Laptop and Mouse

The 3 principles /rules of strategy Pattern
====================================

a) Favour Composition over inheritence
b) Code to interfaces .. Never code to Concrete classes /abstract classes
c) Code must be open for extension and must be closed for modification.

a) Favour Composition over inheritence (strategy Pattern rule no:1)

Inheritence (IS-A relation)

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

B is sub class of A class

Composition (HAS- Relation)

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

B has got A class object.

=> if class1 wants to use entire state and behaviour of class2 then keep them in inheritence relationship.
=> if class1 wants to use specific state and behaviour of class2 then keep them in composition relationship.

Limitations of Inheritence (while using it in deepndency management)
======================

(i) Few languages do not support inheriting from multiple classes (including java)
(ii) Code becomes fragile (Easy breakable)
(iii) Testing of code becomes very complex.

(i) Few languages do not support inheriting from multiple classes (including java)

problem ::

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

Not allowed in most of the OOP languages.

=>Do not look at interfaces in the angle of inheritence..becoz they does not contain any thing exclusively to inherit.. Always look at them in the angel of polymorphism.

Solution::

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

                            }
```

Can work with multiple other classes objects using composition.

(ii) Code becomes fragile (Easy breakable)

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

class B extends A{
   public int m1(){
    ...
    return 1000;
   }
}

class C extends B{
  ....
  ....
}

class D extends B{
  ....
  ..
}
```

problem :: if we modify the return type of m1() in "A" class from int to float the all classes that inheriting from "A" class directly or indirectly will be distrubed..and we can get lots of inconsistency problem.

note:: if the signature of java.lang.Object class methods are modified bit .. then all all java apis will be collapsed ..becoz java.lang.Object is the top most class in the inheritence hierarchy of every java class.

Solution:: Use the composition , then we can restrict the problem to certain 1 or 2 methods.

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

class B{
 private A a=new A();

 public int m1(){
  int x=a.m1();
int   x=Math.round(a.m1());
          (or)
 var x=a.m1(); //from java 10
  ...
  return x+1000;
 }

class C extends B{
  ....
  ...
}
class D extends B{
  ....
  ....
}
```

=>Here B class has the object of A class (composition and no inheritence b/w A and B classes), So if we change the return type of m1() in A class from int to float .. we just need to modify the code in B class or other classes when that m1() of A class is being invoked i.e we can restrict the code change problem propagation to certain method call area .. All classes in the hierarchy will not be distrubed.

=> declarting local variable having type "var" in given from Java 10 ..
In Java 10, the var keyword allows local variable type inference, which means the type for the local variable will be inferred by the compiler based on the value that is assigned.

(iii) Testing of code becomes very complex.

=>Programmer's testing on his own piece of code is called unitTesting.
=> We can use Junit tool to perform UnitTesting through coding

Problem::

```
class A{
 public int m1(){
  ...
  return 100;
 }
 public int m2(){
  ...
  return 200;
 }
}

class B extends A{
  public int m3(){
   ...
   return 300;
  }
  public float m4(){
   ....
   return 45.77;
  }
}

class C extends B{
 ...
 ... //more 3 methods
 ...
}

class D extends C{
 ...
 ... //more 3 methods
 ...
}

B b=new B();
b.m1(); b.m2(); b.m3(), b.m4();

D d=new D();
 //we can invoke total 10 methods on this object.
```

UnitTesting deals with Testcases where expected results will be compared with generated Results.

While performing unit Testing of any sub class .. we need to write test cases for the direct methods of the sub class and also for the inherited method to sub class .. This improves burden on the programmer.

Solution
=======

```
class A{
 public int m1(){
  ...
  return 100;
 }
 public int m2(){
  ...
  return 200;
 }
}

class B{
 private A a=new A();

 public int m3(){
   int x=a.m1();
   return x+200;
 }

 public float m4(){
  return 200.6f;
 }
}

class C {
 ...
 ... //more 3 methods
 ...
}

class D {
 ...
 ... //more 3 methods
 ...
}

B b=new B();
 b.m3();
 b.m4();

D d=new D();
  we can only 3 methods on this object of "D" class.
```

=> Here while doing unit testing on any class.. we just need to write test cases only for that class methods.. So burden on the programmer will be decreased.

b) Code to interfaces .. Never code to Concrete classes /abstract classes (strategy Pattern rule no:2)

problem:: (Tight coupling)
========================

```
//target class
class Flipkat { //HAS-A Property
 private DTDC dtdc =new DTDC();
  private BlueDart bDart =new BlueDart()
public String shopping(
    String item[],float prices[]){
  .....
  ..... //calc b.method
  .....
  ------
  ..... //logic for paymemt
  .....
   String status=bDart.deliver(oid);
  String status=dtdc.deliver(oid);

  return status +" "+billAmt;
 }
}

//dependent class
public class DTDC{

 public String deliver(int oid){
   ....
   ...
 }
}

//dependent class
public class BlueDart {

 public String deliver(int oid){
    ....
    ...
  }
}
```

=>To change Depedent "DTDC" to "BlueDart" we need to modify the source code target class every time , So we can say there is tight coupling.

solution :: Loose Coupling through interfaces/abstract classes
========================

=>Make all related Dependent classes implementing the common Interface or extending from Common super class /abstract class
=> Take common Interface or super class or abstract class type HAS-A property in the Target class and use the support setter method or parameter constructor for assinging dependent to target externally.

Soultion code
============

```
     //Common interface
interface Courier{
  public String deliver(int oid);
 }

//Target class
=============
class Flipkart {
 //HAS- A property
 private Courier courier;
       Coding to interfaces
 public void setCourier(Courier courier){
    this.courier=courier;
 }

 public String shopping(
      String item[],float prices[]){
   .....
   ..... //calc b.method
   .....
   ------
   ..... //logic for paymemt
   .....
  String status=courier.deliver(oid);

   return status +" "+billAmt;
 }

//Dependent class1
class DTDC implements Courier{

  public String deliver(int oid){
     .....
     .....
   return "....";
  }
}

//Dependent class2
class BlueDart implements Courier{

  public String deliver(int oid){
     .....
     .....
   return "....";
  }
}
```

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

```
Flipkart fpkt=new Flipkart(); //target class obj
Courier courier=new DTDC(); //Dependent class obj
 flpk.setCourier(courier); // assinging dependent class obj
                               to target class obj
 String resultMsg=fpkt.shopping(new Stirng[]{"shirt","trouser"},
                                 new float[]{5000.0f,4000.of} );
 S.o.p(resultMsg);
```

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

```
  Flipkart fpkt=new Flipkart(); //target class obj
Courier courier=new BlueDart(); //Dependent class obj
   flpk.setCourier(courier); // assinging dependent class obj
                                 to target class obj
   String resultMsg=fpkt.shopping(new Stirng[]{"shirt","trouser"},
                                   new float[]{5000.0f,4000.of} );
   S.o.p(resultMsg);
```

In the Client Apps we can from one Dependent courier and another Dependent courier with out changing the code of target class becoz the coding is done to interfaces.. to achieve loosely coupling.

c) Code must be open for extension and must be closed for modification. strategy pattern rule no:3

=>This rule is extension of rule2 .. if we enhance rule number2 , automatically the rule number3 will be implemented..

=> We can take more dependent classes like DHL, FirstFlight and etc.. implementing the common Interface i.e our code is open for Extension..
=> We can take both target and dependent classes as the final classes or their methods as final methods to make their code "Closed for modifition" (Final classes can not have sub classes , Final mehods can not be overriden in sub classes)

Soultion code
============

```
     //Common interface
interface Courier{
  public String deliver(int oid);
 }

//Target class
=============
final class Flipkart {
 //HAS- A property
 private Courier courier;
       Coding to interfaces
 public void setCourier(Courier courier){
    this.courier=courier;
 }

 public String shopping(
      String item[],float prices[]){
   .....
   ..... //calc b.method
   .....
   ------
   ..... //logic for paymemt
   .....
  String status=courier.deliver(oid);

   return status +" "+billAmt;
 }

//Dependent class1
final class DTDC implements Courier{

  public String deliver(int oid){
     .....
     .....
   return "....";
  }
}

//Dependent class2
final class BlueDart implements Courier{

  public String deliver(int oid){
     .....
     .....
   return "....";
  }
}

//Dependent class3
final class DHL implements Courier{

  public String deliver(int oid){
     .....
     .....
   return "....";
  }
}
```

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

```
Flipkart fpkt=new Flipkart(); //target class obj
Courier courier=new DTDC(); //Dependent class obj
 flpk.setCourier(courier); // assinging dependent class obj
                               to target class obj
 String resultMsg=fpkt.shopping(new Stirng[]{"shirt","trouser"},
                                 new float[]{5000.0f,4000.of} );
 S.o.p(resultMsg);
```

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

```
  Flipkart fpkt=new Flipkart(); //target class obj
Courier courier=new BlueDart(); //Dependent class obj
   flpk.setCourier(courier); // assinging dependent class obj
                                 to target class obj
   String resultMsg=fpkt.shopping(new Stirng[]{"shirt","trouser"},
                                   new float[]{5000.0f,4000.of} );
   S.o.p(resultMsg);
```

Sep 25 Stategy Pattern DP

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

strategy pattern rule no:3

=>This rule is extension of rule2 .. if we enhance rule number2 , automatically the rule number3 will be implemented..

=> We can take more dependent classes like DHL, FirstFlight and etc.. implementing the common Interface i.e our code is open for Extension..

=> We can take both target and dependent classes as the final classes or their methods as final methods to make their code "Closed for modifition" (Final classes can not have sub classes , Final mehods can not be overriden in sub classes)

Soultion code

```
//Common interface
interface Courier{
  public String deliver(int oid);
}

//Target class
final class Flipkart {
  //HAS- A property
  private Courier courier;
          Coding to interfaces
  public void setCourier(Courier courier){
    this.courier=courier;
  }

  public String shopping(
        String item[],float prices[]){
    .....
    ..... //calc b.method
    .....
    ------
    ..... //logic for paymemt
    .....
   String status=courier.deliver(oid);

    return status +" "+billAmt;
  }
```

```
//Dependent class1
final class DTDC implements Courier{

  public String deliver(int oid){
    .....
    .....
   return "....";
  }
}

//Dependent class2
final class BlueDart implements Courier{

  public String deliver(int oid){
    .....
    .....
   return "....";
  }
}
```

```
//Dependent class3
final class DHL implements Courier{

  public String deliver(int oid){
    .....
    .....
   return "....";
  }
}
```

Client App

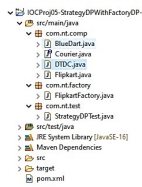
```
Flipkart fpkt=new Flipkart(); //target class obj
Courier courier=new DTDC(); //Dependent class obj
flpk.setCourier(courier);  // assinging dependent class obj
                              to target class obj
String resultMsg=fpkt.shopping(new Stirng[]{"shirt","trouser"},
                               new float[]{5000.0f,4000.of} );
S.o.p(resultMsg);
```

=>To provide abstraction to Client App developers .. It is better to provide Factory to them

Client App|

```
Flipkart fpkt=new Flipkart(); //target class obj
Courier courier=new BlueDart();    //Dependent class obj
flpk.setCourier(courier);  // assinging dependent class obj
                              to target class obj
String resultMsg=fpkt.shopping(new Stirng[]{"shirt","trouser"},
                               new float[]{5000.0f,4000.of} );
S.o.p(resultMsg);
```

example App Strategy DP




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

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

public final class Flipkart {
    //HAS -A  property of type interface
    private Courier courier;

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

    //setter method for setter injection
    public void setCourier(Courier courier) {
        System.out.println("Fliplkart.setCourier(-)");
        this.courier = courier;
    }

    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);
        //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;
    }


}
```

```
package com.nt.comp;

public interface Courier {
    public String  deliver(int oid);
}
```

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

public final class BlueDart implements Courier {

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

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

}
```

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

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";
    }

}
```

```
//FlipkartFactory.java
package com.nt.factory;

import com.nt.comp.BlueDart;
import com.nt.comp.Courier;
import com.nt.comp.DTDC;
import com.nt.comp.Flipkart;

public class FlipkartFactory {

    //static factory method supporting Factory pattern
    public static  Flipkart  createFlipkart(String courierType) {
        //create target class obj
        Flipkart fpkt=new Flipkart();
        Courier courier=null;
        //create Depennt class obj based on given courier type
        if(courierType.equalsIgnoreCase("dtdc"))
            courier=new DTDC();
        else if(courierType.equalsIgnoreCase("bDart"))
            courier=new BlueDart();
        else
            throw new IllegalArgumentException("Invalid courier type");
        //set  Dependent class object to target class obj
        fpkt.setCourier(courier);
      return fpkt;
  }

}
```

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

import com.nt.comp.Flipkart;
import com.nt.factory.FlipkartFactory;

public class StrategyDPTest {

    public static void main(String[] args) {
        try {
            //get target class object form Factory
            Flipkart fpkt=FlipkartFactory.createFlipkart("bDart");
            // invoke the b.method
            String result=fpkt.shopping(new String[] {"shirt","trouser","mobile"},
                                        new float[] {3400.0f , 5600.0f, 60000.0f});
            System.out.println(result);

        }//try
        catch(Exception e) {
            e.printStackTrace();
        }
    }//main
}//class
```

=>The above code is developed by using strategy DP and Factory DP and getting loose coupling with respect to target and dependent classes i.e we can change from depedent class to another dependent class with out changing the source code of target class, But this not 100% loosely coupled code..

Why the above code is not 100% Loosely coupled?

a) afer adding 1 more courier class by implementing Courier(I) ...we need to modify the source code of Factory class

b) if Client App wants to change from 1 courier to another courier then we need to modify the source code of Cleint App.

Read following topics and come for next class

a) Properties file

b) java.util.Properties

c) IO streams

d) static block

Why the above code is not 100% Loosely coupled?

Sep 27 Stategy Pattern DP Using Spring

a) afer adding 1 more courier class by implementing Courier(I) ...we need to modify the source code of Factory class
b) if Client App wants to change from 1 courier to another courier then we need to modify the source code of Cleint App.

To make initial Strategy DP application as 100%Looselycoupled App , we need to use the proeprties file or xml file

properties file
==========
=>The text file that maintain the entries as key=value pairs is called properties file. It is recomanded to take .properties extension for the properties file.

info.properties (com/nt/commons pkg)

```
# Personal info
per.name=raja
per.addrs=hyd
per.age=40
```
Keys values

=>In keys we can take "."," ","_","-" as the seperators ..not the other characters..
=>Keys must be unique.
=>'#' symbol comments the line

=>java.util.Properties is the Map Collection and the sub class of java.util.Hashtable. we can store the keys and values of properties file into java.util.Properties collection using load(-) of same class.

//sample code
==========
```
//Locate proeprties file
FileInputStream fis=new FileInputStream("src/main/java/com/nt/commons/info.properties");
//Load properties file content into java.util.Properties class object
Properties props=new properites();
 props.load(fis);
```

props(java.util.Properties class obj)

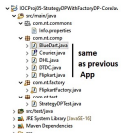
per.name	raja	element1
per.addrs	hyd	element2
per.age	40	element3
keys	values	

```
System.out.println(props);
System.out.println("per.age key value"+props.getProperty("per.age"));
```
gives the value 40

=>java.util.Proepeties Collection elements ,the key and values must be Strings

---

Making Core java based StrategyDP App as loosely coupled App using the support of Properites file
==========



info.properties

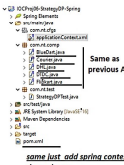
```
#Specify Dependent class
dependent.comp=com.nt.comp.DHL
```

```
//FlipkartFactory.java
package com.nt.factory;

import java.io.FileInputStream;
import java.lang.reflect.Constructor;
import java.util.Properties;

import com.nt.comp.Courier;
import com.nt.comp.Flipkart;

public class FlipkartFactory {
   private static Properties props;
   static {
      System.out.println("FlipkartFactory.static block");
  try{//Load Properties file
            FileInputStream fis=new FileInputStream("src/main/java/com/nt/commons/info.properties");){
            //load properties file content to java.util.Properties class obj
            props=new Properties();
            props.load(fis);
      }//try
      catch(Exception e) {
            e.printStackTrace();
      }
   }

      //static factory method supporting Factory pattern
      public static  Flipkart  createFlipkart()throws Exception {
            //create target class obj
            Flipkart fpkt=new Flipkart();

            // Load Dependent class
            Class c=Class.forName(props.getProperty("dependent.comp"));
            //create object using reflection object
            Constructor cons[]=c.getDeclaredConstructors();
            //create object
            Courier courier=(Courier) cons[0].newInstance();
            //set  Dependent class object to target class obj
            fpkt.setCourier(courier);
         return fpkt;
      }

   }
```

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

import com.nt.comp.Flipkart;
import com.nt.factory.FlipkartFactory;

public class StrategyDPTest {

   public static void main(String[] args) {
      try {
            //get target class object form Factory
            Flipkart fpkt=FlipkartFactory.createFlipkart();
            // invoke the b.method
            String result=fpkt.shopping(new String[] {"shirt","trouser","mobile"},
                                        new float[] {3400.0f , 5600.0f, 60000.0f});
            System.out.println(result);
      }//try
      catch(Exception e) {
            e.printStackTrace();
      }
   }//main
}//class
```

Problems in Strategy DP impl using core Java setup even after improvising with properties file
==========
a) We need to implement factory pattern explicitly
b) No internal cache by default.. If needed we should write those logics explicitly
c) To make code of 100% Loosely coupled code , we need to take properties file and xml file but reading from properties file or from xml file needs lots of api knowledge .. This improves burden on the programmer

To overcome these problem implement strategy DP using Spring framework



same last add spring context support dependency

applicationContext.xml

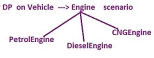
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

   <!-- 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"/>

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

</beans>
```

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

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

import com.nt.comp.Flipkart;

public class StrategyDPTest {

   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 Target class obj
         Flipkart fpkt=factory.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
```

Advantages of developing strategy DP application using spring framework
==========
a) No need of developing Factory pattern ..becoz spring Container/IOC container itself acts factory Patttern
b) Keeps Spring bean class objs in the built-in internal cache of IOC container giving the reusability of spring bean class obj
c) Allows to give Dependency Mamagement inputs in multiple ways like xml file, annotations, java classes
d) Improves the readability of the Application.
e) 100% Loosely copuling promised.
f) It is industry standard to use Spring Framework for Dependency Manageemnty.
and etc..

=>A perfectly develop spring App contains the following DesignPatterns
a) Factory pattern
b) strategy DP
c) Flyweight pattern (keeping objects in the internal cache and reusing the objs is activity of Flyweight patter)

Assignment:: Implement strategy DP on Vehicle --> Engine scenario



Sep 28 Setter Injection Vs Constructor Injection

Q) Where should use setter Injection and where should we contructor Injection?

Ans) if all properties spring bean are mandatory to parcipate in Dependency Injection then go for Constructor Injection .. if the proeprties of spring bean are optional to participate in dependency Injection then go for setter Injection.

=>Spring Bean class with 4 properties
a, b, c, d are bean propeties

Let us assume 4 param constuctor is placed in spring bean class
=>If we do constuctor injection using 4-param constructor then all params must partipate in the injection

=> Using constructor Injection , if want to involve our choice no.of properties in the Injection process then we need multiple overloaded constructors that is 4! no.of constructors (24), if the bean properties count is 4. 10! no.of constructors if the bean properties count 10(36,28,800)

Conclusion:: Performinng constructor Injection using overloaded constructors on our choice no.of bean properties is very complex process. So prefer constructor injection only when all bean proeprties mandatory to particapte in costructor injection.. For this only one parameterized constructor is enough

eg: spring bean with 10 properties
=> contstrctor injection on all bean properties ==> one 10 param consturctor is enough
=> contstrctor injection on my choice bean properties ==> we need 10! no.of overloaded constructors (36,28,800)

=>While workign setter methods seetter Injection we can involve our choice no.of bean proepeties fo setter Injection.

10 spring bean properties in spring bean calss
==========
=>To invole our choice no.of bean properties in setter Injection we just need 10 setter methods
=> All properties must participate in dependency Injection :: Not possible with setter Injection.

Final Conclusion::
==========
spring bean class with 10 bean properties
==========
=>all bean properties must participate in the Dependency Injection then go for one 10 param constructor supporting constructor Injection
=>all bean properties are optional to participate in the Dependency Injection then go for 10 setter methods supporting seetter Injection.

=>In spring bean class few properites are mandatory to participate in dependency Injection another few propeerties optional to participate in the dependency Injection then prefer using both setter and constructor Injection
=>For mandatory properties :: cosntructor Injection
=> for optional oprperties :: setter Injection.

=>Spring's dependency Injection is not given to Inject the enduser non-technical input values to spring bean properties like name,age, addrs and etc.. It is always given to Inject programmer supplied technical input values (liked jdbc properties) or IOC container created spring bean class objs (like DTDC obj to Flipkart)

jdbc properties :: jdbc driver class name, jdbc url, db user, db pwd.

//Employee1.java

```
package com.nt.beans;

public class Employee1 {
    //all are mandtory to participate
    private int eno;
    private String ename;
    private  float billAmt;

    public Employee1(int eno, String ename, float billAmt) {
        System.out.println("Employee1:: 3-param constructor");
        this.eno = eno;
        this.ename = ename;
        this.billAmt =billAmt;
    }


    @Override
    public String toString() {
        return "Employee1 [eno=" + eno + ", ename=" + ename + ", billAmt=" + billAmt + "]";
    }

}
```

Customer.java

```
package com.nt.beans;

public class Customer {
    //cno,cname,billAmt are mandatory properties
    private  int cno;
    private String cname;
    private  float billAmt;
    //caddrs,mobileNo are optional properities
    private String caddrs;
    private long  mobileNo;

    public Customer(int cno, String cname, float billAmt) {
        super();
        this.cno = cno;
        this.cname = cname;
        this.billAmt = billAmt;
    }

    public void setCaddrs(String caddrs) {
        this.caddrs = caddrs;
    }

    public void setMobileNo(long mobileNo) {
        this.mobileNo = mobileNo;
    }

    @Override
    public String toString() {
        return "Customer [cno=" + cno + ", cname=" + cname + ", billAmt=" + billAmt + ", caddrs=" + caddrs
                + ", mobileNo=" + mobileNo + "]";
    }
```

//Student.java

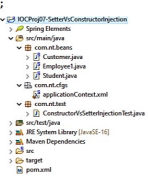
```
package com.nt.beans;

public class Student {
    //let us assume optional to provide
    private  String sname;
    private  String  collegeName;
    private  int age;
    private  String qlfy;

    public void setSname(String sname) {
        this.sname = sname;
    }
    public void setCollegeName(String collegeName) {
        this.collegeName = collegeName;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public void setQlfy(String qlfy) {
        this.qlfy = qlfy;
    }

    @Override
    public String toString() {
        return "Student [sname=" + sname + ", collegeName=" + collegeName + ", age=" + age + ", qlfy=" + qlfy + "]";
    }



}
```

applicationContet.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">

  <bean id="emp1" class="com.nt.beans.Employee1">
    <constructor-arg name="eno" value="1001"/>
    <constructor-arg name="ename" value="raja"/>
    <constructor-arg name="billAmt" value="545.45f"/>
  </bean>

  <bean id="stud1" class="com.nt.beans.Student">
    <property name="sname" value="raja"/>
    <!-- <property name="collegeName" value="CBIET"/>
    <property name="qlfy" value="B.Tech"/> -->
  </bean>

  <bean id="cust1" class="com.nt.beans.Customer">
    <constructor-arg name="cno" value="1001"/>
    <constructor-arg name="cname" value="rajesh"/>
    <constructor-arg name="billAmt" value="67.88f"/>
    <property name="caddrs" value="hyd"/>
  </bean>

</beans>
```

Client App

```
package com.nt.test;

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

import com.nt.beans.Customer;
import com.nt.beans.Employee1;
import com.nt.beans.Student;

public class ConstructorVsSetterInjectionTest {

    public static void main(String[] args) {
        DefaultListableBeanFactory factory=new DefaultListableBeanFactory();
        XmlBeanDefinitionReader reader=new XmlBeanDefinitionReader(factory);
        reader.loadBeanDefinitions("com/nt/cfgs/applicationContext.xml");
        //get Spring bean class obj
        Employee1 emp1=factory.getBean("emp1",Employee1.class);
        System.out.println(emp1);
        System.out.println("====================");
        Student  st1=factory.getBean("stud1",Student.class);
        System.out.println(st1);
        System.out.println("====================");
        Customer  cust1=factory.getBean("cust1",Customer.class);
        System.out.println(cust1);



    }

}
```



Cyclic Dependency Injection/ Circular Inejction

Sep 29 Setter Injection Vs Constructor Injection -part 2

=> Here Target and dependent classes maintain HAS-A property having Circular Dependency i.e Both are dependent to each other.

=>If A,B are the spring Beans the A dependent B and B is dependent to A
=> In real Projects , we do not see the need of Cyclic DI.. It is purely POC (knowledge)

=>setter injection support Cyclic DI and constructor injection does not Support Cyclic DI

Cyclic DI using setter injection

A.java

```
package com.nt.beans;

public class A {
        private  B b;

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

 public void setB(B b) {
        System.out.println("A.setB()");
              this.b = b;
        }

@Override
public String toString() {
      return "A [b=]";
}


}
```

B.java

```
package com.nt.beans;

public class B {
 private A a;


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


public void setA(A a) {
        System.out.println("B.setA()");
        this.a = a;
}


@Override
public String toString() {
      return "B [a=]";
}


}
```

applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans https://www.springframework.org/schema/beans/spring-beans-3.1.xsd">

 <!--Spring beans cfg  -->
 <bean id="a1" class="com.nt.beans.A">
   <property name="b"  ref="b1"/>
 </bean>

 <bean id="b1" class="com.nt.beans.B">
     <property name="a"  ref="a1"/>
    </bean>
</beans>
```

Client App

```
package com.nt.test;

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

import com.nt.beans.A;

public class CyclicDITest {

      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 calss obj
            A  oa=factory.getBean("a1",A.class);
            System.out.println(oa);

      }

}
```

Cyclic DI using constructor injection (not possible .. throws Exception org.springframework.beans.factory.BeanCurrentlyInCreationException: Error creating bean with name 'a1': Requested bean is currently in creation: Is there an unresolvable circular reference?)

A.java

```
package com.nt.beans;

public class A {
        private  B b;

        public A(B b) {
              this.b=b;
              System.out.println("A:: 1-param constructor");
        }


@Override
public String toString() {
      return "A [b=]";
}


}
```

B.java

```
package com.nt.beans;

public class B {
 private A a;


  public B(A a) {
        this.a=a;
        System.out.println("B: 1-param constructor");
  }



@Override
public String toString() {
      return "B [a=]";
}


}
```

applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
https://www.springframework.org/schema/beans/spring-beans-3.1.xsd">

 <!--Spring beans cfg  -->
 <bean id="a1" class="com.nt.beans.A">
   <constructor-arg name="b"  ref="b1"/>
 </bean>

 <bean id="b1" class="com.nt.beans.B">
     <constructor-arg  name="a"  ref="a1"/>
    </bean>
</beans>
```

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

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

import com.nt.beans.A;

public class CyclicDITest {

      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 calss obj
            A  oa=factory.getBean("a1",A.class);
            System.out.println(oa);          throws  Exception::
                                             org.springframework.beans.factory.BeanCurrentlyInCreationExcept
      }                                      ion: Error creating bean with name 'a1': Requested bean is
                                             currently in creation: Is there an unresolvable circular reference?
}
```

Cyclic DI using **one side** ***constructor injection*** **and another side Setter injection**

Possible but we need to pass that class beanid in factory.getBean(-) where setter injection is enabled.

A.java

```
package com.nt.beans;

public class A {
        private  B b;

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

        public  void setB(B b) {
              System.out.println("A.setB()");
              this.b=b;
        }


@Override
public String toString() {
      return "A [b=]";
}


}
```

B.java

```
package com.nt.beans;

public class B {
 private A a;


  public B(A a) {
        this.a=a;
        System.out.println("B: 1-param constructor");
  }



@Override
public String toString() {
      return "B [a=]";
}


}
```

applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
https://www.springframework.org/schema/beans/spring-beans-3.1.xsd">

 <!--Spring beans cfg  -->
 <bean id="a1" class="com.nt.beans.A">
   <property name="b"  ref="b1"/>
 </bean>

 <bean id="b1" class="com.nt.beans.B">
     <constructor-arg  name="a"  ref="a1"/>
    </bean>
</beans>
```

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

```
package com.nt.test;

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

import com.nt.beans.A;
import com.nt.beans.B;

public class CyclicDITest {

      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 calss obj  (works)
            A  oa=factory.getBean("a1",A.class);
            System.out.println(oa);
            /*System.out.println("=======");
            B  ob=factory.getBean("b1",B.class);  //throws exception
            System.out.println(ob); */

      }

}
```

What is difference b/w setter injection and constutor injection?

Setter Injection	Constructor Injection
a) IOC container uses setter method to perform this injection	a) IOC container uses parameterized constructor to perform this injection
b) use <property> tag to cfg this injection	b) use <constructor-arg>tag to cfg this injection
c) place @Autowired on the setter Method to perform this injection *	c) place @Autowired on the parameterized constructor to perform this injection *
d) IOC container first creates target class obj ,then creates Depedent class obj	d)IOC container first creates Dependent class obj ,then creates Target class obj
e) If All bean properties of spring bean class are enabled with setter injection then IOC container creates spring bean class object using 0-param constructor	e) If any bean property of spring bean is enabled with constructor injection then IOC container creates spring bean class object using parameterized constructor
f) It is good to use this setter injection if the bean properties of spring bean class are optionai participare in dependency injection..	f) It is good to use this constructor injection if all the bean properties are mandatory to participate in the injection.
g) To Perform injection on n properties in all permitation and combination we generally need max "n" setter methods	g) To Perform injection on n properties in all permitation and combination we generally need max "n!" overloaded constructor (partically impossible)
h) we can use P name space tags and attributes for simplifying setter injection cfgs **	h) we can use C name space tags and attributes for simplifying constructor injection cfgs **
(i) Supports Cyclic DI	i) Does not support Cyclic DI
j) Bit slow in the injection process becoz injection takes place after creating sprign bean class obj	j) Bit faster compare to setter injection becoz the injection takes place while creating object itself

Realtime DI Background preparation (Mini Project Development) sep 30 Realtime DI

=> A typical project contains the following logics
a)presentation logics (Logic giving user-interface)
b) business logic/ Service logic (Logic performing calculations, analyzations, filtering ,sorting and etc..)
c) Persistece logic performing CURD/CRUD Operations on s/w
(Insert,update,delete and select operations)
and etc..

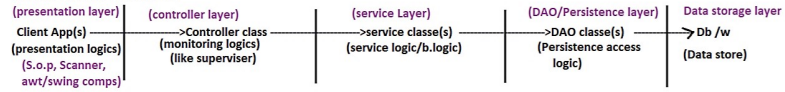
Limitations of keeping multiple logics in single java class (It is like bachelor single room)

=> Multiple logics will be mixedup , So we can say there is no clean seperation b/w logics
=> Code becomes very clumzy
=> The modifications done in one kind of logics effects other kind of logics
=> Maintainace and the enhancement of the Project becomes complex
=>Parellel development is not possible , So the productivity is poor.
=> It is not indurstry standard approach to develop the s/w Apps.

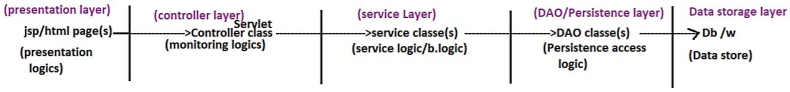
=>To overcome these problems .. Develop the App/project as the Layered App/Project where multiple category logics will be placed in multiple logical or physical partiations (nothing but layers) of the Project.

=> A Layer is logical or physical partiation of the Project having 1 more resources(files/classes) representing certain category logics.
=>Presentation layer (1 or more classes/files) represents presentation logics
=>Service layer (1 or more classes/files) represents service logics
=Persistence layer/DAO Layer (1 or more classes/files) represents Persistence logics
=>Controller layer (generally 1 class) represents controlling or monitoring logics

Standalone Layered Application (It is like 3BHK Flat)



Web based Layered Application



DAO class (Data Access Object class)

=>The java class whose code is written using one or another DataAccess tehchnologies or frameworks to seperate persitence logics from other logics of the Project to make the Persistence logics as the reusable logics and flexible to modify is called DAO class.

=>DAO class should have only persistence logic peforming CURD operations .. it should not have even c=a+b kind of b.logic

=>DAO class can use either direct JDBC con object or Pooled JDBC object for interacting with Db s/w..

=> If project is having <100 db tables then take 1 DAO class per 1 db table
if project is having >=100 db tables then take 1 DAO class per related 4 to 5 db tables

=> Can have perisitence logic calling PL/SQL procedures and functions

=> Evey DAO class contains two parts
a) Query part (All SQL queries will be placed at top of the class in upper case letters as private string constant values (private static final String variable)

eg: private static final String GET_ALL_EMPS="SELECT * FROM EMP";

To make query specifc current class
To access query with out object
To make sure that query is not going be modified in rest of prg
Recomanded to take in upper case to differentiate from the regular java code.

Code part
Java method having persistence logic devleoped using persistence technologies /frameworks like JDBC, hibernate, spring JDBC, spring ORM, spring data jpa.

=>DB s/ws like oracle ,mysql ,mongoDB and etc.. called Data storage technologies

=> JDBC,hibernate, spring JDBC, spring ORM, spring data jpa and etc.. are given to access and manipulate the data of Db s/ws.. So they are called Data Access technologies or frameworks

=>insert ,update, delete and select perations are called CURD Operations

Two types of JDBC con objects
1.Direct JDBC con obj — created by the progammer manually
2.Pooled JDBC con obj — Collected from JDBC con pool (ready made jdbc con obj)

Service class

=>The java class that contains b.logic or service logic representing calculations,analyzations, filtering, sorting and etc..
=> It is 1 class per module
=> purely contains b.logics /service logics

University Project
|---> Admissions module
|--->Examinations module
|--->Academics module
|--->Sports module
|---> Payroll module
|--->Co-Circular activities module
|--->Placements module
|---->HR department module

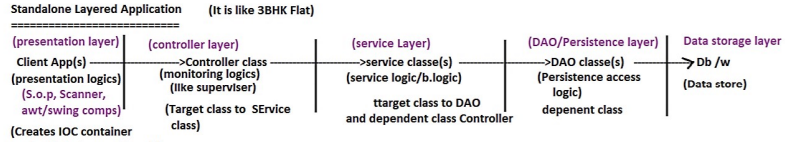
Controller class

=>The java class contains purely monitoring logic
=>This class code controls all activities of the Application
=> It is 1 class Per Project.
=>Makes Approprivate Client Apps/ presentation comp/view comp taking approprivate SErivce class and vice-versa.
=>It is like superviser
=>It contains addtional logics like logging,, auditing and etc.. to enable more control and monitoring on the Project.
=>All the activities of the App must take place under the monitoring Controller.

Client App / View Comps/ html,jsp comps

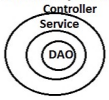
=>Contains presentation logics providign user-interface to enduser
=>Responsible to gather inputs from enduser and to display results for enduser
=> Any exception raised in any layer during the excution of the Project should be propagated/passed to Client App to display to enduser..

Standalone Layered Application (It is like 3BHK Flat)



use
Where did u Realtime DI in your project?

Ans) My Project is Layered App .. In the Project multuple layer classes are there like DAO class, service class,Controller class and etc.. By cfg these classes as spring beans for IOC container we make the IOC container injecting DAO class object to service class obj and Service class object Controller object..
note:: The Client App gets Controller object having injected Service,DAO class objs by calling factory.getBean(-) method



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

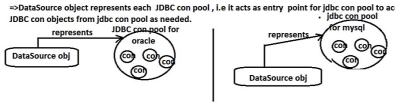
---

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

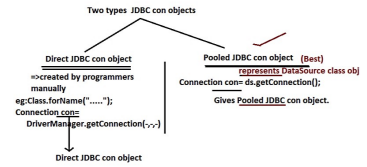
JDBC con pool
==========
=> It is a factory that contains set of readily avalable jdbc con objects before actually being used.
=>the jdbc con pool advantages are
i) resuability of jdbc con objs
ii) with minimum no.of jdbc con objects , we can make max client Apps/requets talking to DB s/w.
iii) creation 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)

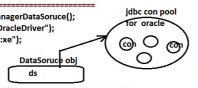


=> By creating object for DataSoruce class (class implementing DataSoruce(I)) and filling with JDBC driver details and db details (jdbc properties) .. we can make dataSoruce creating JDBC con pool having given details based jdbc con objs.. and that jdbc con pool will be represented by DataSource object.

=>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.

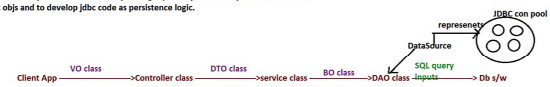
//To get one pooled jdbc con obj
Connection con=ds.getConnection();

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

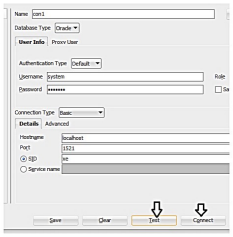
- **|-->eno (pk) (n)**
- **|-->ename (vc2)**
- **|-->desg (vc2)**
- **|-->basicSalary (float)**
- **|-->grossSalary (float)**
- **|-->netSalary (float)**

GUI DB tools make DB operations easy

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

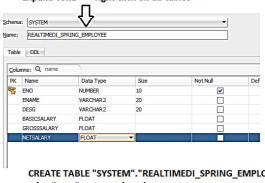
note:: SQL developer is seperate installation by dowloading from
https://www.oracle.com/tools/downloads/sqldev-downloads.html

a) create connection using SQL developer



b) create the above db table

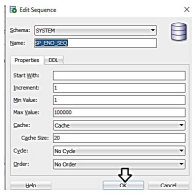
Expand con1 ---> right click on db tables --->




```
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

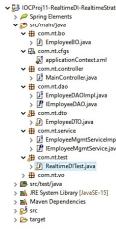
MainController.java

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
==========

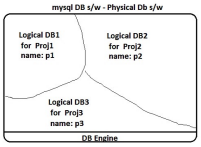
Oct 08 Realtime DI-MySql

=>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



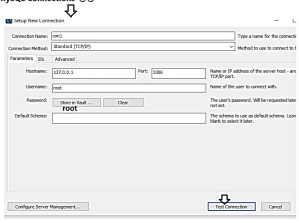
=>Every Project related
db tables , pl/sQL procedures
functions, view ,sequences and etc..
will be created .. in that project related
Logical DB.

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 ⊕ ⇘



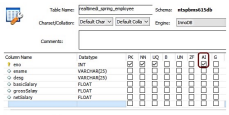
step2) create Logical Db in mysql DB s/w

(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

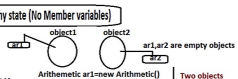
When should we java class as singlton java class or spring bean as singleton scope spring beans?

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;
   }
}
```



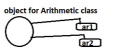

```
Arithmetic ar1=new Arithmetic()
ar1.sum(10,20);
Arithmetic ar2=new Arithmetic()
ar2.sum(20,30);
```

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

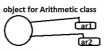
Solution2:

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.



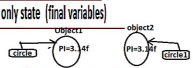
In client app

```
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



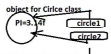

```
public class Circle{
   private float PI=3.14f;

   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;
    }
}
```



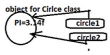

```
Circle circle1 =Circle.getInstance();
circle1.calcArea(10.0f);

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);
```

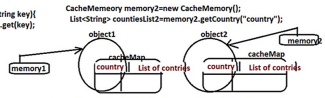
---

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.

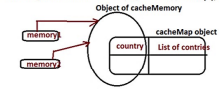
Solution1:: Develop CacheMemory class singleton java class

```
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");
```

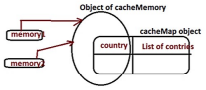


Solution2:: make Cache Memory calss as spring bean having singleton scope

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
==========

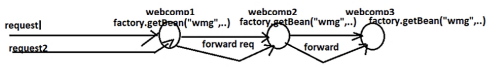
```
//get Spring bean class obj
Printer p1=factory.getBean("p1",Printer.class);
Printer p2=factory.getBean("p1",Printer.class);
System.out.println(p1.hashCode()+"  "+p2.hashCode());  // gives  same  hashCodes
System.out.println("====================");
```

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

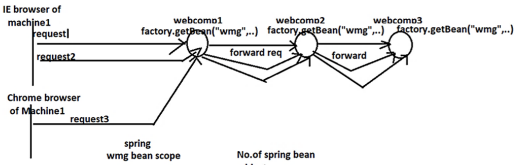
if "wmg" bean id scope is "prototype" then all 3 times 3 diffrent objects will be given for both request and request2 | total 6 objs

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

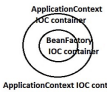


spring wmg bean scope	No.of spring bean objects
"singleton"	1
"prototype"	9
"request"	3
"session"	2
"application"	1

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
```

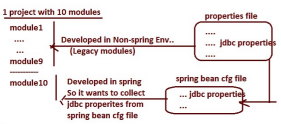
```
==================
Sunday 10:30 am
spring boot Logging
==================
========== Workshop Link =========
Meeting ID 928 0375 4982
Passcode 112233
```

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)

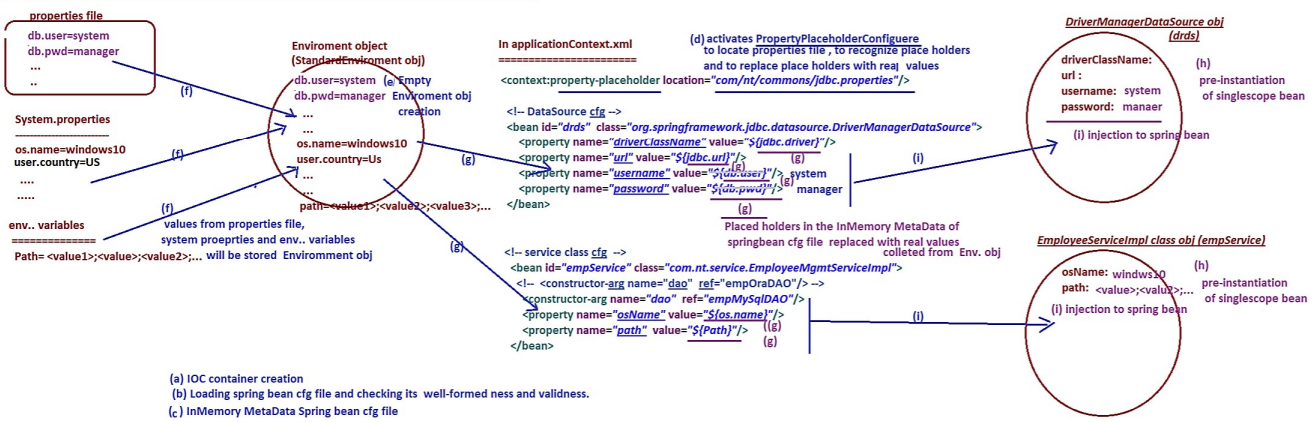
```
<!-- service class cfg -->
 <bean id="empService" class="com.nt.service.EmployeeMgmtServiceImpl">
  <!-- <constructor-arg name="dao" ref="empOraDAO"/> -->
   <constructor-arg name="dao" ref="empMySqlDAO"/>
    <property name="osName" value="${os.name}"/>
    <property name="path" value="${Path}"/>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

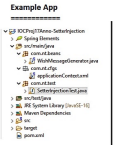
To enable autowiring in xml cfgs use "autowire" attribute <bean> tag .. In annotation driven cfgs use @Autowired annotation.

@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)

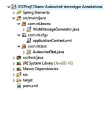
note:: @Service , @Reppository , @Controller are the extentions annotations of @Component

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.

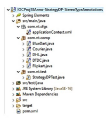
In xml driven cfgs

```
<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
```

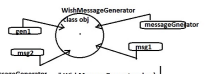
```
DTDC.java
=========
//DTDC.java (Dependent class1)
package com.nt.comp;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.PropertySource;
import org.springframework.stereotype.Component;
@Component("dtdc")
//@Component("courier")
//@Primary
@PropertySource("com/nt/commons/info.properties")
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

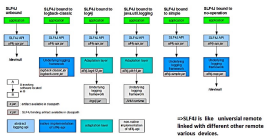
Oct 31 Spring Boot Logging

=>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?

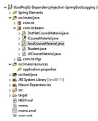
step1) Create Logger object in java classes using LoggerFactory.getLogger() method and use that logger obj in various places of class to write different levels of log messages as shown below.

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..

Spring Bean Life cycle **Nov 01 Spring Bean Life cycle**

=keep track of all activies of the comp from birth to death (Object creation to object destruction) is called life cycle.
=> The important activity of life cycle is called life cycle event .. executing logics for that life cycle event is called as life cycle event handling.. the methods that executes automatically for life cycle events are called life cycle methods.

=>Servlet container manages servlet life cycle by raising 3 life events and calling 3 life cycle methods

Sevlet life cycle event	life cycle method	
=>instantation event	-----> init(SevletConfig cg)	(raises when servlet container cretes our servlet class obj
=> request processing event	----> service(re,res)	(raises when servlet container keeps our servlet class obj ready for request processing)
=> Destruction event	-----> destroy()	(raises when Servlet container is about to destroy our servlet class obj)

=> The life cycle method names servlet comp are fixed name becoz the servlet comp is invasive (Should implement javax.servlet.Servlet(I) directly or indirectly) jakarta.

=>IOC container manages spring bean life cycle raising two life cycle events calling two life cycle methods

a) Instantaition event
=>Raises when IOC containier creates Spring bean class obj, completes all injections on that object
=>Generally useful to initialize those properties that are not participating in any kind injections and also useful to check wheather the bean properties that are partcipated in the injecton process assigned with valid values or not.

=>Generally we take custom method name as init life cycle method becoz spring beans are non-invasive and we try to cfg that as init life cycle method.. This method must follow some standard signature.

```
public void <method name>(){
   .....
   ....
   ...
}
```
This method must be cfg init life cycle method either using annotaitons or xml cfgs

b) destruction event

=>Raises when the IOC container is about to destroy our spring bean class object
=>we can cfg any custom method as destroy life cycle method either using xml fgs or annotations.
=> This custom method signature should be

```
public void <method-name>(){
   ....
   ...
}
```
Since the spring beans are non-invasive we must cfg custom destroy method as destroy life cycle method.

=> The destroy life cycle method generally contains the logics to nullify bean properties to release non-java resources (like jdbc con) associcated with spring bean

---

Spring bean life cycle cfgs can be done in 3 approaches

Approach1) Declarative Approach (Cfg spring bean life methods using xml cfgs)

Approach2) Programatic Approach (Making spring bean to implements spring interfaces executing fiexed methods life cycle methods)

Approach3) Annotatoion Approach (cfg spring life cycle methods using annotations)

=>Approach1, approach3 are allows us to keep spring beans as non-invasive .. So the life cycle methods custom methods and we need to cfg those methods. can be

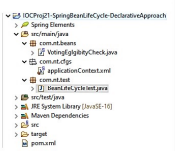
=>Approach2 makes spring bean as invasive .. So the life cyle method names are fixed and we need not to cfg them..

Approach1) Declarative Approach (Cfg spring bean life methods using xml cfgs)

=> We need to use "init-method" attribute of<bean> tag to cfg custom method as init life method
=> We need to use "destroy-method" attribute ot<bean> tag to ctg custom method as destroy life method
=>Both custom init method and custom destory method should have same standard signature.

```
public void <method-name>(){
   ....
   ...
   ..
}
```

note:: Both BeanFActory(BF) and ApplicationContext (AC) containers support bean life cycle cfgs..




```
package com.nt.beans;

import java.util.Date;

public class VotingEglgibityCheck {
        //bean properties
        private String name;
        private int age;
        private String addrs;
        private Date verfiedOn;

        //setter methods for setter injetion
        public void setName(String name) {
                System.out.println("VotingEglgibityCheck.setName(-)");
                this.name = name;
        }
        public void setAge(int age) {
                System.out.println("VotingEglgibityCheck.setAge(-)");
                this.age = age;
        }
        public void setAddrs(String addrs) {
                System.out.println("VotingEglgibityCheck.setAddrs(-)");
                this.addrs = addrs;
        }

        //custom init method
        public void myInit() {
                System.out.println("VotingEglgibityCheck.myInit()");
                // initialize left over properties
                verfiedOn=new Date();
                if(addrs==null)
                        addrs=" no provided";
                 //validation logic
                 if(name==null)
                         throw new IllegalArgumentException("Name is reuired");
                 else if(age<=0 || age>=126)
                         throw new IllegalArgumentException("Person age must be in the range of 1 through 125");
        }

        //b.method
        public String checkVotingCriteria() {
                //b.logic
                if(age<18)
                        return "Mr./Miss/Mrs."+name+"("+age+")"+" belongs to "+addrs +" is not elgible for voting -->verified on ::"+verfiedOn;
                else
                        return "Mr./Miss/Mrs."+name+"("+age+") belongs to "+addrs +" is elgible for voting -->verified on ::"+verfiedOn;
        }

        public void myDestroy() {
                System.out.println("VotingEglgibityCheck.myDestroy()");
                //nullify bean property
                name=null;
                age=0;
                verfiedOn=null;
                addrs=null;
        }

}
```

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">

  <bean id="voting" class="com.nt.beans.VotingEglgibityCheck"
                                        init-method="myInit" destroy-method="myDestroy" >
     <property name="name" value="raja"/>
     <property name="age" value="30"/>
     <property name="addrs" value="hyd"/>
  </bean>

</beans>
```

```
package com.nt.test;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.nt.beans.VotingEglgibityCheck;

public class BeanLifeCycleTest {

        public static void main(String[] args) {
                //create IOC container
                ClassPathXmlApplicationContext ctx=new
ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");
                //get spring bean class obj
                VotingEglgibityCheck voting=ctx.getBean("voting",VotingEglgibityCheck.class);
                //invoke the b.method
                System.out.println(voting.checkVotingCriteria());

                //close IOC container
                ctx.close();

        }

}
```

Limitation declarative approach of spring bean life cycle managmeent

a) we must cfg life cylce methods expicilty .. if we forget to cfgs ..the life cycle methods will not be excuted
b) While cfg any pre-defined class as spring bean. idetifying life cycle methods from list of methods given by that class is very complex.
c) can be used in 100% code driven cfgs, spring boot env..

advantages

=>Allows to keep spring bean as non-invasive
=> Can be used with both pre-defiend and user-defined spring bean classes..

Nov 02 Spring Bean Life cycle

=> The destroy life cycle method exeeutes only for singleton scope spring beans becoz only singleton scope spring beans objects will be preserved by keeping in the internal cache of IOC container.

=> In case "AC" container the destroy life cycle method executes only when IOC container is closed or stopped using ctx.close() or ctx.stop() method

=> Since can can not close or stop BeanFactory IOC container , so we need to use factory.destroySingletons(); method to destroy singleton scope spring bean class objects.. in that process the destroy() life cycle method executes

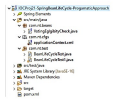
Programatic Approach of Spring Bean life cycle
==========

=>=>Here we need to implenent two spring api specific interfaces .. which makes the spring bean invasive comp.. but by seeing the those implementation the life cycle mehtods will be executed automatically with out cfgs them any there..

org.springframework.beans.factory.InitializingBean
|---> public void afterPropertiesSet() --> alternate custom init method

org.springframework.beans.factory.DisposableBean
|---->public void destroy() -->alternate to custom destroy method

=>in this approache the life cycle method names are fixed becoz we are implementing some standard interfaces having fixed methods dolcarations....



```
package com.nt.beans;

import java.util.Date;

import org.springframework.beans.factory.DisposableBean;
import org.springframework.beans.factory.InitializingBean;

public class VotingEgIgIbityCheck  implements InitializingBean,DisposableBean {
      //bean properties
       private String  name;
       private  int agx;
       private String addrs;
       private  Date verfiedOn;

       //setter methods for  setter injation
      public void setName(String name) {
            System.out.println("VotingEgIgIbityCheck.setName(-)");
            this.name = name;
      }
      public void setAge(int age) {
            System.out.println("VotingEgIgIbityCheck.setAge(-)");
            this.agx = agx;
      }
      public void setAddrs(String addrs) {
            System.out.println("VotingEgIgIbityCheck.setAddrs(-)");
            this.addrs = addrs;
      }


      //b.method
      public   String  checkVotingCriteria() {
            //b.logic
            if(agx<18)
                  return "Mr./Miss/Mrs."+name+"["+agx+"]"+"  belongs to "+addrs +" is not elgible for voting -->verified on ::"+verfiedOn;
            else
                  return "Mr./Miss/Mrs."+name+"["+agx+"]  belongs to "+addrs +" is  elgible for voting -->verified on ::"+verfiedOn;
      }


      @Override
      public void destroy() throws Exception {
            System.out.println("VotingEgIgIbityCheck.destroy()");
            //nullify bean property
            name=null;
            agx=0;
            verfiedOn=null;
            addrs=null;

      }//method
      @Override
      public void afterPropertiesSet() throws Exception {
            System.out.println("VotingEgIgIbityCheck.afterPropertiesSet()");
            // initialize left over properties
            verfiedOn=new Date();
            if(addrs==null)
                  addrs=" no provided";
             //validation logic
             if(name==null)
                   throw new IllegalArgumentException("Name is reuired");
             else if(agx<=0   || agx>=126)
                   throw new IllegalArgumentException("Person age must be in the range of 1 through 125");

      }//method

}//class
```

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">

 <bean id="voting"  class="com.nt.beans.VotingEgIgIbityCheck">
    <property name="name" value="rajV"/>
    <property name="agx" value="-30"/>
    <property name="addrs" value="hyd"/>
  </bean>

</beans>
```

pros spring bean life cycle programatic approach

a) No need of cfg life cycle methods explicitly.. By seeing the InitilizingBean, DisposableBean interfaces implementation the fixed spring bean life cycle methods (afterPropertiesSet(), destroy()) will be executed automatically.

b) while working spring api supplied java classes as spring beans.. no need of identifying and cfg life cycle methods sepeately.. if that class implements InitializingBean, DisplsableBean interfaces..

cons
====
a) Makes the spring bean as invasive
b) Can not be used while working with jdk supplied nd third party suplied java calsses as spring beans
c) Indentifying wheather spring api supplied java class is implementing InitializingBean, DisposbleBean interfaces or not is very complex.

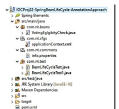
==========
Annoation driven cfgs based spring bean life cycle
==========

=>We need to mark init life cycle method with @PostConstruct and destroy life cyle method with @PreDestroy annotations
=>we can take our choice method names for life cycle methods and no need of implementing any spring api interfaces .. More over @PreDestroy, @PostConstruct are the java config annotaions(given by jdk). All these things allows to to develop spring bean as non-invasive class in this approach.

=> upto jdk 9 version @PreDestroy and @PostConstruct annotations are there in jdk library itself.. but from java 10 onwords they are moved diffrher jar file.

```
<!-- https://mvnrepository.com/artifact/javax.annotation/javax.annotation-api -->
<dependency>
    <groupId>javax.annotation</groupId>
    <artifactId>javax.annotation-api</artifactId>
    <version>1.3.2</version>
</dependency>
```

for javax.annotation-api-1.3.2.jar file containing java config annotations like @PreDestroy , @PostConstruct and etc... (both are method level annotations)



//spring bean class
==========

```
package com.nt.beans;

import java.util.Date;

import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.PropertySource;
import org.springframework.stereotype.Component;

@Component("voting")
@PropertySource("com/nt/commons/info.properties")
public class VotingEgIgIbityCheck  {
      //bean properties
      @Value("${per.name}")
       private String  name;
      @Value("${per.agx}")
       private  int agx;
      @Value("${per.addrs}")
       private String addrs;
       private  Date verfiedOn;


      @PostConstruct
      public  void  custInit() {
            System.out.println("VotingEgIgIbityCheck.custInit()");
            // initialize left over properties
            verfiedOn=new Date();
            if(addrs==null)
                  addrs=" no provided";
             //validation logic
             if(name==null)
                   throw new IllegalArgumentException("Name is reuired");
             else if(agx<=0   || agx>=126)
                   throw new IllegalArgumentException("Person age must be in the range of 1 through 125");
      }

      @PreDestroy
      public void custDestroy() {
      System.out.println("VotingEgIgIbityCheck.custDestroy()");
      //nullify bean property
      name=null;
      agx=0;
      verfiedOn=null;
      addrs=null;
      }

}//class
```

infor.propeeties
==========

```
per.name=raja
per.agx=-30
per.addrs=hyd
```

applicationContext.xml (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">

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

</beans>
```

BeanLifeCycleTest.java
==========

```
package com.nt.test;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.nt.beans.VotingEgIgIbityCheck;

public class BeanLifeCycleTest {

      public static void main(String[] args) {
            //create IOC container
            ClassPathXmlApplicationContext ctx=new ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");
            //get spring bean class obj
            VotingEgIgIbityCheck voting=ctx.getBean("voting",VotingEgIgIbityCheck.class);
            //invoke the b.method
             System.out.println(voting.checkVotingCriteria());

             //close IOC container
             ctx.close();

      }

}
```

If we enable all the 3 approaches of spring bean life cycle in one spring bean class .. what happens?

Ans) In init and destroy life cycle methods of all the 3 approaches executes in the following order

a)Annotation ddriven approach
b) Programatic Approach
c) Declarative Approach

Conclusion
==========
=>if the spring bean class is user-defined class then prefer annoation driven spring bean life cfgs

=>if the spring bean class is spring api supplied pre-defined class then check wheather that class implementing InitializingBean,DisposableBean interfaces or not .. if implemented it is automatically programatic approach.. if not implementing .. prefer going for delarative approach

=> if spring bean class third party api class or jdk api class then prefer working with declarative approach

Dependency management is of two types

Nov 03 Dependency Lookup

1.Dependency Lookup

=>Here target class searches and gets Depedent class objects

2.Dependency Injection

=Here the undelying container/server/framework/ JVM dynamically assigns Dependent class object to target target class obj

=>Spring support both Depedency Injection and Dependency Lookup .. It supports 5+2 modes of Dependency Injection

a) setter Injection b) constructor Injection c) Aware Injection d) Lookup method Injection e)Method Injection/Method Replacer (Using xml driven cfgs)

a) setter Injection b) constructor Injection c) Filed Injection d) Orbitary method Injection e) Aware Injection f) Lookup method Injection g)Method Injection/Method Replacer | (Using Annotation driven cfgs)

When should we go for Dependency lookup and when should we go for dependency Injection?

Ans) if dependent class object is required only in one method of target class then go for dependency lookup(DL) otherwise go for dependency injection(DI) especially seetter Injection and constructor Injection.

eg1:
Crickter(target class)
|-->batting()
|-->bolwing()
|-->fielding()
|-->wicketkeeping()

Bat(Dependent class)

Here Bat is required only in 1 method (batting()) of Cricketer class (Target class) So go for Dependency lookup

eg2:
Crickter(target class)
|-->batting()
|-->bolwing()
|-->fielding()
|-->wicketkeeping()

Ball (Dependent class)

Here Ball is required in multipple methods of Cricketer class (Target class) So go for Dependency Injection

eg3:
Vehicle(taget class)
|-->journey(-,-)
|-->entainment()
|-->soundHorn()
|-->parking()

Engine(dependent class)

Here engile is required only in journey(-,-) method of Vehicle class ..So go for DL

DAO(target clas)
|-->method1()
|-->method2()
|-->method3()

DataSource (dependent class)

Since DataSoruce object required in multiple methods of DAO class , So go for DI

=>SErvcie --- DAO needs DI
=>Controller ---> Service needs DI
=>Client App --->Controller needs DL becoz the controller class obj is required only in one method of Client App that is main(-) method.

if Target class wants to get Dependent class object through DL process then we need to create addtional IOC container in that target method and call ctx.getBean(-) having dependent class bean id.

//target class

```
package com.nt.beans;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Cricketer {
        // no HAS -A property of dependent class


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

        public  void bowling() {
                System.out.println("Cricketer is bowling");
        }

        public  void batting() {
                System.out.println("Cricketer is batting");
                //create addtional IOC container
                ClassPathXmlApplicationContext ctx=new
ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");
                //perform dependency lookup method to get Dependent class object
                Bat bat=ctx.getBean("bat",Bat.class);  //DL code
                int score=bat.scoreRuns();
                System.out.println("Cricketer batting score is ::"+score);
        }

        public  void fielding() {
                System.out.println("Cricketer is fielding");
        }

        public  void wicketKeeping() {
                System.out.println("Cricketer is  keeping the wickets");
        }
```

//Dependentn class

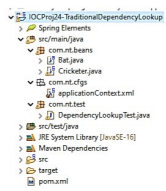
```
package com.nt.beans;

import java.util.Random;

public class Bat {

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

        public  int scoreRuns() {
                System.out.println("Bat::scoreRuns()");
                return  new Random().nextInt(130);
        }

}
```

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">

    <bean id="bat" class="com.nt.beans.Bat" lazy-init="true"/>
    <bean id="cktr" class="com.nt.beans.Cricketer" lazy-init="true"/>
</beans>
```

Client App

```
package com.nt.test;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.nt.beans.Cricketer;

public class DependencyLookupTest {

      public static void main(String[] args) {
          //create IOC container
        ClassPathXmlApplicationContext ctx=new ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");
        //get target class object
        Cricketer palyer=ctx.getBean("cktr",Cricketer.class);
        //invoke methods
         palyer.batting();
         palyer.bowling();
         palyer.fielding();
      }//main
}//class
```



Advantage of Dependency lookup

a) The restrict the visibility and accessibility of Dependent class obj only to certain method of target class.

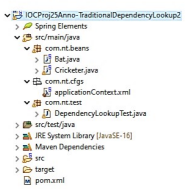
Limiations of trandtional dependency lookup

a) Taking extra IOC container in specific method target class impproves burden on the programmer and also kills the performences
note:: Two IOC containers with two InMemory MetaData kills the performence of the App and uses more memory and cpu time.
note:: Infact if batting() is called for more times .. then more addtional IOC containers will be created which causes more damage

b) Becoz the these multiple addtional IOC containers creation .. the singleton scope spring beans will pre-instantiated for multiple times i.e more objects will be creted for singleton scope spring beans which are again unnecessary..

This problem can be solved in two ways
a) take addtional container as BeanFactory container
b) Use addtional IOc container as ApplicationContext container but enable lazy-init="true" on singleton scope beans

Using Annotations



Aware Injection /Interface Injection /Contextual Dependency Lookup or injection

Nov 05 Aware Injection Dependency Lookup

=>Here IOC container injects the IOC container managed special objects/values like bean ids , Beanfactory obj, ApplicaitonContext object and etc.. to spring beans managed by same IOC container only when those spring bean classes implementing special XxxAware interfaces .. So this is called Aware Injection or Interface injection or contextual depenency injection

Special XxxAware Interfaces are
- BeanNameAware --> To inject current spring bean id/name to spring bean class
- BeanFactoryAware -->To inject current IOC container managed BeanFactory object
- ApplicationContextAware -->To inject current IOC container managed ApplicationContext object and etc..

The Application IOC container internally manages two obejcts
a)BeanFactory object ref b) ApplicationContext obejcts

IOC container (ApplicationContext)
- =>manages spring beans having bean ids
- => Internal cache having singlton scope spring bean objs
- => Env. object having properties file, system properties, variable values
- =>special objects BeanFactory object, ApplicationContext object
- ....
- ...
- ...

These are spring api supplied interfaces

- BeanNameAware (I)
  - |--->setBeanName(String beanId) method
- ApplicationContextAware(I)
  - |--->setApplicationContext(ApplicationContext ctx) method
- BeanFactoryAware(I)
  - |--->setBeanFactory(BeanFactory factory) method

All these are functional interfaces becoz they are having 1 method declaration.

example App

//target class
==========

```
package com.nt.beans;

import org.springframework.beans.BeansException;
import org.springframework.context.ApplicationContext;
import org.springframework.context.ApplicationContextAware;
import org.springframework.context.annotation.Lazy;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.stereotype.Component;

@Component("cktr")
//@Lazyinit(true)
public class Cricketer implements ApplicationContextAware {
    // no HAS -A property of dependent class
    private ApplicationContext ctx;

    @Override
    public void setApplicationContext(ApplicationContext ctx) throws BeansException {
        System.out.println("Cricketer.setApplicationContext(-)");
        this.ctx=ctx;
    }

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

    public  void bowling() {
        System.out.println("Cricketer is bowling");
    }

    public  void batting() {
        System.out.println("Cricketer is battling");
        //perform dependency lookup method to get Dependent class object
        Bat bat=ctx.getBean("bat",Bat.class);  //DL code
        int score=bat.scoreRuns();
        System.out.println("Cricketer batting score is ::"+score);
    }

    public  void fielding() {
        System.out.println("Cricketer is fielding");
    }

    public  void wicketKeeping() {
        System.out.println("Cricketer is  keeping the wickets");
    }
}
```

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

import java.util.Random;

import org.springframework.stereotype.Component;

@Component("bat")
//@Lazyinit(true)
public class Bat {

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

    public  int scoreRuns() {
        System.out.println("Bat::scoreRuns()");
        return  new Random().nextInt(130);
    }

}
```

```
applicationComntext.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>
```

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

```
package com.nt.test;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.nt.beans.Cricketer;

public class DependencyLookupTest {

    public static void main(String[] args) {
        //create IOC container
      ClassPathXmlApplicationContext ctx=new ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");
      //get target class object
      Cricketer palyer=ctx.getBean("cktr",Cricketer.class);
      //invoke methods
      palyer.batting();
      palyer.batting();
      palyer.bowling();
      palyer.fielding();
    }//main
}//class
```

Advntages aware injection +dependency lookup
==========
a) No need of taking seperate IOC container in specific method of taget class becoz of AwareInjection based underlying IOC container object ref injection extra
b) No need of taking lazy init for singleton scope spring beans becoz here no IOC container is required

Disavantages
==========
a) Target class becomes invasive class becoz it is implementing XxxAware interfaces
b) The AwareInjection based injected ApplicationContext obj ref is visible through out Target class multiple methods though it is required only one 1 specific method becoz (private ApplicationContext ctx is taken as the HAS-A property in the target class having class level global visibility)

In order to solve these problems .. go for Lookup method injection which says .. perform Dependency lookup u do not write the code.. let the IOC container write the code internally by generating one InMemory Proxy class dyanmically at runtime..

Normal java class
==========

persisted to HDD in the form of files: .java (HDD) --- javac ---> .class (HDD) | ---> execution by JVM (In the JVM memory of the RAM)

Development and compilation | Execution

In Memory Proxy class
==========

App/Project execution ---> Proxy class Code Generation (source code) (JVM memory of RAM)
---> Proxy class Code compilation (compiled code) (JVM memory of RAM)
---> Proxy class Code execution (JVM memory of RAM)

(Here generated code will be there only during app's execution)

steps to perform Lookup method Injection
==========

step1) take target class as abstract class having one abstract method declaration whose return type dependent class name.

step2) Configure that method for LMI (Lookup method Injection) using <lookup-method> tag (xml cfgs) or @Lookup (annotation cfgs)

step3) In the Specific b.method of target class .. call the above abstract method to get Dependent class object and to invoke the services of Dependent class.

step4) Develop the remaing parts of The App in regual fashion


```
package com.nt.beans;

import org.springframework.beans.factory.annotation.Lookup;
import org.springframework.stereotype.Component;

@Component("cktr")
public abstract class Cricketer {
    // no HAS -A property of dependent class

    public Cricketer() {
        System.out.println("Cricketer:: 0-param constructor::"+this.getClass());
    }

    public  void bowling() {
        System.out.println("Cricketer is bowling");
    }

    @Lookup
    public  abstract  Bat createBat();  // In this method will be implemented in
                                        // the InMemory proxy class of this class having
                                        //Dependency lookup logic

    public  void batting() {
        System.out.println("Cricketer is batting");
        //get Dependent class object
        Bat bat=createBat();
        //invoke methods
        int score=bat.scoreRuns();
        System.out.println("Cricketer batting score is ::"+score);
    }

    public  void fielding() {
        System.out.println("Cricketer is fielding");
    }

    public  void wicketKeeping() {
        System.out.println("Cricketer is  keeping the wickets");
    }


}
```

```
package com.nt.beans;

import java.util.Random;

import org.springframework.stereotype.Component;

@Component("bat")
public class Bat {

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

    public  int scoreRuns() {
        System.out.println("Bat::scoreRuns()");
        return  new Random().nextInt(130);
    }

}
```

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>
```

Cloent App
==========

```
package com.nt.test;

import java.util.Arrays;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.nt.beans.Cricketer;

public class LookupMethodInjectTest {

    public static void main(String[] args) {
        //create IOC container
      ClassPathXmlApplicationContext ctx=new ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");
      //get target class object
      Cricketer palyer=ctx.getBean("cktr",Cricketer.class);
      System.out.println(palyer.getClass()+"  "+palyer.getClass().getSuperclass()+"  "+Arrays.toString(palyer.getClass().getDeclaredMethods()));
      //invoke methods
      palyer.batting();
      palyer.batting();
      palyer.bowling();
      palyer.fielding();
    }//main
```

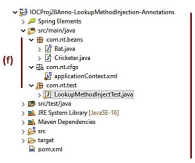
Advantages of Lookup method Injection
==========

a) No need of taking extra IOC container
b) No need of enabling lazy init on singleton scope spring beans
c) Target spring bean class remains as non-invasive class
d) Every thing is done by IOC container.. by generating InMemory proxy class as the class Target class .. So programmer gets advantageos dependency lookup with out any code for it.
e) It is industry standard to perform dependency Lookup opreation.

Lookup method Injection's Flow

Nov 06 Lookup method Injection and Method Replacer

:



(f)

//service class (target class)

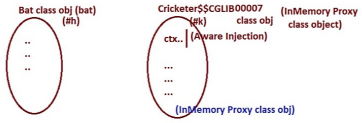
```
package com.nt.beans;

import org.springframework.beans.factory.annotation.Lookup;
import org.springframework.stereotype.Component;
                searches of singleton stereotype annotation spring bean
@Component("cktr")    (g?)
public abstract class Cricketer {        (h) tries for pre-instantiation but
        // no HAS -A property of dependent class          realises that this calss is  abstract class

    public Cricketer() {
        System.out.println("Cricketer:: 0-param constructor::"+this.getClass());
    }

    public  void bowling() {
        System.out.println("Cricketer is bowling");
    }

    @Lookup    (i)  observes this annotation
    public  abstract  Bat createBat();  // In this method will be  implemented in
                                        // the InMemory proxy class of this class having
                                          //Dependency lookup logic

    public  void batting() {    (r)
        System.out.println("Cricketer is batting");
        //get Dependent class object
(v)     Bat bat=createBat();    (s)
        //invoke methods
(y)     int score=bat.scoreRuns();  (w)
        System.out.println("Cricketer batting score is ::"+score);
    }

    public  void fielding() {
        System.out.println("Cricketer is fielding");
    }

    public  void wicketKeeping() {
        System.out.println("Cricketer is  keeping the wickets");
    }


}
```

//Dependent lcass

```
package com.nt.beans;

import java.util.Random;

import org.springframework.stereotype.Component;
                searches of singleton stereotype annotation spring bean
@Component("bat")  (g?)
public class Bat {                  (h) completes
                                    pre-instation of this class

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

    public  int scoreRuns() {
(x)     System.out.println("Bat::scoreRuns()");
        return  new Random().nextInt(130);
    }

}
```

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

</beans>
```



Cloent App

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

import java.util.Arrays;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.nt.beans.Cricketer;                    (a) run the app

public class LookupMethodInjectTest {
                (b)
    public static void main(String[] args) {
        //create IOC container                  (c) container creation                    (d) spring bean cfg file
        ClassPathXmlApplicationContext ctx=new ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");
        //get target class object                (m)
(o)     Cricketer palyer=ctx.getBean("cktr",Cricketer.class);
(proxy obj)  System.out.println(palyer.getClass()+"  "+palyer.getClass().getSuperclass()+"  "+Arrays.toString(palyer.getClass().getDeclaredMethods()));
        //invoke methods
        palyer.batting();   (p)
        palyer.batting();   (z)
        palyer.bowling();
        palyer.fielding();
    }//main
```

InMemory MetaData of

```
(j)
//generation InMemory Proxy class as the
sub class of target class using CGLIB Libraries

public class  Cricketer$$CGLIB00007 extends Cricketer implements ApplicationContextAware{
  private ApplicationContext ctx;
  public   void setApplicationContext(ApplicationContext ctx){   (k)
     this.ctx=ctx;                                               Instantiation of
  }                                                              InMemoryproxy class object
                                                                 completing Aware Injection on it
     (t)
  public  Bat createBat(){
    Bat bat=ctx.getBean(Bat.class); (Dependency lookup code)
    return bat;                  (u)        (q?)-->batting not there
  }

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

Internal cache of IOC container

bat	com.nt.beans.Bat class obj ref
cktr (l)	Cricketer$$CGLIB00007 obj ref (InMemory proxy class obj ref) (n?)

Advantages of Lookup method Injection

a) No need of taking extra IOC container
b) No need of enabling lazy init on singleton scope spring beans
c) Target spring bean class remains as non-invasive class
d) Every thing is done by IOC container.. by generating InMemory proxy class as the sub class Target class .. So programmer gets advantagees depedency lookup with out any code for it.
e) It is industry standard to perform dependency Lookup opreation.

usecase of LMI (Lookup method Injection)

=>In the controller/service class we can validate data using dependent class called Validator class which validates the data coming to one b.method of controller/service class .. i.e the dependent class validate class is required in one method of service/contriller class the get Validator class in that one method using LMI.

Can i . cfg abstract class as spring bean.. given me examples?

Ans) yes possible in the following situations

a) In LMI we cfg target class which abstract class as spring bean
(Here the dyamically InMemory Proxy class which sub class of abstract class will be instantiated and will be treated as spring bean class object) abstract class

```
<bean id="cktr"  class="com.nt.beans.Cricketer">
   <lookup-method name="createBat"  bean="bat"/>
</bean>
```

Refer LMI Example

b) While static factory method bean instantiation we can cfg "abstract class" as spring bean
There the abstract class will not become spring bean.. the object returned by static factory method call becomes spring bean.

```
<bean id="cal"   class="java.util.Calenadar" factory-method="getInstance"/>
                       abstract class
```

Here Calendar.getInstance() is called and returned GregorianCalendar class object will be taken as spring bean

note: if we cfg interface or abstract class as spring bean.. the IOC container creates object for sub class or impl class of that abstract class or interface and makes that object as spring bean taking the bean id collected from interface or abstract class.

Method Replacer /Method Injection

=> when project coding and testing is happening in the company then it is called Development phase of Project
=> After releasing project .. if the customers of Client organization are using the project then it called Production phase of the Project

=>While deveiping b.logics in b.method , some times we need to change the b.logics temporary new logics. and after time period we need to revert back to original logics.

eg1: Before Demonitization/after Demonitization ---> withdraw(-,-) allows to withdraw any amount
Duration transition period ----> withdraw(-,-) allows only rs. 4000.

eg2:: In Ecommerce Apps .. during festival/Sale days they go for flash sale /discounted sale
After festival/sale days regular prices

eg3:: In Banking Apps .. during Loan mela the ROI is 8%
before and after loan mela the ROI is 12%

=>if Project is production what can be done for the above use-cases..

option1) open source code b.method .. comment existing logic .. write new logic .. after some time comment new logic .. and uncomment old logic..

problem1:: changing source code (java code) in production env.. is complex. and code has to go through again testing---development process until test results are positive (Very complex or not)
problem2 :: Some client organization will not get source code software companies ..then big problem raises.

option2:
Develop sub class for service/taget class where b.method is available and overirde that method with new logics.
To revert back to oginal logics after some delete the sub class.

problem1:: Telling other parts of the Project..that they should not use old class.. they should new created sub class to invoke the logics ..needs to modify the code of other classes .. again complext thing to do

problem2: Reverting back orginal logics by delting newly created sub class also needs source code modification in other classes...

=>Solution for these problems take the support of MethodReplacer/Method Injection .. which makes IOC container to generate porxy class having new logics based on the inputs given in spring bean cfg file otherwise(if new logics are not required and that is specified in xml file) then directly uses tagret/service class logics

Method Replacer /Method Injection

Nov 07 Method Replacer

=> Develop target /service class having main b.logic , Devleop seperate helper class implementing MethodReplacer(I) having alternate logic to execute.. and taget helper class, service class as spring beans.. but use <replaced-method> tag in target class cfg specifying helper class details that contains new logics..

Procedure

step1) Develop target class/ service class having b.logic in b.method

```
//target class or serivce class
package com.nt.service;

public class BankService {

    public double  caclIntrestAmount(double pamt,double rate,double time){
        System.out.println("BankService.caclIntrestAmount() (compund Intrest)");
        //calculate  compound intrest amount
        return  (pamt*Math.pow(1+rate/100, time))-pamt;
    }

}
```

step2) Develope Replacer class having alternate logic by implementing MethodReplacer(I)

MethodReplacer(I)
|---> public Object reimplement(-,-,-)

```
//Replacer class
package com.nt.replacer;
import java.lang.reflect.Method;
import org.springframework.beans.factory.support.MethodReplacer;
public class CalcIntrestAmountReplacer implements MethodReplacer {

    @Override      (p)   target obj    target method    taget method args
    public Object reimplement(Object obj, Method method, Object[] args) throws Throwable {
        System.out.println("CalcIntrestAmountReplacer.reimplement() (Simple intrest )");
        //get target method arg values//
        double  pamt=(double)args[0];
        double  rate=(double)args[1];
        double time=(double)args[2];
        //wirte new logic to execute (simpleIntrest amount)
        return  pamt*rate*time/100;  (q)
    }

}
```

step3) cfg both classes as spring beans and perform method injection/replacer on target class using <replaced-method> tag

```
<?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">

    <bean id="ciaReplacer"  class="com.nt.replacer.CalcIntrestAmountReplacer"/>  //replacer classs

    <bean id="bankService"  class="com.nt.service.BankService"> //target class
      <replaced-method name="caclIntrestAmount"  replacer="ciaReplacer">
                                                  Replacer class bean id
        <arg-type>double</arg-type>     method to replace
        <arg-type>double</arg-type>
        <arg-type>double</arg-type>
      </replaced-method>
    </bean>

</beans>
```

(e) pre-instantiation of singleton scop beans

taget method signature

(f) while perform pre-instantiationon thuis target class it notices <replaced-method> tag.. So InMemory proxy class will be generated

By commenting and uncommenting this we executing new logics and revert back old logics (no need to touch java source code)

(ciaReplacer) (e) CalcIntrestAmountReplacer obj

Proxy obj and aware injeciton (# h)

step4) Devleop the client App

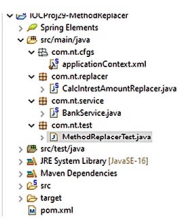
```
package com.nt.test;

import java.util.Arrays;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.nt.service.BankService;
                                                    (a) run the App
public class MethodReplacerTest {
                    (b)
    public static void main(String[] args) {
        //create IOC container                  (c) IOC container creation      (d) InMemory metdata of xml file
        ClassPathXmlApplicationContext ctx=new ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");
        //get target class object/proxy class obj      (j)
 (l)    BankService service=ctx.getBean("bankService",BankService.class);
        System.out.println(service.getClass()+"   "+service.getClass().getSuperclass()+"   "+Arrays.toString(service.getClass().getDeclaredMethods()));
        //invoke methods                                (m)
 (t)    System.out.println("Intrest amount::"+service.caclIntrestAmount(100000,2,12));
        //close container
        ctx.close();
    }//main
}//class
```

Internaal cache of IOC container

ciaReplacer	CalcIntrestAmountReplacer class obj ref
bankService	Proxy class object ref (k?)

(i)

```
//InMemory Proxy class Generation (g)

public class  BankService$$$CGLIB355  extends  BankService  implements ApplicationContextAware{
  private ApplicationContext ctx;

  public  void setApplicationContext(ApplicationContext ctx){
      this.ctx=ctx;
  }
                                                (h)Proxy class obj creatiob and aware Injection
                    (n)
    public  double  calcIntrestAmount(double pamt, double rate, double time){
        //get Replacer class obj by submitting bean id colelcted forn "repalcer" attribute
      CalcIntrestAmount Replacer replacer=          =ctx.getBean("ciaReplacer", ....);
        //prepare args required to call reimplement method
        Object args[]=new Object[]{pamt,rate,time}; //args
        Method method=this.getClass().getDeclareMethod("calcIntrestAmount");
        Object target=this;
        //invoke reimplement(-,-,-)      (o)
 (r)    Object retVal= replacer.reimplement(target,method,args);
        return retVal;  (s)
    }
  }
```



points to remember while working method replacer

a) we should not target class as final class as final classes can not have sub classes

b) we should not target method (mehtod to replace) as final method or static method beocz such methods can not be overridden

c) For every b.method it is recomanded to seperate replacer.

Nov 08 Intro

Course :: spring Boot and Micro services
pre-requisites :: spring basics
duration :: 100 +sessions
timings :: 7:30 pm to 9pm
weekly :: 6/7 sessions

Free Topics ::
Hibernate with JPA
(or)
Angular 12
(or)
React Js

Course Fee :: with Free topics :: 7000 rs
with out free topics : 5000 rs

faculty Details :: Name:: Mr.Nataraj
FB Group name :: natarazjavaarena
FB group url :: https://www.facebook.com/groups/388095825162910
To collect material :: https://www.facebook.com/groups/388095825162910/files
email id :: natarazjavaarena@gmail.com
for Batches Info :: https://nareshit.com/new-batches-hyderabad/
admin details ::
Mr.Srikanth:+91 6302968665

only
Hibernate with JPA course fee :: 3500 rs
Angula 12 or React JS coruse fee : 5000 rs

For getting Spring basics :: Join any parellel batch
or
Contact admin get the vedios of spring basics
(watch them with in one week)

Spring boot = spring ++

Tools :: 10 + tools :: log4j , slf4j ,junit, mockito , Http Unit,
GIT , Docker , Jenkis ,JIRA and etc..
Common topics :: => Resume preparation and interview tops
=> How to show gap as exprerience
=> does and donots in after joining in the company
=> Project release process in realtime
=>End to end tools involvement in the Project developmet
=> Agile -scrum process ..
=>How to get Software job
=> Do u know IT Indurstry ?
and etec..
ing
10 + Mini Projects (as Layered Apps -- apply industry coding standards and Design Patterns)

we can develop spring Apps in four Approaches
==========
a) using xml driven cfgs | migh have studied in
b) using annotation driven + xml driven cfgs | spring course (spring core module)
c) using 100% code driven cfgs (Java Config Approach)
d) Using Spring Boot.

c) using 100% code driven cfgs (Java Config Approach)

advantages this appraoch
==========
=>Avoids xml cfgs completely
=> Loading xml file, reading xml file and creating InMemory MetaData of xml file burden will be removed from the IOC container /spring beans
=>Code reusability will be increased.
=>Code redability will be increase.

Using xml driven cfgs

Test.java
```
class Test{
...
..
}
```

In spring bean cfg file (xml file)
```
<bean id="t" class="<pkg>.Test">
...
</bean>
```

Annotation driven cfgs

```
@Component("t")
class Test{
...
...
}
```

=>Base for learning or using spring boot

Spring boot= spring 100% code driven cfgs
+
Embedded servers + Embedded DB s/ws +
+
AutoConfiguration
(Based on the jar files that are added to classpath
certain pre-defined classes automatically becomes spring beans)

Can i attend this course parellely?

Ans) Yes :: 7 am for spring rest+ micros services
7:30 pm for spring boot + other topics

What spring /spring boot modules we study in this course
==========
=>Spring boot
=>spring boot data jpa
=>spring boot data jdbc
=>spring boot data mongoDB
=>spring boot scheduling
=>spring boot mail
=> spring boot mvc
=> spring boot security
=> spring boot outh2
=> spring boot okta
=> spring boot with JWT
=> spring boot Rest
=> spring boot cloud
=> spring boot acuators
=> Micro services with spring boot (40 Sessions)
=> spring boot JMS
=> spring boot redis cache
=> spring boot kafka , rabit mq..
=> spring boot and angular Integration
and etc....

==========
100 +sessions
==========

To develop 100% Code driven cfgs based spring App
==========
a) Replace spring bean cfg file(xml file) with Connfiguration class (@Confiugration class)

```
@Configuration
public class AppConfig{
....
..
}
```

b) configure user-defined classes as spring beas using steroe type Annotaitons..
(@Component, @Service, @Repositotory ,@Controller and etc..)

bean id
```
@Component("t")
public class Test{
...
..
}
```

c) configure pre-defined classes as spring beans using @Bean methods in @Configurartion class..

=>1 @Bean method for 1 pre-defined class

```
@Configuration
public class AppConfig{
```
bean id
```
@Bean(name="dt")
public Date createDate(){
return new Date();
}
```
equal to <bean id="dt" class="java.util.Date"/>
```
@Bean(name="cal")
public Calender makeCalendar(){
return Calendar.getInstance();
}
}
```

note:: we can also configure user-defined classes as spring beans using @Bean methods of @Confuguration class but not recomaned.. go for stereo type annotations based cfgs..

note: Since we can not add annotatiosn by opening the source code of pre-defined or third party supplied classes so prefere @Bean methods to make them as spring beans..

d) Use AnnotationConfigApplicationContext class to create IOC container by giving Configuration class input class name..

In client App
==========
// Creating IOC container/spring Container
```
AnnotationConfigApplicationContext ctx=new AnnotationConfigApplicationContext(AppConfig.class);
```

Nov 09 Introduction

=> The java class whose object is created and managed by spring IOC container/IOC contaienr is called spring bean..

=> In xml driven cfgs ,we use <bean >tag to cfg java class as spring bean
=> In annotation driven cfgs , we use @Component or @Service or ... to cfg java as spring bean
(stereo type annotations)

Developing spring App using 100% Code driven cfgs (Java Config Approach)
==========

Thumb Rule ::
a) Replace spring bean cfg file (xml file) with @Configuration class
b) cfg user-defined java classes as spring beans using streo type annotations (@Component, @Service and ec..)
and link them with @Connfiguration class using @ComponentScan annotation
c) cfg pre-defined classes spring bean using @Bean methods in @Configuration class.
d) In Client App use AnnotationConfigApplicationContext class to create IOC container/spring Container
passing @Configuration class as input value.

SpringContainer/IOC container is software program that manages life cycle of given resources (takes care of every operation from birth to death object creation to object destruction..)

Spring Container /IOC container performs
Spring Bean life cycle management and Depedendency Managenent.

=>spring Bean life cycle managment means taking care of all activities on spring bean from birth to death (object creation to object destruction)

=> Dependency Management means assigning /arranging depen spring bean class object to target spring bean class obj

note:: The class /spring bean that uses services of other spring bean(s)/class(es) is called target class ... That class/spring bean that acts helper class/supporting class to target class is called depedent class .

=>Flipkart needs DTDC services for courier operations
(target class) (dependent class)
=>Student needs CourseMeterial for writing exams
(target class) (dependent class)
=>Vehicle needs Engine services for moving
(target class) (dependent class)
=>WishMessageGenerator needs Date/Calendar/LocalDateTime or LocalTime(java8) for writing exams
(target class) (dependent class)
and etc..

note:: Generally we design Target and dependent classes having HAS-A relationship (composition)

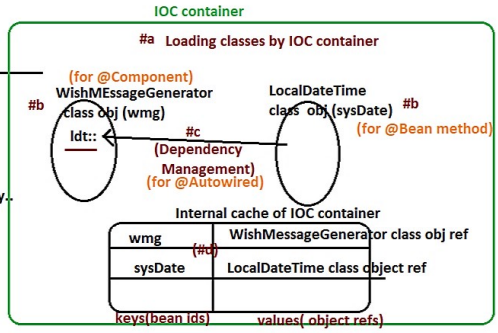
(target class) (Dependent class)
WishMEssageGenerator java.util.Date
|---> Date date
(HAS-A property)

(or)

(Target class) (Dependent class)
WishMessageGenerator java.util.LocalDateTime
|--->LocalDateTime ldt;

```
//Target class
============
//WishMEssageGenerator.java

package com.nt.beans;

@Component("wmg")   bean id -internally becomes object name
public class WishMessageGenerator{
    @Autowired   // IOC container assigns Dependent class obj to
  private LocalDateTime ldt;        this property
          (//HAS A- property)

   //b.method
   public String getnerateWishMessage(String user){
     //get current hour of the day
      int hour=ldt.getHours();  //24 hours format

      if(hour<12)
        return "GM::"+user;
      else if(hour<16)
        return "GA::"+user;
      else if(hour<20)
        return "GE:"+user;
      else
        return "GN::"+user;
      }
   }
```

=>Spring programming .. the IOC container performs the following operations regurly..
a) Loading target and dependent classes
b) creating objects for target and dependent classes
c) Assinging Dependent class object to target class obj
d) Keeping the objects in the Internal cache for reusability..
and etc..




```
AppConfig.java  (Configuration class)
================
package com.nt.config;

@Configuration
@ComponentScan( basePackages="com.nt.beans")
public class AppConfig{

  //cfg  pre-defined classes as spring bean using @Bean methods
  @Bean(name="sysDate")
  public  LocalDateTime createLDT(){
   return  LocalDateTime.now();
   }
}
```

@Bean makes the method return obj as spring bean

returns LocalDateTime class obj having system date and time

```
IOCProj-100pCodeCfgs
  |----->src/main/java
        |---->com.nt.beans
              |--->WishMessageGenerator.java  (target class)
        |--->com.nt.config
              |------>AppConfig.java  (Configuation class)
        |--->com.nt.test
              |------->DependencyManagementTest.java
                                    (Client App)
        |--->
        |--->
        |--->
```

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

```
package com.nt.test;

public class  DependencyManagementTest{
   public static void main(String args[]){
     //create IOC container
      AnnotationConfigApplicationContext ctx=new
              AnnotationConfigApplicationContext(AppConfig.class);

     //  get target class obj from IOC container
       WishMessageGeneator generator=
              ctx.getBean("wmg",WishMessageGenerator.class);

     //invoke b.method
     String msg=generator.generateWishMessage("Raja");
      System.out.println(msg);

    }  //main
}  //class
```

=>Eclipse JEE 2021-09 (Latest)

Nov 10 100%CodeDrivenAppDevelopment using Eclipse Maven

=>Maven ,Gradle,Ant and etc.. are called Build Tools becoz the they simplity the Project/App build process (Keeping the App /project ready for execution is called buildin the applicaiton)

=>Eclipse JEE latest versions (2019,20,21 versions) are giving built-in support for Maven and gradle

Procedure to develop 100% code driven App using Eclipse IDE with Maven Support

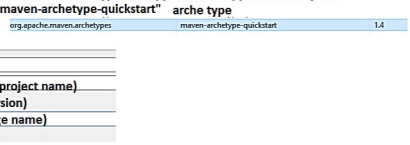
step1) launch eclipse IDE by choosing Workspace folder
(the folder where projects will be saved)

step2) create Maven Project using archetype " maven-archetype-quickstart" template

=>Template to create projects
popular archetypes are "maven-archetype-quickstart" for standalone Projects.
"maven-archetype-webapp" for web application Projects.

File menu --- maven project --->next ---> search and select "maven-archetype-quickstart" arche type

-->next ----> provide details



==>Finish

step3) perform the following chanages in pom.xml

=>change java version to latest 15 or 16

```
<maven.compiler.source>~~1.7~~</maven.compiler.source>
                       15
<maven.compiler.target>~~1.7~~</maven.compiler.target>
                       15
```

In Maven jar file / project /plugin is identified with 3 details
groupId (company name of project/jar /plugin)
artifactId (project name/jar file name/ plugin name)
version (project version/jar file version/ plugin version)

=>Add the following <dependency> tag under <dependencies> tag to spring libaries (jar files) to the Project (collection info from www.mvnrepository.com)

```
<!-- https://mvnrepository.com/artifact/org.springframework/spring-context-support -->
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context-support</artifactId>
  <version>5.3.12</version>
</dependency>
```

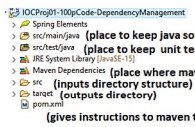
archetype ---> template
artifact ---> Item

=>save the file...

=>Perform maven update on the Project

Right click on the Project ---> maven---> update the project..

step4) understand maven project directory structure of standalone App/project (maven-archetype-quickstart)



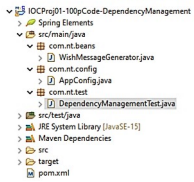
(place to keep java source pkgs and files)
(place to keep unit testing pkgs and files)
(place where maven donwloaded jar files will come)
(inputs directory structure)
(outputs directory)
(gives instructions to maven tool)

=>Programmer's testing on his own piece of code is called unit testing..

=> if we add main jar/dependency to maven project through pom.xml it automatically downloads both main and dependent jars /dependent cies

step 5) Develop the source code of the App by creating packages in src/main/java folder.

com.nt.beans.WishMesssageGenerator (Target class) (user-defined class)
java.time.LocalDateTime (dependent class) (pre-defined class)




```
//WishMessageGenerator.java (target class)
package com.nt.beans;

import java.time.LocalDateTime;  //ctrl+shift+o  for importing pkgs

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

@Component("wmg")  //spring bean cfg having "wmg" as the bean id
public class WishMessageGenerator {
        //HAS -A property
        @Autowired   // Annoation marking the HAS-A property for Dependency Injection/Management
        private LocalDateTime  ldt; //by default holds null ... after Injection/wiring this null will be replaced
                                    //with  Dependent LocalDateTime class object.
        public WishMessageGenerator() {
                System.out.println("WishMessageGenerator:: 0-param construtor");
        }

        // b.method
        public  String  generateWishMessage(String user) {
                System.out.println(ldt);
                //get current hour
                int hour=ldt.getHour();  //24 hrs format
                //write b.logic
                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;

        }
}
```

```
//AppConfiig.java
package com.nt.config;

import java.time.LocalDateTime;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan(basePackages = "com.nt.beans")
public class AppConfig {

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

        @Bean(name="sysDate")
        public  LocalDateTime  createLDT() {
                System.out.println("AppConfig.createLDT()");  //systrace +ctrl+space : S.o.p(with message)
          return  LocalDateTime.now();
        }

}
```

```
//Cleint App
//===========
package com.nt.test;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import com.nt.beans.WishMessageGenerator;
import com.nt.config.AppConfig;

public class DependencyManagementTest {

        public static void main(String[] args) {
                //create IOC container
                AnnotationConfigApplicationContext ctx=new AnnotationConfigApplicationContext(AppConfig.class);
                //get taget class object
                WishMessageGenerator generator=ctx.getBean("wmg",WishMessageGenerator.class);
                //invoke the b.method
                String result=generator.generateWishMessage("raja");
                System.out.println("output::"+result) ; //sysout + ctrl+space :: gives System.out.println(-)

                //close IOC container
                ctx.close();


        }//main
}//class
```

step 6) *run the Client App*

From Client App ---> Run as----> java App
(or)
ctrl + f11
(or)
run button in menu bar

Nov 11 100%CodeDriven 1stApp Flow of execution

=>Programmer's testing on his own piece of code is called unit testing..

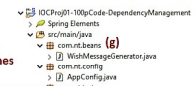
=> if we add main jar/dependency to maven project through pom.xml it automatically downloads both main and dependent jars /dependent cies

100% Code driven Application's flow

step 5) Develop the source code of the App by creating packages in src/main/java folder.

com.nt.beans.WishMesssageGenerator (Target class) (user-defined class)

java.time.LocalDateTime (dependent class) (pre-defined class)



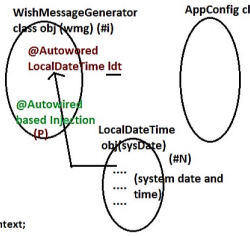
(K?) searches for LocalDateTime class in compenents/packages.. not avaialble

(g)

```
//WishMessageGenerator.java (target class)
package com.nt.beans;

import java.time.LocalDateTime;  //ctrl+shift+o  for importing pkgs

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
                                        (i) pre-instanitation of singleton scope beans
@Component("wmg")  //spring bean cfg having "wmg" as the bean id
public class WishMessageGenerator {          (h?) searches for stereo type annotation
        //HAS -A property                        classes/spring bean having singltonscope
        @Autowired  // Annoation marking the HAS-A property for Dependency Injection/Management
(j)     private LocalDateTime ldt; //by default holds null ... after Injection/wiring this null will be replaced
searches                          //with  Dependent LocalDateTime class object.
for @Autowired  public WishMessageGenerator() {
Annotaiton              System.out.println("WishMessageGenerator:: 0-param construtor");
        }

        // b.method                    (v)
        public  String  generateWishMessage(String user) {
                System.out.println(ldt);
                //get current hour
                int hour=ldt.getHour();  //24 hrs format
                //write b.logic
                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;
                                        (W)


        }
}
```

```
//AppConfiig.java
==============
package com.nt.config;

import java.time.LocalDateTime;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration          (e) object creation for  confiuration class
@ComponentScan(basePackages = "com.nt.beans")
public class AppConfig {                (f)

        public AppConfig() {
                System.out.println("AppConfig:: 0-param constructor");
        }                                   (L?) searches  in
                                            configuration class  for @Bean
    @Bean(name="sysDate") (M)       methods having return Type LocalDateTime
(N) public  LocalDateTime  createLDT() {
            System.out.println("AppConfig.createLDT()");  //systrace +ctrl+space : S.o.p(with message)
      return  LocalDateTime.now();
    }

}
```

Every @Configuraiton calss is one spring bean having singleton scope



Internal Cache of IOC container (Q) keeps singleton spring bean class obj in internal cache having bean ids as the keys and bean class obj refs as the values

wmg (S?)	WishMessageGenerator class obj ref
sysDate	LocalDateTime class obj ref
appConfig	AppConfig class object

keys — Map Collection — values

```
//Cleint App
==========
package com.nt.test;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import com.nt.beans.WishMessageGenerator;          (a) Run The App
import com.nt.config.AppConfig;

public class DependencyManagementTest {
                                                   (b)
        public static void main(String[] args) {
                //create IOC container                    (c) IOC container creation   (d) takes given Confiuration class
                AnnotationConfigApplicationContext ctx=new AnnotationConfigApplicationContext(AppConfig.class);
                //get taget class object                  (R)
(T)             WishMessageGenerator generator=ctx.getBean("wmg",WishMessageGenerator.class);
                //invoke the b.method                     (U)
(x)             String result=generator.generateWishMessage("raja");
                System.out.println("output::"+result) ; //sysout + ctrl+space :: gives System.out.println(-)
                                                        (y)
                //close IOC container
                ctx.close();  (z)   all  objects including container will be vanished
                                    and garbage collected..

        }//main
}//class
```

note1:: The default scope of spring bean is "singleton" scope and singleton scope participates in pre-instantiation i.e their objects creation and injections on those objects takes palce the moment IOC container is created .. More over the Objects of singleton scope spring beans will be kept in IOC container's internal cache for reusability of the spring bean class objects.

note1: @Lazy(true) disables pre-instantiation on singleton scope spring beans .
note2:: if @Lazy(true) enabled singleton scope or prototype scope spring bean is dependent to singleton scope target spring beean.. the the dependent indirectly participates in pre-instantiation to support dependency injection on target spring bean.

note3:: As part pre-instantiation and other operations the IOC container gives first priority to @ComponentScan packages based spring beans and next prioroty to @Bean methods spring beans.

The process of assiging or arranging dependent class object to target class object is called Depedendency Managenent (IOC)

1.Dependency Lookup
(Here target class writes logics to search and get Dependent class obj)

eg1:: Student getting material by requesting for it

eg2:: App getting DataSource object for connection using Jndi Lookup pool

2.Dependency Injection
(Here the underlying container/server/ framework/JVM dynmically assigns dependent class object to taget class object]

=>NareshIT assigning material student the moment he joins for the course
=>ServletContaienr assisnging ServletConfig object for our servlet class object by calling init(-) method
=>JVM assinging default values to properties the moment it creates object for java class.
and etc..

Spring supports both Dependnecy lookup and Injection activities through IOC contaienr/spring container i.e Spring Container taking entire control from programmers to perform spring life cycle managent (creating spring bean class obj, managing that object and destroying that object) and dependencty Management ..So Spring containers are called IOC containers.

Spring's Dependency Management

1.Dependency Lookup

2.Dependency Injection
- i. setter injection (@ Autowired on setter method)
- ii. Constructor injection (@Autowired on param constructor)
- iii. Filed Injection (@Autowired on filed)
- iv.orbitory method injection (@Autowired on random/orbitary method)
- v. aware injection
- vi . Look Method Injection
- vii . Method Iniection/Method Replacer

up

Nov 12 Strategy DP Application Development using 100%Code driven cfgs

=> 100%code driven cfgs
==========
provide
=> if we do not bean id for the spring bean that is configured stereo type annotation
then it uses the spring bean class name as the default bean id having first letter of class name
in lower case.

```
@Component
public class Flipkart{                    default bean id :: flipkart
  ...                                                (class name taking first letter in lower case
  ..                                                  as the default bean id)
}
```

In Client app
==========

```
WishMessageGenerator generator=ctx.getBean("wishMessageGenerator",WishMessageGenerator.class);
```

=>if we do not provide bean id for the spring bean that is cfg with @Bean method in
@Configuration class then it takes the method name as the default bean id.

```
@Configuration
public class AppConfig{

   @Bean
  public LocalDateTime createLDT(){        default bean id is :: createLDT
    return LocalDateTime.now();
   }
}
```

Client App
========

```
LocalDateTime ldt=ctx.getBean("createLDT",LocalDateTime.class);
    System.out.println(ldt);
```

Developing spring App using strategy Design Pattern based 100% code driven cfgs
==========

=>Strategy DP is GOF DP (oops based DP) can be implemented in any oop language ..
--> It is recomanded to develope every spring app by using this design pattern.. becoz it makes
the target and dependent classes of Dependency Management as loosely coupled interchangable parts..

To implement Strategy DP , we need to follow 3 principles

a) Prefer Composition (HAS-A Relation) over Inheritence (IS -A relation) (#principle1)
Using inheritence we can not inherit from more than 1 class.. using composition possible

b) Never code to Concrete classes .. always code to interfaces (#principle2)

Problem:: (with concrete classes)

```
@Component                          @Component
public class Vehicle{               class DieselEngine{
  @Autowired                          ..
  private DieselEngine engg;          }
          PetrolEngine
  ....                                                @Component
  ..                                                  class PertrolEngine{
}                                                       ...
                                                        ..
                                                        }
```

To chanage the dependent from DieselEngine to PetrolEngine
we need modif the source code of Vechile class (target class)
(So it is having tight coupling)

Solution: (Using intefaces)

```
@Component
public class Vehicle{
 @Autowired
 private Engine engg;
  ...
  ...
}
```

```
interface Engine{
}
    @Component
class PetrolEngine implements Engine{
  ...
}
    @Component   @Primary
class DieselEngine implements Engine{
  ...
}
```

=>With out changing the source code of taget class we can change dependent
form DieselEngine to PetrolEngine and vice-vera.. For this
we need to use @Primary or @Qualifer or other concepts..

note:: In @Autowired based injection.. if IOC container finds more than 1 possible
dependent then we get "AmbiguityError" .. To overcome that problem we can
@Primary or @Qualifier or other concpets..

---

c) Our code must be open for extension and must be closed for modification (#Priciple3)

=>By coding with interface , our code automatically becomes ready for extension
for example we can devleop more Dependent classes implementing Engine (I)
like CNGEngine implements Engine(I)
GasEngine implements Engine(I)
EletricEngine implements Engine(I)
and etc..

=> By taking both target and dependent classes as final classes or their methods as
final methods we can keep our code closed for modification

note1:: final classes can not have sub classes (so methods can not overriden)
final methods of a class can not be overriden in sub classes.

---

when we dont give Bean Id's the IOC container will create Default Bean Id's internally:-

in Xml approach if the spring bean id is not configured then default beanId is fully qualified class name i.e., nothing but com.nt.beans.Data#0

and if same class configured multiple times then com.nt.beans.Data#n where n=0,1,2,3,...

```
ex:-com.nt.beans.Data#0   <bean class="com.nt.beans.Data"/>
    com.nt.beans.Data#1   <bean class="com.nt.beans.Data"/>
    com.nt.beans.Data#2   <bean class="com.nt.beans.Data"/>
    com.nt.beans.Data#3   <bean class="com.nt.beans.Data"/>
```

=> configured in applicationContext.xml

in Annotation approach nothing but @Component then bean id is first letter of the classname is appears in lowercase and all other letters will be same

ex:- MyClass then bean id is myClass
StudentData then beanId is studentData

in Annotation approach nothing but @Bean method name will be becomes the beanId where these @Bean annotation uses only for predefined classes

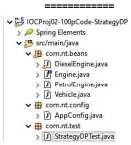
Ex:-

```
@Configuration
class AppConfig{
@bean
public Date getDate(){   here the default bean id is "getDate"
return ----;
   }
}
```

Note:- please refer the page no.15 in October pdf for full clarity

---

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



//common interface
---
package com.nt.beans;

```
public interface Engine {

        public  void start():
        public void stop();

}
```

//Dependent class1

```
package com.nt.beans;

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

@Component("petrol")
@Primary
public final class PetrolEngine implements Engine {

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

        @Override
        public void start() {
  System.out.println("PetrolEngine:: engine started");
        }

        @Override
        public void stop() {
                System.out.println("PetrolEngine:: engine stopped");
        }

}
```

//Dependent class2
==========

```
package com.nt.beans;

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

@Component("diesel")
//@Primary
public final class DieselEngine implements Engine {

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

        @Override
        public void start() {
  System.out.println("DieselEngine:: engine started");
        }

        @Override
        public void stop() {
                System.out.println("DieselEngine:: engine stopped");
        }

}
```

//Vechicle.java (target class)
==========

```
package com.nt.beans;

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

@Component("vehicle")
// A all spring beans are taken as final class ( StrategyDP Rule#3)
public final class Vehicle {
        @Autowired  // For Filed Level Dependency Injection
        @Qualifier("diesel")
        private Engine engg;  //HAS- A Property (compositon : StrategyDP Rule#1)
                //HAS A property type is Interface type StrategyDP Rule#2)
                // All Dependent classes are implementing common Interface StrategyDP Rule#2)

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

        //b,method usng the dependent
        public void journery(String startPlace, String destPlace) {
                engg.start();
                System.out.println("Vechicle:: journey started at::"+startPlace);
                System.out.println("journey is goining on.....");
                engg.stop();
                System.out.println("Vechicle:: journey stopped at::"+destPlace);
        }
}
```

//AppConfig.java
==========

```
package com.nt.config;

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan(basePackages = "com.nt.beans")
public class AppConfig {
        public AppConfig() {
                System.out.println("AppConfig:: 0-param constructor");
        }

}
```

Client App

```
package com.nt.test;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import com.nt.beans.Vehicle;
import com.nt.config.AppConfig;

public class StrategyDPTest {

        public static void main(String[] args) {
                // create IOC container
                AnnotationConfigApplicationContext ctx=new AnnotationConfigApplicationContext(AppConfig.class);
                //get Target class object
                Vehicle vehicle=ctx.getBean("vehicle",Vehicle.class);
                //invole method
                vehicle.journery("delhi", "london");
                //close container
                ctx.close();
        }

}
```

properties file Nov 15 Working Properties file

The text file that contains entries in the form of key=value pairs is called properties file

info.properties

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

=>In spring Apps to cfg properties file (best)
use either <context:propertyplaceholder location="...."/> tag (or)
use @PropertySource annotation

=>The properties file cfg in <context:propertyplaceholder ...."/> tag is visible and accessiable in both spring beans that are user-defined classes and pre-defined classes..

where as property file cfg using @PropertySource is visible and accessible only in the user-defined spring bean classes..

=> In 100% code driven cfgs based spring App (Java Config Approach) and in spring boot App we do not use xml file (spring bean cfg file).. if needed we can link spring bean cfg file (xml file) with @Configuration class using @ImportResource Annotation

com/nt/cfgs/applicationContext.xml (spring Bean cfg file)

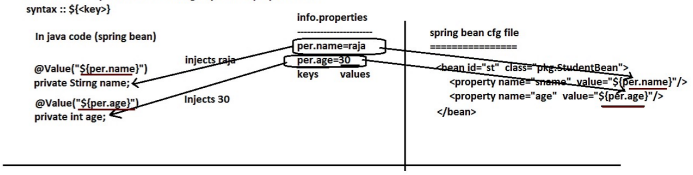
```
<bean ....>
  ....
  ....
</bean>
```

AppConfig.java

```
@Configuration
@ComponentScan(basePackages=".....")
@ImportResource("com/nt/cfgs/applicationContext.xml");
public class AppConfig{
   ...
   ...
}
```

=>In order inject properties file values to any place of spring bean cfg file(xml) or spring bean code (java code) we need to use place holders having keys of the properties file

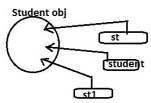
syntax :: ${<key>}



=> if we need we can give alian name for bean id /name using <alias> in spring bean cfg file (we do not have any Annotation alternate for this)

in spring bean cfg file (xml file)

```
<bean id="student"  class="com.nt.beans.Student">
  ....
  ...
</bean>
<alias name="student" alias="st"/>
<alias name="st"   alias="st1"/>
```



What is procedure we need to follow in 100% code driven cfgs App and in spring boot App to change one Dependent to another dependent from Target class with 100% loose coupling(with out touching the source code any spring bean classes while implementin strategy DP)?

Ans) a) use Spring profiles + spring Boot Profiles (future topic)
(Best becoz do not need of involving xml file in any angel)

b) Using properties file + sporing bean cfg file + <alias> tag
(xml is involved , So not much greately recomanded)

---

b) Using properties file + sporing bean cfg file + <alias> tag (xml is involved , So not much greately recomanded)

=>Provide alias name for dependent class bean id by getting that bean id from properties file and pass that alias name to @Qualifier(".....") annotation of Target class HAS -A property as shown below.

a) prepare properties file having key holding u r choice dependent class bean id as value

info.properties

```
# pass dependent class bean id here
choose.engine=diesel    (i)
```

b) cfg properties file in spring bean cfg file and provide fixed alias name for the dependent class bean id collected from properties file

com/nt/config/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/info.properties"/>   (properties file cfg)
<alias name="${choose.engine}" alias="engineType"/>   providing alia name
</beans>
```

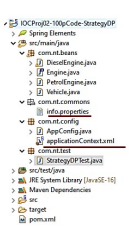
c) Link spring bean cfg file with @Configuration class

```
package com.nt.config;

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.ImportResource;

@Configuration
@ComponentScan(basePackages = "com.nt.beans")
@ImportResource("com/nt/config/applicationContext.xml")
public class AppConfig {
        public AppConfig() {
                System.out.println("AppConfig:: 0-param constructor");
        }

        }
```

d) On taget class HAS-Property place the fixed alias name in @Qualifier annotation

```
// A all spring beans are taken as final class ( StrategyDP Rule#3)
public final class Vehicle {
    @Autowired   // For Filed Level Dependency Injection
    @Qualifier("engineType")  // works becoz engineType collected from <alias> tag
    //@Qualifier(@Value( "${choose.engine}" )   // Does not work becoz @Value can not placed inside another annotation
    private Engine engg;  //HAS- A Property  (compositon : StrategyDP Rule#1)
        //HAS A property type is  Interface type  StrategyDP Rule#2)
        // All Dependent classes are implementing common Interface StrategyDP Rule#2)

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

    //b,method usng the dependent
    public void journery(String startPlace, String destPlace) {
            engg.start();
            System.out.println("Vechicle:: journey started at::"+startPlace);
            System.out.println("journey is  goining on.....");
            engg.stop();
            System.out.println("Vechicle:: journey stopped at::"+destPlace);
    }
}
```

(iii) (diesel)



e) run the App by changing the dependent class bean id in properties file time to time