Spring JDBC

Spring JDBC (Spring DAO)

|-->It provides abstration on plain JDBC Technology and simplifies jdbc style pesistence
logic development by avoiding boilter plate code..

plain jdbc code (java JDBC code)
==============
-> Load jdbc driver class (To regigter jdbc driver with DriverManager Service)
-> Establish the connection
-> create Jdbc Statement object
(common logics)

->send and execute SQL query
-> gather results and process results
(if necessary iterate through RS)
(App specific logics)

-> perform exception handling
-> perform TxMgmt (optional)

->close jdbc objs (including jdbc con)
(common logics)

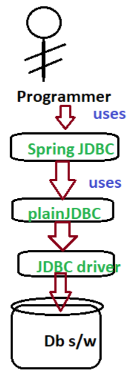
common logics --> these are same in all jdbc apps (boilerplate code)
app specific logics --> will change based on the Db s/w we use.

note:: The code that repeates across the multiple parts of Project /application either with
no changes or with minor changes is called boilerplate code..

spring JDBC App
===============
(It internally takes care of boiler plate code(common logics)
--> Inject JdbcTemplate class obj having Datasoruce obj
--> send and execute SQL query
--> Gather results and process resutls
(application spcific logics)

=>JdbcTemplate /spring jdbc
is given based on Template Method
Design Pattern

[This DP says provide template/
algorithm to perform series operation where
commong things will be taken care internally and
specific things will be given to programmer to
implement]

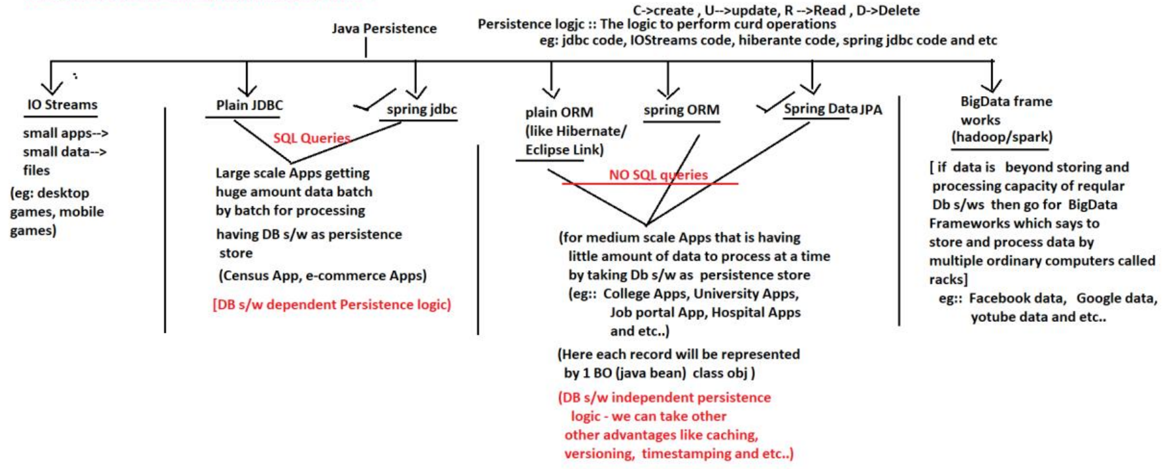


plain jdbc code ==> java code + SQL queries
(DB s/w dependent Persisttence logic
becoz SQL queries are DB s/w dependent)

spring Jdbc code ==> spring code+ jdbc code + SQL queries
(DB s/w dependent persistence logic)

persistence :: The process of saving and managing dta for long time
is called persistence
Persistence store:: The place where pesistence takes place
eg:: files , DB s/w (Best)
Persitence operations :: insert,update,delete,select operations
are called persistence operations

Where should we use which persistence api of Java ?



these are also called CURD/CRUD operations

C->create , U-->update, R -->Read , D->Delete

Java Persistence

Persistence logjc :: The logic to perform curd operations
eg: jdbc code, IOStreams code, hiberante code, spring jdbc code and etc

IO Streams

small apps-->
small data-->
files

(eg: desktop games, mobile games)

Plain JDBC / **spring jdbc**

SQL Queries

Large scale Apps getting huge amount data batch by batch for processing

having DB s/w as persistence store

(Census App, e-commerce Apps)

[DB s/w dependent Persistence logic)

plain ORM (like Hibernate/ Eclipse Link) / **spring ORM** / **Spring Data JPA**

NO SQL queries

(for medium scale Apps that is having little amount of data to process at a time by taking Db s/w as persistence store
(eg:: College Apps, University Apps, Job portal App, Hospital Apps and etc..)

(Here each record will be represented by 1 BO (java bean) class obj)

(DB s/w independent persistence logic - we can take other other advantages like caching, versioning, timestamping and etc..)

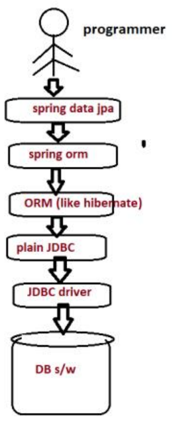
BigData frame works (hadoop/spark)

[if data is beyond storing and processing capacity of reqular Db s/ws then go for BigData Frameworks which says to store and process data by multiple ordinary computers called racks]
eg:: Facebook data, Google data, yotube data and etc..

note:: spring Data module is having capability of generating 100% Basic CURD operations code dynamically for the given db tables..

note:: spring jdbc internally uses plain JDBC and just simplifies jdbc style persistnece logic i.e (70% spring jdbc will take care and 30% should be taken care by programmer)

note:: spring ORM internally uses plain ORM and just simplifies ORM style objects based persistnece logic i.e (70% spring ORM will take care and 30% should be taken care by programmer)



Limitations with Plain JDBC
=========================

a) uses Db s/w dependent SQL Queries in the development of Persistence logic, So the persistence logic DB s/w dependent
b) Supports only positional params (?) i.e does not support named params
c) ResultSet obj that represents the "SELECT SQL Query" execution is not Serializable object to send its data over the network.
d) We need to write explicit logic to convert RS object records to diff formats like ListColleciton, Map collection, simple values and etc..
e)Gives boilter plate code problem (i.e we need write to common logics in every jdbc app)
f)Throws SQLException which is checked Exception and limitations are
  i) For all problems of jdbc code same exception
  ii) We should explicitly catch and handle the exception
  iii) Does not support Exception Propagation naturally
g) Customization results is very complex..
and etc..

spring JDBC advantages
====================

(a) Supports both postional (?) and named parameters
(b) we can get "SELECT Query " Results in diffrent formats directly with the support of query(),queryXxx()[queryForList(),queryForMap(), queryForObject() and etc..] methods
(c) Customization results is bit easy. with the support of Callback Interfaces..
(d) Provides abstraction on plain jdbc code and avoids the boiler plate code (common logics will be generated internally)
(e) Gives Detailed Exeception classes hierarchy which is called DataAccessException classes hierarchy
the advantages are
  i) These exceptions are unchecked exceptions
  ii) Exception handling is optional
  iii) suppors exception propagration by default..
  iv) raises different exceptions for different problems.
  v) these are same execeptions for spring jdbc ,spring orm and spring data modules..
  vi) Spring JDBC internally uses Exception rethrowing concept to convert checked exceptions (SQLException) into Unchecked Exceptions (DataAccessException and its sub classes)

=>The direct sub classes of java.lang.Exception class are called Checked Exception..
=>The direct sub classes of java.lang.RuntimeException class are called UnChecked Exception..

JdbcTemplate class

```
public   Object  queryForObject(String query)throws DataAccessException{
        try{
          ....
          ....  //plain jdbc code
           ....
            ...
          }
        catch(SQLException  se){
          throw new DataAccessException(se.getMessage());
        }              ==>Exception rethrowing is happening here
      }
```

(f) Simplifies the process of calling PL/SQL Procedures and functions..

(g) Gives great support to work with Generics, var args and etc.. (java5,6 features)
(h) Allows to work with java8,9,10 and etc. features..
(i) Can generate insert SQL query dyamically based on the given db table name, col names and col values.

and etc...

note:: spring JDBC Persistence logic is still DB s/w dependent Persistence logic becoz its SQL queries based Persistnece logic..

Different Approaches of developing Persistence logic in spring JDBC
===

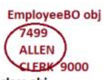
a) Using JdbcTemplate
b) Using NamedParameterJdbcTemplate
c) Using SimpleJdbcTemplate (depreacted in spring 4.x and removed in spring 5.x)
d) Using SimpleJdbcInsert, SimpleJdbcCall
e) MappingSQLOperations as sub classes

a) Using JdbcTemplate

=> It is central Class/API class for entire Spring JDBC i.e remaining approaches of spring JDBC programming internally uses this JdbcTemplate
=> Designed based on Template method design pattern which says define a algorithm to complete a task where common aspects will be taken care by spring jdbc and lets the programmer to take care of only specific activities.
=> Neeed DataSource obj as Dependent object..
=>gives query(-) and queryForXxx(-) for select Queries Exception and gives update(-) method for non-select Queries exeception..
=>JdbcTemplate supports only Positional params

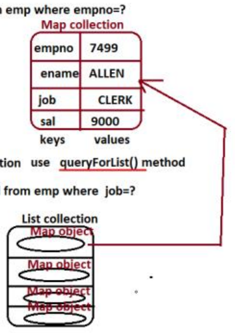
Different query(), queryForXxx() of JdbcTemplate
==

1) To get Single value or single object use queryForObject()
    eg1:: select count(*) from emp
    eg2: select ename from emp where empno=?
    eg3:: select empno,ename,job,sal from emp where empno=?
        |-->to get these values of a record into BO class obj



2) To get single record into Map Collection use queryForMap()
    eg1:: select $\overset{\text{no}}{\text{emp}}$,ename,job,sal from emp where empno=?

3) To get multiple records into List Collection use queryForList() method
    eg1:: select empno,ename,job,sal from emp where job=?



and etc...

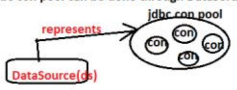
We can use JdbcTemplate in two ways
============================

Using Direct methods with out callback interfaces
=>To get results as given by methods like object, Map , list and etc..

Using methods having callback interaces
=> To get customize resutls as needed by writing partial jdbc code.. by utilizing the internally create jdbc objs..

What is the jdbc con pool/DataSource that u $\overset{\text{have}}{\text{used}}$ in spring Project?

Ans) DataSoruce object represents jdbc con pool i.e all operations on jdbc con pool can be done through DataSoruce object.



=>if the project or application is standalone then use hikaricp(best) or apache dbcp, or c3p0 or vibur cp or tomcat cp and etc..

=>if the Project or application is web application and deployable in the server like tomcat,weblogic, glassfish and etc.. then use Server managed jdbc con pool like tomcat managed jdbc con pool, weblogic jdbc con pool and etc..

```
persistence-beans.xml
====================
<beans>
   <bean id="hkDs"  class="pgk.HikariDataSource">
        ....
        ....
        ....
   </bean>
    <bean id="template"  class="pkg.JdbcTemplate">
         <constructor-arg ref="hkDs"/>
   </bean>
    <bean id="empDAO"  class="pkg.EmployeeDAOImpl">
      <constructor-arg ref="template"/>
    </bean>
</beans>
```

```
applicationContext.xml
======================
<import resource="persistence-beans.xml"/>
<import resource="service-beans.xml"/>
```

```
public  class EmployeeDAOImpl implemetns EmployeeDAO{
    private JdbcTemplate jt;
  public   EmployeeDAOImpl(JdbcTemplate jt){
       this.jt=jt;
    }

   ....
   ....  //methods with persistnece logic
   ...
}
```

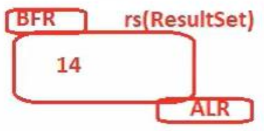
Client App -----------> SErvice class ---------------->DAO class -------------->DB s/w
(presentation logic) (b.logic) (persistence logic)

DS → JdbcTemplate → EmployeeDAOImpl

eg:: JdbcTemplateTest ---------->EmployeeMgmtSErviceImpl ------------>EmployeeDAOImpl -------> DB s/w

jar files :: spring-jdbc-<ver>.jar, ojdbc8.jar , hikaricp-<ver>.jar

refer :: DAOProj1-Xml-JdbcTemplateDirectMethods



```
int count=0;
if(rs.next()){
  count=rs.getInt(1);
 }
```

```
int count=jt.queryForObject("SELECT COUNT(*) FROM EMP",Integer.class);
```

=>queryForObject(-) gets the Injected DS from jt --->Ds collects one jdbc con obj from jdbc con pool ---> creates PS(PreparedStatement obj) having given SQL query as pre-compiled SQL query--> executes query using ps.executeQuery() and get RS(ResultSet) object--> calls rs.next() and rs.getInt(1) method to get result as int value becoz of required type Integer.class --> gives result to DAO method -->DAO method gives to the caller service class method.

=>JdbcTemplate internally uses SimpleStatement object to execute the given SQL if the query is not having any positional (?) params.. otherwise it uses PreparedStatement object

```
int count=jt.queryForObject("SELECT COUNT(*) FROM EMP",Integer.class);
```

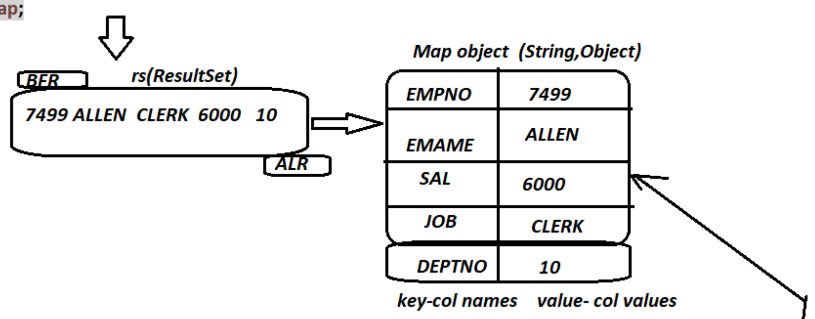
It internally uses SimpleStatement object (static query)

```
String name=jt.queryForObject("SELECT ENAME FROM EMP WHERE EMPNO=?", String.class,eno);
```

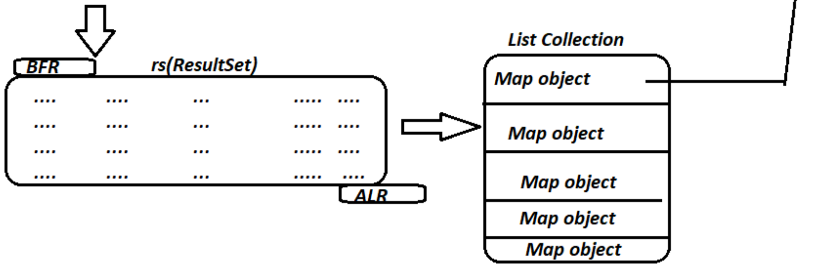
It internally uses PreparedStatement obj becoz the SQL query is dyamic SQL query (query with parameter)

supplies query param values

```
@Override
public Map<String, Object> getEmpDetailsByNo(int no) {
        Map<String,Object>  map=null;
        map=jt.queryForMap(SELECT EMPNO,ENAME,SAL,JOB,DEPTNO FROM EMP WHERE EMPNO=?" ,no);
        return map;
}
```



```
@Override
        public List<Map<String, Object>> getEmpDetailsByDesgs(String desg1, String desg2) {
                return jt.queryForList(SELECT EMPNO,ENAME,SAL,JOB,DEPTNO FROM EMP WHERE JOB IN(?,?)
ORDER BY JOB",desg1,desg2);
        }
```



Converting spring JDBC App into annotation driven cfg based App
==
Thumb rule ::
=>configure pre-defined classes as spring beans using <bean> tags
=> confugure user-defined classes as spring beans using stereo annotations and link them
with configuration file (xml file) using <context:component-scan> tag..

refer DAOPoj2-Anno-JdbcTemplate-DirectMethods

Converting Spring JDBC App into 100%Code Driven cfgs based App
==
Thumb rule::
==>Configure user-defined classes as spring beans using stereo type annotations and link them with @Configuration class using @ComponentScan Annotation
==>Configure pre-defined classes as spring beans using @Bean methods in @Configuration classes
==> Use AnnotationConfigApplicationContext class to create IOC container ..

refer DAOPoj3-100pCode-JdbcTemplate-DirectMethods

Converting Spring JDBC App into Spring Boot App
==

Thumb rule::

==>Configure user-defined classes as spring beans using stereo type annotations
=>make sure that all packages are placed under starter/main class package as sub packages
=>Configure pre-defined classes as spring beans using @Bean methods in @Configuration classes only if they are not coming through AutoConfiguration.
=> get IOC container using SpringApplication.run(-) method..

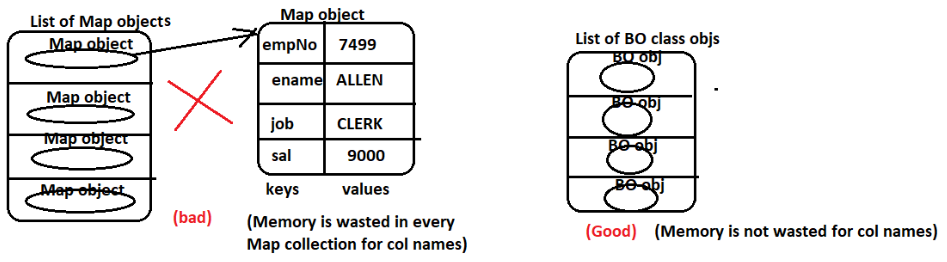
if add spring-boot-starter-jdbc to spring boot project ,we following classes as spring beans through AutoConfiguration
a) HikariDataSource
b) JdbcTemplate
c)NamedParameterJdbcTemplate
d) DataSourceTransactionManager
and etc...

refer DAOPoj4-Boot-JdbcTemplate-DirectMethods

Limitations with JdbcTemplate Direct methods
==
a) query() ,qyeryForXxx(-,-) are giving "SELECT "SQL query results in diffrent formats.. but still they not are industry standard becoz of memory wastage..

=>queryForMap(-) gives single record as Map object having col names as keys and col vlaues as values.. here col names wasting the memory.. The industry standard is getting record into BO class obj

=>queryForList() gives multiple records as Map objects stored into the List Collection .. But col names in all Map objs are wasting the memory. The Indurstry standard is List Collection with BO class objects..



b) No ability to use our choice JDBC statement objs to send and execute SQL queries in DB s/w..

c) Customization of Results (SQL Query results) is more required..

note:: To overcome the above problems.. use JdbcTemplate methods with Callback Interfaces..

with out callback Interfaces

=> No Boilerplate code
=> provides abstraction on plain JDBC
=>Fixed Custom Results like record as Map object , records as List of Map objs which are not indurstry standard..

(Staying in hostel with hostel food)

Callback Interfaces
==============================

=>No Boilerplate code problem
=>Provides abstraction on plain JDBC code
=> Exposes required jdbc objs as the parameters of callback methods by creating them internally to customize the results as needed i.e we can get industry standard results.. like BO, ListBO and etc..

[Staying in hostel taking hostel food and kitchen facility to prepare our own food]

plain JDBC code

=> Boilerplate code problem
=> No Abstraction
=>Pain to programmer to do everthing
=>Results Customization is completly in our choice .. So we can get results as per industry standard like BO obj, ListBO objs and etc..

(staying in Flat with self made food)

Callback method :: The method that executes automatically is called Callback method i.e this
============== method will be called by underlying env.. like Spring JDBC or Container or F/w or server automatically...

Callback Interface :: The interface that contains the decl of callback methods is called Callback Interface..
=================

Servlet life cycle methods are called Cotnainer callback methods.. becoz we do not call them they will be called ServletContainer automatically for different life cycle events..

Spring JDBC is providing multiple callback Interfaces , they are

a) RowMapper :: Gives RS obj to customize single record/row (like BO)
b)ResultSetExtractor :: Gives RS object to customize multiple records (like ListBO) ->stateless
c)RowCallbackHandler :: Gives RS object to customize multiple records (like ListBO)->statefull
d)PreparedStatementCreator :: gives jdbc con object to create PreparedStatement obj
e) PreparedStatementSetter :: gives jdbc PreparedStatement obj to set values to query params and to execute Query
f)StatementCallback
g) PreparedStatementCallback
h)CallbableStatementCallable
i)PreparedStatementBatchSetter
and etc..

RowMapper<T>
=============

=> callback method is
public <T> mapRow(ResuletSet rs, int index)

=>very useful to convert RS obj single record to BO class obj

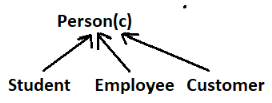
=> queryForObject(-,-,-) is having .. overloaded formshaving RowMapper as the parameter type

```
         @Nullable
         public <T> T queryForObject(String sql,
(a)                            RowMapper<T> rowMapper,          | for query with
                               @Nullable                        | params (?)
                              Object... args) throws DataAccessException

        @Nullable
        public <T> T  queryForObject(String sql,
(b)                         RowMapper<T> rowMapper)throws DataAccessException | for query with out
                                                                               | params (?)
```

```
(b)                     RowMapper<T> rowMapper)throws DataAccessException |  params (?)

      @Nullable
      public <T> T queryForObject(String sql,
                               Object[] args,
 (c)                           int[] argTypes,                          | for query with
                               RowMapper<T> rowMapper)  throws DataAccessException | params (?) same as  (a)
```

=> The above methods must be called RowMapper(I) Impl class obj as the argument value..
becoz if java method paramter type is an interface.. we should call method having impl class obj of that interface as an argument value.

=>public Student process(String data) -->Method returns Student class obj always
(Bad)
or
=> public Person process (String data) --> Method returns PErson its sub class obj
(Good)



To make process(data) method returning any object randomly then we should take Object as return type

public Object process(String data)
(BAD --> while calling we should go type casting, So there is a chance of getting ClassCastException)

```
Student st=(Student)process(...);            | code is not type safe
Employee emp=(Employee) process(..);         |
```

<T> -->Type/Template

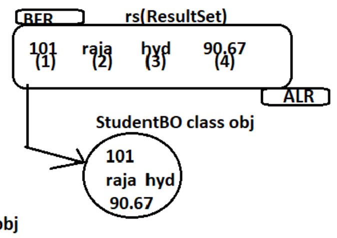
To avoid type castings go for Generics based method designing

```
public  <T>T     process(String data , Class<T> clazz);

Student st=process("..",Student.class);          | Code is typesafe..
Employee emp=process("..",Employee.class);       |
```

Example on queryForObject(-,-,-) having RowMapper to get record as StudentBO class obj
==

```
StudentBO  bo=jt.queryForObject("SELECT * FROM STUDENT WHERE SNO=?",
                                 new StudentMapper(),
                                 101);

 // nested class
private  static class  StudentMapper implements RowMapper<StudentBO>{

    public StudentBO  mapRow(ResultSet rs,int index)throws SQLException{
        //copy ResultSEt object record to StudentBO class obj
        StudentBO bo=new StudentBO();
          bo.setSno(rs.getInt(1));
          bo.setSname(rs.getString(2));
          bo.setSadd(rs.getString(3));
          bo.setAvg(rs.getFloat(4));
          return bo;
      }
  }//inner class
```

```
public class StudentBO{
  private int sno;
   private String sname;
   private String sadd;
   private  float avg;
     //setters  && getters
      ....
      ....
  }
```



Code Flow :: queryForObject(-) method gets the injected DS from jt(JdbcTemplate obj) --> Using that DS gets one jdbc con object from jdbc con pool --> Using that con object creates PreparedStatement obj haivng given SQL query as the pre-compiled SQL query --> set given var args as query param values--> executes the query and gets RS obj with one record--> calls rs.next() method , gets retrieved record index from DB table -->takes second argument (StudentMapper obj-->RowMapper obj) --> calls mapRow(-,-) on that object having RS,record index as the argument values -->mapRow(-,-) copy RS object record to STudentBO class obj and returns that object queryForObject(-,-,-) method and this method returns its caller (generally the DAO class method)

=>instanceOf is java operator to check wheather given reference variable/object is pointing to certain type class object or not .. it returns boolean value.. (true/false)

4 types inner classes
==================
a)Normal inner class [To use its logics in multiple non static methods outer class]
b)Nested inner class/static inner class [To use its logics in multiple static,non-static methods outer class]
c)Local inner class [To use its logics in a method definition in multiple method calls]
d) Anonymous inner class [To use its logics only in one method call]

note:: The methods of JdbcTemplate class will throw DataAccessException and its sub classes related Exceptions based on the problem that is raised..

Anonymous inner class based logic while working with queryForObject(-,-,-) having RowMapper
==

```
                StudentBO bo=null;
                  bo=jt.queryForObject(GET_STUDENT_BY_NO,   // arg1
                                  new RowMapper<StudentBO>() {
                             @Override
                        public StudentBO mapRow(ResultSet rs, int rowNum) throws SQLException {
                                  StudentBO bo=null;
 arg2                             bo=new StudentBO();
                                  bo.setSno(rs.getInt(1));
                                  bo.setSname(rs.getString(2));
                                  bo.setSadd(rs.getString(3));
                                  bo.setAvg(rs.getFloat(4));
                                  return bo;
                                    }//mapRow(-,-)
                             }//anonymous inner class   //arg2
                                  ,
                               no //arg3
                            );
```

In arg2 total 3 things are happening
(a) One anonymous(name less) inner class createed implementing RowMapper(I)
(b) mapRow(-,-) is implemented inside that Anonymous inner class
(c) Object is created for anonymous inner class and passed it as second argument to queryObject(-,-,-) method.

LAMDA Expression based
Anonymous inner class based logic while working with queryForObject(-,-,-) having RowMapper
==

```
       StudentBO bo1=null;
                bo1=jt.queryForObject(GET_STUDENT_BY_NO,   // arg1
                          (rs, rowNum)->{
                                StudentBO bo=null;
                                bo=new StudentBO();
                                bo.setSno(rs.getInt(1));
                                bo.setSname(rs.getString(2));
                                bo.setSadd(rs.getString(3));
                                bo.setAvg(rs.getFloat(4));
                                return bo;
                          }//mapRow(-,-)
                          ,
                          no //arg3
                          );
```

arg2

LAMDA based Anonymous inner class

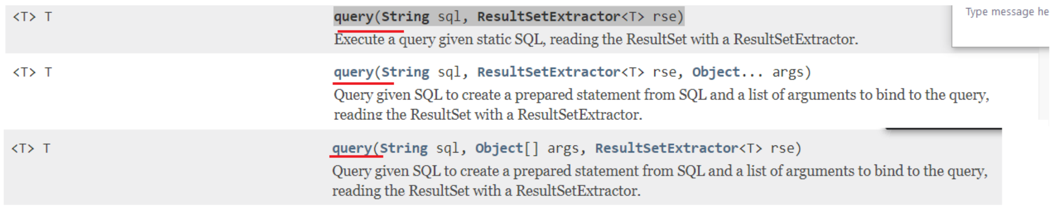
BeanPropertyRowMapper<T> is pre-defined Impl class of RowMapper<T>(I) having logic to copy RS object record given Java Bean class object properties but RS object record db table col names and java Bean class property names must match..

```
       StudentBO bo1=null;
                bo1=jt.queryForObject(GET_STUDENT_BY_NO,   // arg1
                 new BeanPropertyRowMapper<StudentBO>(StudentBO.class),  //arg2
                  no //arg3
                   );
```

Working with ResultSetExtractor<T> callback Interface
===

=> if SELECT SQL Query execution gives multiple records to process then go for ResultSetExtractor<T> or RowCallbackHandler<T>

=> The Best usecase is getting List of BO class objects from RS after executing Select SQL query that gives multiple records..



`<T> T`	`query(String sql, ResultSetExtractor<T> rse)` Execute a query given static SQL, reading the ResultSet with a ResultSetExtractor.
`<T> T`	`query(String sql, ResultSetExtractor<T> rse, Object... args)` Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, reading the ResultSet with a ResultSetExtractor.
`<T> T`	`query(String sql, Object[] args, ResultSetExtractor<T> rse)` Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, reading the ResultSet with a ResultSetExtractor.

requirememt :: DAO class should gives bunch of records as List<StudentBO> objs from Student DB table based on the given student addresses (sadd) city1,city2,city3 values..

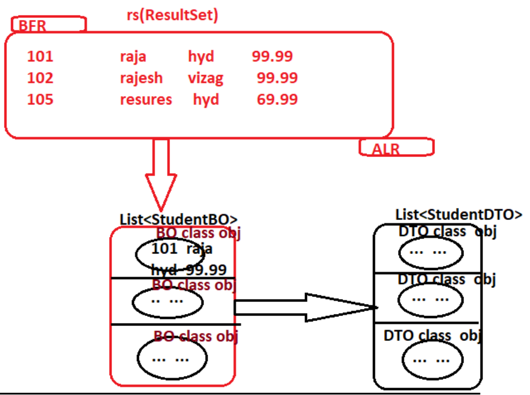
note:: RowMapper<T> , ResultsetExtractor<T> , RowCallbackHandler<T> are functional interfaces becoz they are having only one method declaration.. directly or indirectly..

```
ResultSetExtractor<T> (I)    [Callback inteface)
      |---> public <T> extractData(ResultSet rs)throws SQLException    [Callback method]
```

According to the above require method we should take <T> as List<StudentBO> .

Writing ResultSetExtractor<T> (I) Impl class as Nessted inner class in DAO class
==

```
   @Override
        public List<StudentBO> getStudentsByCities(String city1, String city2, String city3) {
              List<StudentBO> listBO=null;
              listBO=jt.query(GET_STUDENTS_BY_CITIES, //arg1
                                         new StudentExtractor(), //arg2
                                         city1,city2,city3 //args3 (var args)
                                         );
              return listBO;
        }

        //nested inner class /static inner class
        private static class  StudentExtractor implements ResultSetExtractor<List<StudentBO>>{

              @Override
              public List<StudentBO> extractData(ResultSet rs) throws SQLException, DataAccessException {
                    List<StudentBO> listBO=null;
                    StudentBO bo=null;
                    //copy RS object records to List of StudentBO collection
                    listBO=new ArrayList();
                    while(rs.next()) {
                          //get each record into StudentBO class object
                          bo=new StudentBO();
                          bo.setSno(rs.getInt(1));
                          bo.setSname(rs.getString(2));
                          bo.setSadd(rs.getString(3));
                          bo.setAvg(rs.getFloat(4));
                          //add each BO class obj to List colleciton
                          listBO.add(bo);
                    }//while
                    return listBO;
              }//extractData(-)

        }//inner class
```



ResultSetExtractor<T>(I) impl by using anonymous inner class of DAO
===

```
 @Override
      public List<StudentBO> getStudentsByCities(String city1, String city2, String city3) {
            List<StudentBO> listBO=null;
            listBO=jt.query(GET_STUDENTS_BY_CITIES,  //arg1
                           new ResultSetExtractor<List<StudentBO>>() {
                                                      @Override
                                                      public List<StudentBO> extractData(ResultSet rs)throws SQLException{
                                                            List<StudentBO>  listBO=null;
                                                            StudentBO bo=null;
                              arg2                          //copy  RS object records to  List of StudentBO collection
                                                            listBO=new ArrayList();
                        Anonymous                           while(rs.next()) {
                        inner class implementing                  //get each record into StudentBO class object
                        ResultSetExactor<T>(I)                    bo=new StudentBO();
                                                                  bo.setSno(rs.getInt(1));
                                                                  bo.setSname(rs.getString(2));
                                                                  bo.setSadd(rs.getString(3));
                                                                  bo.setAvg(rs.getFloat(4));
                                                                  //add each BO class obj to List colleciton
                                                                  listBO.add(bo);
                                                            }//while
                                                            return listBO;
                                                      }//extratData(-)
                            }, // anonymous inner class arg2
                     arg3        city1,city2,city3  //args3 (var args)
                                 );
            return listBO;
      }//method
```

LAMDA Expression based Anonymous inner class represening ResultExtractor<T> implementation

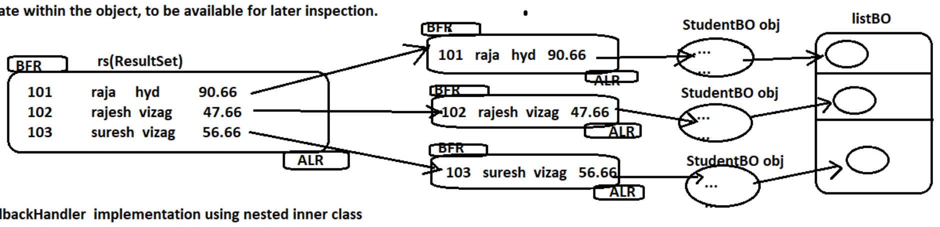
```
    public List<StudentBO> getStudentsByCities(String city1, String city2, String city3) {
            List<StudentBO> listBO1=null;
            listBO1=jt.query(GET_STUDENTS_BY_CITIES, //agrg1
                              rs->{
                                List<StudentBO>  listBO=null;
                                            StudentBO bo=null;
                                            //copy  RS object records to  List of StudentBO collection
                                            listBO=new ArrayList();
        arg2-LAMDA expression based         while(rs.next()) {
        Anonymous inner class for                 //get each record into StudentBO class object
        ResultSetExtactor<T> (I)                  bo=new StudentBO();
                                                  bo.setSno(rs.getInt(1));
                                                  bo.setSname(rs.getString(2));
                                                  bo.setSadd(rs.getString(3));
                                                  bo.setAvg(rs.getFloat(4));
                                            //add each BO class obj to List colleciton
                                                  listBO.add(bo);
                                            }//while
                                        return listBO;
                              }, //args
                              city1,city2,city3 // arg3(var ..args)
                             );//method
            return listBO1;
        }//method
```

REsultSetExtractor<T>(I) impl using the predefined RowMapperResultSetExctor (c)

```
      @Override
      public List<StudentBO> getStudentsByCities(String city1, String city2, String city3) {
            List<StudentBO> listBO=null;
            BeanPropertyRowMapper<StudentBO> bprm=null;
            //create BeanPropertyRowMapper class obj  that hepls to copy each record into  one BO class obj
            bprm=new BeanPropertyRowMapper<StudentBO>(StudentBO.class);
            listBO=jt.query(GET_STUDENTS_BY_CITIES, //arg1
                              new RowMapperResultSetExtractor<StudentBO>(bprm), //args2
                              city1,city2,city3 //arg3 (Var args)
                              );

            return listBO;
      }
```

RowCallbackHandler
==================

|-->Callback method is :: public void processRow(ResultSet rs)throws SQLException
|--->It is stateful becoz implementation class obj remembers the state across the multiple executions of processRow(-) method.
|-->In contrast to a ResultSetExtractor, a RowCallbackHandler object is typically stateful: It keeps the result state within the object, to be available for later inspection.



RowCallbackHandler implementation using nested inner class
===

```
	@Override
	public List<StudentBO> getStudentsByCities1(String city1, String city2, String city3) {
		List<StudentBO> listBO=new ArrayList();
		jt.query(GET_STUDENTS_BY_CITIES,
				new StudentCallbackHandler(listBO),
				city1,city2,city3);
		return listBO;
	}
```

jt.query(-,-,-) gets injected DS --> gets con obj from DS --> creates PS having given Query as pre-compiled query -->set city1/2/3 as the query paa, values -->executes Query and gets RS(main RS)--> In a loop gets each record frm mainRS and creates seperate RS and calls processRow(RS) method for multipletimes.In the Proess ListBO is filledup BO objs given by processRow(-,-) method.

```
	private static class  StudentCallbackHandler  implements RowCallbackHandler{
	  private List<StudentBO> listBO;
	  public StudentCallbackHandler(List<StudentBO> listBO) {
			this.listBO=listBO;
		}
		@Override
		public void processRow(ResultSet rs) throws SQLException {
			System.out.println("StudentDAOImpl.StudentCallbackHandler.processRow(-)");
			StudentBO bo=null;
			//covert RS record into BO clss object
			bo=new StudentBO();
			bo.setSno(rs.getInt(1));
			bo.setSname(rs.getString(2));
			bo.setSadd(rs.getString(3));
			bo.setAvg(rs.getFloat(4));
			listBO.add(bo);
		}//method

	}//inner class
```

Anonymous inner class based RowCallbackHandler(I) Implementation
==

```
   @Override
         public List<StudentBO> getStudentsByCities1(String city1, String city2, String city3) {
            List<StudentBO> listBO=new ArrayList();
            jt.query(GET_STUDENTS_BY_CITIES,
                                  new RowCallbackHandler() {
                                     @Override
                                     public void processRow(ResultSet rs) throws SQLException {
                                          System.out.println(
                                         "StudentDAOImpl1.getStudentsByCities1(...).new RowCallbackHandler() {...}.processRow()");
                                         //get each record into StudentBO class object
                                          StudentBO bo=new StudentBO();
                                                   bo=new StudentBO();
                                                   bo.setSno(rs.getInt(1));
                                                   bo.setSname(rs.getString(2));
                                                   bo.setSadd(rs.getString(3));
                                                   bo.setAvg(rs.getFloat(4));
                                                   //add each BO class obj to List colleciton
                                                   listBO.add(bo);
                                     }
                                 },
                                     city1,city2,city3);
               return listBO;
         }
```

LAMDA expression based Anonymous inner class implementation for RowCallbackHandler(-)
==

```
   @Override
         public List<StudentBO> getStudentsByCities1(String city1, String city2, String city3) {
              List<StudentBO> listBO=new ArrayList();
              jt.query(GET_STUDENTS_BY_CITIES,rs->{
                   //get each record into StudentBO class object
                   System.out.println("StudentDAOImpl2.getStudentsByCities1()...lAMDA...");
                   StudentBO bo=new StudentBO();
                   bo.setSno(rs.getInt(1));
                   bo.setSname(rs.getString(2));
                   bo.setSadd(rs.getString(3));
                   bo.setAvg(rs.getFloat(4));
                   //add each BO class obj to List colleciton
                   listBO.add(bo);

                               },
                    city1,city2,city3);
              return listBO;
         }
```

What is difference b/w ResultSetExtractor and RowCallbackHandler callback Interfaces

ResultSetExtractor(I)	RowCallbackHandler (I)
(a) it stateless in nature becoz there is no need of remebering state across the mutiple executions of Impl class object	(a) It is statefull in nature becoz it remembers the given state like listBO across the multiple executions of the Impl class object.
(b) exractData(-) is the callback method and it executes only for 1 time	(b) processRow(-) is callback method and it executes for multiple times
(c)Invovles only one ResultSet object in the entire process	(c) Involves multiple RS Objects (n+1) in the entire process n--> records count given by "SELECT SQL Query"
(d) Good in performence	(d) bad in performence
(e) Support for Generics	(e) No support for Generics
(f) we have multiple useful readymade impl classes	(f) we do no have here

NamedParameterJdbcTemplate
==========================

=> It is given to support named parameters in the SQL query..
=> The Limitation with positional params (?) is providing index and setting values to those parameters according to the index is bit complex.. especially if the query having multiple positional parametes..

=>To overcome the above problem use named parameters (:<name>) which gives name to each parameter and we can set values to parameters by specifying their name..

query with positional params
==========================

SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE EMPNO>=$\underbrace{?}_{1}$ AND EMPNO<=$\underbrace{?}_{2}$

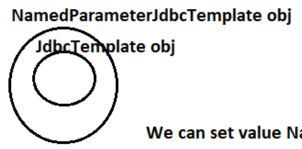
query with named params
==========================

SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE EMPNO>=<u>:min</u> AND EMPNO<=<u>:max</u>

Named Parameter

note:: JdbcTemplate does not support Named Parameters.. it supports only Positional parameters
note :: NamedParameterJdbcTemplate supports named Parameters but does not support positional parameters..

This class delegates to a wrapped JdbcTemplate once the substitution from named parameters to JDBC style '?' placeholders is done at execution time. It also allows for expanding a List of values to the appropriate number of placeholders.



NamedParameterJdbcTemplate object has JdbcTemplate object
i.e composition (Has-A Relation)

We can set value NamedParameters in 2 ways while working with NamedParameterJdbcTemplate

a)Using Map<String,Object> obj
=>here the named parameter names are keys and param values are values

b) Using SqlParameterSource(I) Implementations

i) MapSqlParameterSource (c)
|-->use its addValue(-,-) method having param name, param value as the arguments

ii)BeanPropertySqlParameterSource (c)
|-->Allows set JavaBean obj values as the named parameter values but the names of named parameters and the names java bean class properties must match

=>To create NamedParameterJdbcTemplate we need DS object as the dependent object
=>NamedParameterJdbcTemplate also gives support to work with callback interfaces..

=>NamedParamers are case-sensitive..

Example code
==============
EmployeeDAOImpl.java
======================

```
package com.nt.dao;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.namedparam.BeanPropertySqlParameterSource;
import org.springframework.jdbc.core.namedparam.MapSqlParameterSource;
import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;
import org.springframework.stereotype.Repository;

import com.nt.bo.EmployeeBO;

@Repository("empDAO")
public class EmployeeDAOImpl implements IEmployeeDAO {
	private static final String  GET_EMPNAME_BY_NO="SELECT ENAME FROM EMP WHERE EMPNO=:no";
	private static final String  GET_EMPDETAILS_BY_DESGS="SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE  JOB IN(:desg1,:desg2,:desg3)";
	private static final String  INSERT_EMPLOYEE="INSERT INTO EMP(EMPNO,ENAME,JOB,SAL) VALUES(:empNo,:ename,:job,:sal)";

	@Autowired
	private  NamedParameterJdbcTemplate npjt;

	@Override
	public String getEnameByNo(int no) {
		/*Map<String,Object> paramMap=new HashMap();
		paramMap.put("no",no);*/
		Map<String,Object> paramMap=Map.of("no",no);  //java9 feature

		String name=npjt.queryForObject(GET_EMPNAME_BY_NO,
		                                            paramMap,
		                                            String.class);
		return name;
	}//method

	@Override
	public List<EmployeeBO> getEmpDetailsByDesgs(String desg1, String desg2, String desg3) {

		//prepare MapSqlParameterSource obj having the names,values of the named parameters
		MapSqlParameterSource  msps=new MapSqlParameterSource();
		msps.addValue("desg1",desg1); //namedparam, value
		msps.addValue("desg2",desg2);
		msps.addValue("desg3",desg3);
		List<EmployeeBO> listBO=npjt.query(GET_EMPDETAILS_BY_DESGS,
		                                             msps,
		                                             rs->{
		                                              List<EmployeeBO> listBO1=new ArrayList();
		                                              while(rs.next()) {
		                                                   EmployeeBO bo=new EmployeeBO();
		                                                   bo.setEmpNo(rs.getInt(1));
		                                                   bo.setEname(rs.getString(2));
		                                                   bo.setJob(rs.getString(3));
		                                                   bo.setSal(rs.getFloat(4));
		                                                   listBO1.add(bo);
		                                              }//while
		                                              return listBO1;
		                                             });

		return listBO;
	}//method

	@Override
	public int insertEmployee(EmployeeBO bo) {
		//create BeanPropertySqlParameterSource object
		BeanPropertySqlParameterSource bpsps=new BeanPropertySqlParameterSource(bo);
		//execute query
		int count=npjt.update(INSERT_EMPLOYEE, bpsps);
		return count;
	}

}//class
```

SimpleJdbcTemplate
==================

=>Introduced in spring 2.x as alternate to JdbcTemplate supporting new features of that time like generics , var args and etc..
=>continued and deprecated in spring 3.x becoz they upgraded JdbcTemplate itself supporting features like generics ,var args and etc..
=>In Spring 4.1, the SimpleJdbcTemplate is removed..

SimpleJdbcInsert
==============

A SimpleJdbcInsert is a multi-threaded, reusable object providing easy insert capabilities for a table. It provides meta-data processing to simplify the code needed to construct a basic insert query . All you need to provide is the name of the table and a Map containing the column names and the column values.

5 approaches of writing logic in spring jdbc
- a) Using JdbcTemplate
- b) Using NamedParameterJdbcTemplate
- c) Using SimpleJdbcJdbcTemplate
- d)Using SimpleJdbcInsert,SimpleJdbcCall
- e) Mapping SQL Operations as sub classes.

JdbcTemplate,NamedParameterJdbcTemplate, SimpleJdbcTemplate are threadsafe i.e they are single threaded objects. so they allow only one thread at time to perform persistence operation i.e they are not suitable multi-threaded time critical web applicaiton env... like online auction/bidding and online Counselling, online shopping and etc..

=> In the above situations, we can use "SimpleJdbcInsert" for insert persistence operations..becoz it is multithreaded.. i.e multiple threads can be perform insert operation simultaenously..
=> While woking with "SimpleJdbcInsert" we do not write "INSERT SQL Query" Seperately.. we just provide DS, Db table name, Map of Col names, values .. then insert SQL Query will be generated dynamically

```
SimpleJdbcInsert
    |--->DS (as dependent obj)
    |-->setTable(-)
    |--->int  execute(Map<String,Object> map)
    |
    |         (or)          takes colnames and col values.
    |-->int  execute(SqlParameterSource source)
                              |-->MapSqlParameterSource (c)
                                     -->using addValue(-,-) we need to pass col names and col values
                              |-->BeanPropertySqlParameterSource(c)
                                     -->Here we can  pass JavaBean object as input for col names and
                                        col values.. but  db table col names and Java Bean property
                                        names must match.
```

The actual insert is being handled using Spring's JdbcTemplate.

Q) SimpleJdbcInsert Internally uses JdbcTemplate for completing generated insert SQL query execution then how can say it is multi-threaded as we know JdbcTemplate is singleThreaded?

Ans) if we call execute(-) method on SimpleJdbcInsert for multiple times.. then multiple JdbcTemplate class objects will be used internally to execute the generated Insert SQL query for multiple times, So the SimpleJdbcInsert becomes multi-threaded.

Q)Why Spring JDBC is not providing "SimpleJdbcUpdate", "SimpleJdbcDelete" and "SimpleJdbcSelect" classes?

Ans) update, delete and select SQL queries execution takes place along with conditions .. Based on given table name, col names and col values .. these conditions can not be generated dynamically.. So There are no "SimpleJdbcUpdate","SimpleJdbcDelete","SimpleJdbcSelect" classes..
note:: Insert SQL query executes with out any condition i.e it can be generated dynamically based on the given Db table name, col names, col vlaues.. So "SimpleJdbcInsert" is given..

Q) How to execute update, delete ,select SQL queries in multi-threaded env...?

Ans) SimpleJdbcCall multi-threaded object having ability to call PL/SQL procedures or functions So keep u r update, delete ,select SQL queries inside PL/SQL procedure or functions and call them by using SimpleJdbcCall object..

Q)will SimplejdbcInsert support positional/named parameters?
=>Since programmer is not preparing query .. and query is generated dynamically.. So there is no posibility placing of any kind of parameters..

Example Code
=============

```
 @Repository("bankDAO")
 public class BankAccountDAOImpl implements IBankAccountDAO {
        @Autowired
        private  SimpleJdbcInsert sjc;

        @Override
        public int register(BankAccountBO bo) {
               //prepare Map object having  col names and value
               Map<String,Object>  map=Map.of("acno", bo.getAcno(), "holderName",
                                          bo.getHolderName(), "balance", bo.getBalance(), "status",bo.getStatus()); //java9 feature
               //set db table name
               sjc.setTableName("BANK_ACCOUNT");
               //execute  query by generating the query dynamically
               int  count=sjc.execute(map);
               return count;
        }
 }
```

persistence-beans.xml

```
<beans ....>
<!-- DataSource cfg -->
  <bean id="hkDs"  class="com.zaxxer.hikari.HikariDataSource">
     <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
     <property name="jdbcUrl"  value="jdbc:oracle:thin:@localhost:1521:xe"/>
     <property name="username" value="system"/>
     <property name="password" value="manager"/>
     <property name="minimumIdle" value="10"/>
     <property name="maximumPoolSize" value="100"/>
  </bean>

  <!-- Cfg SimpleJdbcInsert -->
  <bean id="sjc"  class="org.springframework.jdbc.core.simple.SimpleJdbcInsert">
       <constructor-arg  ref="hkDs"/>
  </bean>

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

</beans>
```

if Db table col names are matching BO class obj property names then we can call execute(-) of SimpleJdbcInsert having BeanPropertySqlParametersource object as shown below.

```
@Override
    public int register(BankAccountBO bo) {
        //prepare BeanPropertySqlParameterSource  object having  BO class obj (here col names must match bo class property names)
      BeanPropertySqlParameterSource bpsps=new BeanPropertySqlParameterSource(bo);
        //set db table name
        sjc.setTableName("BANK_ACCOUNT");
        //execute  query by generating the query dynamically
        int  count=sjc.execute(bpsps);
        return count;
    }
```

SimpleJdbcCall
=============

A SimpleJdbcCall is a multi-threaded, reusable object representing a call to a stored procedure or a stored function. It provides meta-data processing to simplify the code needed to access basic stored procedures/functions. All you need to provide is the name of the procedure/function and a Map containing the parameters when you execute the call. The names of the supplied parameters will be matched up with in and out parameters declared when the stored procedure was created.

=> Instead of writing same persistence logic/b.logic in multiple modules as sqlqueries/java code ,it is recomanded to write only for 1 time in Db s/w as stored procedure /function..and use it multiple modules.

eg1: Authentication logic as PL/SQL procedure /function
eg2: Attendence calculation logic as PL/SQL procedure function

=>PL/SQL procedure does not return a value.. but to get multiple results from PL/SQL procedure we need to use multiple OUT params
=>PL/SQL function returns a value.. So to get multiple results from PL/SQL function we need to get 1 result as return value and remaining results as OUT params

=>IN PL/SQL procedure or function the params will have type(data type),mode.

The modes are ::
IN (default)
OUT
INOUT

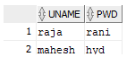
eg:: PL/SQL Logic in oracle
y:= x*x; x as in mode param, y as out mode param
eg:: PL/SQL logic in oracle
x:=x*x ; x as INOUT param

PL/SQL programming syntaxes are specific to each DB s/w..

PL/SQL procedure for Authentication
================================

step1) make sure that one DB table is taken having usernames and passwords..

USERINFO (db table)



	UNAME	PWD
1	raja	rani
2	mahesh	hyd

step2) create PL/SQL procedure having authentication logic

```
CREATE OR REPLACE PROCEDURE P_AUTHENTICATION
  ( USERNAME IN VARCHAR2 , PASSWORD IN VARCHAR2 , RESULT OUT VARCHAR2 ) AS
  CNT NUMBER(3);
BEGIN
  SELECT COUNT(*) INTO CNT FROM USERINFO WHERE UNAME=USERNAME AND
PWD=PASSWORD;
  IF(CNT<>0) then
   RESULT:='VALID CREDENTIALS';
  ELSE
    RESULT:='INVALID CREDENTIALS';
  END IF;
 END P_AUTHENTICATION;
```

SQL developer ---> Procedures ---> right click-> new Procedure ---> .....

SimpleJdbcCall
```
  |-->DS (required as dependent obj)
  |-->setProcedureName(-)  [To specify PL/SQL procedure name]
  |-->setFunction(true/false) [ set to true . if the above name is  PL/SQL function name otherwise set to false (deafult)]
  |-->Map<String,Object> execute(Map<String,Object>  inparams)
  |
  |------>  Map<String,Object>  execute(SqlParameterSource source)
  |---->   T<T>   executeFunction(Class <T> returnType, Map<String,?> inParams)
  |---->   T<T>   executeFunction(Class <T> returnType, SqlParameterSource inParams)
```

gives OUT Param names and values asMap obj

To supply IN param names and values as Map object

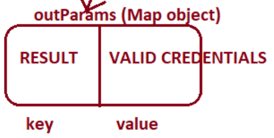
To call PL/SQL procedure

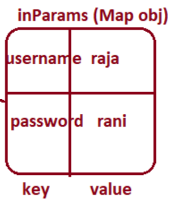
To call PL/SQL function.

---

example code
============

```java
@Repository("authDAO")
public class AuthenticationDAOImpl implements IAuthenticationDAO {
	@Autowired
	private SimpleJdbcCall sjc;

	@Override
	public String authentication(String user, String pwd) {
		//set procedure name
		sjc.setProcedureName("P_AUTHENTICATION");
		//prepare Map of IN Params
		Map<String,?> inParams=Map.of("USERNAME",user,"PASSWORD",pwd);  //java 9 feature
		//call PL/SQL procedure
		Map<String,?> outParams=sjc.execute(inParams);
		return (String) outParams.get("RESULT");
	}

}
```





Approach5 :: Mapping SQL Operations(Queries) as sub classes using SqlQuery<T>,SqlUpdate classes
===

Another limitation with JdbcTemplate/NamedParameterJdbcTemplate/SimpleJdbcTemplate

in DAO class

```
 public   String  getEmpNameByNo(int no){

  String name=jt.queryObject("SELECT ENAME FROM EMP WHERE  EMPNO=?",
                                               String.class,
                                               no);
   return name;
   }
```

if the the above DAO method/jt.queryObject(-,-,-) is called for multiple times then
- (a) Gatheres the injected DS obj from JdbcTemplate object for multiple times (ok)
- (b) Gahters jdbc con object from jdbc con pool for multiple times (ok)
- (c) makes the given SQL query as pre-compiled Query for multiple times by creating PreparedStatement object for multiple times (no ok) => making the same SQL query as pre-compiled SQL query for multiple times is unneccessary and also degrades the performence
- (d) sets the values to query param for multiple times and exeutes query for multiple times (ok)
- (e) procsess/convert the results for multiple times (ok)

To overcome the above problem use "Mapping SQL Operations as sub classes" approach .. which says for every SQL query develope one sub class extending SqlQuery<T>(AC) (for select query) or from SqlUpdate<T> for non-select query . we generally these sub classes as inner classes in the DAO class.

=>In these sub classes we give DS,SQL query to their super classes (SqlQuery<T>/SqlUpdate classes) only 1 for time , so that collecting con object from jdbc con pool, creating PreparedStatement object having given query as pre-compiled SQL query happens only for 1 time and sub classes objes start representing pre-compiled Queries, So that DAO class methods can use the objects of sub classes for multipletimes to execute the pre-compiled SQL queries for multipletimes..

SqlQuery<T>(AC) is having more abstract methods to implement.. So prefer using MappingSqlQuery<T> (AC) which having less no.of abstract methods to implement..

```
java.lang.Object
    org.springframework.jdbc.object.RdbmsOperation
        org.springframework.jdbc.object.SqlOperation
            org.springframework.jdbc.object.SqlQuery<T>
                org.springframework.jdbc.object.MappingSqlQueryWithParameters<T>
                    org.springframework.jdbc.object.MappingSqlQuery<T>
```

On the each select SQL query related sub class obj of SqlQuery<T>/MappingSqlQuery<T> we can call
- a) List<T> execute(...) :: if the Select Query gives bunch of records.
- b) <T> findObject(...) :: if the Select Query gives single record.

```
@Reposity("studDAO")
public class StudentDAOImpl implemetns  StudentDAO{
   private static final String  GET_STUDENTS_BY_ADDRS="SELECT SNO,SNAME,SADD,AVG FROM STUDENT WHERE SADD=?";

    @Autowired
    private   DataSource ds;
     private  StudentSelector1  selector1;

     //constructor
    public  StudentDAOImpl(){
       selector1=new StudentSelector1(ds,GET_STUDENTS_BY_ADDRS);

     }


        //method
      public  List<StudentBO>  getStudentsByAddrs(String addrs){

           List<StudentBO> listBO=selector1.execute(addrs);

            return listBO;
       }

      //sub class as inner class in DAO

        private class   StudentSelector1 extends  MappingSqlQuery<StudentBO>{

                   //constructor
                  public  StudentSelector1(DataSource ds, String query){
                      super(ds,query);
                      super.declareParameter(new SqlParameter(Types.VARCHAR));  //registrering param(?) with  jdbc data type
                      super.compile();
                  }

                 public  StudentBO mapRow(ResultSet rs, int rowNum) throws SQL Exception{
                    //convert  RS record to BO class obj
                          StudentBO bo=new StudentBO();
                            bo.setSno(rs.getInt(1));
                            bo.setSname(rs.getString(2));
                            bo.setSadd(rs.getString(3));
                            bo.setAvg(rs.getFloat(4));
                        return bo;
                 }//mapRow(-,-)
         }//inner class

     }//DAO class
```

Flow of execution
==================

IOC container creation ---> pre-instantiation of singleton scope beans, So DS, DAO classes pre-instantiated and DS is injected to DAO ---> In that process DAO constructor executes and calls sub class cum inner class (StudentSelector1) constructor due to this sub class cum inner class (StudentSelector1) gives DS,query to its super class(MappingSQLQuery) only for 1 time and creates PreparedStatement obj by making given SQL query as pre-compiled Query becoz of super.compile() only for 1 time (At the end the sub class cum inner class (StudentSelector1) represents pre-compiled SQL query)

Service class method calls DAO method (getStudentsByAddrs(-)) for multiple times, so selector1.execute(-) also called for multiple times-->In this process values to query params will be set for multiple times --> query exection takes place for multiple times ---> gathering RS obj processing that obj to ListBO by calling mapRow(-,-) takes place for multiple times -->returns ListBO back to DAO class method for multiple times..

a)Gathering and using DS happens for 1 time (OK)
b) Gathering jdbc con object from jdbc con pool happens for 1 time (OK)
c) creating PreparedStatement obj by making the SQL query as pre-compiled SQL query happens for 1 time (OK)
d) setting values query params and executing query happens for multiple times (OK)
e) gathering results and processing results happens for multiple times (OK)

Woking with properties file and yml/yaml files in spring/spring boot
==

we can read inputs from properties file/yml file to spring bean properties in two ways

a) using @Value   (given spring framework) 3.0 (Does not support bulk reading)

->we should add on the top of each property

->reading values into array/list/set/map and HAS-A Object is complex (not recomanded to do)

->property name in bean class   and key in propperties /yml file need to not match

yml -->yiant markup language/yamaling markup language.
yaml -->yet another markup language.

A different approach maintaing key=values pairs..

application.properties

(In spring boot application applicaiton.properties/yml file will be detected and loaded automatically as part application flow from src/main/resources folder.)

```
per.info.id=101
per.info.name=raja
```

[note: The properties/yml files having other name or location must be configured explicitly using @PropertySource annotation]

```
@Component    @Data
public class  Person{

    @Value("${per.info.id}")
    private int pid;
    @Value("${per.info.name}")
    private   String  pname;

}
```

a) @ConfigurationProperties (supports Bulk reading)

=>Given by spring boot 1.0

=>Allows to read values into simple, array/list/set/map , HAS-A object properties

=>We need to apply only the top of spring bean class by specifying prefix , So values will be bound to spring bea class properties at once..

=>Here keys in properties/yml file must match with spring bean class property names.

=>All keys in properties file must have commonon prefix.. and that common prefix must be specified in @ConfiguratinProperties (prefix="....")

While working with @ConnfiguationProperties it is recomanded to add the following dependency in pom.xml file to generate MetaData about entries /keys of proeprties/yml file.. Due to this all warnings in properties will go off ..

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-configuration-processor</artifactId>
    <optional>true</optional>
</dependency>
```

In properties file the allowed special characters in keys are ".","-","_"

=>On certain bean propeperty of spring bean class if we place both @Value , @ConfigurationProperties(indirectly from top of the class) effect with two different keys and value.. the @Configuration value will be taken as the final value..

application.properties

```
per.info.id=101
per.info.no=102
```

Person.java

```
@Data
@Component("per")
@ConfigurationProperties(prefix = "per.info")
public class Person {
@Value("${per.info.no}")
private int id;   // holds 101 as  final value.

}
```

application.properties
====================

```
#simple properties (prefix.var=value)
per.info.id=101
#per.info.no=102
per.info.name=rakesh
per.info.addrs=hyd

#arrays (prefix.var[index]=value)
per.info.marks1[0]=40
per.info.marks1[1]=50
per.info.marks1[2]=60

#List Collection (prefix.var[index]=value)
per.info.marks2[0]=50
per.info.marks2[1]=60
per.info.marks2[2]=70

#Set Collection (prefix.var[index]=value)
per.info.marks3[0]=60
per.info.marks3[1]=70
per.info.marks3[2]=80

#Map Collection/Properties [prefix.var.ke=value]
per.info.phones.residence=9999999
per.info.phones.office=8888888
per.info.phones.personal=777777

# HAS- Relation Object type property [prefix.Has-Avar.var=value]
per.info.job.company=HCL
per.info.job.desg=Programmer
per.info.job.deptno=9001
per.info.job.salary=67788.6
```

Person.java
==========

```java
@Data
@Component("per")
@ConfigurationProperties(prefix = "per.info")
public class Person {
	//@Value("${per.info.no}")
	private int id;
	private String name;
	private  String addrs;
	private  int[] marks1;
	private List<Integer> marks2;
	private  Set<Integer> marks3;
	private  Map<String,Long> phones;
	private  Job   job;  //HAS-A relation property

}
```

=>In most cases we use pre-defiend keys and their values in application.properfies to provides instructions /inputs related autoconfiguration..
=>The Beans of Autoconfiuration internally uses this @ConfigurationProperties to read values from properites..

example
=========

```java
@ConfigurationProperties(prefix = "spring.datasource")
public class DataSourceProperties implements BeanClassLoaderAware, InitializingBean {
  ...
}
```

while preparing element values to array/list/set collection inline syntax in properties file as shown below
==

in application.properties

```
Array/Set/List/ Collection (prefix.var[index]=value1,value,value3)
per.info.marks3=60,70,80
```

YML/YAML
===========
=> Yiant markup language / YAMiling markup language (yml)
=> Yet Another Markup language (YAML)
=> Alternate to proeperties file , very useful when lengthy keys at same level becoz it avoids duplicates from the keys by maintaning key and values in hierarchy manner.

application.properties

```
info.per.id=101
info.per.name=raja
info.per.addr=hyd
```

the word "info.per" is repeated for multiple times in the keys there is duplication in the keys.

application.yml

```
info:    | level1 node
  per:   | level2 node
    id: 101
    name: raja
    addr: hyd
```

Here the word "info.per" is not repeated in the keys by maintaing data as hierarchal data..

=>extension can be .yml or .yaml
=> spring boot internally uses "snack yaml api" to parse and convert yml file into properites file
=> while writing "nodes" in yml file you must give minimum one space and allowed special symbols in the keys are "_" ,"-","."
=> same levels nodes must started at same place(same col number in the file).. if not errors will comes (this indicates we must maintain proper indentation)
=> Both application.yml or application.properties will dected and loaded by spring boot automatically during the appplication startup.. from main/java/resources folder.
=> we can bind yml file data to spring bean class properties/variables either using @Value(given by spring) or using @ConfiurationProperties(given spring boot) annotations
=>yml files are node based , space sensitive and indentation based files.

application.properties

```
info.per.id=101
info.per.name=raja
info.job.desg=clerk
info.job.salary=9000
company.location=hyd
company.name=HCL
```

$\Rightarrow$

application.yml

```
info:
  per:
    id: 101
    name: raja
  job:
     desg: clerk
     salary: 9000
company:
  location: hyd
  name: HCL
```

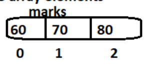
symbol in properties file, yml file indicates comment

array/List /Set Collection
----
applicaiton.properties

```
info.per.marks[0]=60
 info.per.marks[1]=70
 info.per.marks[2]=80
```

application.yml

```
info:
   per:
      marks:
         - 60
         - 70
         - 80
```
these array elements

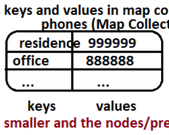


Map/Properties Collection
========================
application.properties

```
# prefix.var.key=value
info.per.phones.residence=999999
info.per.phones.office=88888888
info.per.phones.personal=7777777
```

application.yml

```
info:
  per:
    phones:
        residence: 999999
        office: 888888
        personal: 777777
```
acts keys and values in map collection



=>use properties file if the keys are smaller and the nodes/prefixes are no repeating..
=>use yml file if the keys are lengthy and the nodes/prefixes are repeating

for Has-A relation property $\overset{\text{Object type}}{}$
========================
application.yml

```
info:
  per:
    job:
      desg: manager
      salary: 8999.5
      company: HCL
       skills:
         - java
         - spring
         - hibernate
```

```
@Data
@Component("per")
@ConfigurationProperties(prefix="info.per")
public class Person {
  private Job job;
 }
```

```
@Data
public class Job {
       private String desg;
       private float salary;
       private String company;
       private String[] skills;
}
```

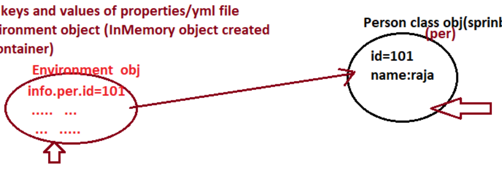
=>In Eclipse IDE therre is bult-in convertor to given .properties file to .yml file (to convert)
right click on properties file --->convert to .yml file..

Internal flow of @ConfigurationProperties and @Value
==

#1 Spring boot detects and loads
applicatiion.properties/yml file
[if it yml file it will converted into
propperties file intenrally using
snackyml]

#2 reads keys and values of properties/yml file
into Environment object (InMemory object created
in IOC container)

#3 Collectis the values from Envirmoment object
and binds to Spring Bean class obj properties
based on @Value or @ConfigurationProperties
annotation..



```
@Component("per")
@Data
@ConfigurationProperties(prefix="info.per")
public class Person{
  private int id;
   private String name;
   ...
   ..
}
```

This env.. object holds multiple details
=>properties/yml file data
=> system properties
=> env.. variables info like PATH, CLASSPATH and etc..
=> profiles info

if we place both application.properties and application.yml files in spring boot application having same keys and different values then what happens?

ans) The values kept properties file will be taken as final values..

note:: if certain key is not avaiable in application.properties file , it will be gathered from application.yml

What are the differences and similiraties between properties file and yml file?

Table of Difference:

YAML(.yml)	.properties
=> Spec can be found here	It doesn't really actually have a spec. The closest thing it has to a spec is actually the javadoc.
=> Human Readable (both do quite well in human readability)	Human Readable
=> Supports key/val, basically map, List and scalar types (int, string etc.)	Supports key/val, but doesn't support values beyond the string
=> Its usage is quite prevalent in many languages like Python, Ruby, and Java	It is primarily used in java
=> Hierarchical Structure	Non-Hierarchical Structure
=> Spring Framework doesn't support @PropertySources with .yml files	supports @PropertySources with .properties file
=> If you are using spring profiles, you can have multiple profiles in one single .yml file	Each profile need one separate .properties file
While retrieving the values from .yml file we get the value as whatever the respective type (int, string etc.) is in the configuration	While in case of the .properties files we get strings regardless of what the actual value type is in the configuration

When should I use .properties or .yml file?

=> if keys are lengthy having multuple common modes then for yml files becoz it avoids the repeatation of common nodes .. otherwise go for properties file

What is the difference b/w   @Value   and   @ ConfigurationProperties?

@Value ========	*@ConfigurationProperties* ======================
a) given by spring framework 2.0, So it can be used in both spring and spring boot programming	a) given by spring boot 1.x , So it can be used only in spring boot programming
b)useful for reading single value from .proeprties or yml file	b) useful for reading bulk values by giving common prefix from properties or yml file.
c) can be applied at .method level and field level ,param level and etc.. (but not at class level)	c) can be applied only on class level and method level
d) Common prefix is not required to read values from properties or yml files	d) common prefix is required
e) allows to use SPEL (spring expression language)	e) Not possible to work with SPEL

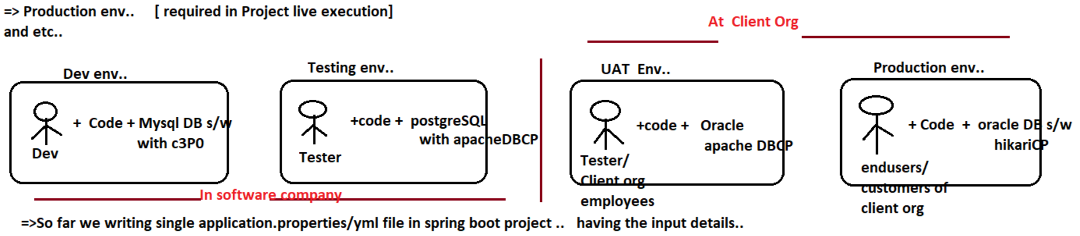
```
   @Value("#{2*10}")
   private  int age;
```
$\Downarrow$

SPEL :: allows to work with airthmetic and logical operators

Profiles in spring /Spring boot
==========================

=>Envirmonment is the setup that required to execute/test the application/project
=>For a s/w project we need to have different enviroments or profiles they are

=>Developmen Env..   [required in project development)
=>Testing   env..   [required in Project Testing]
=>UAT env../Pilot   [required in UAT -->after releasing testing at client side]
=> Production env..   [required in Project live execution]
and etc..


At Client Org
Dev env..
+ Code + Mysql DB s/w with c3P0
Dev
Testing env..
+code + postgreSQL with apacheDBCP
Tester
UAT Env..
+code + Oracle apache DBCP
Tester/ Client org employees
Production env..
+ Code + oracle DB s/w hikariCP
endusers/ customers of client org
In software company

=>So far we writing single application.properties/yml file in spring boot project ..   having the input details.. but that properites file must be changed env.. to env.. or profile to profile as discussed above..   Instread of the we can develop multiple properites files for multiple envs../profiles on 1 per env../profile basis and we can activate one env../profile based on the requirement.

syntax::
  application-<env/profilename>.properties (or) application-<env/profile>.properties

eg::
  application.properties/yml (base/default properies file)
  application-dev.properties/yml (for dev env/profile)
  application-test.properties/yml (for test env/profile)
  application-uat.properties/yml (for uat env/profile)
  application-prod.properties/yml (for production env/profile)

=>To make spring beans working for certain profile we can use @Profile annotation on the top of stereotype annotation based spring bean classes or @Bean methods of @Configuration class.

```
@Profile({"uat","prod"})
@Repository("oraCustDAO")
public  class OracleCustomerDAOImpl implenents  CustomerDAO{
 ...
 ...
 ..
}
```

```
@Profile({"dev","test"})
@Repository(" mysqlCustDAO")
public  class MySQLCustomerDAOImpl implenents  CustomerDAO{
 ...
 ...
 ..
}
```

```
@Configuration
@ComponentScan(basePackages="com.nt.dao")
public  class  PersistenceConfig{


 @Profile({"uat","test"}
 @Bean
 public   DataSource  createApacheDBCPDS(){
   ...
   ...
 }

@Bean
@Profile("dev")
public  DataSoruce  createC3PODs (){
   ...
   ..
}

@Bean
@Profile("prod")
public  DataSource  createHKCPDS (){
   ...
   ..
}
}//class
```

To activate speficic profile dynamically at runtime
==

using base/default profile/yml file   (best)
==============================

application.properties	application.yml

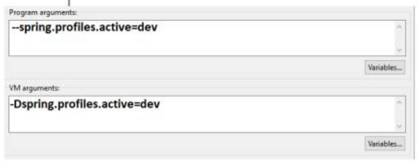
spring.profiles.active=dev	spring:
	  profiles:
	     active: dev

Using command line args (optional args)
======================
--spring.profiles.active=dev

Using System properties (VM arguments)

-Dspring.profiles.active=dev

In eclipse IDE run AS--->Run configurations --->
arguments tab ---> program arguments (command line args)
VM arguments (system properites)



```
--spring.profiles.active=dev
```

```
-Dspring.profiles.active=dev
```

Example App on spring profile using spring boot
==

a) keep spring boot mini Project ready

b) add adtional jars/dependencies in pom.xml

=> c3p0 , apachedbcp2

c) Go to DAO classes ..   write code with JdbcTemplate and also specify
@Profile on the top of classes..

```
@Profile({"uat","prod"})
@Repository("oraCustDAO")
public  class OracleCustomerDAOImpl implements  CustomerDAO{

 @Autowired
 private JdbcTemplate jt;
 ...
 ..
}
```

```
@Profile({"dev","test"})
@Repository(" mysqlCustDAO")
public  class MySQLCustomerDAOImpl implements  CustomerDAO{
@Autowired
private JdbcTemplate jt;
  ...
  ...
  ..
}
```

d) develop multiple proeprties files for multiple profiles as show below.

application-dev.properites
```
#Datasoruce cfg for  dev env.. (C3PO , mysql)
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql:///ntsp713db
spring.datasource.username=root
spring.datasource.password=root
#make spring boot work with c3p0 by breaking default algorithm
spring.datasource.type=com.mchange.v2.c3p0.ComboPooledDataSource
```

application-test.properties
```
#Datasoruce cfg for  dev env.. (apacheDBCP , mysql)
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql:///ntsp713db
spring.datasource.username=root
spring.datasource.password=root
#make spring boot work with c3p0 by breaking default algorithm
spring.datasource.type=org.apache.commons.dbcp2.BasicDataSource
```

application-uat.properties
```
#Datasoruce cfg for  dev env.. (apacheDBCP , oracle)
spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe
spring.datasource.username=system
spring.datasource.password=manager
#make spring boot work with c3p0 by breaking default algorithm
spring.datasource.type=org.apache.commons.dbcp2.BasicDataSource
```

application-prod.properties
```
#Datasoruce cfg for  dev env.. (HkiariCP , oracle)
spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe
spring.datasource.username=system
spring.datasource.password=manager
```

step4) activate one profile from application.properties

application.properties
```
#Activate profile
spring.profiles.active=dev
```

Taking spring boot profiles as yml files
==================================

application-dev. yml

```yaml
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    password: root
    type: com.mchange.v2.c3p0.ComboPooledDataSource
    url: jdbc:mysql:///ntsp713db
    username: root
```

application-test.yml

```yaml
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    password: root
    type: org.apache.commons.dbcp2.BasicDataSource
    url: jdbc:mysql:///ntsp713db
    username: root
```

application-uat.yml

```yaml
spring:
  datasource:
    driver-class-name: oracle.jdbc.driver.OracleDriver
    password: manager
    type: org.apache.commons.dbcp2.BasicDataSource
    url: jdbc:oracle:thin:@localhost:1521:xe
    username: system
```

application-prod.yml

```yaml
spring:
  datasource:
    driver-class-name: oracle.jdbc.driver.OracleDriver
    password: manager
    url: jdbc:oracle:thin:@localhost:1521:xe
    username: system
```

application.yml

```yaml
spring:
  profiles:
    active: dev
```

Writing multiple profiles using single yml file
=======================================

```yaml
spring:
 profiles: dev
 datasource:
   driver-class-name: com.mysql.cj.jdbc.Driver
   password: root
   type: com.mchange.v2.c3p0.ComboPooledDataSource
   url: jdbc:mysql:///ntsp713db
   username: root
---    # acts seperator .. must be at begining

spring:
 profiles: test
 datasource:
   driver-class-name: com.mysql.cj.jdbc.Driver
   password: root
   type: org.apache.commons.dbcp2.BasicDataSource
   url: jdbc:mysql:///ntsp713db
   username: root
---
spring:
 profiles: uat
 datasource:
   driver-class-name: oracle.jdbc.driver.OracleDriver
   password: manager
   type: org.apache.commons.dbcp2.BasicDataSource
   url: jdbc:oracle:thin:@localhost:1521:xe
   username: system
---
spring:
 profiles: prod
 datasource:
   driver-class-name: oracle.jdbc.driver.OracleDriver
   password: manager
   url: jdbc:oracle:thin:@localhost:1521:xe
   username: system
---

spring:
  profiles:
    active: prod
```

Working with Profiles in 100%Code driven confugurations
==

step1) keep 100% code driven configurations MiniProejct ready

step2) add apache dbcp2 , c3p0 ,hikaricp jars build.gradle or pom.xml file

step3) make sure that DAO classes are linked to profiles properly..using @Profile

```
@Profile({"uat","prod"})
@Repository("oraCustDAO")
public  class OracleCustomerDAOImpl implements CustomerDAO {
    ...  ......
}

@Repository("mysqlCustDAO")
@Profile({"dev","test"})
public  class MysqlCustomerDAOImpl implements CustomerDAO {\
  ....
  ....
}
```

step4) Develop PersisteConfig class having @Bean methods linked with Profiles...

PersistenceConfig.java

```
@Configuration
@ComponentScan(basePackages = "com.nt.dao")
public class PersistenceConfig {

    @Bean
    @Profile("dev")
    public  DataSource  createC3PODS() throws Exception {
        System.out.println("PersistenceConfig.createC3PODS()");
        ComboPooledDataSource ds=new  ComboPooledDataSource();
        ds.setDriverClass("com.mysql.cj.jdbc.Driver");
        ds.setJdbcUrl("jdbc:mysql:///ntsp713db");
        ds.setUser("root");
        ds.setPassword("root");
        return ds;
    }

    @Bean
    @Profile({"test"})
    public  DataSource  createApacheDBCPDSMysql() throws Exception {
        System.out.println("PersistenceConfig.createApacheDBCPDSMysql()");
        BasicDataSource bds=new BasicDataSource();
        bds.setDriverClassName("com.mysql.cj.jdbc.Driver");
        bds.setUrl("jdbc:mysql:///ntsp713db");
        bds.setUsername("root");
        bds.setPassword("root");
        return bds;
    }

    @Bean
    @Profile("uat")
    public  DataSource  createApacheDBCPDSOracle() throws Exception {
        System.out.println("PersistenceConfig.createApacheDBCPDSOracle()");
        BasicDataSource bds=new BasicDataSource();
        bds.setDriverClassName("oracle.jdbc.driver.OracleDriver");
        bds.setUrl("jdbc:oracle:thin:@localhost:1521:xe");
        bds.setUsername("system");
        bds.setPassword("manager");
        return bds;
    }

    @Bean
    @Profile("prod")
    public  DataSource  createHKCPDS() throws Exception {
        System.out.println("PersistenceConfig.createHKCPDS()");
        HikariDataSource hds=new HikariDataSource();
        hds.setDriverClassName("oracle.jdbc.driver.OracleDriver");
        hds.setJdbcUrl("jdbc:oracle:thin:@localhost:1521:xe");
        hds.setUsername("system");
        hds.setPassword("manager");
        return hds;
    }

    @Bean
    @Profile("dev")
    public  JdbcTemplate createJTUsingC3PODs() throws Exception {
        System.out.println("PersistenceConfig.createJTUsingC3PODs()");
        return new JdbcTemplate(createC3PODS());
    }

    @Bean
    @Profile("uat")
    public  JdbcTemplate createJTUsingApacheDBCPDsWithOracle() throws Exception {
     System.out.println("PersistenceConfig.createJTUsingApacheDBCPDsWithOracle()");
        return new JdbcTemplate(createApacheDBCPDSOracle());
    }

    @Bean
    @Profile("test")
```

Taking spring boot profiles as yml files
===================================

application-dev. yml

```
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    password: root
    type: com.mchange.v2.c3p0.ComboPooledDataSource
    url: jdbc:mysql:///ntsp713db
    username: root
```

application-test.yml

```
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    password: root
    type: org.apache.commons.dbcp2.BasicDataSource
    url: jdbc:mysql:///ntsp713db
    username: root
```

application-uat.yml
```
spring:
  datasource:
    driver-class-name: oracle.jdbc.driver.OracleDriver
    password: manager
    type: org.apache.commons.dbcp2.BasicDataSource
    url: jdbc:oracle:thin:@localhost:1521:xe
    username: system
```

application-prod.yml

```
spring:
  datasource:
    driver-class-name: oracle.jdbc.driver.OracleDriver
    password: manager
    url: jdbc:oracle:thin:@localhost:1521:xe
    username: system
```

application.yml

```
spring:
  profiles:
    active: dev
```

Writing multiple profiles using single yml file
==
```
 spring:
  profiles: dev
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    password: root
    type: com.mchange.v2.c3p0.ComboPooledDataSource
    url: jdbc:mysql:///ntsp713db
    username: root
---    # acts seperator .. must be at begining

 spring:
  profiles: test
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    password: root
    type: org.apache.commons.dbcp2.BasicDataSource
    url: jdbc:mysql:///ntsp713db
    username: root
---
 spring:
  profiles: uat
  datasource:
    driver-class-name: oracle.jdbc.driver.OracleDriver
    password: manager
    type: org.apache.commons.dbcp2.BasicDataSource
    url: jdbc:oracle:thin:@localhost:1521:xe
    username: system
---
 spring:
  profiles: prod
  datasource:
    driver-class-name: oracle.jdbc.driver.OracleDriver
    password: manager
    url: jdbc:oracle:thin:@localhost:1521:xe
    username: system
---

spring:
  profiles:
    active: prod
```

Working with Profiles in 100%Code driven congugurations
===

step1) keep 100% code driven configurations MiniProejct ready

step2) add apache dbcp2 , c3p0 ,hikaricp jars build.gradle or pom.xml file

step3) make sure that DAO classes are linked to profiles properly..using @Profile

```
@Profile({"uat","prod"})
@Repository("oraCustDAO")
public class OracleCustomerDAOImpl implements CustomerDAO {
   ...  ......
}

@Repository("mysqlCustDAO")
@Profile({"dev","test"})
public class MysqlCustomerDAOImpl implements CustomerDAO {\
  ....
  ....
}
```

step4) Develop PersisteConfig class having @Bean methods linked with Profiles...

PersistenceConfig.java

```
@Configuration
@ComponentScan(basePackages = "com.nt.dao")
public class PersistenceConfig {

    @Bean
    @Profile("dev")
    public  DataSource  createC3PODS() throws Exception {
        System.out.println("PersistenceConfig.createC3PODS()");
        ComboPooledDataSource ds=new  ComboPooledDataSource();
        ds.setDriverClass("com.mysql.cj.jdbc.Driver");
        ds.setJdbcUrl("jdbc:mysql:///ntsp713db");
        ds.setUser("root");
        ds.setPassword("root");
        return ds;
    }

    @Bean
    @Profile({"test"})
    public  DataSource  createApacheDBCPDSMysql() throws Exception {
        System.out.println("PersistenceConfig.createApacheDBCPDSMysql()");
        BasicDataSource bds=new BasicDataSource();
        bds.setDriverClassName("com.mysql.cj.jdbc.Driver");
        bds.setUrl("jdbc:mysql:///ntsp713db");
        bds.setUsername("root");
        bds.setPassword("root");
        return bds;
    }

    @Bean
    @Profile("uat")
    public  DataSource  createApacheDBCPDSOracle() throws Exception {
        System.out.println("PersistenceConfig.createApacheDBCPDSOracle()");
        BasicDataSource bds=new BasicDataSource();
        bds.setDriverClassName("oracle.jdbc.driver.OracleDriver");
        bds.setUrl("jdbc:oracle:thin:@localhost:1521:xe");
        bds.setUsername("system");
        bds.setPassword("manager");
        return bds;
    }

    @Bean
    @Profile("prod")
    public  DataSource  createHKCPDS() throws Exception {
        System.out.println("PersistenceConfig.createHKCPDS()");
        HikariDataSource hds=new HikariDataSource();
        hds.setDriverClassName("oracle.jdbc.driver.OracleDriver");
        hds.setJdbcUrl("jdbc:oracle:thin:@localhost:1521:xe");
        hds.setUsername("system");
        hds.setPassword("manager");
        return hds;
    }

    @Bean
    @Profile("dev")
    public JdbcTemplate createJTUsingC3PODs() throws Exception {
        System.out.println("PersistenceConfig.createJTUsingC3PODs()");
        return new JdbcTemplate(createC3PODS());
    }
@Bean
@Profile("prod")
public  JdbcTemplate createJTUsingHKCPDs() throws Exception {

 System.out.println("PersistenceConfig.createJTUsingHKCPDs()");
          return new JdbcTemplate(createHKCPDS());
     }
}
```

note:: if u do not put spring bean in any profile. then it will be used for all profiles.. in our miniProjects we can use all service, controller classes with placing in profiles to make them common for all profiles..

step5) Activate profile from client App...

```
// create BEanFacory IOC container
   AnnotationConfigApplicationContext ctx=new AnnotationConfigApplicationContext();
   //get Enviroment object from  IOC container
   ConfigurableEnvironment env=ctx.getEnvironment();
   env.setActiveProfiles("prod");
   //provide configuration class
   ctx.register(AppConfig.class);
   ctx.refresh();
   //get  Controller class object
   MainController controller=ctx.getBean("controller",MainController.class);
```

$$\begin{tikzcd}
\text{org.sf.core.env.Enviroment(I)} \\
\text{org.sf.core.env.ConfigurableEnviroment(I)} \arrow[u, "\text{exteneds}"']
\end{tikzcd}$$

Enviromnent object is IOC Container maintained internal object having profiles info, properties file info, system properties info and env..variable info..

Improved Persistenceconfig.java
==============================

```
 @Configuration
 @ComponentScan(basePackages = "com.nt.dao")
 @PropertySource("com/nt/commons/jdbc.properties")
 public class PersistenceConfig {
        @Autowired
        private   Environment env;

        @Bean
        @Profile("dev")
        public   DataSource  createC3PODS() throws Exception {
                System.out.println("PersistenceConfig.createC3PODS()");
                ComboPooledDataSource ds=new  ComboPooledDataSource();
                ds.setDriverClass(env.getRequiredProperty("jdbc.mysql.driverclass"));
                ds.setJdbcUrl(env.getRequiredProperty("jdbc.mysql.url"));
                ds.setUser(env.getRequiredProperty("jdbc.mysql.username"));
                ds.setPassword(env.getRequiredProperty("jdbc.mysql.pwd"));
                return ds;
        }

        @Bean
        @Profile("test")
        public   DataSource  createApacheDBCPDSMysql() throws Exception {
                System.out.println("PersistenceConfig.createApacheDBCPDSMysql()");
                BasicDataSource ds=new BasicDataSource();
                ds.setDriverClassName(env.getRequiredProperty("jdbc.mysql.driverclass"));
                ds.setUrl(env.getRequiredProperty("jdbc.mysql.url"));
                ds.setUsername(env.getRequiredProperty("jdbc.mysql.username"));
                ds.setPassword(env.getRequiredProperty("jdbc.mysql.pwd"));
                return ds;
        }

        @Bean
        @Profile("uat")
        public   DataSource  createApacheDBCPDSOracle() throws Exception {
                System.out.println("PersistenceConfig.createApacheDBCPDSOracle()");
                BasicDataSource ds=new BasicDataSource();
                ds.setDriverClassName(env.getRequiredProperty("jdbc.oracle.driverclass"));
                ds.setUrl(env.getRequiredProperty("jdbc.oracle.url"));
                ds.setUsername(env.getRequiredProperty("jdbc.oracle.username"));
                ds.setPassword(env.getRequiredProperty("jdbc.oracle.pwd"));
                return ds;
        }

        @Bean
        @Profile("prod")
        public   DataSource  createHKCPDS() throws Exception {
                System.out.println("PersistenceConfig.createHKCPDS()");
                HikariDataSource ds=new HikariDataSource();
                ds.setDriverClassName(env.getRequiredProperty("jdbc.oracle.driverclass"));
                ds.setJdbcUrl(env.getRequiredProperty("jdbc.oracle.url"));
                ds.setUsername(env.getRequiredProperty("jdbc.oracle.username"));
                ds.setPassword(env.getRequiredProperty("jdbc.oracle.pwd"));
                return ds;
        }


        @Bean(~~autowire~~ = Autowire.BY_TYPE)
        @Autowired
        public  JdbcTemplate createJT() throws Exception {
                return new JdbcTemplate();
        }
 }
```

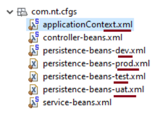
Spring profiles in xml +annotation or xml configurations based spring app development
==

we need to use "profile" attribute of <beans> tag..

step1) kepp MiniProject ready (xml+ annotations based)

step2) add apache dbcp2, c3p0 jar files

step3) takes multiple persistence-beans.xml files from for multiple profiles and import them in applicationContext.xml



```
com.nt.cfgs
    applicationContext.xml
    controller-beans.xml
    persistence-beans-dev.xml
    persistence-beans-prod.xml
    persistence-beans-test.xml
    persistence-beans-uat.xml
    service-beans.xml
```

persistence-beans-dev.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans  profile="dev" .....>
 <bean id="c3P0Ds"  class="com.mchange.v2.c3p0.ComboPooledDataSource">
    <property name="driverClass"  value="com.mysql.cj.jdbc.Driver"/>
    <property name="jdbcUrl" value="jdbc:mysql:///ntsp713db"/>
    <property name="user" value="root"/>
    <property name="password" value="root"/>
 </bean>

 <bean id="jt"  class="org.springframework.jdbc.core.JdbcTemplate">
    <constructor-arg  ref="c3P0Ds"/>
 </bean>

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

persistnece-beans-test.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans  profile="test" .....>
 <bean id="dbcpDs"  class="org.apache.commons.dbcp2.BasicDataSource">
    <property name="driverClassName"  value="com.mysql.cj.jdbc.Driver"/>
    <property name="url" value="jdbc:mysql:///ntsp713db"/>
    <property name="username" value="root"/>
    <property name="password" value="root"/>
  </bean>
  <bean id="jt"  class="org.springframework.jdbc.core.JdbcTemplate">
    <constructor-arg  ref="dbcpDs"/>
  </bean>
  <context:component-scan base-package="com.nt.dao"/>
</beans>
```

persistence-beans-uat.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans  profile="uat" ....>
 <bean id="dbcpDs"  class="org.apache.commons.dbcp2.BasicDataSource">
    <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="jt"  class="org.springframework.jdbc.core.JdbcTemplate">
    <constructor-arg  ref="dbcpDs"/>
  </bean>

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

persistence-beans-prod.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans  profile="prod"   ....>
 <bean id="hkDs"  class="com.zaxxer.hikari.HikariDataSource">
    <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="jt"  class="org.springframework.jdbc.core.JdbcTemplate">
    <constructor-arg  ref="hkDs"/>
 </bean>

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

step4) Develop DAO classes ready to work for profiles having @Profile

```
@Profile({"uat","prod"})
@Repository("oraCustDAO")
public  class OracleCustomerDAOImpl implements CustomerDAO {
    ...   ......
}

@Repository("mysqlCustDAO")
@Profile({"dev","test"})
public  class MysqlCustomerDAOImpl implements CustomerDAO {\
  ....
  ....
}
```

step5) Activate profile from Client App

```
ClassPathXmlApplicationContext ctx=new ClassPathXmlApplicationContext();
        //get Envirmment object
        ConfigurableEnvironment env=(ConfigurableEnvironment) ctx.getEnvironment();
        //set active profile
        env.setActiveProfiles("prod");
        //set spring bean cfg file
        ctx.setConfigLocation("com/nt/cfgs/applicationContext.xml");
        ctx.refresh();
        // get Controller Bean class object..
        MainController controller = ctx.getBean("controller", MainController.class);
```