Nov 16 Spring Boot Intro

Spring Boot

==========

=> It is another framework that is developed on the top of spring framework.

=> spring boot = spring ++ (Either avoid xml cfgs or minimize xml cfgs]

spring boot = spring + Embedded server + Embedded DB s/ws + AutoConfiguration + no xml cfgs

(Lots common things will come automatically ..Such as
=>classes as spring beans
=>Injections on spring beans
=>jar files
=>DB setup
=> servers
=> plugins
=>Configurations
and etc..]

=>Working with technologies is like washing clothes with hands (Every thing we should take care)

=>Working with spring framewrok is like washing clothes with semi-automated washin machine

=>Working with spring boot framewrok is like washing clothes with fully-automated washin machine

JSE,

=>Project1 with JEE Technologies (JDBC, Servlet,Jsp,Jndi and etc..) (Assume 2000 LOC is required)

=>Project1 with spring framework (Assume 800 LOC is required)

=>Project1 with spring boot framework (Assume 300 LOC is required)

JSE: Java Standard edition
JEE : Jakarta Enteprise Edition (new)
Java Enteprise Edition (old name)

Plain Jdbc App (Java App with Technologies)

==========

=>Load jdbc driver class (Activate jdbc driver)
=>Establihs the Connection with DB s/w
=> Create JDBC Statement object
[Common logics)

=>SEnd and execute SQL Query in Db s/w
=>Gather results and process Results
[Application specific logics)

=> Exception handling
=>Close connection with Db s/w
[Common logics)

Here Programmer needs to take care of both Common logics and Application specific logics.

Spring JDBC App (Spring FrameWork App)

==========

=>set of items is called pool
=>jdbc con pool contains set of readily avaiable jdbc con objects.

DataSource (Jdbc con pool) providers in the market

---

apache dbcp , hikaricp , c3p0 and etc..
(best)

//AppConfig.java

==========

```
@Configuration
@ComponentScan(base-packages="......")
public class AppConfig{
    @Bean(name="hkDs")
   public  HikariDataSource createDs(){
     HikariDataSource hkDs=new HikariDataSource();
      hkDs.setDriverClassName("oracle.jdbc.driver.OracleDriver");
      hkDs.setUrl(".....");
      ....
       ...
      return HkDs;
    }
}
```

java.sql.DataSource(I)
↑ implements
<pkg>.HikariDataSource(c)

@Component:: makes java class as spring bean
@Repository :: makes java class spring bean cum DAO class having capability translaing jdbc exceptions to spring exceptions

=>In spring env... we get the following support from spring's IOC container
a) Loading spring bean classes and creating their objects
b) Injecting Dependent class object to target class obj

=>Spring env.. Developing classes having logics is the repsonsbility of the Programmers.. creating objects for those classes and injecting one object with another objct will be taken by Spring IOC contaienr.

```
//DAO class (Class having persistence logic)
@Repository (or) @Component
public calss StudentDAOImpl implements IStudentDAO{
   @Autowired
   private  DataSource ds;    //Injects DataSource object

   public int getStudentsCount(){
     //get pooled jdbc con
     Connection con=ds.getConnection();
       ...
       ... //write the remaining jdbc code
       ...
    }
}
```

required jars :: spring-jdbc-<ver>.jar
ojdbc6.jar , hikari cp jar ,
+ other spring jar files

Nov 16.1 Spring Boot Intro

Spring Boot JDBC App (Developed using spring env..)

=>Spring boot provides ready made spring boot projects called spring boot starters ..These starters are jar files supporting AutoConfiguration i.e based on staters we add lots of common logics as disccussed above will taken care by spring boot itself

(Lots common things will come automatically ..Such as
=>classes as spring beans
=>Injections on spring beans
=>jar files
=>DB setup
=> servers
=> plugins
=>Configurations
and etc..]

if we add "spring-boot-starter-jdbc" to the Project (add to CLASSPATH/BuildPATH) it will given lots jar files to work with spring jdbc and gives multiple classes as spring beans through AutoConfiguration.

=>DataSource (HikariDataSource)
=>JdbcTemplate
=>NamedParameterJdbcTemplate
and etc..

These are classes automatically and internally becoming spring beans (No need of using @Bean methods)

if we are using maven/gradle add spring boot starters to pom.xml (maven) build.gradle(gradel)

application.properties

(Gives inputs/instructions to spring boot related AutoConfiguration and for other operations)

```
spring.datasource.driver-class-name=oracle.jdbc.driver.Oracle
spring.datasoruce.url=.....
spring.datasource.username=....
.....
....
```

note:: we can given jdbc properties instructions to "spring-boot-starter-jdbc" to create DataSource object pointing jdbc con pool of certain Db s/w. (if give oracle details .. the jdbc con pool will be created for oracle .. and ....)

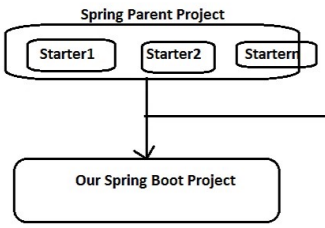
=>In spring boot App
spring Benifits +
Autoconfiguration benifits

```
//DAO class (Class having persistence logic)
@Repository (or) @Component
public calss StudentDAOImpl implements IStudentDAO{
    @Autowired
    private DataSource ds;    //Injects DataSource object

    public int getStudentsCount(){
        //get pooled jdbc con
        Connection con=ds.getConnection();
          ...
          ... //write the remaining jdbc code
          ...
        }
    }
```

required jars :: spring-boot-starter-jdbc jar
ojdbc6.jar

What is Spring Boot

Spring boot is collection of parent pojects (Spring boot Starter Projects) that providees multiple Commom logics to our project (our spring Boot Project nothing but child Project) throuhg Auto Configuration process



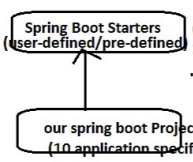
Available in Maven Repository/Gradle Repositiry (Internet Based)

Starters perform AutoConfigiration Activites

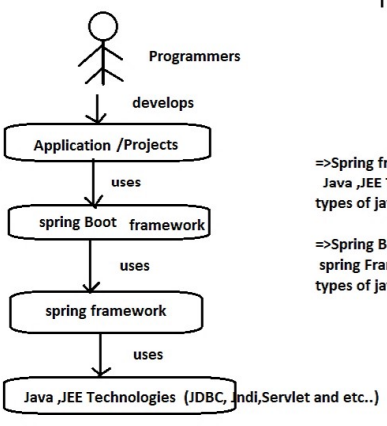
(Lots common things will come automatically ..Such as
=>classes as spring beans
=>Injections on spring beans
=>jar files
=>DB setup
=> servers
=> plugins
=>Configurations
and etc..]

(Bank1)
Project 1 (spring)
10 common logics
10 specific logics

(Bank2)
Project2 (spring)
10 common logics
10 specific logics



(Takes care of 10 Common Logics through AutoConfiguation)



=>Spring fraramework provides abstraction on Java ,JEE Technologies and simplifies various types of java,JEE Project development..

=>Spring Boot fraramework provides abstraction on spring Framework and simplifies various types of java,JEE Project development..

Nov 17 Spring Boot Intro2

Is spring boot alternate/ replacement for spring framework?

Ans) No , Spring boot internally uses spring framework and provides abstraction spring framework programming .. So spring boot is not replacement for spring framework .. Infact it compliments spring framework and simplfies the Application devleopment process.

Spring boot = spring ++

What is spring boot starter?

Ans) spring boot starter in ready made spring project available in Internet repositories like maven /gradle repositories in the form of advanced jar files (dependencies) having capability to perform common task of the Project or App development through AutoConfiguration fetaures.

=> if u r arranging apis/ librraries manually then we need arrange both main and dependent jar files of those apis /libraries which is very complex.

spring-context-support (main jar file for spring api)
|--->it is having nearly 5/6 dependent jar files (searching and arranging them is very complext)

=> if u r using maven /gradle tool to create and develope projects they will take care arranging both main and dependent jar files from internet repositories.

=> if ur using maven/gradle tool to add spring boot starters to the project the we will get main jar files, dependent jar files and relevant jar files

if we add "spring -boot-starter-jdbc " to the project
=> we get spring-jdbc-<ver>.jar (main jar file)
spring-tx-<ver>.jar (dependent jar file)
spring-context-<ver>.jar file and its dependents jar files (5 to 6 jar files)
hikaricp-<ver>.jar and its dependent jar files
and other jar files

main and dependent jar files.

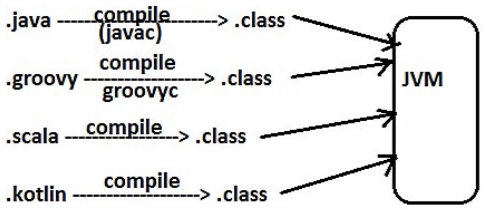
relavent jar files

spring starter jar or dependency syntax
spring-boot-starter-<variant>-<ver>.jar

eg: spring -boot-starter-jdbc-<ver>.jar
spring-boot-starter-web-<ver>.jar
spring-boot-starter-mail-<ver>.jar
spring-boot-starter-batch-<ver>.jar
spring-boot-starter-security-<ver>.jar
and etc.. (1000+ starters are available)

---

java , groovy ,koltin ,scala,Go and etc.. are called JVM based languages



the .class files generated by the compilers of different JVM based languages can be executed using same JVM.. So these are called JVM based Lanauges..

=>@SpringBootApplication = @Configuration + @ComponentScan +@EnableAutoConfiguration + others..
=>Insteead of using @Value annotations for multiple time to read values from properites file.. we can use @ConfigurationProperties annotation only for 1 time on top of class.

Pros/Cons of Spring Boot

Pros of Spring Boot:

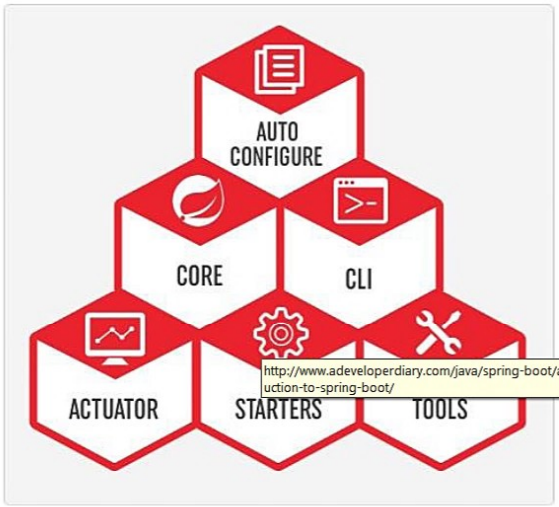
- It is very easy to develop Spring Based applications with Java or Groovy.
- It reduces lots of development time and increases productivity.
- It avoids writing lots of boilerplate Code, Annotations and XML Configuration.
- It is very easy to integrate Spring Boot Application with its Spring Ecosystem like Spring JDBC, Spring ORM, Spring Data, Spring Security etc.
- It follows "Opinionated Defaults Configuration" Approach to reduce Developer effort
- It provides Embedded HTTP servers like Tomcat, Jetty etc. to develop and test our web applications very easily.
- It provides CLI (Command Line Interface) tool to develop and test Spring Boot (Java or Groovy) Applications from command prompt very easily and quickly.
- It provides lots of plugins to develop and test Spring Boot Applications very easily using Build Tools like Maven and Gradle.
- It provides lots of plugins to work with embedded and in-memory Databases very easily.

Limitation of Spring Boot:

It is very tough and time consuming process to convert existing or legacy Spring Framework projects into Spring Boot Applications. It is applicable only for brand new/Greenfield Spring Projects.

Nov 17.1 Spring Boot Intro2.png

Spring Boot Components



Spring Boot Auto Configure

Module to auto configure a wide range of Spring projects. It will detect availability of certain frameworks (Spring Batch, Spring Data JPA, Hibernate, JDBC). When detected it will try to auto configure that framework with some sensible defaults, which in general can be overridden by configuration in an application.properties/yml file.

Spring Boot Core

The base for other modules, but it also provides some functionality that can be used on its own, eg. using command line arguments and YAML files as Spring Environment property sources and automatically binding environment properties to Spring bean properties (with validation).

Spring Boot CLI

A command line interface, based on ruby, to start/stop spring boot created applications.

Spring Boot Actuator

This project, when added, will enable certain enterprise features (Security, Metrics, Default Error pages) to your application. As the auto configure module it uses auto detection to detect certain frameworks/features of your application.For an example, you can see all the REST Services defined in a web application using Actuator.

Spring Boot Starters

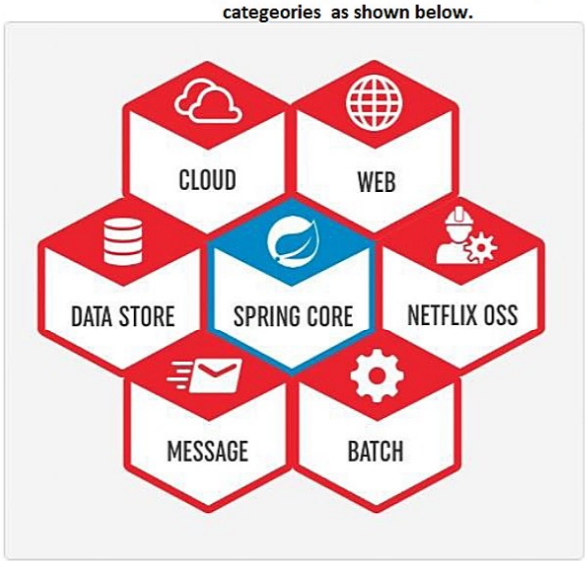
Different quick start projects to include as a dependency in your maven or gradle build file. It will have the needed dependencies for that type of application. Currently there are many starter projects (We will learn about few of them in the next section) and many more are expected to be added.

Spring Boot Tools

The Maven and Gradle build tool as well as the custom Spring Boot Loader (used in the single executable jar/war) is included in this project.

Spring Boot starters categorization
==========

spring is giving 1000+ starters following in to different categeories as shown below.



Nov 17.2 Spring Boot Intro2

Different ways developing spring boot App

==========

a) Using CLI (Command Line Interface- Non IDE env..)

b) Using start.spring.io website (Best3)

(which creates project by taking ur inputs and take that project to add more code)

c) Using IDEs

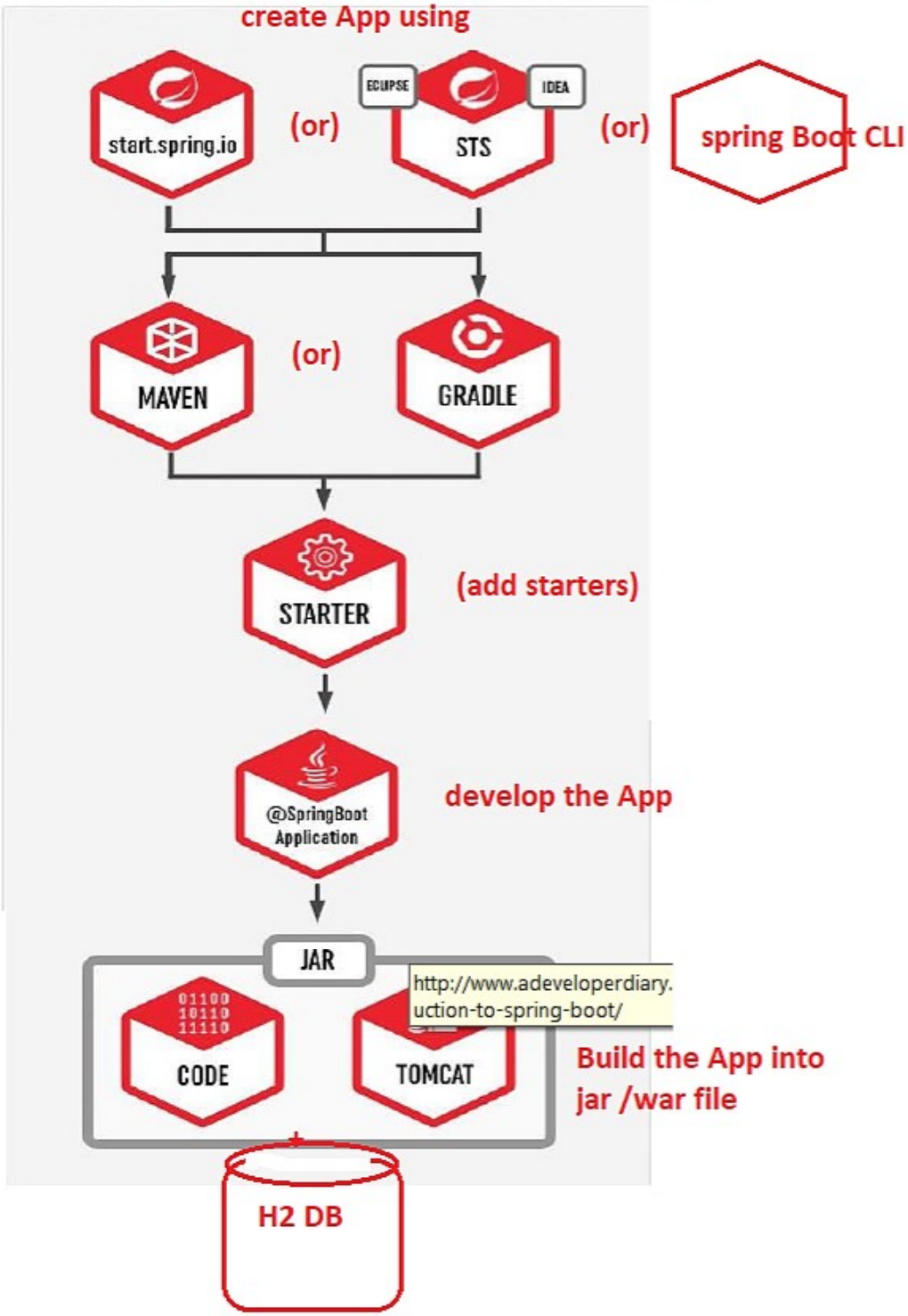
i) use STS IDE directly (Best2)

ii) use Eclipse IDE with STS plugin in (Best1)

iii) IntelliJ IDE

Spring Boot App devleopment process

==========



Nov 18 Spring Boot Intro3

Using Spring boot , we can develop standalone Apps, web applications, Distributed apps and Enterprise apps app

standalone :: The app that is specific to one computer and alllows only one user at a time to manipulate the app is called standalone app

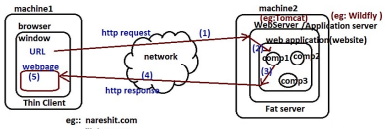
eg:: calculator app , Desktop game , Antivirus s/w , Awt/swing frame App and etc..

singleline statement : In Java Standalone App means it is class with main(-) method

To develop these Apps we need core java(jdk) (or) Spring core (or) spring boot core

web application: It is Client -server where Server is a software App and client is browser (web browser) having capability to process the Http requests and to generate http responses.

=>web applications are two- tier Apps falling under Thin Client -Fat server App



eg:: nareshit.com
flipkart.com
citibank.com and eetc..

=>webServer or Application server automates the execution of web application and its web comps 24/7 (24 hours in a day and 7 days in a week)

'eb server ++

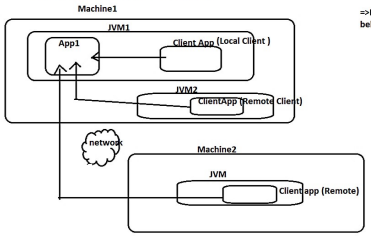
Application server = w

To develop web application in Java we use
servlet, jsp technologies
spring MVC /spring web MVC framework
spring boot MVC framework
JSF (old) , struts (old)

Distributed App
==========

The App that Allows both local and remote Clients of different types to access and invoke b.logics is called Distributed App

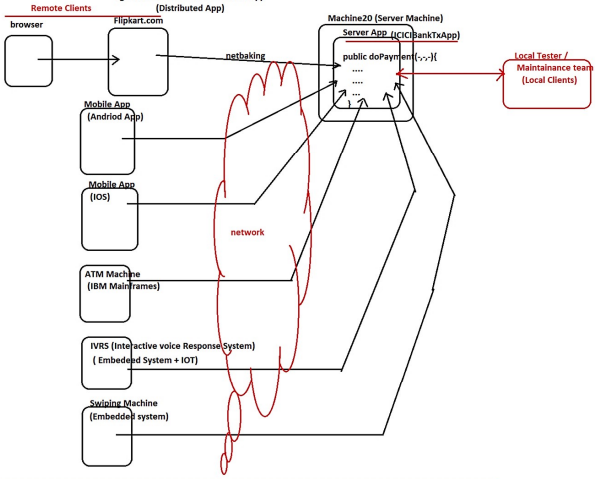
=>if App and its client are there in the same JVM then that client is Local client to App
=>if App and its client are there in two different JVMs same computer or different computer then the Client is called Remote Client to App..

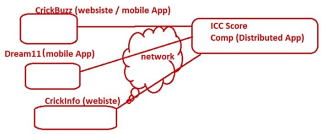


=>In Computer we can launch multiple JVMS beloging to same versions or diferent versions at a time

=> Distributed Apps are also Client-Server App where both Client and server Apps are s/w apps distributing work .. So we can these Apps Fat Client -Fat Server App.

eg:: ICICI Bank Transcation server App (Distributed App)





Simple Object Access Protocol (SOAP)

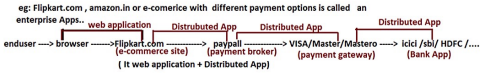
To develop Distributed Apps in java
=> RMI ,EJB, CORBA (outdated)
=> Jax-RPC, JAX-WS,Axis, Apache CXF and etc,, (SOAP based web services) (Loosing fame)
=> Jax-RS , Spring Rest, Rest Easy and etc.. (Restfull web services) (Doing well)

REST :: Representional State

Entterprise Apps
==========

An Entprise App can be a web application or distriuted App or combination of both dealting with compltext logics.

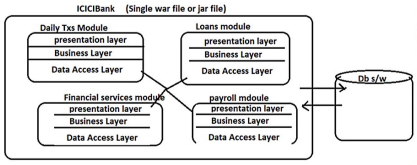
eg: Flipkart.com , amazon.in or e-comerice with different payment options is called an enterprise Apps..



=>Spring Boot can be used to develop the above said apps either in Monolithic Architecture and MicroServices Architecture.

Monolithic Architecture
==========

=>All the services /operations /modules will be placed in single unit as single Project having layers



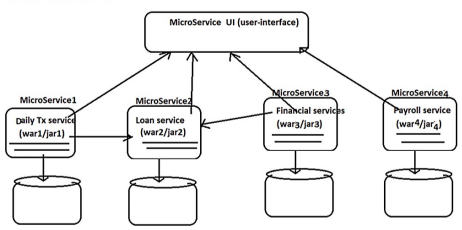
=>All these services/module of the Project should handled, deployed and mantained togather as single Project.
=>we can not controll memory management , CPU Utilization and others services seperately for each module or service (we must do for all modules/service go gather)

=> if we do modifications in one module/service that effect other other modules/ sevices

=> if service /module is down then it effects all other services / modules

[To overcome these problems go for Microservices architecture]

MicroServices Architecture
==========



=> Here each service/operation is a seperate project called micro service
=> we can give more memory and cpu time certain micor service u want

=> if one micro service is donwn .. only service releated operations will be effected .. other services will not be affected.

=> we can sell each micro service to other companies becoz it running indepedendly

=>This architecture is very good in Cloud env ...

Spring Boot is in Microservice architecture based application devleopment becoz every microservice needs some common logics like DB setup or security and etc.. but we can get them through auto configuraiton.. So boiler plate code development by programmer will be reduced and the productivity will be improved..

Nov 19 Spring Boot First App

Thumb rule to followo while developing spring Boot App

a) Cfg user-defined classes as spring beans using @Stereo type annotations
b) Cfg pre-defined classes as spring beans using @Bean methods of @Configuraiton class
if they are not coming as spring bean through AutoConfiguration based on starter
we add (Provide inputs/instructions of autoConfiguration using application.properites file)
c) get IOC container given by SpringApplication.run(-) method and use it to get spring beans and to invoke b.methods (static factory method)

This method bootstraps (starts) spring App by doing multiple activities internally in a sequence including IOC container creation and returns the referecne ponting to that IOC container.

=>Since spring boot is spring ++ we can perform all Dependency Management activities of spring in spring boot including both Dependency Lookup and Dependency Injection

Dependency Management (arraging dependent class obj to tarager class object)

1.Dependency Lookup
=>Target class object searches and gets Dependent class obj

2.Dependency Injection
=>Here the underlying IOC container dynamically assigns Dependent class object to target class obj injects

(a) setter Injection
(b) constructor Injection
(c) Field Injection
(d) Arbitarty method injection
(e) Aware Injection /Interface injection
(f) LookUp Method Injection
(g) Method Injection/Method Replacer

=>The Main class (where main(-) is placed) of spring boot App will be annotation with @SpringBootApplication Annotation whic internall contains other 3 annotations

a) @Configuration :: Makes main class as Confiuguration class to make place @Bean methods where pre-defined classes will be made as spring beans
b) @ComponentScan :: Automatically scans current pkg and its sub pkgs to make streo type annotation classes as spring beans automatically . In that process it automatically recognizes other @Configuration classes avaiable in the same pkgs.
c) @EnableAutoConfiguration :: performs AutoConfiguration during the app startup based on spring boot starters that are added. In that process certain pre-defined classes given spring boot starters become spring beans and participates in neccessaryInjections.

note:: In new version @SpringBootApplicaiton annotaiton internally uses @SpringbootConfiguration which internally uses @Configuration annotation .

@SpringBootConfiguration = @Configuration (from 2.2+)

First spring boot Standalone App (No AutoConfiguration , Only Dependency Injection (Filed Injection))

	target class	Dependent class	[use @Bean method to cfg as spring bean]
Stereo type annotaiton (@Component)	WishMessageGenerator	java.util.Calendar (AC) pre-defined	

Should get sys date and time from the Injected Calendar obj (GregorianCalendar obj) to generate wish messsage based on the current hour of day

When we ask container to make Calendar class as spring bean.. it actually creates its sub class GregorianCalendar and makes its object as spring bean.

Using xml cfgs :: 5 (Filed Injection , arbitary method injections are not possible)
using annotation cfgs :: 7

=>cyclic injection is not a seperate injection
it also internally uses setter or contructor Injection.

Procedure to develop spring boot based Standand App

step1) keep the folowing software setup ready
Eclipse latest version (2019,20,21)

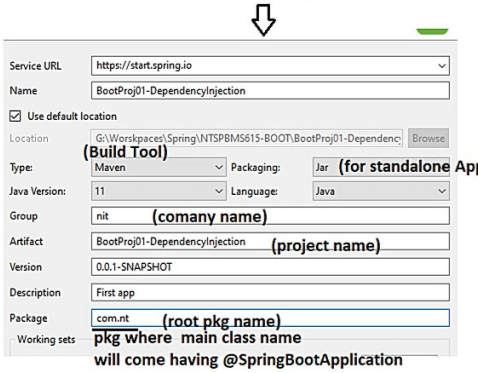
2
step) make sure that STS plugin installed to the Eclipse IDE

Help --- eclipse market palce ----> STS ---> go ---> select
Spring Tools 3 (Standalone Edition) 3.9.14.RELEASE
next ---> accept terms and conditions --->next --->
restart IDE ...


(these icons will come menu bar
support spring,spring boot dash board activities)

step3) create spring boot starter Project (user-defined) adding no pre-defined starters..

File menu --->new ---> spring starter project --->



(Build Tool)
(for standalone App)
(comany name)
(project name)
(root pkg name)
pkg where main class name will come having @SpringBootApplication

spring Boot Latest version is : 2.6.0 and internally uses spring 5.3.13 version

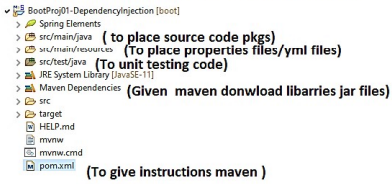
Nov 19.1 Spring Boot First App

In spring boot App it is recomand default pkg name as the root pkg name like "com.nt" and remaining pkgs as the sub packages , So the @ComponentScan annotation of @SpringBootApplication can scan root pkg and those sub pkgs automatically to recognizing streo type annotaiton classe and @Configuration classes.

```
com.nt                    @SpringBootApplication
  |----> BootProj01-DependencyInjection.java (main class)
  |-->com.nt.dao
        |--->all dao clsses
  |-->com.nt.service
        |--->all services classes
```

note:: place main class in root pkg and other spring bean classes and configuration classes in the sub pkgs of root pkg.

step4) understand directory structure of spring boot Project.



(to place source code pkgs)
(To place properties files/yml files)
(To unit testing code)
(Given maven donwload libarries jar files)
(To give instructions maven)

step5) create target class in the com.nt.beans pkg placed src/main/java folder.

```
//target class
package com.nt.beans;

import java.util.Calendar;

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

@Component("wmg")
public class WishMessageGenerator {
	@Autowired
	private Calendar calendar;

	//b.method
	public String generateWishMessage(String user) {
		//get current hour of the day
		int hour=calendar.get(Calendar.HOUR_OF_DAY);
		//generate wish message
		if(hour<12)
			return "Good Morning::"+user;
		else if(hour<16)
			return "Good AfterNoon::"+user;
		else if(hour<20)
			return "Good Evening::"+user;
		else
			return "Good Night::"+user;
	}

}
```

step6) make java.util.Calendar as spring bean using @Bean method in main class (@SpringBootApplication class)

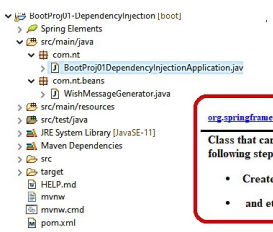
```
In main class
=============
@Bean(name="cal")
	public Calendar createCalendar() {
		return Calendar.getInstance(); //return GregorianCalendar class(sub class of Calenar) obj
	}
```

step7) get IOC container obj ref from SpringApplication.run(-,-) method and get target class object to invoke the b.method

```
//main class
package com.nt;

import java.util.Calendar;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;
import org.springframework.context.ConfigurableApplicationContext;
import org.springframework.context.annotation.Bean;

import com.nt.beans.WishMessageGenerator;

@SpringBootApplication
public class BootProj01DependencyInjectionApplication {

	@Bean(name="cal")
	public Calendar createCalendar() {
		return Calendar.getInstance(); //return GregorianCalendar class(sub class of Calenar) obj
	}

	public static void main(String[] args) {
		//Bootstrap /boot spring boot App and get IOC container ref
		ApplicationContext ctx=SpringApplication.run(BootProj01DependencyInjectionApplication.class, args);
		//get target class object
		WishMessageGenerator generator=ctx.getBean("wmg",WishMessageGenerator.class);
		//invoke b.method
		String result=generator.generateWishMessage("raja");
		System.out.println("Wish Message is ::"+result);

		//close container
		((ConfigurableApplicationContext) ctx).close();

	}

}
```

Current class as cfg class for IOC container that is created internally

step8) Run the applicaiton...
(main class as spring boot app or as Java Application)



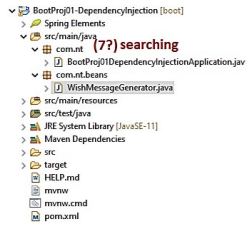
org.springframework.boot.SpringApplication

Class that can be used to bootstrap and launch a Spring application from a Java main method. By default class will perform the following steps to bootstrap your application:

- Create an appropriate `ApplicationContext` instance (depending on your classpath)
- and etc...

Nov 20 Spring Boot First App Flow

Spring Boot Application Flow



//target class

```
package com.nt.beans;

import java.util.Calendar;

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

@Component("wmg")
public class WishMessageGenerator {
    @Autowired
    private Calendar calendar;

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

    //b.method
    public String generateWishMessage(String user) {
        System.out.println("WishMessageGenerator.generateWishMessage()");
        //get current hour of the day
        int hour=calendar.get(Calendar.HOUR_OF_DAY);
        //generate wish message
        if(hour<12)
            return "Good Morning::"+user;
        else if(hour<16)
            return "Good AfterNoon::"+user;
        else if(hour<20)
            return "Good Evening::"+user;
        else
            return "Good Night::"+user;
    }

}
```

(8) finds this class

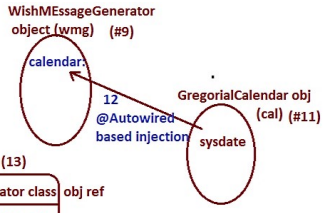
(9) Loads the and creates the object supporting pre-instantation.

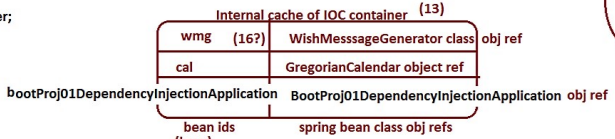
(10) detects @Autowired

(19)

(20)

11) Searches for Calendar type spring bean in com.nt and its sub pkgs (not there) and in the spring beans that coming through auto configuration (not there) and searches of @Bean methods whose return type Calendar (avaialble)






```
package com.nt;

import java.util.Calendar;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;
import org.springframework.context.ConfigurableApplicationContext;
import org.springframework.context.annotation.Bean;

import com.nt.beans.WishMessageGenerator;

@SpringBootApplication
public class BootProj01DependencyInjectionApplication {

    public BootProj01DependencyInjectionApplication() {
        System.out.println("BootProj01DependencyInjectionApplication:: 0-param construtor");
    }

    @Bean(name="cal")
    public Calendar createCalendar() {
        System.out.println("BootProj01DependencyInjectionApplication.createCalendar()");
        return Calendar.getInstance(); //return GregorianCalendar class(sub class of Calenar) obj
    }

    public static void main(String[] args) {
        System.out.println("main(-) method");
        //Bootstrap /boot spring boot App and get IOC container ref
        ApplicationContext ctx=SpringApplication.run(BootProj01DependencyInjectionApplication.class, args);
        System.out.println("IOC container class name ::"+ctx.getClass());
        System.out.println("=========================================");
        //get target class object
        WishMessageGenerator generator=ctx.getBean("wmg",WishMessageGenerator.class);
        System.out.println("============================");
        //invoke b.method
        String result=generator.generateWishMessage("raja");
        System.out.println("Wish Message is ::"+result);

        //close container
        ((ConfigurableApplicationContext) ctx).close();
        System.out.println("end of main(-)");

    } // (23)
}
```

(1) run App (spring boot App/Java App)

(6) @ComponentScan annotation of @SpringBootApp Annotaion looks for stereo type singleton scope spring bean classes in the current pkg and sub pkgs

(12)

(2)

(5) Checks the spring-boot-starters (jar files) that added to buildpath/classpath for AutoConfiguraiton activitity (In this application No specital starters add)

gets IOC container obj ref (14)

AnnotationConfig ApplicationContext obj ref)

(3)

(4)

(15)

(17)

(18)

(21)

(22)

In Process of closing Container all spring bean objects and Internal cache IOC container and many other internally creates create objects will vanished/destroyed

End of the App.

#3 & #4) SpringApplication.run(-,-) bootstraps (starts/activaties) the sprign boot App by doing multiple things internally

i) creating IOC container of type ApplicationContext

ii)IOC container takes the current main class as the configuration class .. In that process the configuration class objec will be created

ii) recognize the Properties application.properties that is placed in src/main/resources folder (in our app it an empty file .. nothing takes place)

and many other operations...

=>Every @Configuration is spring bean internally with singleton having class name as default bean id So it also will be stored in the internal cache of IOC container for reusability

```
@SpringBootApplication
    |-->uses @SpringBootConfiguraiton
            |-->uses @Configuraiton
                    |--->@Component
```

F3:: To source code any code in eclipse IDE

Nov 22 Spring Boot SEcond App AutoConfiguration

Spring Boot App using AutoConfiguration

==========

AutoConfiguration

==========

=> The Process performing the common operations based on the spring boot starters that are added is called AutoConfiguration

The Common opeartions/logics are

a) Making certain pre-defined classes as spring beans
b) Giving embedded servers
c) Giving Embedded DB s/ws
d) Giving DB cfg setup (like giving datasoruces and con pools)
e) plugins cfgs
f) getting main ,dependent and relavent jar files and etc..

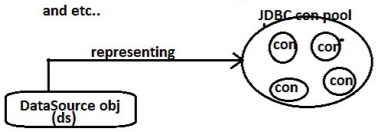
=> When spring-boot-stater-jdbc we get follwing jar files and java classes spring beans automatically through AutoConfiguration process.

jar files :: spring jar files , spring jdbc jars
Hikari cp jar files
maven /gradel puggin jars
logging jar , junit jars
and etc..

Here we get all main jar files, dependent jar files and relavent jar files

pre-defined classes as spring beans
=>HikariDataSource class (DataSource)
=>JdbcTemplate class
=>NamedParamterJdbcTemplate class
=> DasourceTransactionManager class
and etc..

javax.sql.DataSource(I)
↑ implements
HikariDataSoruce (c)



DataSource object acts entry point for jdbc con pool i.e each jdbc con object will be collected from the jdbc con pool using the support of DataSource object.

```
Connection con=ds.getConnection();
```
gets one jdbc con object from jdbc con pool

```
con.close();
```
=>returns the jdbc con obj back to jdbc con pool

Two types of jdbc con objects

created by programmer manually

1.Direct jdbc con (Not recomanded)

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

2.Pooled jdbc con object (Best)

```
Connection con=ds.getConnection();
```
Pooled jdbc con object.

JDBC con pool advantages

==========

a) gives the reusablity of jdbc con objects
b) With Minimun jdbc con objects we can make max clients talking to Db s/w
c) jdbc con object creation , managing jdbc con object and destroying jdbc con object is the responsbility of Datasource .. not the programmer.

=> In spring boot autoConfiguration , the jdbc con objects in jdbc con pool that is pointed DataSource object will be created for certain Db s/w by collection jdbc properties from application.properties file..

=> For example ,if jdbc properties are placed for oracle then the DataSource object creation in autoconfiguration creates the jdbc con pool having jdbc con objects for oracle s/w

=> For example, if jdbc properties are placed for mysql then the DataSource object creation in autconfiguration creates the jdbc con pool having jdbc con objects for mysql Db s/w

application.properties (src/main/resources)

```
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
```

=>applicaiton.properties can have both fixed keys and user-defined keys .. For autoconfuration instructions we need to fixed keys

Example App (POC) on AutoConfuration

==========

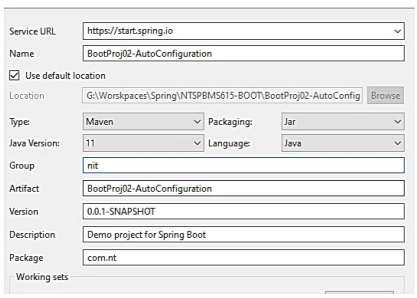
=>Develop DAO (DataAceess object) Class to get emps Count from "emp" db table of oracle

//target class
DAO class(EmployeeDAO) ---------- //Dependent class
DataSource (HikariDataSource)

=>DAO class needs Datasource obj to get one jdbc con object from jdbc con pool and to write JDBC code as persistence logic
So DAO class taerget class and Datasoruce is Dependent class

The java class class that seperate persistence logics from other logics and makes that persistence logics as the reusable logics and flexible logics to modify is called DAO class

step1) Create s spring starter project adding the follwing spring boot starters
a) JDBC API b) oracle driver



next ---->select JDBC API , oracle driver -----> next ---> Finish..

Nov 22.1 Spring Boot 2nd App AutoConfiguration

step2) add the following entries in application.properites file

application.properties

```
#DataSource cfg
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
```

jdbc properties

fixed keys

Collection key from here (https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html#application-properties.data)

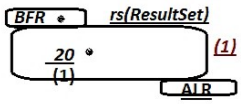
(These entries in application.properites given instruction to spring autoconfiguration process towards creating JDBC DataSource object and jdbc con objects in the JDBC con pool)

step3) Develop the DAO class injecting DataSoruce object (created through AutoConfiguration) using @Autowired property.

//DAO class

```
package com.nt.dao;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

import javax.sql.DataSource;

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


@Component("empDAO")
public class EmployeeDAO {
private static final String GET_EMPS_COUNT="SELECT COUNT(*) FROM EMP";

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

 // method with persistence logic
public  int  getEmpsCount() throws Exception {
 //get poold jdbc con object
 Connection con=ds.getConnection();
 //create PreparedStatmnent object having SQL query
 PreparedStatement ps=con.prepareStatement(GET_EMPS_COUNT);
 //exceute the Query
 ResultSet rs=ps.executeQuery();
 //process the resultSet
  rs.next();
  int count=rs.getInt(1);
  //close close jdbc objects
    rs.close();
    ps.close();
    con.close();
  return count;
 }//method
}//classs
```

BFR:: Before First record
ALR : After Last Record

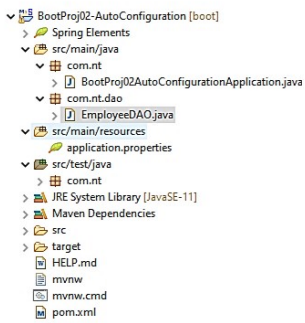


rs.next()
int count=rs.getInt(1);

step3) In the main class get access to IOC container from springApplication.run(-) method and call ctx.getBean(-) to access DAO class object ref and invoke method on it.

main class

```
package com.nt;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;
import org.springframework.context.ConfigurableApplicationContext;

import com.nt.dao.EmployeeDAO;

@SpringBootApplication
public class BootProj02AutoConfigurationApplication {

    public static void main(String[] args) {
        //get IOC container
        ApplicationContext ctx=SpringApplication.run(BootProj02AutoConfigurationApplication.class, args);
        //get DAO class obj ref
        EmployeeDAO dao=ctx.getBean("empDAO",EmployeeDAO.class);
        //invoke the method
        try {
        System.out.println("emps  count ::"+dao.getEmpsCount());
        }
        catch(Exception e) {
            e.printStackTrace();
        }

        //close container
        ((ConfigurableApplicationContext) ctx).close();

    }//main
}//class
```



step5) Run the App

Run as spring boot App

Nov 23 AutoConfiguration Internals

Q) How does spring boot picks up the classes from spring boot starter jar files to make then spring bean as part AutoConfiguration operation.

Ans) Based on the starter jar file we add to the spring boot Project , we will make the classes as spring beans through AutoConfiguration process by collecting class names from META-INF/spring.factories file of spring-boot-autoconfigure-<ver>.jar file

In spring.factories file

```
83 org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration,\
84 org.springframework.boot.autoconfigure.jdbc.JdbcTemplateAutoConfiguration,\
85 org.springframework.boot.autoconfigure.jdbc.JndiDataSourceAutoConfiguration,\
86 org.springframework.boot.autoconfigure.jdbc.XADataSourceAutoConfiguration,\
87 org.springframework.boot.autoconfigure.jdbc.DataSourceTransactionManagerAutoConfiguration,\
```

All these are configuration classes in which @Bean methods will be there directly or indirectly to make corresponding classes as spring beans

For Example DatasourceAutoConfiguration class contains @Bean method to make HikariDataSource class as spring bean through AutoConfiguration class.

sample code DataSourceAutoConfiuation class

```
@Configuration(proxyBeanMethods = false)
    @Conditional(PooledDataSourceCondition.class)
    @ConditionalOnMissingBean({ DataSource.class, XADataSource.class })
    @Import({ DataSourceConfiguration.Hikari.class, DataSourceConfiguration.Tomcat.class,
            DataSourceConfiguration.Dbcp2.class, DataSourceConfiguration.OracleUcp.class,
            DataSourceConfiguration.Generic.class, DataSourceJmxConfiguration.class })
    protected static class PooledDataSourceConfiguration {

    }
```

```
@Bean
    @ConfigurationProperties(prefix = "spring.datasource.hikari")
    HikariDataSource dataSource(DataSourceProperties properties) {
        HikariDataSource dataSource = createDataSource(properties, HikariDataSource.class);
        if (StringUtils.hasText(properties.getName())) {
            dataSource.setPoolName(properties.getName());
        }
        return dataSource;
    }
```

In spring boot 2.x , if spring -boot-starter-jdbc is added the DataSource object comes as spring bean through AutoConfiguration based on given algorithm (priority order)

a) Hikari DataSource
b) TomcatCp Datasource (if HikariDatasource is not avaiable)
c) apache dbcp2 (if HikariDataSource, Tomcatcp Datasource not avaialble)
d) Oracle UCP (if HikariDataSource, Tomcatcp Datasource, apache dbcp2 are not avaialble)

How to make spring boot app giving Tomcat CP DataSource through Auto Configuration?

step1) exclude hickari cp jar from project build path using maven <exclusion> tags

Go to dependency hieararchy ---> seaach HikariCP jar file ---> right click ---> exclude maven artificat ---> ok ---> reflected code in pom.xml file

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-jdbc</artifactId>
        <exclusions>
            <exclusion>
                <groupId>com.zaxxer</groupId>
                <artifactId>HikariCP</artifactId>
            </exclusion>
        </exclusions>
    </dependency>
```

=>Tomcat CP /Tomcat JDBC can used indenpendently or can be used with Tomcat server.
=> Tomcat CP/Tomcat JDBC is no way tightly coupled Tomat server.

step2) Add Tomcat CP jar file to pom.xml file

```
<!-- https://mvnrepository.com/artifact/org.apache.tomcat/tomcat-jdbc -->
<dependency>
  <groupId>org.apache.tomcat</groupId>
  <artifactId>tomcat-jdbc</artifactId>
</dependency>
```

Collect from mvcrepository.com by searching for tomcat-jdbc

Making App Working with apache dbcp2 DAtaSource

step1) make sure that hikari cp and tomcat cp jar files not there in the project In pom.xml

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-jdbc</artifactId>
        <exclusions>
            <exclusion>
                <groupId>com.zaxxer</groupId>
                <artifactId>HikariCP</artifactId>
            </exclusion>
        </exclusions>
    </dependency>

    <!-- https://mvnrepository.com/artifact/org.apache.tomcat/tomcat-jdbc -->
<!-- <dependency>
   <groupId>org.apache.tomcat</groupId>
   <artifactId>tomcat-jdbc</artifactId>
</dependency>
    -->
```

step2) add apache dbcp2 jar file (dependency) to pom.xml file by collecting mvnrepository.com

```
<!-- https://mvnrepository.com/artifact/org.apache.commons/commons-dbcp2 -->
<dependency>
   <groupId>org.apache.commons</groupId>
   <artifactId>commons-dbcp2</artifactId>
</dependency>
```

---

if we place all the 4 DataSources related jar files (dependencies) in pom.xml file then which data source comes through AutoConfiguration?

Ans) Since HikariCP having high priority , So it will be taken

How to make spring Boot AutoConfuguration process breaking the Default DataSource priority algorithm and How to configure our choice DataSource class as first Priority DataSoruce class of AutoConfiguration?

Ans) specify DAtaSource type in application.properties as shown below

applicaiton.properties

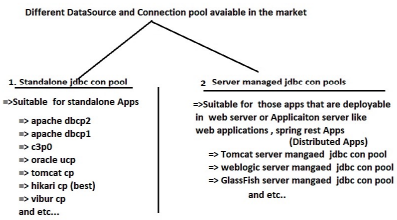
```
spring.datasource.type=oracle.ucp.jdbc.PoolDataSourceImpl
```

This dataSource must be one of 4 dataSource clases of Datasource priority algorithm..

make sure this class related jar file added to project through pom.xml

```
<!-- https://mvnrepository.com/artifact/com.oracle.database.jdbc/ucp -->
<dependency>
  <groupId>com.oracle.database.jdbc</groupId>
  <artifactId>ucp</artifactId>
</dependency>
```

note:: while adding spring boot starter jar files and other relavent jar files there is no need of specifying version becoz based the spring boot version the other jar files version will be decided.



How to configure our DataSource class in spring boot Project which is not part DataSource priority algorithm Like we want to use c3P0 , vibur and etc dataSource in spring boot Project?

Ans) place u r choice Datasource jar file to classpath /build path using pom.xml and write @Bean method in main class to make that DataSource obj as spring bean.

In pom.xml

```
<!-- https://mvnrepository.com/artifact/com.mchange/c3p0 -->
<dependency>
  <groupId>com.mchange</groupId>
  <artifactId>c3p0</artifactId>
  <version>0.9.5.5</version>
</dependency>
```

In Main class (@SpringBootApplication class)

```
@Bean
    public DataSource createDs() throws Exception{
        ComboPooledDataSource cds=new ComboPooledDataSource();
        cds.setDriverClass("oracle.jdbc.driver.OracleDriver");
        cds.setJdbcUrl("jdbc:oracle:thin:@localhost:1521:xe");
        cds.setUser("system");
        cds.setPassword("manager");
        return cds;
    }
```

Nov 24 Spring Boot Layered Application

How to make certain class of spring boot starter not participating in Autoconfiguration though that spring boot starter is added to class path or build path?

Ans) pass relavent AutoConfiguration class name which is present in spring.factories file of spring-boot-autoconfigure-<ver>.jar file in the "exclude" param of @Spri ngBootApplication annotation.

In main class
==========

```
@SpringBootApplication(exclude = DataSourceAutoConfiguration.class)
```

execlude is Class[] type param, so we can multiple values here..

=>if DataSourceAutoconfiguration is execlued and @Bean method is placed making DataSource object as spring bean then what happens?

Ans) App uses the DataSource object created by @Bean method

Q) Spring Boot AutoConfiguration is giving certain DataSource object through AutoConfiguration and programmer have placed @Bean method explicitly to make another DataSource object as spring bean .. Tell which DataSource object will be used by spring boot App?

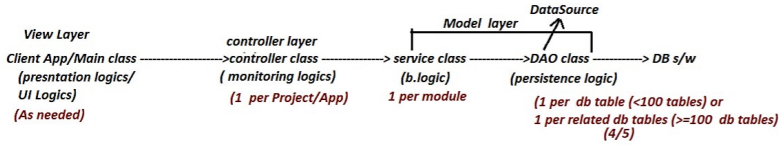
Ans) @Bean method DataSource object gets priority.

Spring Boot Mini Project as Layered Application
==========

Layer : The Logical partiation of the Project/ App that represents speciific logics is called layer
eg: serivce layer (contains b.logic)
eg: persistence layer/DAO layer (contains persistence logics)
eg: controller layer (monitors the application flow)
eg: presntation layer (contains UI logics)
and etc..

Standalone Layered App (BASED ON MVC Architecture)
==========

MVC
====
M--->Model layer ---> represents data generated by b.logic +persinstenc logic
V --->View layer ---> represents presentation logics/UI Logics
C----> Controller layer --->represents Monitoring logics

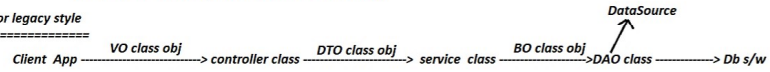


=> if 1 layer /class wants to pass more than 3 details/values to anothe layer or class then we need to pass them as Java Bean class object (java class with setter and getter methods)

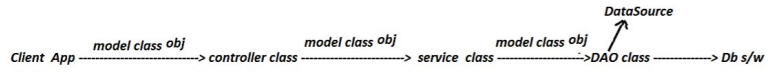
=>Earlier we use work with 3 types of Java beans to carry data across the layers
VO class (Value Object class)
DTO class (DataTransfer Object class)
BO class/Entity class/Model class (Business Object clas)

=> Instead of taking diffeent types of Java beans in one project .. now they are preffering to take only one java bean to pass the data across multiple layeres of a Project that is Model class (java bean class) [According to requirement this class acts as VO class ,DTO class and also BO class)

Old or legacy style
==========



Modren Approach
==========



=>Model means data ...The class that carrying is called Model class

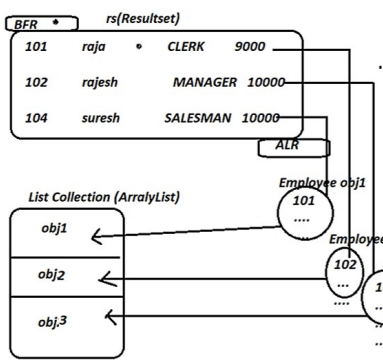
In this layered App I can spring Boot supplied IOC container
a) To give Datasource object through Autoconfiguration
b) To Inject DataSource obj to DAO class using @Autowired
c) To Inject DAO class obj to Service class using @Autowired
d) To Inject service class obj to controller class using @Autowired
e) To get controller class obj from IOC container and to invoke the b.method on it.

if Layered App requirement is getting Employee details from DB table baased on the given 3 desgs/jobs then

Employee ---> Model class

According above code in DAO class we need convert ResultSet object records into List of Model class objects (List of Employee class objs) suing the following logics
==========

```
Connection con=ds.getConnection(); //pooled JDBC connection object
//create Statement object
PreparedStatement ps=con.prepareStatement("SELECT EMPNO,ENAME,JOB,SAL FROM EMP
                                           WHERE JOB IN(?,?,?)");

//set query param values
  ps.setString(1,"CLERK");
  ps.setString(2,"MANAGER");
  ps.setString(3,"SALESMAN");

//execute the Query
  ResultSet rs=ps.executeQuery();
List<Employee> listEmps=new ArrayList();
While(rs.next()){
  Employee e=new Employee();
  e.setEno(rs.getInt(1));
  e.setEname(rs.getString(2));
  e.setDesg(rs.getString(3));
  e.setSalary(rs.getFloat(4));
  listEmps.add(e);
}
```




```
//Model class
public class Employee{
  private Integer eno;
  private String ename;
  private String desg;
  private Float salary;
  //setters and getters
  .....
  ....
}
```

When should i use which collection ? (How to decide to work with synchronzied collection and non-synchronized collection)

Ans) After putting data into collection, if u r planning to perform only read operations on the collection then prefer working with non-synchronized collection like ArrayList , HashMap and etc.. for better performence

After putting data into collection, if u r planning to perform both read and write operations on the collection then prefer working with synchronized collections like Vector,Hashtable and etc.. for thread safety...

story board of layred Application (spring boot app)

Nov 27 Spring Boot Layered Application

oracle Db s/w
emp(db table)

```
//Model class
@Data
public class Employee{
private Integer empno;
private String ename;
private  String job;
private Float sal;
 }
```

DAO Interface

```
public  interface IEmployeeDAO{
   public List<Employee>  getEmpsByDesgs(String desg1,String desg2,String desg3);
  }                                                                  throws Exception;
```

DAO Impl class

```
   @Repository ("empDAO")
   public class EmployeeDAOImpl implements IEmployeeDAO{
      @Autowired
      private  DataSource ds;

                                        (t)
      public   List<Employee> getEmpsByDesgs(String desg1,String desg2,String desg3)throws Exception{
         //get pooled Connection

            Connection con=ds.getConnection(); //pooled JDBC connection object
            //create Statement object
            PreparedStatement ps=con.prepareStatement("SELECT EMPNO,ENAME,JOB,SAL  FROM  EMP
                                                              WHERE JOB IN(?,?,?)");

           //set query param values
             ps.setString(1,  desg1.  );
             ps.setString(2,    desg2       );
              ps.setString(3,'.  desg3    .');

           //execute the Query
            ResultSet rs=ps.executeQuery();
List<Employee>  listEmps=new ArrayList();

  While(rs.next()){
    Employee e=new Employee();
    e.setEno(rs.getInt(1));
    e.setEname(rs.getString(2));
    e.setDesg(rs.getString(3));
    e.setSalary(rs.getFloat(4));
     listEmps.add(e);
    }

   return  listEmps;  (u)
}
```

service Interface

```
public interface IEmployeeMgmtService{
   public List<Employee>  fetchEmpsByDesgs(String desg1,
                                           String desg2,String desg3)throws Exception;
 }
```

// seviceImpl class

```
@Service("empService")
public   class EmployeeMgmtServiceImpl implements  IEmployeeMgmtService{
    @Autowired
     private  IEmployeeDAo dao;

                                  (r)
     public List<Employee>  fetchEmpsByDesgs(String desg1,
                                           String desg2,String desg3)throws Exception {

        //use DAO                                  (s)
(v)|   List<Employee> listEmps=dao.getEmpsByDesgs(desg1,desg2,desg3);
       return listEmps;
      }
   }
```

```
//Controller class

@Controller ("controller")
public  class  PayrollSystemController{
    @Autowired
  private  IEmployeeMgmtService service;

                                (p)
public  List<Employee>  gatherEmpsByDesgs(String desg1,String desg2,String desg3){
                                                                    throws Exception
   //use service class
                                (q)
(w)|  List<Employee>  listEmps=service.fetchEmpsByDesgs(desg1,desg2,desg3);
      return listEmps;
   }
 }
```

Main class / Client app

(a)run the App

```
@SpringBootApplication
SpringBootTestApp {  (b)
  p s v main(String args[]){                    (c) IOC container creation and others
(k)  ApplicationContext ctx=SpringApplication.run(SpringBootTestApp.class,args);
     //get Controller class obj                  (l)
(n)  PayrollSystemController  controller=ctx.getBean("controller",PayrollSystemController.class);

     //invoke b.method
     try{                                       (o)
(x)|  List<Employee> listEmps=controller.gatherEmpsByDesgs("CLERK",
                                                           "MANAGER",
                                                           "SALESMAIN");

(y)|  listEmps.forEach(emp->{
          System.out.println(emp);
        });
      }//try
      catch(Exception e){
        e.printStackTrace();
      }
    }

  //close container
  ctx.close();  (z)
 }
}
```

(f) Autoconfiguration activites of @SprinbBootApplication
HikariDataSource,
JdbcTemplate
and etc.. object creation
spring.factories
(uses)
(In this processs application.properites file info
will be used through Environment obj)

(h) @ComponentScan of
@SpringBootApplicaiton activites
->DAO class, service class and controller
classes object creation

(i) @Autowired activites
=>DS Injection to DAO obj
=>DAO injection to service obj
=>service injection to controller obj

(g) created beans will be kept in
Internal cache of IOC container

Internal cache of IOC container

hikariDataSource	HikariDatasource obj ref
springBootTestApp	SpringBootTestApp obj ref
...	.....
empDAO	EmployeeDAOImpl obj ref
empService	EmployeeServiceImpl obj ref
controller (m)	PayrollSystemcontroller obj ref

(j) Keeping new set of spring bean objects
in the internal cache

(z)
(All objects including
IOC container will be vanished)

application.properties (src/main/resources) Environment obj (intera object)
(d)

```
#DataSource cfg
spring.datasource.driver-class-name=.....
spring.datasource.url=.....
spring.datasource.username=.....
spring.datasource.password=.....
```

(e)

jars/starters :: spring-boot-starter-jdbc , oracle driver, lombok api

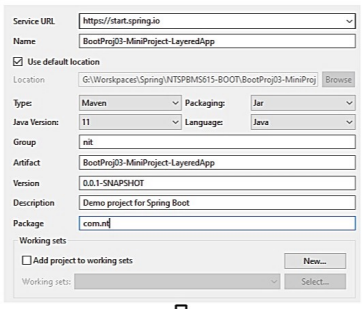
Nov 29 Spring Boot Layered Application

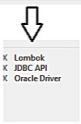
Procedure to develop spring boot Mini Project as Layered Application

==========

step1) create spring boot starter project adding the following starters
a) jdbc api b) lombok api c)oracle driver

File menu ----> new ----> project --->spring starter project --->





next ----> finish.

step2) add the following entries in application.properties file to provide inputs for Datasource cfg

application.properties

```
#jdbc proeprties for  DataSource AutoConfiguration
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
```

step3) create fllowing pkgs in src/main/java folder

step4) develop the Model class using lombok api annotation

Employee.java (model class)

```
package com.nt.model;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class Employee {
        private  Integer empno;
        private  String ename;
        private  String job;
        private Float  sal;

}
```

lombok api annotations generating commonly required code like setters ,getters , constructors, toString and etc..

step5) Develop DAO Interface and DAO Impl class

DAO Interface

```
package com.nt.dao;

import java.util.List;

public interface IEmployeeDAO {
    public  List<Employee>  getEmpsByDesgs(String desg1,String desg2,String desg3)throws
Exception;
}
```

//DAO Impl class

```
package com.nt.dao;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
import javax.sql.DataSource;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;
import com.nt.model.Employee;
@Repository("empDAO")
public class EmployeeDAOImpl implements IEmployeeDAO {
        private static final String GET_EMPS_BY_DESGS="SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE JOB IN(?,?,?) ORDER BY JOB";
        @Autowired
        private DataSource ds;

        @Override
        public List<Employee> getEmpsByDesgs(String desg1, String desg2, String desg3) throws Exception {
                List<Employee> listEmps=null;
          //get pooled jdbc con object
                try(Connection con=ds.getConnection();
                                PreparedStatement ps=con.prepareStatement(GET_EMPS_BY_DESGS)){  // try with resource
                        //set qury param values
                        ps.setString(1, desg1);
                        ps.setString(2, desg2);
                        ps.setString(3, desg3);

                        //execute the query
                        try(ResultSet rs=ps.executeQuery()){
                                // copy  ResultSet object records to List collection as Employee class objects
                                listEmps=new ArrayList();
                                  while(rs.next()) {
                                        Employee emp=new Employee();
                                        emp.setEmpno(rs.getInt(1));
                                        emp.setEname(rs.getString(2));
                                        emp.setJob(rs.getString(3));
                                        emp.setSal(rs.getFloat(4));
                                        listEmps.add(emp);
                                  }//while
                        }//try2
                }//try1
                catch(SQLException se) {
                        se.printStackTrace();
                        throw se; // exception rethrowing
                }
                catch(Exception e) {
                        e.printStackTrace();
                        throw e; //exception rethrowing
                }
        return listEmps;
        }//method
}//class
```

Nov 29.1 Spring Boot Layered Application

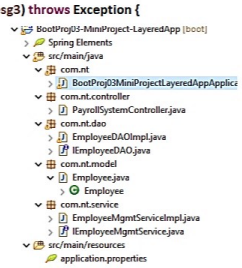
step6) develop Service Interface and service impl class

service Interface

```
package com.nt.service;
import java.util.List;
import com.nt.model.Employee;

public interface IEmployeeMgmtService {
   public  List<Employee> fetchEmpDetailsByDesgs(String desg1,String desg2,String desg3)throws Exception;
}
```

service Impl class

```
package com.nt.service;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.nt.dao.IEmployeeDAO;
import com.nt.model.Employee;
@Service("empService")
public class EmployeeMgmtServiceImpl implements IEmployeeMgmtService {
      @Autowired
      private  IEmployeeDAO dao;
      @Override
      public List<Employee> fetchEmpDetailsByDesgs(String desg1, String desg2, String desg3) throws Exception {
        //use  DAO
          List<Employee> listEmps=dao.getEmpsByDesgs(desg1, desg2, desg3);
          return listEmps;
      }
}
```



step7) Develope the controller class

```
//controller class
==================
@Controller("controller")
public class PayrollSystemController {
      @Autowired
      private  IEmployeeMgmtService service;

      public  List<Employee> showEmpsDetailsByDesgs(String desg1,String desg2,String desg3)throws Exception{
            //use serivce
            List<Employee>  listEmps=service.fetchEmpDetailsByDesgs(desg1, desg2, desg3);
            return  listEmps;
      }
  }
```

step8) Write Client app code in the main(-) of main class

```
package com.nt;
import java.io.Closeable;
import java.sql.SQLException;
import java.util.List;
import java.util.Scanner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;
import org.springframework.context.ConfigurableApplicationContext;
import com.nt.controller.PayrollSystemController;
import com.nt.model.Employee;
@SpringBootApplication
public class BootProj03MiniProjectLayeredAppApplication {

      public static void main(String[] args) {
            //get IOC container
            ApplicationContext ctx=SpringApplication.run(BootProj03MiniProjectLayeredAppApplication.class, args);
            //get Controller class object
            PayrollSystemController controller=ctx.getBean("controller",PayrollSystemController.class);
            //gather inputs  from enduser
            Scanner sc=new Scanner(System.in);
            System.out.println("enter desg1");
            String desg1=sc.next().toUpperCase();
            System.out.println("enter desg2");
            String desg2=sc.next().toUpperCase();
            System.out.println("enter desg3");
            String desg3=sc.next().toUpperCase();
            //invoke method
            try {
              List<Employee> listEmps=controller.showEmpsDetailsByDesgs(desg1, desg2, desg3);
                  /* for(Employee e:listEmps) {   // Enhanced for loop
                        System.out.println(e);
                   }//for
                  */
                  /*listEmps.forEach(emp->{     //Java8 forEach syntax1
                    System.out.println(emp);
                  });*/

                  /* listEmps.forEach(emp->          //Java8 forEach syntax2
                        System.out.println(emp)
                        );*/
      listEmps.forEach(System.out::println);  // java8   forEach with method reference concept
            }//try
            catch(SQLException se) {
                  se.printStackTrace();
                  System.out.println("Internal DB Problem  ");
            }
            catch(Exception e) {
                  e.printStackTrace();
                  System.out.println("Internal unknown Problem  ");
            }

            //close container
            ((ConfigurableApplicationContext) ctx).close();
      }//main
}//class
```

step9) run the application... Right click on Project --->Run as ---> JAva App/sprinb boot App

---

Can we cfg Controller, service , DAO calsses by just using @Component Annotation?

Ans) yes , but not recomanded

@Service = @Component + abilibity to perform Tx Mgmt
@Repository =@Component + abilibity to underlying Persistence technology exceptions to spring specific exceptions
@Controller = @Component + ability to take http requests..

can we interchange stereo type annotation utilization in layered App?

Ans) Yes , but not recomanded.

Nov 30 spring boot starter-parent

Understanding "spring-boot-starter-parent" of spring boot app

The maven's pom.xml of spring boot Project contains following details

a) importing spring-boot-starter-parent as parent project/starter to our spring boot Project
b) Current project details (like proj name(artifact id), group id ,version)
c) overirding of common property details (like java version ,encoding and etc..)
d) dependencies details (main jar file details)
e) Dependenccy management details (optional -- dependent jar file details)
f) maven plugins (which required to run the spring boot App)
and etc..

In Maven Project / plugin /dependent (jar file) is identified with 3 details
a) group id (project/plugin /jar file company name)
b) artifact id (project /plugin/ jar file name)
c) version (project/plugin/jar file version)

plugin : it is patch s/w that provides addtional fuctionalaties on the top of exisiting functionalaties..

sample pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <parent>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-parent</artifactId>
                <version>2.6.0</version>
                <relativePath/> <!-- lookup parent from repository -->
        </parent>
        <groupId>nit</groupId>
        <artifactId>BootProj02-AutoConfiguration</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <name>BootProj02-AutoConfiguration</name>
        <description>Demo project for Spring Boot</description>
        <properties>
                <java.version>11</java.version>
        </properties>
        <dependencies>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-jdbc</artifactId>
        <!--            <exclusions>
                                <exclusion>
                                        <groupId>com.zaxxer</groupId>
                                        <artifactId>HikariCP</artifactId>
                                </exclusion>
                        </exclusions> -->
                </dependency>

                <!-- https://mvnrepository.com/artifact/org.apache.tomcat/tomcat-jdbc -->
        <dependency>
          <groupId>org.apache.tomcat</groupId>
          <artifactId>tomcat-jdbc</artifactId>
        </dependency>

        <!-- https://mvnrepository.com/artifact/com.oracle.database.jdbc/ucp -->
        <dependency>
          <groupId>com.oracle.database.jdbc</groupId>
          <artifactId>ucp</artifactId>
        </dependency>

        <!-- https://mvnrepository.com/artifact/com.mchange/c3p0 -->
        <dependency>
          <groupId>com.mchange</groupId>
          <artifactId>c3p0</artifactId>
          <version>0.9.5.5</version>
        </dependency>


                <!-- https://mvnrepository.com/artifact/org.apache.commons/commons-dbcp2 -->
         <dependency>
          <groupId>org.apache.commons</groupId>
          <artifactId>commons-dbcp2</artifactId>
        </dependency>

                <dependency>
                        <groupId>com.oracle.database.jdbc</groupId>
                        <artifactId>ojdbc8</artifactId>
                        <scope>runtime</scope>
                </dependency>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-test</artifactId>
                        <scope>test</scope>
                </dependency>
        </dependencies>

        <dependency-management>
          ....
          ....
          ....
        </dependency-management>
        <build>
                <plugins>
                        <plugin>
                                <groupId>org.springframework.boot</groupId>
                                <artifactId>spring-boot-maven-plugin</artifactId>
                        </plugin>
                </plugins>
        </build>

</project>
```

parent spring boot Project inherited from Maven central repository

Current/our spring boot details

overriling some property values thare collected parent project

depdenciers (about main jar files)

about dependent jar files info

addional maven plugins

class plus

spring-boot-starter-parent

=> It is readily available spring boot starter /Project having lots of common things that required to build every spring boot App /Project..

=> It is placed as readily available project in maven central repository and we can inherit that to our spring Boot project /App using <parent> tag.
(Maven Inheritence)

```
<parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.6.0</version>
        <relativePath/> <!-- lookup parent from repository -->
</parent>
```

This spring -boot-starter -parent peforms the following operations on our spring boot

a) Inherits dependencies
and gives control on versions of various dependencies (jar files)
b) Inherites common Project properties to our spring boot project
c) Inherits common plugins to our spring boot Project

Nov 30.1 spring boot starter-parent

a) Inherits dependencies and gives control on versions of various dependencies (jar files)

=>Based on the spring-boot-start-parent version all the spring related main jar files and dependent jars version will be decided automatically .. This brings lots compitablity among the various starters and jar that we get automatically and that we add explicitly..

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

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-jdbc</artifactId>
    </dependency>
```

No need of adding any version here bcoz these jar files versions will be decided based on spring -boot-starter-parent version

parent starter version is 2.6.0 the we get spring jar files, stater jar files having 5.3.13 version

if want to change version of particular jar file/stater we can add <version> tag explcitly but not recomanded becoz it may lead to compitable isssues.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-jdbc</artifactId>
    <version>2.5.1</version>
</dependency>
```

(Not recomanded)

chanages only this starter version irrespective of parent starter version that we have specified

(This is for changing version of specific main starter that is spcified in <dependencies>

if u want to change one of depedent jar file or related jar file version of specific main jar file or starter then we place jar file information inside <dependencyManagement> tag

```
<dependencyManagement>
    <dependencies>
        <!-- https://mvnrepository.com/artifact/org.springframework/spring-aop -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-aop</artifactId>
            <version>5.3.8</version>
        </dependency>
    </dependencies>
</dependencyManagement>
```

(Not recomanded to use compability issues may come)

chanages the spring-aop jar file version to 5.3.8 irrepsective of its main jar /starter version and parent stater version

URL to see pom.xml file "spring-boot-starrter-parent" project from maven central repository

https://repo1.maven.org/maven2/org/springframework/boot/spring-boot-starter-parent/2.6.1/spring-boot-starter-parent-2.6.1.pom

b) Inherites common Project properties to our spring boot project

properties in spring-boot-starter-parent project

```
<properties>
  <java.version>1.8</java.version>
  <resource.delimiter>@</resource.delimiter>
  <maven.compiler.source>${java.version}</maven.compiler.source>
  <maven.compiler.target>${java.version}</maven.compiler.target>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
</properties>
```

These properties will be inherited our spring boot project automatically.. if want override some of these propeties we can place <properties> tags explicitly in the pom.xml file of our spring boot project.

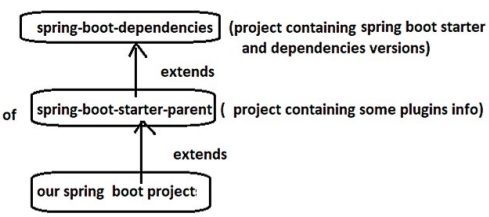
our project's pom.xml file

```
<properties>
        <java.version>15</java.version>
    </properties>
```

perform maven update (right click on project ---> maven -->update --> ....)

c) Inherits common plugins to our spring boot Project

=> To build spring Application , to run spring boot app , to generate report on spring boot app , to pack spring app into jar file or war file , To run test cases(junit) of spring boot App and etc ... we neend lots maven plugings .. But do not add those plugins to our pom.xml becoz they will be inherited from spring-boot-stater-parent and its parent spring-boot-dependencies



Dec 01 Spring vs spring boot and properties file

What is the difference between spring and Spring boot?

spring boot =spring ++

spring boot = spring + auto configuration + Embedded servers + Embedded Db s/ws + plugins -- spring bean cfgs (xml cfgs)

boilterplate code =repeatative code (The code that repeates in multiple parts of the project either with no changes or with minor changes is called boilerplate code)

Spring	spring boot
a) provides abstraction on JSE,JEE technologies	a) provides abstaction on spring framework itself
b) Avoids boilerplate code with respect to JSE, JEE technologies based programming	b) Avoids boilter place that is happening with spring framework programming
c) No Support of Autoconfiguration (Here main feature is Dependency Management)	c) Supports AutoConfiguration as the main feature
d) Supports xml files based cfgs	d) avoids or minimizes xml files based cfgs
e) we can develop spring app in 3 ways a)using xml driven cfgs b) xml + annotation driven cfgs c) 100% code driven cfgs	e) spring boot allows 100% code driven cfgs with Autoconfiguration support
f) Does not given embeeded servers ...So we need to cfg exteranl servers to run the App	f) Gives Tomat, jetty,undertow and etc.. as EmbeddedServer
g) Does not give InMemory or Embedded Db s/ws	g) Gives H2, HSQL and etc.. as the InMemory Db s/ws based on the starters we add
h) In most of cases we create IOC container explicitly here	h) We do not IOC container explicitly ... The springApplicaiton.run(-) takes care of this process
i) Suitable for new projects , migration projects (good for both green field (new), brown field(enhancement) projets	i) Not suitable for migrating spring project into spring boot project but suitable for new projects (scratch level devleopment) (green field projects)
j) In spring projects we need to main jar files and related jar files explicitly but the dependent jar files will be given maven/gradle build tools	j) The spring boot starters takes care of getting all main jar files, dependent jar files and relevant jar file dyamically the to project.
k) does not give default propperties file	k) gives application.properties file as default properties which can be used to provide inputs for auto configration.
l) does not give any non-functional features like health metrics , logging and etc..	l) spring boot actuators provide multiple non-funcational features which can used to monitor various stats of the Project.
m) we can not spring web application as standalone Apps (jar file)	m) we can run spring boot web application standalone App(jar) and also web application (war)
n) Spring not so suitable to develop micro services architecture based applications..	n) Spring boot Exclusiy designed to support micro service archicture based Appliations
o) every spring project is direct project to develop	Every service will be developed as seperate project and all those service will be integrated as need. o) spring boot project is child for spring -boot-starter-parent project available in central repositories
(p) takes bit extra time in development , So productivity is not so good	(p) reduces the application devleopment becoz autoconfiguration.. So we get good amount productivity
q) WE need to use multiple detailed annotations for configurations	q) lots of new annotations are given by combining related spring annotations to simplify the process (eg: @SpringBootApplication)
r) Application is bit light weight (No Autoconfiguration , So unneccessary objects will not created)	r) Applications bit heavu weight becoz the AutoConfiguation created both needed and unwanted obejcts.

Working properties file in spring boot App

properties file

The text file that maintains entries as the key=value pairs

=> Spring Boot App by default recognized application.propeties file of src/main/resoruce folder while executing SpringApplication.run(-,-) method.

=> if want to cfg user-defined properites file .. then it must be cfg explicitly by using @PropertySource annotation

=> if want to change the name of application.properties then it must be cfg explicitly by using @PropertySource annotation

pllace this annotation either on main class or on spring bean class.

=> In application.properties we can place only pre-defined keys and values or only user-defined keys and vlaues or both

https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html

=>In application.properties we can add the following category pre-defined keys having values accordingly.

1. Core Properties
2. Cache Properties
3. Mail Properties
4. JSON Properties
5. Data Properties
6. Transaction Properties
7. Data Migration Properties
8. Integration Properties
9. Web Properties
10. Templating Properties
11. Server Properties
12. Security Properties
13. RSocket Properties
14. Actuator Properties
15. Devtools Properties
16. Testing Properties

What is difference b/w AutoWiring and AutoConfiuration?

Ans) Autowiring makes IOC container to detect dependent spring bean dynamically either based on type or based on name and injects that object to target spring bean class obj

@Autowired performs Auotowiring based dependency Injection

AutoConfiguration makes the IOC container to cfg certains pre-defined classes as spring beans based on the starters(jar files) that are added to the Project .. AutoConfiguraion also gives embedded servers, Db s/ws , plugins , configurations and etc..

@EnableAutoconfiguration of @SpringBootApplication performs AutoConfiguration activities during the startup of spring boot App.

Benefits Of spring boot Over Spring:-

1) Dependency Injection
2) minimum configuration (AutoConfiguration)
3) Embedded server for testing
4) Bean auto scan (@ComponentScan will be hidden by @SpringBootApplication)
5) health metrics (actuators)

=>we can provide inputs to spring boot Application

a)from properties files/yml
b) from form pages (in web applications)
c) from Db s/ws
d) from enduser throuhg keyboard (Scanner, cmd line args ,system props and etc..) and etc..

Dec 01.1 Spring vs spring boot and properties file

=> To read values from properties file and to Inject to spring bean properties
we can use the following two Annotations
a) @Value (supplied by spring and can be used in spring boot)
(Given for single value injection)
b) @ConfigurationProperties (supplied by spring boot and can not be used in spring)
(Given for bulk injection)

@Value
=======

=>It is given to inject direct values , values gathered from properties file(s)/ system properties/
env variables to spring bean properties .. Can inject one value at a time
=> Also useful to evaluate expression using SPEL (spring Expression Language) and to inject the
results to spring bean properties.
(both simple and object type variables of spring bean class)

note:: using SPEL we can perform
arithemetic and logical operations
before injectiing value to spring
bean property.. Inspired by JSP EL

@Value injecting hard coded value to spring bean property
==========

```
@Component
public class Hospital{
    @Value("Apollo")
   private String name;
   @Value("30")
   private int age;

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

@Value Injecting value to spring bean property by collecting the value from the properties file
==========

application.properties (src/main/resoruces)

```
# hospital info
hsptl.name=Apollo
hsptl.age=30
```

Injects Apollo

```
@Component
public class Hospital{
   @Value("${hsptl.name}")
   private String name;
   @Value("${hsptl.age}")
   private int age;
```

Injects 30

if the name or location or
extension of application.properties
is changed it must be cfg as user-defined
properties file using @PropertySource

${<key>}--represents
place holder having key whose
value will be collected from
properties file by submiting the give
key and the gathered value will be
injected spring bean property.

What is the dfference b/w @Value and @Autowired?

Ans) @Value injects only simple values to bean properties by collecting from different places
where @Autowired injects other spring bean class obj as dependent class object to a HAS-A property
of target spring bean class object .. In this proccess it dynamicaly searches and gets dependent
spring bean class obj either by Type or by name.

tentive dates:
5.45pm :: adv.java (JDBC) ---> from dec13th (monday)
4pm :: hibernate(***) /Design Pattern -->from dec16th (thrus day)

@Value
=======

=>It is given to inject direct values , values gathered from properties file(s)/ system properties/
env variables to spring bean properties .. Can inject one value at a time
=> Also useful to evaluate expression using SPEL (spring Expression Language) and to inject the
results to spring bean properties.
(both simple and object type variables of spring bean class)

note:: using SPEL we can perform
arithemetic and logical operations
before injectiing value to spring
bean property.. Inspired by JSP EL

@Value injecting hard coded value to spring bean property
==========

```
@Component
public class Hospital{
    @Value("Apollo")
   private String name;
   @Value("30")
   private int age;

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

@Value Injecting value to spring bean property by collecting the value from the properties file
==========

application.properties (src/main/resoruces)

```
# hospital info
hsptl.name=Apollo
hsptl.age=30
```

Injects Apollo

```
@Component
public class Hospital{
   @Value("${hsptl.name}")
   private String name;
   @Value("${hsptl.age}")
   private int age;
```

Injects 30

if the name or location or
extension of application.properties
is changed it must be cfg as user-defined
properties file using @PropertySource

${<key>}--represents
place holder having key whose
value will be collected from
properties file by submiting the give
key and the gathered value will be
injected spring bean property.

Dec 02 @Value and ConfigurationProperties

What is the dfference b/w @Value and @Autowired?

Ans) @Value injects only simple values to bean properties by collecting from different places where @Autowired injects other spring bean class obj as dependent class object to a HAS-A property of target spring bean class object .. In this proccess it dynamicaly searches and gets dependent spring bean class obj either by Type or by name.

tentive dates:
5.45pm :: adv.java (JDBC) --> from dec13th (monday)
4pm :: hibernate(***) /Design Pattern -->from dec16th (thrus day)

@Value can be used to inject env.. variable value to spring bean proeprty

```
@Component
public class Hospital{
  @Value("${Path}")                    ------> environment variable name (fixed)
  prvIate String pathData;

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

@Value can be used to inject system property value to spring bean proeprty

```
@Component
public class Hospital{
    @Value(${os.name}")  ------> The fixed system property name
    private osName;

}
```

System.getProperties() method gives all the system properties and their values and key and value pairs..

=> To add new starter to the existing spring boot Project we can right click on project --> starters--> select starter -->next --> select pom.xml --> finish.

@Value can can be used place SPEL code where the given expression having Logical and Arithmetic opeations can be evaluated and the results can be injected spring bean properties

place holder syntax :: ${<key>}
SPEL syntax :: #{ <expr> }

SPEL is very much specific to spring and spring boot env..

the SPEL code placed in @Value makes the expession to execute and the generated result will be injected to spring bean property and using SPEL we can do injection of object type property like @Autowired Annotation

Ex:- @Value("#{ bean id }") //using SPEL Perfoming injection to object type property
private ClassName refVarName;

note: @Value performs Field Injection on simple data type properties.

example App

application.properties

```
# user-defined key=value pairs
hsptl.name=Apollo
hsptl.age=30

lab.bpPrice=5000
lab.rtpcrPrice=1300
lab.echo2DPrice=1300
```

LabTestsInfo.java

```
package com.nt.beans;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

import lombok.Data;

@Component("lInfo")
@Data
public class LabTestsInfo {

    @Value("${lab.bpPrice}")
    private float bloodProfilePrice;

    @Value("${lab.rtpcrPrice}")
    private float rtpcrPrice;

    @Value("${lab.echo2DPrice}")
    private float echo2DPrice;

}
```

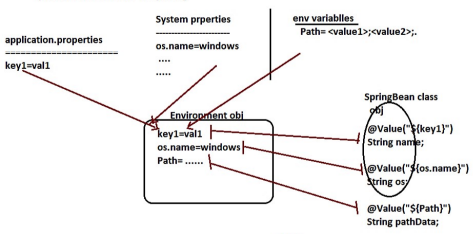
Hospital.java

```
package com.nt.beans;

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

import lombok.Data;

@Component("hsptl")
@Data
public class Hospital {

    @Value("KIMS") //hard coding
    private String name;
    @Value("30") //hard coding
    private int rank;

    @Value("${hsptl.name}") //collecting from properties file
 private String name1;

    @Value("${hsptl.age}") //collecting from properties file
     private int age;


    @Value("${Path}") //injecting from env.. variable
    private String pathData;

    @Value("${os.name}") // Injecting system property value
    private String os;

    //@Autowired //Autowing
    @Value("#{lInfo}") //using SPEL Perfoming injection to object type property
    private LabTestsInfo info;

    @Value("#{lInfo.bloodProfilePrice+lInfo.rtpcrPrice}") //SPEL based arithmetic operation and injection
    private float labTestsBillAmt;

}
```

Main. class

```
package com.nt;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;
import org.springframework.context.ConfigurableApplicationContext;

import com.nt.beans.Hospital;

@SpringBootApplication
public class BootProj04ValueAnnotationApplication {

public static void main(String[] args) {
// get IOC container
ApplicationContext ctx=SpringApplication.run(BootProj04ValueAnnotationApplication.class, args);

//get SpringBean class object
 Hospital hospital=ctx.getBean("hsptl",Hospital.class);
 //invoke the method
 System.out.println(" spring bean class obj data::"+hospital);

 //close container
 ((ConfigurableApplicationContext) ctx).close();
 System.out.println("=============");
// System.out.println("system properties are ::"+System.getProperties());
    }
}
```

The spring boot App /spring App collects values from properites/yml file(s), system properties , env . variables to store Into the IOC container managed Environment object. From there they will be passed to different properties of different various spring beans based place holders that we are placing



name

if the key of property file is matching with system property or env variable name can u tell me which value will be injected as the final value?

Ans) The value kep in system propety or env variable will be taken as the final value..

=>if application.properties and user-defined propertie file are having same keys with different values then the application.properties value will be taken as the final value.

application.properties

```
# user-defined key=value pairs
hsptl.name=Apollo
hsptl.age=30

lab.bpPrice=5000
lab.rtpcrPrice=1300
lab.echo2DPrice=1300

os.name=win11
Path=hello1
```

input.properties (src/main/resources folder)

```
os.name=Linux

hsptl.name=Global hospital
```

```
@Component("hsptl")
@Data
@PropertySource("classpath:input.properties")
public class Hospital {

    @Value("KIMS") //hard coding
    private String name;
    @Value("30") //hard coding
    private int rank;

    @Value("${hsptl.name}") //collecting from properties file
 private String name1;   takes Apollo from application.properties

    @Value("${hsptl.age}") //collecting from properties file
     private int age;


    @Value("${Path}") //injecting from env.. variable
    private String pathData;

    @Value("${os.name}") // Injecting system property value
    private String os;   takes from system properties (windows10)

    //@Autowired //Autowing
    @Value("#{lInfo}") //using SPEL Perfoming injection to object type property
    private LabTestsInfo info; // HAS -A property

    @Value("#{lInfo.bloodProfilePrice+lInfo.rtpcrPrice}") //SPEL based arithmetic operation and injection
    private float labTestsBillAmt;

}
```

Order of searching for the values based on the key of place holder

a) user-defined properties file(s)
b) application.properties /yml
c) system properties
d) env.. variables

note:: IOC container internally maintains lots ready made objects like
a) Its internal cache
b) Environment object
c) ApplicationContext, BeanFactory objects
d) Lots of bean ids
and etc..

Dec 03 @Value and @ConfigurationProperties

=>we can inject values to spring bean properties from properties file
usisng either @Value or @ConfigurationProperties annotations

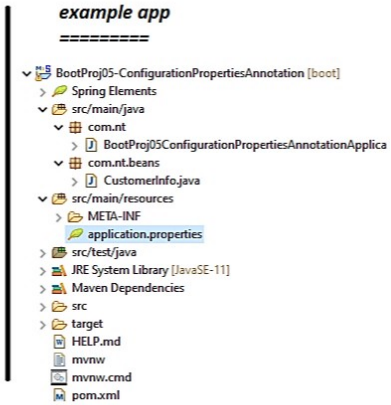
@Value for single vlaue injection given by spring
@ConfigurationProperties for bulk Injectio given by spring boot

Working @ConfigurationProperties
==========
=>Given by spring boot , so can not be used spring apps
=>Useful to Injection properties file/yml file values to spring bean properties
in bulk manner .. For this
a) The keys in properties/yml file should have common prefix
b) The last words in keys (after common prefix) must match with spring bean property names
c) place @ConfigurrationProperties on the top spring Bean class specifying that common prefix
d) Spring bean class must have setter methods becoz it performs bulk injection as setter Injection

application.properties

```
#customer info
cust.info.name=raja
cust.info.age=40
cust.info.addrs=hyd
cust.info.billAmt=5000
```

example app
=========



CustomerInfo.java

```
package com.nt.beans;

import lombok.Setter;

@Component("custInfo")
@ConfigurationProperties(prefix = "cust.info")
@Setter
public class CustomerInfo {
	//spring bean properties
	private String  name;
	private int  age;
	private String addrs;
	private  float billAmt;

@Override
public String toString() {
	return "CustomerInfo [name=" + name + ", age=" + age + ", addrs=" + addrs + ", billAmt=" + billAmt + "]";
}

}
```

@While working with @ConfigurationProperties it is recomanded to added the following dependency in pom.xml file to generate metdata about properties placed in properies file or yml file (we can get the source code of spring bean for the keys placed in properites file using ctrl+ click)

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

main class
==========

```
ackage com.nt;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;
import org.springframework.context.ConfigurableApplicationContext;
import com.nt.beans.CustomerInfo;
@SpringBootApplication
public class BootProj05ConfigurationPropertiesAnnotationApplication {

   public static void main(String[] args) {
        //get IOC Container
       ApplIcationContext ctx=
       SpringApplication.run(BootProj05ConfigurationPropertiesAnnotationApplication.class, args);

       //Get Spring class object
       CustomerInfo info=ctx.getBean("custInfo",CustomerInfo.class);
       //invoke method b.method
       System.out.println("CustomerInfo object data::"+info);

       //close container
       ((ConfigurableApplicationContext) ctx).close();
    }

    }
```

Dec 03.1 @Value and @ConfigurationProperties

What is the difference between @Value and @ ConfigurationProperties ?

@Value	@ConfigurationProperties
a) given by spring ,so can be used in both spring and spring boot apps	a)Given by spring boot ,So can be used only in spring boot Apps ..not in spring apps
b) can inject only value at a time	b) can be used for bulk injection
c) Needs to use place holders to inject the values to spring bean properties	c) No need of working with place holders
d) performs Filed level Injection so setter methods are not required in spring beans	d) performs setter Injection , So settet methods are required in spring beans
e) key in properties file and spring bean property names need not to match	e) last word in the keys (after prefix) and spring bean property names need to match
f) Generally applied at fileld level	f) generally applied at class level , method level (for @Bean methods)
g) Can work with SPEL during the Injection porcesss	g) Can not be used
h) Can be used to Inject system property values and env.. variable values	h) can be used to Inject only certain common prefix system property values (Not all at time) ... and can be used to inject env.. variable values (@ConfigurationPropertis should be taken with out specifying prefix)
i) In single spring bean we can inject values to spring bean properties by collection from properties file, system properites and env. variables	i) not possible (becoz @ConfigurationProperties allows to specify "single prefix" at a value and not repetable annotation i.e can not be repeated at same level for multiple times)
j) if the key in properties file and key in place holder of @Value is not matched then we get Illegal argument excpetion	j) if the last word in the key of properties file (after common prefix) and the spring bean property name is not matched then value Injection propecess does not take place on that specific spring bean property

=>One spring boot Project can have only one application.properties/yml in src/main/resoruce folders .. if any addtional application.properties files are taken in other location they must cfg explicitly.. using @PropertySource

=> In one properties file we can place different sets of commin prefixes to perform multiple bulk Injection on spring beans using @ConfigurationProperties

Employee.java
```
package com.nt.beans;

import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;

import lombok.Data;

@Component("emp")
@ConfigurationProperties(prefix = "emp.details")
@Data
public class Employee {
        //spring bean properties
        private String  name;
        private int  age;

}
```

CustomerInfo.java
```
package com.nt.beans;

import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;

import lombok.Data;

@Component("custInfo")
@ConfigurationProperties(prefix = "cust.info")
@Data
public class CustomerInfo {
        //spring bean properties
        private String  name1;
        private int  age;
        private String addrs;
        private  float billAmt;

}
```

application.properties
```
#customer info
cust.info.name1=raja
cust.info.age=40
cust.info.addrs=hyd
cust.info.billAmt=5000


#employee info
emp.details.name=ramesh
emp.details.age=30
```

note:: While working @ConfigurationProperties also , the Environment object will be involved to collecting values from different places to spring bean properties

note:: Lots of spring boot supplied prefined class that are involved in Autoconfiguration process directly or indirectly uses @ConfiguationProperties for bulk injection to inject the values spring bean properties by colelcting from fixed keys of application.properites.

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

        private ClassLoader classLoader;
        private boolean generateUniqueName = true;
        private String name;
           ....
            ...
}
```

@) if i try to inject two different values to same spring bean property usisng @Value and @ConfigurationProperties can u tell me which value will be injected as the final value ?

Ans) @ConfigurationProperties based Injection value will be taken as final value becoz it performs setter Injection by calling method which executes after filed Injection done by @Value Annotation.

order
=====
=> constructor (Construction Injection)
=>Filed Injection (using reflection api internally) (default of @Value Injection)
=> setter Injection (Default for @ConfigurationProperties bsaed Injection)
=> orbitary method Injection

```
@Component("emp")
@ConfigurationProperties(prefix = "emp.details")
@Data
public class Employee {
        //spring bean properties
        @Value("${emp.info.name}")
        private String  name;
        private int  age;

        Employee(){
                System.out.println("Employee::0-param construtor ::"+name);
        }

        public void setName(String name) {
                System.out.println(this.name);
                this.name=name;
                System.out.println("Employee.SetName(-) ::"+name);
        }

}
```

application.propereties
```
#employee info
emp.details.name=ramesh
emp.details.age=30

emp.info.name=suresh
```

Dec 04 Working with different types of properties with @ConfigurationProperties

Injecting values to array, Collection, HAS-A type spring bean properties using @ConfugurationProperties

The class /java bean class/spring bean class can have the following type properties

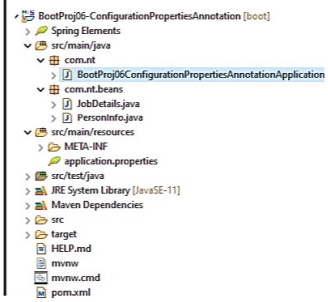
a) simple type properties
b) array type properties
c) collection type properties (List/Set/Map)
d) HAS-A property (other class type -composition)
type

note:: The following special characters allowed in the keys of properties file "." ,"-" ,"[" ,"]"

PersonInfo.java

```
package com.nt.beans;

import java.util.Arrays;
import java.util.List;
import java.util.Map;
import java.util.Set;

import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;

import lombok.Data;

@Data
@Component("pInfo")
@ConfigurationProperties(prefix = "per.info")
public class PersonInfo {
        //simple/wrapper type
        private Integer pid;
        private String  pname;
        //array type
        private  String[] favColors;
        //List-Collection type
        private  List<String>  studies;
        //Set- Collection type
        private  Set<Long>  phoneNumbers;
        //Map-Collection type
        private  Map<String,Object> idDetails;

        //HAS-A type  (composition)
        private JobDetails  jobInfo;

    @Override
    public String toString() {
            System.out.println("PersonInfo.toString()");
            return "PersonInfo [pid=" + pid + ", pname=" + pname + ", favColors=" + Arrays.toString(favColors)
                            + ", studies=" + studies + ", phoneNumbers=" + phoneNumbers + ", idDetails=" + idDetails + ", jobInfo="
                            + jobInfo + "]";
    }
 }
```

example App



application.properties

```
# personal  Info

#simple  type :: <prefix>.<propname>=<value>
per.info.pid=1001
per.info.pname=raja

#array type :: <prefix>.<propname>[index]=<value>
per.info.favColors[0]=red
per.info.favColors[1]=green
per.info.fav-colors[2]=blue
# invalid becoz the index must be given in sequence
#per.info.favColors[4]=orange

#List type :: <prefix>.<propname>[index]=<value>
per.info.studies[0]=10+2-MPC
per.info.studies[1]=B.Tech-CSE
per.info.studies[2]=M.Tech-IT

#Set type :: <prefix>.<propname>[index]=<value>
per.info.phoneNumbers[0]=9999999999
per.info.phoneNumbers[1]=8888888888
per.info.phoneNumbers[2]=7777777777
per.info.phone-numbers[3]=7777777777

#Map type :: <prefix>.<propname>.<key>=<value>
per.info.idDetails.aadharNo=5543577778
per.info.idDetails.panNo=AJXPRH6707H
per.info.id-details.voterId=5457788885

#HAS-A type(Composition type) ::<prefix>.<propname>.<subpropname>=<value>
per.info.jobInfo.desg=CLERK
per.info.jobInfo.company=HCL
per.info.job-info.deptNo=6678
```

if index in the array type or collection type property by missing its index then we get

```
Binding to target [Bindable@2d66530f type = java.lang.String[],
value = 'provided', annotations = array<Annotation>[[empty]]] failed:

Property: per.info.favcolors[4]
Value: orange
Origin: class path resource [application.properties] - 13:23
Reason: The elements [per.info.favcolors[4]] were left unbound.
```

Client App /main class

```
package com.nt;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;
import org.springframework.context.ConfigurableApplicationContext;

import com.nt.beans.PersonInfo;


@SpringBootApplication
public class BootProj06ConfigurationPropertiesAnnotationApplication {

 public static void main(String[] args) {
        //get IOC Container
        ApplicationContext ctx=  SpringApplication.run(BootProj06ConfigurationPropertiesAnnotationApplication.class, args);
        //get  Spring Bean class object
        PersonInfo person=ctx.getBean("pInfo",PersonInfo.class);
        //invoke the methods
        System.out.println("PersonInfo Object details::"+person);

        //close container
        ((ConfigurableApplicationContext) ctx).close();
  }
 }
```

JobDetai;ls.java

```
package com.nt.beans;

import lombok.Data;

@Data
public class JobDetails {
        private  String desg;
        private  String company;
        private  Integer deptNo;
}
```

Dec 06 YML fles

YML /YAML Files

==========

=>Given alternate to properties file but which will be internally converted into properties file

=> YML/YAML : Yet Another Markup Language
YML/YAML : Yamaling Language
YML/YAML : Yain't markup Language
YML/YAML : YAML ain't markup language (official)

YAML is a human-friendly data serialization language for all programming languages.
(www.yaml.org)

=>The Biggest Limitation of properties file repeataion of nodes/levels /prefixes in multiple keys for simple or collection or HAS-A properties while perform Bulk Injection using @CofigurationProperties

=> yml/yaml file can have extension of .yml or .yaml
=> The spring Boot App recognize application.yml or application.yaml file automatically as the applications startup activties.

application.properties
---
per.info.name=raja
per.info.addrs=hyd
per.info.desg=Programmer

node/level/prefix is repeating in multiple keys

application.yml

```
per:
  info:
    name: raja
    addrs: hyd
    desg: Programmer
```

(node/level/prefix is not repeating)

=> While procecessing .yml file .. it will converted in to properties file intenally ..For this conversion and process of yml file/yaml file we need seprate api called snackyaml api (jar snakeyml-<ver>.jar)

(part of every spring boot project given by the basic spring-boot-starter)

==========

While converting properties file to yml file or while yml file we need to keep the following points in mind

==========

=>same level/node will not repeated i.e each node/level will be written only for one time
=>Replace "." symbol of node or level seperation with ":" symbol and place next node/child node or level in the next line having atlease single space (recomanded to take 3 spaces for better redability)
=>Replace "=" symbol with ":" symbol and give minimum single space before placing value

=> Comments can be placed using "#" symbol
=> While placing array/List/set element values use "-" symbols before the values having minimum single space
=> Take Map collection elements keys and HAS-A property related sub properties as the sub nodes/levels and etc..

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

```
# personal Info

#simple type :: <prefix>.<propname>=<value>
per.info.pid=1001
per.info.pname=raja

#array type :: <prefix>.<propname>[index]=<value>
per.info.favColors[0]=red
per.info.favColors[1]=green
per.info.fav-colors[2]=blue
# invalid becoz the index must be given in sequence
#per.info.favColors[4]=orange


#List type :: <prefix>.<propname>[index]=<value>
per.info.studies[0]=10+2-MPC
per.info.studies[1]=B.Tech-CSE
per.info.studies[2]=M.Tech-IT

#Set type :: <prefix>.<propname>[index]=<value>
per.info.phoneNumbers[0]=9999999999
per.info.phoneNumbers[1]=8888888888
per.info.phoneNumbers[2]=7777777777
per.info.phone-numbers[3]=7777777777

#Map type :: <prefix>.<propname>.<key>=<value>
per.info.idDetails.aadharNo=5543577778
per.info.idDetails.panNo=AJXPRH6707H
per.info.id-details.voterId=5457788885

#HAS-A type(Composition type) ::<prefix>.<propname>.<subpropname>=<value>
per.info.jobInfo.desg=CLERK
per.info.jobInfo.company=HCL
per.info.job-info.deptNo=6678
```

application.yml
==========

```
per:
  info:
    #simple properties
    pid: 1001
    pname: raja
     # array propery
    favColors:
      - red
      - green
      - blue
     # List collection
    studies:
        - 10+2-MPC
        - B.Tech-CSE
        - M.Tech-IT
       #Set collection
    phoneNumbers:
          - 9999999
          - 8888888
          - 7777777
         #  Map collection
    id-details:
          aadhar: 545435435
          panNo: AJXPR0886H
          voterId: 45677655
         #HAS- A property
    job-info:
        desg: Programmer
        company : HCL Technologies
        deptNo: 56778
```

In yml file
==========

```
per:
  info:
    favColors:
      - blue
      - red
      - green
```

can be written as

```
per:
  info:
    favColors: [bule,red,green]
```

In line Formatting

This can be applied on array , List ,Set type properties

In application.properties

```
per.info.favColors[0]=red
per.info.favColors[1]=green
per.info.fav-colors[2]=blue
```

is equal to

```
per.info.favColors=red, green,blue
```

In line Formatting

same thing is applicable for List ,Set Properties

---

note:: we can use STS plugin supplied wizard to properties file format content yml format content
procedure:: right click properites file -->convert .properties to .yaml file

Dec 06.1 YML fles

note:: WE can cfg user-defined properties file using @PropertySource Annotation.. we can also cfg user-defined yml file using @PropertySource Annotation but we need to place One PropertySourceFactory to process that yml document.

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

```
@Configuration
@ConfigurationProperties(prefix = "yaml")
@PropertySource(value = "classpath:foo.yml", factory = YamlPropertySourceFactory.class)
public class TestInfo {

   private String name;

   private List<String> aliases;

   // standard getter and setters
}
```

sample PropertySoruceFactory class
==================================

```
public class YamlPropertySourceFactory implements PropertySourceFactory {

  @Override
  public PropertySource<?> createPropertySource(String name, EncodedResource encodedResource)
    throws IOException {
     YamlPropertiesFactoryBean factory = new YamlPropertiesFactoryBean();
     factory.setResources(encodedResource.getResource());

     Properties properties = factory.getObject();

     return new PropertiesPropertySource(encodedResource.getResource().getFilename(), properties);
  }
}
```

This class contains logic to convert yml file content into java.util.Properties class object and gives spring boot for further processsing.

---

if we try to execute spring boot with application.yml/yaml file by excluding snakeyaml-<ver>.jar file then we get the fllowing exception

java.lang.IllegalStateException: Attempted to load Config resource 'class path resource [application.yml]' via location 'optional:classpath:/' but snakeyaml was not found on the classpath

Whe[n] should we use properties file and when should use yml file in our spring boot App?

Ans) if the keys are more and repeative while providing information then prefer using yml/yaml file becoz yml avoids duplication of nodes/levels/prefixes.. if keys are less and not so repeatative then prefer using .properties file

to

=>once yml file info is coverted in Properties collection(map collection) then will be stored environment object automatically.
(form here they will injected to spring bean properites based @Value or @ConfigurationProperites annotation)

note:: Since the spring boot supplied pre-defined keys are having more repeated nodes/levels it is recomanded to use yml files alot in spring boot Applications..

application.propeties
=====================

```
emp.info.name=raja
emp.info.age=30
emp.details.addrs=hyd
emp.details.salary=50000
spring.datasource.driver-class-name=oracle.jdbc.driver.OraclerDriver
```

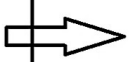

appplcation.yml

```
emp:
  info:
    name: raja
    age: 30
  details:
    addrs: hyd
    salary: 50000
spring:
  datasource:
    driver-class-name: oracle.jdbc.driver.OracleDriver
```

---

application.properties

```
emp.info.name=raja
cust.info.name=ramesh
stud.info.name=rajesh
server.port= 4040
emp.info.addrs=hyd
server.status=running
```



application.yml

```
emp:
  info:
    name: raja
    addrs:  hyd
cust:
  info:
    name: ramesh
stud:
  info:
    name: rajesh
server:
   port: 4040
   status : running
```

---

application.yml
===============

```
emp:
  info:
    name: raja
    addrs: hyd
    marks:
       - 20
       -30
       -40
  details:
     desg: CLERK
     deptNo: 3200
```

application.properties

```
emp.info.name=raja
emp.info.adds=hyd
emp.info.marks[0]=20
emp.info.marks[1]=30
emp.info.marks[2]=40
emp.details.desg=CLERK
emp.details.deptno=3200
```

emp.info.marks=20,30,40

---

application.properties

```
spring.datasource.driver-class-name=oracle.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe
spring.datasource.username=raja
spring.datasource.password=rani
server.port=5050
mail.trasport.protocol=SMTP
```

appplication.yml

```
spring:
  datasource:
    driver-class-name:  oracle.jdbc.driver.OracleDriver
    url: jdbc:oracle:thin:@localhost:1521:xe
    username: raja
    password: rani
server:
   port: 5050
mail:
   transport:
      portocol:  smtp
```

note:: STS is not providing any plugin to convert yml file into properties .. but online tools are avaliable

For yml to properties file and vice-versa conversion use this online convertor tool http://mageddo.com/tools/yaml-converter

Dec 07 YML fles and Profiles

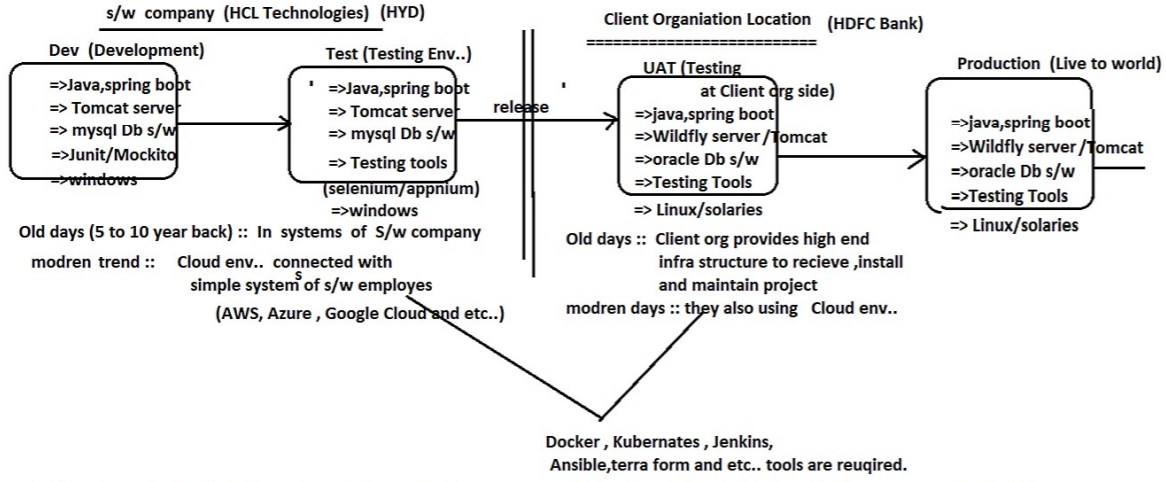
What is the difference b/w yml file and properties file?

properties file	yml file
(a) Here the nodes/levels in the keys will be repeated	(a) will not be repeated
(b) Can be used in both spring Apps and Spring boot Apps with out any Libraries in the Build path	(b) Can be used in spring boot Apps directly but can not be used in spring Apps directly (addtional libraries and api are required in spring apps)
(c) There is no specification to follow while developing properites file .. It just writing key=value pairs	(c) There is specification for yml file .. we can see that in yaml.org website (latest version is 1.2.2)
(d) Can be used only in java env..	(d) so useful in java, ruby ,groovy, python and etc. env..
(e) The data is placed as key=values pairs	(e) The data is placed having nodes hierarchal manner
user-defined (f)To cfg properties file we need to use @PropertySource specifying filename, location and no need of specifying factory class (In Sprng, spring boot Apps)	user-defined (f)To cfg yml file we need to use @PropertySource specifying filename, location and factory class that implements PropertySourceFactory(I) (In spring boot Apps)
(g) The Allowed characters are . , = , [,], - and etc..	g) The allowed characters are : , - , [,] and etc..
(h) Intenrally the properties file content will be read directly	(h) yml content will converted into propeties file sytle key=values before reading them into application
one j) In Properties file we can not place multiple profiles (In one file we can not keep multiple sections wize data)	j) In one yml file we can place multiple profiles sperateed "---" symbols (In one file we can keep multiple sections wize data)
k) needd more memory becoz the nodes/levels in keys will be repeated	k) needs less memory becoz nodes/levlels will not be repeated
(I) Propeties file content is no way related to JSON format JSON :: Java Script Object Notation	(I) it is super set of JSON file
(m) Working with simple Java apis or other language apis (other than spring) every value for kept in key will be retrieved as String value and should be converted to other later types manually [Spring also internally read every value of every key as String values but interally uses built-in PropertyEditors /convertors to traslate them as required for spring bean properties]	(m) Can be retrived different types of from yml file into application with out any explicit conversion process i.e yml allows to place init,float, String, char and etc.. type values directly in the file itself.
(n) use properties file , if the data contains either less keys or less no.of repeated keys	(n) use yml file , if the data contains more keys or more repeated no.of repeated keys

==========
Profiles in spring boot
==========

=> profile is also as called environment
=> The setup required to execute the Application or Project is called environment or Profile
This includes .class files, properites/yml file, libraries(jar files) , servers (like Tomcat), DB s/w (like oracle) and etc..
=> From development to release /maintainace the Project code/App code has to executed in different environmments/profiles.. like Dev , Test , UAT , Prod profiles/environments

UAT :: User Acceptence Test



=>Both spring and spring boot supports profiles creation i.e we can not instaitation activate certains spring beans , properites/yml file in certain profile though project sporing beans ,files required for all the profiles /envirnomnets..

Dec 08 Spring Boot Profiles

=> To make a spring bean working for only for certain profile activation we can use @Profile annotation along with stereotype annotation or @Bean annotation

eg1:
===

(for user-defined classes)

```
@Repository("empDAO")
@Profile({"uat","prod"})
public class OracleEmployeeDAOImpl
             implements IEmployeeDAO{
  ....
  ...
  ...
}
```

```
@Repository("empDAO")
@Profile({"dev","test"})
public class MysqlEmployeeDAOImpl
             implements IEmployeeDAO{
  ....
  ...
  ...
}
```

=> The spring bean certain profile will be loaded and instantiated only when that profile is active

=> if do not give any @Profile annotation for any spring bean then it works all the profiles.

eg2:

```
@Configuration
public class AppConfig{

                                        //for pre-defined classes
                                          which not coming in autoConfiguration
 @Bean(name="dbcpDs")                     Process.
 @Profile("dev")
 public DataSource createApacheDBCPDS(){
   BasicDataSource bds=new BasicDataSource();
     ...
     ..
  return bds;
  }

   @Bean(name="hkDs")
   @Profile("prod")
   public DataSource createHikariDS(){
     HikariDataSource hkds=new HikariDataSource();
       ...
       ..
    return hkds;
    }
```

=>Spring boot spring bean classes will be kept ready through AutoConfiguration to make such classes participating only for certain profile activation we need to need to multiple properties files or yml files by following some naming convetions..

application-dev.properties/yml
application-test.properties/yml
application-uat.properties/yml
application-prod.properties/yml

notation: application-<profilename>.properties/yml
(can be any thing; fixed one; fixed)

wrong notations
==========
dev-application.properties/yml
prod-dev-application.propeties/yml

Profile name can be anythig while creating yml/properties file involving profiles.. but the same profile names should also reflect in @Profile annotation and while actvating certain profile

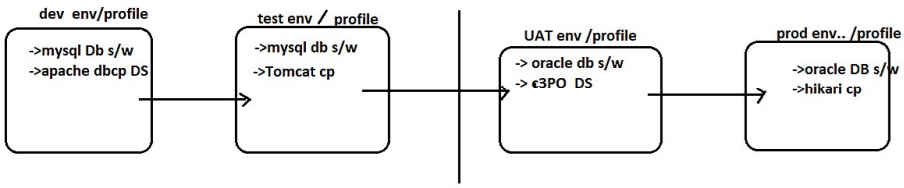
note:: While with profiles we take multiple properties files and yml files by following naming conventions ..So there is no need cfg them as user-defined properties/yml files.

we can actiavate profile from application.yml/properties file

application.properties/yml (Base file)
---
spring.profiles.active= test

Example App (making our MiniProject working for different profiles)
========== ==========

UAT = client org side testing with client org people

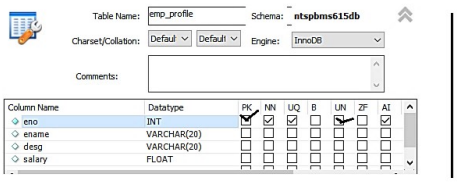


=>In Project , we can create any no.of profiles
=>Generally , we see 4 to 6 profiles in real projects

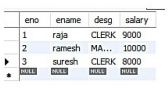
step1) Keep spring MiniProject ready as seperate project

step2) make sure that emp db table with records in avaiable in mysql DB s/w

mysql workbench ---> login ---> go to logical db --> tables --> create new table



apply --->next--> finish



eno	ename	desg	salary
1	raja	CLERK	9000
2	ramesh	MA...	10000
3	suresh	CLERK	8000
NULL	NULL	NULL	NULL

Dec 08.1 Spring Boot Profiles

step3) Add the following dependencies in pom.xml file

```
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
</dependency>
<dependency>
  <groupId>org.apache.commons</groupId>
  <artifactId>commons-dbcp2</artifactId>
</dependency>
<dependency>
  <groupId>com.mchange</groupId>
  <artifactId>c3p0</artifactId>
  <version>0.9.5.5</version>
</dependency>
<dependency>
  <groupId>org.apache.tomcat</groupId>
  <artifactId>tomcat-jdbc</artifactId>
</dependency>
```

step4) Develop seperate DAO class for mysql persistence operations

```
@Repository("mysqlEmpDAO")
@Profile({"dev","test"})
public class MysqlEmployeeDAOImpl implements IEmployeeDAO {
      private static final String GET_EMPS_BY_DESGS="SELECT ENO,ENAME,DESG,SALARY FROM
                                                     EMP_PROFILE WHERE DESG IN(?,?,?) ORDER BY DESG";
      @Autowired
      private DataSource ds;
  ...
  ...
  ...
 }
```

```
@Repository("oraEmpDAO")
@Profile({"uat","prod"})
public class OracleEmployeeDAOImpl implements IEmployeeDAO {
      private static final String GET_EMPS_BY_DESGS="SELECT EMPNO,ENAME,JOB,SAL
                                                     FROM EMP WHERE JOB IN(?,?,?) ORDER BY JOB";
      @Autowired
      private DataSource ds;

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

plan:: OracleEmployeeDAOImpl (for uat, prod profiles)
MysqlEmployeeDAOImpl (for test, dev profiles)
EmployeeMgmtServiceImpl (No @Profile -common for all profiles)
PayRollSystemController (No @Profile -common for all profiles)

step5) In Main class place @Bean method to make C3PO related ComboPooledDataSource class obj as spring bean only for "uat" profile

In Main class

```
==============
    @Bean(name="c3p0Ds")
    @Profile("uat")
    public ComboPooledDataSource createC3PODs() throws Exception {
          System.out.println("BootProj03MiniProjectLayeredAppApplication.createC3PODs()");
           ComboPooledDataSource cds=new ComboPooledDataSource();
           cds.setDriverClass("oracle.jdbc.driver.OracleDriver");
           cds.setJdbcUrl("jdbc:oracle:thin:@localhost:1521:xe");
           cds.setUser("system");
           cds.setPassword("manager");
           return cds;
    }
```

note:- in the interviews dont give urls of your daily practicing things bcoZ in real time Projects things will happen in a different way so just refer the below blue box for url of DB driver give like=>>>jdbc:oracle:thin:@<hostname>:1521:<sid>

just collect from TL/PL

sample url In real projects)
jdbc:mysql://polaris-lms.ct94gqg5c35w.ap-south-12.rds.amazonaws.com:3306/mgapidb
(mysql) — *logical DB*
jdbc:oracle:thin:@polaris-lms.ct94gqg5c35w.ap-south-12.rds.amazonaws.com:1521:mgapidb
(oracle) — *SID*

generally it'll be project name

step6) develop multiple properties files for mutlipel profiles by following naming conventions

application-dev.properties

```
#jdbc proeprties for DataSource AutoConfiguration
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql:///ntspbms615db
spring.datasource.username=root
spring.datasource.password=root

# Activate specific type of DataSource for autoConfiguration by
  breaking the DS algorithm
spring.datasource.type=org.apache.commons.dbcp2.BasicDataSource
```

application-test.properties

```
#jdbc proeprties for DataSource AutoConfiguration
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql:///ntspbms615db
spring.datasource.username=root
spring.datasource.password=root

# Activate specific type of DataSource for autoConfiguration by
  breaking the DS algorithm
spring.datasource.type=org.apache.tomcat.jdbc.pool.DataSource
```

application-uat.properties

```
#jdbc proeprties for DataSource AutoConfiguration
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
```

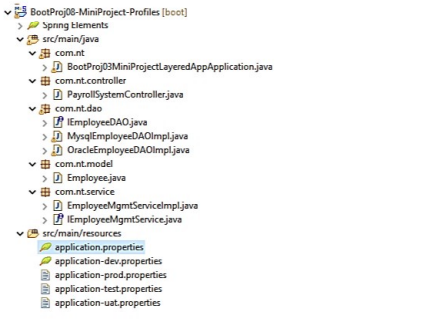
application-prod.properties

```
#jdbc proeprties for DataSource AutoConfiguration
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
```

step7) activate profile from application.properties file

```
applicaiton.properties
======================
#activate profile
spring.profiles.active=dev
```

step8) Run the Client App



Dec 10 Spring Boot Profiles using yml

We can activate profile in spring boot in 3 ways

a) using spring.profiles.active=<val> key of application.properties/yml file (best)

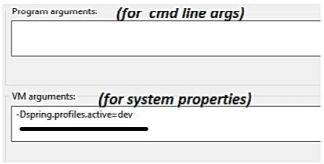
b) using the system property -Dspring.profiles.active=<val>

c) Using Programatic approach (By writing java code in main class)

---

b) using the system property -Dspring.profiles.active=<val>

Right click on Project ---> run as ---> Run cfgs --->



==>Apply --->run

System proerpties	cmd line args
(a) Optional to use thoguh they expected in the application	(a) mandatory to pass when are specfied and expected in the application otherwise ArrayIndexOutOfBoundsException will come
(b) Will be read in the application using System.getProperty(-) method	(b) will be read using main method String[] param like args[0],args[1] and etc
(c) Indentified with their names	c) Indentfied with their indexes of args[]
(d) We have pre-defined system properties like os.name and etc.. and also create user-defined system properties using cmd>java -D option	d) only user-defined cmd line args are avaiable.
(e) In eclipse IDE use VM arguments box for supplying system properties	(e) In Eclipse IDE use Programm argumentx box to supply cmd line args

c) Using Programatic approach (By writing java code in main class)

Spring boot style

In Main class

```
SpringApplication application=
        new SpringApplication(BootProj03MiniProjectLayeredAppApplication.class);
application.setAdditionalProfiles("prod");
ApplicationContext ctx=application.run( args);
```

Q) if we acivate 3 different profiles in 3 different approaches , can u tell me what will happen?

ans) Since 3 profile will be activated at a time ..there is possibility getting ambiguity problem..

Field dao in com.nt.service.EmployeeMgmtServiceImpl required a single bean, but 2 were found:

Q) can we activate multiple profiles at a time using single approach ?

Ans) yes , we can pass list of profiles as comma seperated values in all 3 approaches

in application.properties

```
#activate profile
spring.profiles.active=test,dev
```

if both activated profiles refering not giving any conflicts then lastly specified profile settings will be taken

eg: (test,dev) since both are refering to mysql Db s/w and both are having spring.datasoruce.type , So "dev" profile settings will be taken

note:: when specified profiles activated becarefull there is possiblity of getting abmguity problem (esecially DataSources and DAO classes refering to Db s/ws)

eg: (dev,prod) since both are refering to two different DB s/ws like oracle, mysql Db s/w , So the ambiguity problem will be raised with respect to DAO classes)

In Programtic approach

```
//Sprinb boot style
SpringApplication application=new SpringApplication(BootProj03MiniProjectLayeredAppApplication.class);
application.setAdditionalProfiles("test","dev");
ApplicationContext ctx=application.run( args);
```

In System properties approach

=>we generally activate one profile at time.. if need we can activate multiples on the copy of the project as shown above

(Performing Development and testing parellel needs needs activating multiple profiles at time)

---

if no profile activated or given profile is not there to activate then what happends?

Ans) The infomation placed in application.properties file will be used as the default fallback content so we can say application.properties/yml is default fallack profile file

Dec 11 Spring Boot Profiles using yml

Spring Boot Profiles using yml files

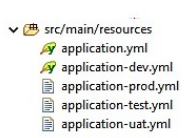
==========

=> While working yml we supply profile details in two ways

a) Using multiple yml fies for multiple profiles

b) Using single yml file for all the profiles

a) Using multiple yml fies for multiple profiles



application.yml

```
spring:
  profiles:
    active: prod
```

applicaiton-dev. yml

```
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    password: root
    type: org.apache.commons.dbcp2.BasicDataSource
    url: jdbc:mysql:///ntspbms615db
    username: root
```

applicaiton-test.yml

```
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    password: root
    type: org.apache.commons.dbcp2.BasicDataSource
    url: jdbc:mysql:///ntspbms615db
    username: root
```

applicaiton-uat.yml

```
spring:
 datasource:
  driver-class-name: oracle.jdbc.driver.OracleDriver
  password: manager
  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
```

b) Using single yml file for all the profiles

=>we can uses --- as the seperator to place multiple profiles info in single yml file and same seperate can also be used to activate the profile

To specify active profile:

```
spring:
  profiles:
    active: ...
```

To specify each profile name

```
spring:
   profile: dev
```

example App

=======

appplication.yml

==========

```
spring:
  profiles:
    active: dev
---
spring:
  profiles: dev
  datasource:
     driver-class-name: com.mysql.cj.jdbc.Driver
     password: root
     type: org.apache.commons.dbcp2.BasicDataSource
     url: jdbc:mysql:///ntspbms615db
     username: root
---
spring:
  profiles: test
  datasource:
     driver-class-name: com.mysql.cj.jdbc.Driver
     password: root
     type: org.apache.tomcat.jdbc.pool.DataSource
     url: jdbc:mysql:///ntspbms615db
     username: root
---
spring:
  profiles: uat
  datasource:
      driver-class-name: oracle.jdbc.driver.OracleDriver
      password: manager
      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
```

Here spring boot internally uses snakeyml to convert single yml file content into multiple properties files by taking "---" as the delimeter or seperator.

Dec 11.1 Spring Boot Profiles using yml

Q) if spirng Beans are abgutity Problem two Injecting to target spring bean class object how can we choose only certain spring bean as dependent spring bean with out touching the source code of spring beans?

Ans1) Using @ImportRespurce(to add spring cfg file) + <alias> tag spring bean cfg file + @Qualifer
Ans2) Using Spring Boot Profiles (Best)

Q) What is the meaning @Profile("default")?

Ans) It keep spring bean in default profile /env.. which will be activated automatically when no active profile is specified.

```
@Repository("oraEmpDAO")
@Profile({"uat","prod","default"})
public class OracleEmployeeDAOImpl implements IEmployeeDAO {
  ...
  ...
  ..
 }
```

note:: The spring bean of @Profile("default") will not activated when wrong profile is specified as the active profile i.e the default spring bean will be activated only when no active profile is specified and correct available profile is specified.

Q) What to do if want to use other than application.properties file content as the fallback profile file?

=>we can mention some profile name as the default fallback profile name in spring.profiles.default key of application.properties

```
In application.properties
---------------------------------
  #spring.profiles.active=uat   (No active profile)
  spring.profiles.default=prod  (takes  prod default fallback profile)
```

=>if do not have active profile and default profile then the content specified in application.properties will be used as fallback profile content.. if content keys and values matching with other profile specific properties file /yml file keys and values then instead of writing in application.properites , we can include from profile specific properties/yml file by using spring.profiles.include key of application.properties (This is making profile specific properties file/yml file as the child file to application.properties/yml file)

```
applicatin.propeties
====================
#activate profile
#spring.profiles.active=prod  (no active profile)

#spring.profiles.default=prod  (no default profle)


spring.profiles.include=uat   (including child profile)
```

if we specify active profile , default profile and child profile at a time in the application then what happens?

Ans) if there is no ambiguirty Problems or Spring bean clashes then Include profile (child profile) will be taken as final profile

```
application.properties
====================
 spring.profiles.include=uat

 spring.profiles.default=prod

 spring.profiles.active=prod
```

What is the difference between keeping spring Bean in no profile (not adding @Profile) and default profile (@Profile("default")) ?

ans) No @Profile Spring bean class object is creted when there is no active profile and when there is active profile

@Profile("default") spring bean class object is created only where there is no active profile

(note:: if comment spring.profiles.active key in the properties file/yml then we can there is no active profile)

What is the difference b/w default profile and include/child profile?

Ans) The Profile that executes when there is no active profile is called default profile we use spring.profiles.default key for this.

The Profile that is linked with applicaiton.properties to reuse key and values when there is no active and default profile is called child /include profile .For that we use spring.profiles.include key.

usecase :: Generally we keep Controller and serivce classes in No @Porfile env.. becoz Thery are not specfic to any Db s/w and commonly required in all profiles to excute

=>we keep DataSources and DAO classes in specific profile becoz we prefer working with different DAO classes, DataSoruces in different profile like dev, test profiles we use mysql Db s/w and uat,prod profiles we use oracle Db s/w.

Runners in spring Boot Application

Dec 13 Runners in Spring Boot

=>Runner is a spring bean of spring boot App that contains one time excution logics in its run(-) method.

=>We do not call the run(-) of the Runner class manually ..It will be called by Spring Boot automatically as part of Application's startup prcoess that is started by SpringApplication.run(-) method Right after pre-instataition of singleton scope beans and completing the neccessary dependency Injections.

=>The Spring Boot App recognizes the spring bean class as the Runner class by seeing itx XxxRunner (I) implementation.

Two types Runners

a) CommandLineRunner (Spring bean class implementing org.springframework.boot.CommandLineRunner(I))

b) ApplicationRunner (Spring bean class implementing org.springframework.boot.ApplicationRunner(I))

What is the difference b/w placing static block in spring bean class and working Runner class?

static block of spring bean class	Runner class
(a) It is java feature , can be used in both spring and spring boot Apps	(a) It is a feature of spring boot Can be used only in spring boot app
(b) Executes automatically when the IOC container Loads the spring bean calss	(b) Executes automatically as part of Spring boot Application startup activity performed by SpringApplication.run(-) method
(c) Does not allow to pass data/args from outside	(c) Allows to pass data/args to run(-) method (command line args can be passoed)
(d) Non-static data can not be accessed in this static block	(d) Allows us to access both static and non-static data
(e) usefull to place class level one -time executing logics.	(e) Useful to place spring boot Application level one time executing logics (eg:: Executing the DB script during the Appliation startup to create db table in Db s/w)
(f) Static block does not support Exception Propagration	(f) run(-) method Runner class supports the Exception Propagations

note:: if we place multiple Runner classes the logics of all runner classes execute only for 1 time during the application startup prcoess.

org.springframework.boot.ApplicationRunner(I)

|--->public void run(ApplicationArguments args) throws Exception

=>This run method allow to get cmd line args as catergorized arg like option args(with name,value) and non option args(only value)

Modifier and Type	Method and Description
boolean	containsOption(String name) Return whether the set of option arguments parsed from the arguments contains an option with the given nam
List<String>	getNonOptionArgs() Return the collection of non-option arguments parsed.
Set<String>	getOptionNames() Return the names of all option arguments.
List<String>	getOptionValues(String name) Return the collection of values associated with the arguments option having the given name.
String[]	getSourceArgs() Return the raw unprocessed arguments that were passed to the application.

org.springframework.boot.CommandLineRunner(I)

|---->void run(String... args) throws Exception

=>Command line args will come directly..
No Categorization for cmd line args.

What is the difference b/w CommandLineRunner and ApplicationRunner ?

=>Both are one time execution classes /spring bean having run(-) method ..But the way they get cmd line args to the run (-) is different

CommandLineRunner gets the given cmd line args in the form of String[] where as

ApplicationRunner gets the given cmd line args in the ApplcationArguments object where we can categorize the cmnd line into optional and non-optional args

//CommandLineRunner example

TestRunner.java

```
package com.nt.runner;

import java.util.Arrays;

import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

@Component
public class TestRunner implements CommandLineRunner {

	@Override
	public void run(String... args) throws Exception {
		System.out.println("Runner to Test...."+Arrays.toString(args));
	}

}
```

Run as ---> run cfgs ----> arguments tag

Program arguments: `101 --name=raja --addrs=hyd`

non option arg: 101

option arg1: --name=raja

option arg2: --addrs=hyd

Variables...

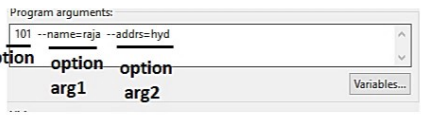
output:: Runner to Test....[101, --name=raja, --addrs=hyd]

Dec 13.1 Runners in Spring Boot

ApplicationRunner example
==========

```
package com.nt.runner;

import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.ApplicationRunner;
import org.springframework.stereotype.Component;

@Component
public class Test1Runner implements ApplicationRunner {

    @Override
    public void run(ApplicationArguments args) throws Exception {
        System.out.println("non option args values::"+args.getNonOptionArgs());

        System.out.println("Option arg names and values::");
        for(String name:args.getOptionNames()) {
            System.out.println(name+"-----> "+args.getOptionValues(name));
        }
    }
}
```

Run as ---> run cfgs ----> arguments tag



non option arg — option arg1 — option arg2

oulput: non option args values::[101]
Option arg names and values::
addrs-----> [hyd]
name-----> [raja]

note:: ComandLineRunner, ApplicationRunner are two indenpendent interfaces.

note:: Mostly used Runner in the industry is CommandLineRunner

note:: Both CommandLineRunner and ApplicationRunner interfaces are Functional Interfaces

Functional Interface (Java8 features)
==========

The interface that conatains only one abstract method declaration either directly or indirectly is called Funcational Interface/

eg::

```
public interface Demo{
  public void hello();
}
```

"Demo" is Functional Interface (FI)

```
public interface Demo1{
  public void m1();
  default void m2(){
     ...
  }
}
```

"Demo1" is FI

```
public interface Demo2 extends Demo{
  public void m2();
}
```

"Demo2" is not FI

```
interface Demo3 extends Demo{
}
```

"Demo3" is FI

```
@FunctionalInterface
interface Demo4{
  public void m1();
}
```

(Guides the compiler to take this interface Functional interface)

=>This annotation will not be recorded .class file .. It is just to guide the compiler.

Demo4 is "FI"

```
@FunctionalInterface    // compiler gives error
interface Demo5 extends Demo{
  public void m2();
}
```

Demo5 is not "FI"

Why Functional Interface concept has come from Java 8 onwords?

Ans) Java8 Lamda expression simplifies the way of providing implementation class and overrding for the interface that is having single method..(i.e Funcationl interface) .. Lamda expression can not be applied on interface with more that one method.. This made Java8 peope to given new identify for interface with single method that is funcational interface.

What is the difference b/w Functional interface and abstract class class?

=>Functional Interface contains only abstract method directly or indirectly where as abstract class can have 0 or more abstract methods

=> Funcational interface can extend from multiple other interfaces where as abstract class can extend from a class

=> Functionla interface is given from java8 and abstract class is available from beginign of java

=>Functional interface does not default super interface
Abstract class is having java.lang.Object class as the default super class

=> FI can have 0 or more default methods where as AC can not have default mmethods

=> FI does not allow to keep constructors .. where as AC allows to keep constructors

=> FI's one abstract methods is public static by default and member variable are public static final by default .. where as Abstract class gets no access modifier for methods and member variabkles by default.

=> In FI we can not place static , instance blocks where in AC we can place instance, static blocks.

Dec 14 Runners in Spring Boot Real Time Usecases

What are the realtime usecases of Runners in spring boot Projects?

=>We place initial or application startup logics in the run(-) method of Runner class..these usecase are

a) Creating db tables with initial row by executing DB Scripts during the application startup process

b) Performing unit Testing activities related to AutoConfiguration based Instantiations and Injections

c) Creating some dummy email ids by using java mail api during the application startup process

d) Triggering EMail messages to imp stack holders when important services are activated like Payment gateway service , UPI Payment service and etc.. through autoConguration Process.

e) Though there multiple modules in u r Project.. but if u r looking to invoke/activate only few modules then also we can use this Runner classes. and etc..

What is the difference b/w starters and Runners?

Ans) Staters are jar files/libaries in spring boot App having capability to perform to AutoConfiguration actvities by collecting the inputs from application.properties/yml file

=>Runners are spring bean classes having one time execution logics in their run(-) methods as intial logics or startup logics of the spring boot application

=> if we place multiple runners in spring boot Project .. first all ApplicationRunners will execute in the alphabetic order of their class names then all CommandLineRunner execute in the alphabetic order of their names.

[A-Z, a-z , numbers,symbols]

=>if u want to control the order of executing Runners then take the support @Order(<n>) or implement Ordered(I) on Runner class (Best)

=> we can pass any numeric value as prority value..

=>The prority value is Integer max value(2147364748) is Integer.MAX_VALUE

=>Low value indicates high priority
=>Hight value indicates low prority

(It does not check type of runner here whenever we are using @Order or Ordered(I)

=>If two Runner classes having same prirotity value then the spring Boot gives priority to Runner classes based their aphabets in the class names .. [the type of runner does not matter here]

=>if u place @Order with out priority value then it takes Integer.MAX_VALUE as the default value.

Class Runner name	Prority value
ARunner (CR)	-10 (II)
BRunner (AR)	-20 (I)
CRunner (CR)	1 (IV)
DRunner (CR)	30 (V)
XRunner (CR)	0 (III)

Class Runner name	Prority value
ARunner (CR)	10 (IV)
BRunner (AR)	10 (V)
CRunner (CR)	1 (II)
DRunner (CR)	1 (III)
XRunner (CR)	-5 (I)

Class Runner name	Prority value
ARunner (CR)	10 (II)
BRunner (AR)	<no value> (IV)
CRunner (CR)	20 (III)
DRunner (CR)	-20 (I)
XRunner (AR)	<no-value> (V)

Class Runner name	Prority value
ARunner (CR)	10 (III)
BRunner (AR)	20 (IV)
CRunner (CR)	<no-value> (V)
DRunner (CR)	<no-value> (VI)
XRunner (AR)	-1 ((I))
FRunner (AR)	0 (II)
ZRunner (CR)	No @Order (IX)
YRunner (AR)	No @Order (VIII)
FZRunner (AR)	<no-value> (VII)

Conclusion Prority Order of Runners

=> Based given Prority value using @Order or Ordered(I)
(High vlaue --> low prority and Low Value --High Priority)
(if multiple runners having same priroty value then go by alphabetic order Runner class names)

=> Based on Runner types
(First All Application Runners in alphabetic order will execute then All Command Line Runners in alphabetic order will execute)

```
@Component
@Order(-20)
public class ABTest1Runner implements ApplicationRunner {
 ....
 ...
}
```

```
@Component
@Order
public class CTestRunner implements CommandLineRunner {
 ...
}
```

Providing Prority Value for runner using Odereed(I) implementation

```
@Component
public class CTestRunner implements CommandLineRunner, Ordered{

    @Override
    public void run(String... args) throws Exception {
        System.out.println("CTestRunner.run()");
        System.out.println("Runner to Test...."+Arrays.toString(args));
    }
    @Override
    public int getOrder() {        --> Method of Ordered(I)
        return 10;
    }
}
```

if we provide two different prority values for a Runner using @Order annotation and Ordered(I) can u tell me which will be taken as the final prority value

Ans) *Ordered (I) implementation based priority will be taken as the final proirity value*

```
@Component
@Order(-30)
public class CTestRunner implements CommandLineRunner, Ordered{
    @Override
    public void run(String... args) throws Exception {
        System.out.println("CTestRunner.run()");
        System.out.println("Runner to Test...."+Arrays.toString(args));
    }
    @Override
    public int getOrder() {
        return 10;
    }
}
```

Why @Order.is recomanded to place prority value for Runner?

Ans) =>Easy to use
=>Gives good readability becoz it will placed on the top of the class
=>No need of implementing Interface and method
=> One Line code
=> Can get Prority value from application.properties/yml file if needed *
and etc..

Dec 15 Runners in Spring Boot By Using Fucntional Interface

Functional Interface

=>The interface that contains only one abstract method directly or indirectly is called Functional Interface
=>We can mark interface as Funtional Interface @FunctionalInterface annotation .. But the compiler checks weather the interface is really functional interface or not?

=> CommandLineRunner, ApplicationRunner interfaces are Functional Interfaces becoz they are having single abstract method declaration called run(-)

Upto Jdk 1.7
==========

```
interface Arithemtic{
 public int add(int a,int b);
}
```

```
public class ArithemeticImpl implements Arithmetic {
    public int add(int x,int y){
       return a+b;
    }
}

Arithmetic ar=new ArithmeticImpl();
 ar.add(10,20);
```

From Java8 LAMDA expression simplifies the process of creating implemtation classes for functional interfaces.. it Internally generates anonymous inner class based Impl class for the Functional Interface

From Jdk1.8 onwords
==========

```
interface Arithemtic{
 public int add(int a,int b);
}
```

Impl class + object creation (LAMDA style)
==========

```
Arithemetic ar=(int a, int b)->{
                    int c=a+b;
                    return c;
                };
ar.add(10,20);
```

Anonymous inner class based impl class for the functional interface Arithmetic and also creating object for that class

---

Improvement1

```
Arithmetic ar=(int a,int b)->{return a+b;};
ar.add(10,20);
```

taking method body as single statement

Improvement2

```
Arithmetic ar=(int a,int b)->  a+b;
ar.add(10,20);
```

flower bracket is optiona for single line statement and also return statement

---

improvement3

```
Arithemetic ar=(a,b)  -> a+b
ar.add(10,20);
```

Data types are optional to place

note:: it it is single param then "()" are also optional

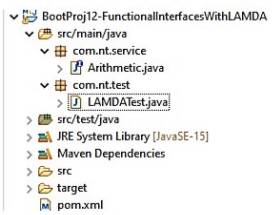
---

improvement 4

```
Arithmetic ar=(x,y) -> x+y
 ar.add(10,20);
```

note: parameter names can be changed..

All these are possible only on Functional interfaces



//Airthmetic.java
==========

```
package com.nt.service;

public interface Arithmetic {
  public int add(int a,int b);
}
```

LAMDATest.java
==========

```
package com.nt.test;

import com.nt.service.Arithmetic;

public class LAMDATest {

        public static void main(String[] args) {
                //style1
                /*Arithmetic ar=(int a,int b)->{
                                int c=a+b;
                                 return c;
                };
                 System.out.println("sum ::"+ar.add(10,20));*/
                //style2
                Arithmetic ar=(int a,int b)->{
                     return a+b;
                };
                System.out.println("sum ::"+ar.add(10,20));

                /*Arithmetic ar=(int a,int b)-> a+b;
                System.out.println("sum ::"+ar.add(10,20));*/

                /*Arithmetic ar=(a, b)-> a+b;
                System.out.println("sum ::"+ar.add(10,20));
                */

                /*Arithmetic ar=(x, y)->; x+y;
                System.out.println("sum ::"+ar.add(10,20));*/
        }
    }
```

Dec 15.1 Runners in Spring Boot

important java8 features to learn
a) LAMDA Expressions
b) Functional Interfaces
c) default methods & static methods
d) Java8 date and time api
e) method reference (::)
f) consructor reference
g) Stream api
h) Optional API
i) Repetable Annotation
j) forEach method
k) Future API
and etc..

=>since CommandLineRunner,ApplicationRunner interfaces of spring boot are Functional interfaces we can implement them in main class it self with out seperate files using the support of LAMDA Expressions..

```
package com.nt;

import java.util.Set;
import java.util.stream.Stream;

import org.springframework.boot.ApplicationRunner;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
public class BootProj12RunnersLamdaApplication {

@Bean
public CommandLineRunner createCLRunner() {

/* // LAMDA Impl for functional Interface
   CommandLineRunner clr=(String ...args)->{

          System.out.println("From CommandLineRunner:"+Arrays.toString(args));
};
return clr;*/

/* // LAMDA Impl for functional Interface (improvision)
 CommandLineRunner clr=(args)-> System.out.println("From CommandLineRunner:"+Arrays.toString(args));
return clr;*/

/*
  // LAMDA Impl for functional Interface (improvision)
 CommandLineRunner clr=args-> System.out.println("From CommandLineRunner:"+Arrays.toString(args));
return clr;*/

/*      // LAMDA Impl for functional Interface with for each /enhance for loop (
       CommandLineRunner clr=args->{
                  System.out.println("Command LineRuner");
                       for(String arg:args) {
                          System.out.println(arg);
                       }
              };
              return clr;
       */

        // LAMDA Impl for functional Interface+ stream api +static method reference
        CommandLineRunner clr=args-> {
               System.out.println("commadn Line Runner");
                Stream.of(args).forEach(System.out::println);
        };

        return clr;

/*
 // LAMDA Impl for functional Interface+ stream api +static method reference (improvistion)
  CommandLineRunner clr=args->  Stream.of(args).forEach(System.out::println);
return clr;
*/

// LAMDA Impl for functional Interface+ stream api +static method reference (improvision)

//return args->Stream.of(args).forEach(System.out::println);

}
      @Bean
      public ApplicationRunner createAppRnner() {

            return aargs->{
                  System.out.println ("application runnner");
                    System.out.println("non option arg values:"+aargs.getNonOptionArgs());
                    System.out.println("option arg names and values");
                    Set<String> optionArgs=aargs.getOptionNames();
                   optionArgs.forEach(name->{
                          System.out.println(name);
                          System.out.println(aargs.getOptionValues(name));
                   });
            };
      }
      public static void main(String[] args) {
            SpringApplication.run(BootProj12RunnersLamdaApplication.class, args);
      }
}
```

note:: In real time Projects seperate files will be taken for Runner classes for better redability

Dec 16 Spring data Intro

Spring Data (main module having multiple sub modules)
|-->spring data jpa
|-->spring data jdbc
|--->spring data mongodb
|--->spring data cassendra
30+ sub modules are.. and etc..

Before the arrival of Spring and Spring boot
==========

IO Streams
Java App ----------> Files
(stream based pesistence logic)

JDBC +SQL
Java App ----------> RDBMS DB s/w
(SQL querues jdbc code)

O-R Mapping
Java App ----------> RDBMS DB s/w
(hiberante)
(Objects based Persistence logic)

Basic 3 approaches
of interacting with
Files and RDBMS DB s/w

RDBMS DB s/ws :: oracle,mysql,postgre sql
and etc..

=>Files are not great Persistence stores becoz of security problem, redundency problem, consistency problem and etc.. So we use Files and IO streams only in small scale Apps like desktop games, mobile Apps , IOT /Embedded System apps and etc..

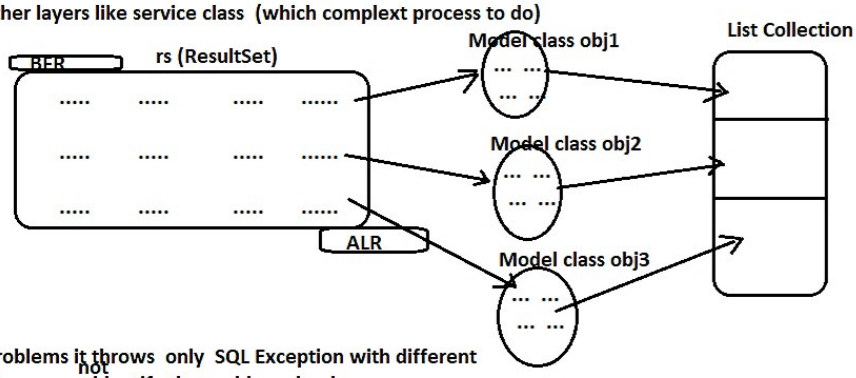
=> Since java is known for medium and large scale projects devleopment , we generally do not use Files as Persistnece stores in Java Projects.. we use only Db s/ws as persistence stores..

Limitations of JDBC style Persistence logic
==========

=> jdbc code needs SQL queries in persistence logic devleopment, but 30% to 40% SQL queries Db s/w dependent SQL queries , So JDBC persistecen logic is DB s/w dependent (or) not portable across the multiple Db s/w.

=> Boiler plate code problem
(loading driver class, creating jdbc con, creating jdbc statement, closing jdbc objs, exception halding and etc.. statements repeat in multiple parts of the Project either with no change or with minor change.. So this called boilerplate code problem)

=> ResultSet obj is not Serializable object to send over the network. We should convert RS Object records into List of Model class objects to send over the network or to send other layers like service class (which complext process to do)



=> For all problems it throws only SQL Exception with different messages i.e we can identify the problem clearly..
not

=> SQLException is a checked exception , So we will be forced to catch and handle Exception whichs creates the issues towards Exception propagation
(For this we should remember and to exception re throwing operation)

=> There is not proper Trsanaction Magmenet support (commit , rollback operations)

=> Most of the features we need to implement by wring the logics.
and etc..

=>To overcome these problems take the support of O-R Mapping (ORM)

ORM (O-R Mapping/ Object-Relattional Management)
=====

=> The Process of mapping java classes with Db tables and the properties of java classes with the columns of db tables and representing the objects of java classes with the rows of db table having synchrnozization among them is called o-r mapping

1 class ---------> 1 db table
1 property ----> 1 column
1 object ------> 1 record

=>synchronization b/w objects and db table rows is nothing but modifications done in object reflect to db table row and vice-versa.

Dec 16.1 Spring Data Intro

//Entity class/Model class/Persistence class
==========

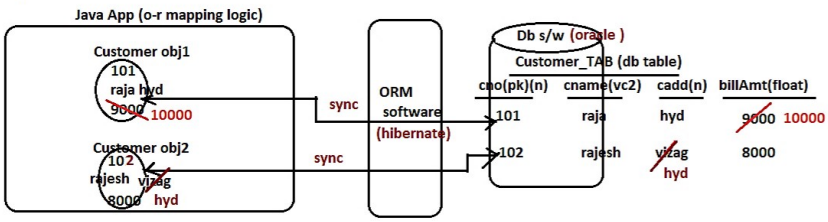
Customer.java

```
@Data
public class Customer{
 private  Integer cno;
 private String cname;
 private  String cadd;
 private  Float billAmt;

 }
```

o-rmapping cfgs (xml cfgs, or annotations)

Customer(java class) <----------> Customer_TAB (db table)

java class properties		Db table columns
	cno <----------> cno	
	cname <----------> cname	
	cadd <----------> cadd	
	billAmt <----------> billAmt	



=> In o-r mapping the all persistence opeartions (CURD Operations) will take place through objects with out SQL queries by giving objects based instructions to ORM frameworks, So the objects based persistence logic in o-r mapping will become Db s/w indepedent persistence logic.

In O-R mapping the ORM software takes objects based persistence instructions and generates the JDBC code +SQL query internally to complete that persistence operation on Db s/w.

In o-r mapping
=>save object means making the ORM software collecting data from object inserting that data to db table as record by generating JDBC code +INSERT SQL Query internally
=>load object means making the ORM software collecting data from db table record and storing into the object by generating JDBC code +SELECT SQL query internally
=>update object means making the ORM software updating the record of db table represented by the object by generating JDBC code +UPDATE SQL query internally
=>deleting object means making the ORM software deleting the record of db table represented by the object by generating JDBC code +DELETE SQL query internally

List of ORM frameworks
==========

hiberante ---------> from softreee/redhat (Best)
Link ----------> from eclipse
iBatis ----------> from apache
Toplink -------> from oracle corp
OJB ----------> from apache
and etc..

When should we use JDBC and Where should use ORM (hibernate) in the persistence logic devleopment?

And) if app gets huge amount of data batch by batch for Processing like Census information or surey information or marketing information or covid information then using ORM is problematic becoz to represent 1 record 1 object is required ..suppose if the batch contains 80k records to process then 80k objects are required to create at time. which lead Application crash.. So prefre JDBC for this batch processing where single ResultSet object will be created to prcoess huge no.of records..

Batch Prcoessing ---> JDBC is good

=> if app is getting little amount of data to process (30 to 40 records at a time) then prefer ORM becoz we can enjony other features of ORM like Db portability , Versioning, timestampng, caching ,locking and etc..

Little amount of data to process ----> ORM is good.

(Naresh it registrataions , online shopping carts, university apps, ticket booking apps and etc..)

note:: JDBC is not outdated and will not be outdated becoz all ORMs intenrally uses JDBC code

note:: if data of app is beyond storing and processsing capacity in regular DB s/ws then use BigData concepts to store and process data ..
eg:: hadoop , spark (Big dataframeworks)
BigData applications :: youtube , amazon, facebook,google, flipkart and etc..

=> hadoop, spark provides both storing and processing mechanizm by taking multiple ordinary computers of a network as cluster..

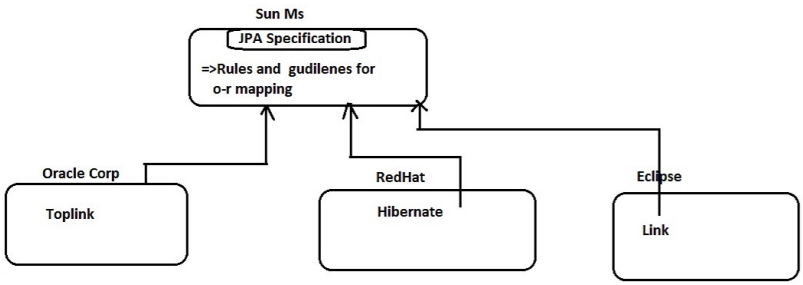
what is the difference JPA and ORM ?

JPA (Theory)
ORM (praticals)

JPA is planning/specification
ORM is implementation of JPA planning

ORM is pratical Impl of JPA concepts

=>JPA provides set of rules and guildelines to develop objects Persistence logic
=>ORM is the real implemetation of JPA Rules and guidelines . all ORM softwware (like hibernate) are developed based JPA specification rules and guidelines .. So we use ORM software directly to JPA rules based o-r mapping persitence logic.



Hibernate programming means writing o-r mapping logics in hibernate style
hibernate with JPA programming means o-r mapping logics in JPA-hiberante style

Dec 17 Spring Data Intro

Different types of DB s/ws

1.SQL DB s/ws (Relational Db s/w)

eg::oracle,mysql, postgreSQL, DB2 and etc..

2.NoSQL DB s/ws (Non-relational Db s/w)

eg: MongoDB, cassendra, Neo4j , Firebase, Redis , dynamo DB and etc..

When should we go SQL Db s/w and when should we go for NoSQL Db s/w?

=> if Data is having fixed structure with fixed attributes and DB schemas (table structures) not changing dynamically then go for SQL Db s/w.

eg:: Employee db table with 20 colums

=>Here every employee can have max of 20 details
=>if certain employee want to have more than 20 details ..then it is not recomanded to work with SQL Db s/w
=> if certain employees are having less than 20 details then memory remaing columns will be wasted.

Employee infomaton In employee db table with 20 columns
employee1 with 20 details (good)
employee2 with 10 details (waste of memory)
employee3 with 25 details (we can not store last 5 details)

all records allocates memory for 20 records irrespective 20 details are given or not

CUSTOMER_ID	PRODUCT_ID	ORDER_DATE	TOTAL
6	11	01-NOV-17	213601.66
6	278	01-NOV-17	94240.61
1	4		299698.89
1	186	27-OCT-17	211697.08
1	181	27-OCT-17	142998.57
1	218	27-OCT-17	119444.54
1		27-OCT-17	75130.37
1	12	27-OCT-17	27224.34
1	172	27-OCT-17	13264.13
1	258	27-OCT-17	

=>if the data is unstructured and dynamically growing or decreasing i.e shema is is changing dynamically then prefer working no-SQL db s/w like Mongo DB..
=>The MongoDB DB s/w stores data in the form of documents(records) in place called Collection(db table)
Here doc will not have fixed stucture .. can increase or decrease data as needed.

Employee collection in MongoDb s/w
doc1 with 5 details
doc2 with 10 details
doc3 with 20 details
doc4 with 5 details different from doc1
doc5 with 50 details.

Each doc allocates memory as need becoz docs are unstrcuted.

```
> db.employee.find().pretty()
{
        "_id" : ObjectId("5e49177592e6dfa3fc48dd73"),
        "name" : "Sonu",
        "age" : 26,
        "branch" : "CSE",
        "department" : "HR",
        "salary" : 30000
}
{
        "_id" : ObjectId("5e49813692e6dfa3fc48dd74"),
        "name" : "Rohit",
        "branch" : "CSE",
        "department" : "Development",
        "salary" : 40000

}
{
        "_id" : ObjectId("5e49813692e6dfa3fc48dd75"),
        "name" : "Mohit",
        "age" : 26,
        "department" : "Development",
        "desg" : "PROGRAMMER"
}
>
```

Every document (record) is in JSON format

Conclusion: Data is structured go for SQL Db s/w
Data in unstructured go ro NO SQL db s/w
note:: Maintaining sturucutured data in Mongo DB can be done but not recomanded

Before arrival of spring data to the market

Java App ----------> SQL Db s/w
(jdbc code)

Java App ----------> SQL Db s/w
(spring jdbc code)

Spring JDBC provides the abstraction plain jdbc code by taking care of common logics internally.. nothinb but avoids boiler plate code.
note:: Instead of using plain jdbc code prefer spring JDBC style jdbc code.

Java App ----------> SQL Db s/w
(hibernate or ibatis or link code)

Java App ----------> SQL Db s/w
(spring based hibernate logics)

plain hibernate code contains lots of common logics (boilerplate code)
So use spring ORM that provides abstraction and simplifies O-R mapping persistence logic development with support Template clsses like HiberanteTemplate classes.

Java App ----------> MongoDB Db/w (No SQL)

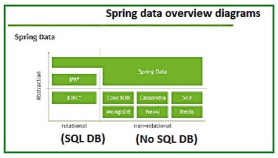
Java App ----------> Cassendra Db/w (No SQL)

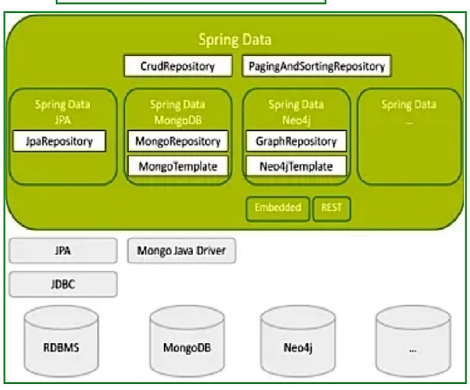
Java App ----------> Couchbase DB s/w (NOSQL)

note:: Before the arrival of In spring data module .. there is no module/provision in spring to interact with No SQL DB s/w.. and while interacting with SQL Db s/w for jdbc style logics seperate modules(spring JDBC) are given and for o-r mapping logics seperate modules(spring ORM) are given
i.e There is no uniformity/unified model towards interacting with SQL DB s/w using jdbc style and orm style.

=>Spring Data module is given providing uniformity and unifiled model to interact both SQL DB s/ws (JDBC style or ORM style) and No SQL Db s/ws.

Spring data overview diagrams





Dec 17.1 Spring Data Intro

sub modules of spring Data module

==========

Main modules

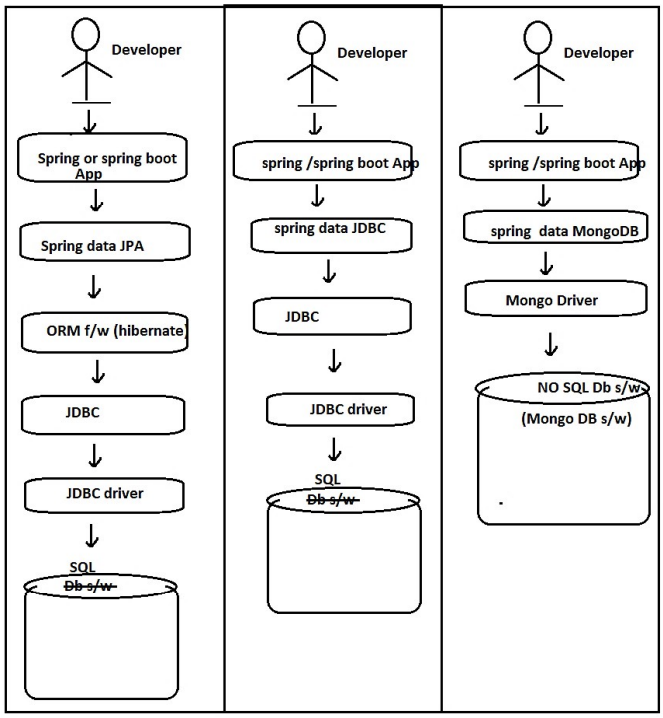
- Spring Data Commons - Core Spring concepts underpinning every Spring Data module.
- Spring Data JDBC - Spring Data repository support for JDBC.
- Spring Data JDBC Ext - Support for database specific extensions to standard JDBC including support for Oracle RAC fast connection failover, AQ JMS support and support for using advanced data types.
- Spring Data JPA - Spring Data repository support for JPA.
- Spring Data KeyValue - `Map` based repositories and SPIs to easily build a Spring Data module for key-value stores.
- Spring Data LDAP - Spring Data repository support for Spring LDAP.
- Spring Data MongoDB - Spring based, object-document support and repositories for MongoDB.
- Spring Data Redis - Easy configuration and access to Redis from Spring applications.
- Spring Data REST - Exports Spring Data repositories as hypermedia-driven RESTful resources.
- Spring Data for Apache Cassandra - Easy configuration and access to Apache Cassandra or large scale, highly available, data oriented Spring applications.
- Spring Data for Apache Geode - Easy configuration and access to Apache Geode for highly consistent, low latency, data oriented Spring applications.
- Spring Data for Pivotal GemFire - Easy configuration and access to Pivotal GemFire for your highly consistent, low latency/high through-put, data-oriented Spring applications.

Activate Windows

Community modules

- Spring Data Aerospike - Spring Data module for Aerospike.
- Spring Data ArangoDB - Spring Data module for ArangoDB.
- Spring Data Couchbase - Spring Data module for Couchbase.
- Spring Data Azure Cosmos DB - Spring Data module for Microsoft Azure Cosmos DB.
- Spring Data Cloud Datastore - Spring Data module for Google Datastore.
- Spring Data Cloud Spanner - Spring Data module for Google Spanner.
- Spring Data DynamoDB - Spring Data module for DynamoDB.
- Spring Data Elasticsearch - Spring Data module for Elasticsearch.
- Spring Data Hazelcast - Provides Spring Data repository support for Hazelcast.
- Spring Data Jest - Spring Data module for Elasticsearch based on the Jest REST client.
- Spring Data Neo4j - Spring-based, object-graph support and repositories for Neo4j.
- Oracle NoSQL Database SDK for Spring Data - Spring Data module for Oracle NoSQL Database and Oracle NoSQL Cloud Service.
- Spring Data for Apache Solr - Easy configuration and access to Apache Solr for your search-oriented

Activate W



---

plain jdbc code

==========

=>load driver class
=>establish the conneciton
=>create statement obj
| common logics

=> send and execute SQL query
=>Gather results and process results
| App Specific logics

=> Exception handling
=> close jdbc objs
| common logics

note: Boilerplate code problem

spring JDBC App

==========

=>cfg and Inject JdbcTemplate class obj (takes care of common logics)
=>send and execute SQL query
=>gather results and process results
| specific logics

note:: No boilerplace code problem

---

spring data JDBC App

==========

=> create interface extending ready RepositoryInterface
(Our Repository) And
like CrudRepsitory/PagingSortingRepository
(common methods declaration)

(note:: Spring /spring boot internally generates Impl class as InMemory Proxy class for the above interface .In that process method inherited fromCrudRepository or PagingandSortingRepository will be implemented auomatically having jdbc style persistence logic)

plain jdbc code = common logics + app specific logics
spring jdbc code = only application specfic logics
spring data jdbc = no logics (only interface declaring/ defining becoz spring/spring boot generates jdbc style logics in the dyamically In Memoru proxy class for that interface)

note:: While working spring data any sub modile there will not having any impl class having persitence logics .. only interface declaration will be there ..guidling spring/spring boot to gnerate Persistnece logic in the dyamically generated inMemory proxy class.

Dec 18 Spring Data JPA

plain hibernate code /app
==========

=> create Configuration object giving hibernate cfg file and mapping file
=> create HB SessionFactory obj
=> create Session object
=> Begin Tx
| common logics

=> use Hibernate object to peform
Entity object based persistence operations
=> Gather results and process results
| (Application specific logics)

=> commit or rollback
=> Close Session obj
=>Close Sessionfactory obj
| common logics

=>Programmer needs to write both common logics + application specific logics
i.e there is boilerplate code problem

Spring ORM App with hibernate
==========

=>Cfg and Inject HibernateTemplate class obj (takes care of common logics)

=> use Hibernate object to peform
Entity object based persistence operations
=> Gather results and process results
| (Application specific logics)

note:: programmer needs to write only applicaiton specific logics
i.e there is no boilerplate code problem

Spring data JPA with Hiberante App
==========

=> create user-defined Repository Interface extending CrudRepository/PagingAndSortingReporsitory
=> declare custom method for custom logics (optional) (contains mutliple common methods)

(note: spring boot internall generats the impl class for our Repository interface as InMemory proxy class containing logics for Crud/PASRepository Methods as hibernate persistence logics)

Biggest Limitation with plain hibernate/spring orm
==========

=> if project contains 500 db tables and if u r looking to perform
6 to 10 persinstence operations (CURD operations) on each db table
then should 500 * 6 or 500 * 10 methods having persistence logics

=>Technically speacking we need to 500 DAO Interfaces, 500 DAO Impl classes and each Impl class contains 6 to 10 methods having persistence logic

Biggest Advantages with spring data jpa
==========

=> if project contains 500 db tables and if u r looking to perform
6 to 10 persinstence operations (CURD operations) on each db table
then we just need to take 500 user-defined Reposoty interfaces extending from Crud/PASRepository becoz the spring or spring boot itself generates the 500 Impl classes as InMemory proxy class having hibernate logics inside the methods.

Normal java class
==========

.java (source code) ---->javac----> .class(compiled code) ----> JVM loads the class as runtime & excutes class
(stores in HDD) (stores in HDD) (In the JVM memory of RAM)

InMemory Proxy java class
==========

Run the App ---> Proxy class generation ----> prxy calss compilation ---> Proxy class load & execution
(JVM memory of the RAM) (nothing but source code) (JVM memory of the RAM) (JVM memory of the RAM) (JVM memory of the RAM)

InMemory metdata :: The infomation is read from xml file or annotations to the JVM memory of the App where the application executes

InMemory Proxy class :: the dynamically created and cmpiler proxy class in the JVM memory of the RAM where the app executes.

Conclusion
==========

hibernate code :: both common logics + application specific logics
spring ORM code :: only application specific logics
spring data JPA code :: No logics (the internally geneated In memory proxy class takes care of the process)
| To learn and use spring data jpa the hiberante knowledge is mandatory

Benifits of ORM
==========

a) Dynamic scheema generation for RDBMS Db s/ws

b) Persinstence logic is object based persistnce logic ,
So the persistence logic DB portable acosss the multiple DB s/ws.

c) The select operations can get the results in desired fomat like Int, foloat, Entity object , list of Entity objs and etc..

d) Generators are given to generate PK column values dynamically.

e) Good Transaction mgmt support

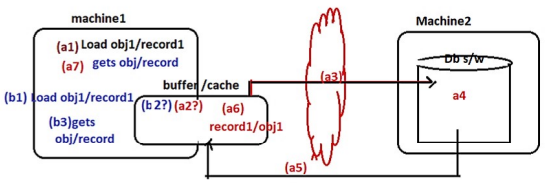
f) Versioning and time stamp support.

=>Versoning keeps track of how many times the obj/record is modified through ORM logic

=>Keeps track of when the object or record is lastly accessed and updated.

g) caching/Buffering :

=>Cache/buffer is a temporary method that holds data for temporary period .. The hibernate cache resides at Client side and uses that cache content acorss the multiple same requests to reduce network traffic b/w Client and server.



note:: It is recomanded to empty the cache at regular inervals for better results.

h) The Entity classes we develop using JPA annotations can be moved to any other ORM F/w becoz JPA annotations are Db s/w independent and the ORM sofware independent ;

i) Tries to avoid mismatches b/w Java features and RDBMS Db s/w fetures..

=>java classes can be in the inheritence where Db tables can not placed in inheritence but still the classes of inheritence should be mapped with 1 or more db tables ...for ORM framework provide special cfgs

=>java classes can not be the in relationship using FK column where as Db tables can .. i.e db tables relationship model is different from java classes relationship model .. but we still need to seem then participating in mapping activities.
(ORM frameworks given special cfgs to address this problem)

Dec 20 Spring Data JPA Basic interface and implementations

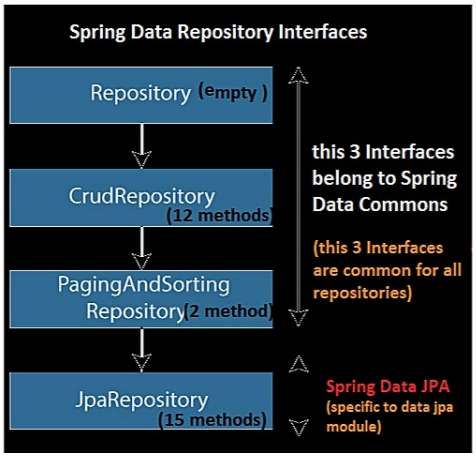
we can spring data jpa coding in 4 approaches
=> using spring driven xml cfgs
=> using spring driven xml +annotations cfgs
=> using spring 100% code driven cfgs
=> using spring boot (spring boot data jpa) (We are Going to learn this approach)

Repisotories in spring or spring boot data jpa

=> It is an interface extending from marker/tag interface (empty interface)
called org.springframework.data.repository.Repository<T,ID>
(Empty Interface)

=> The spring/spring boot dyanmically generate InMemory proxy class implementing our Interface keeping the necessary Persistence logic using hibernate ..

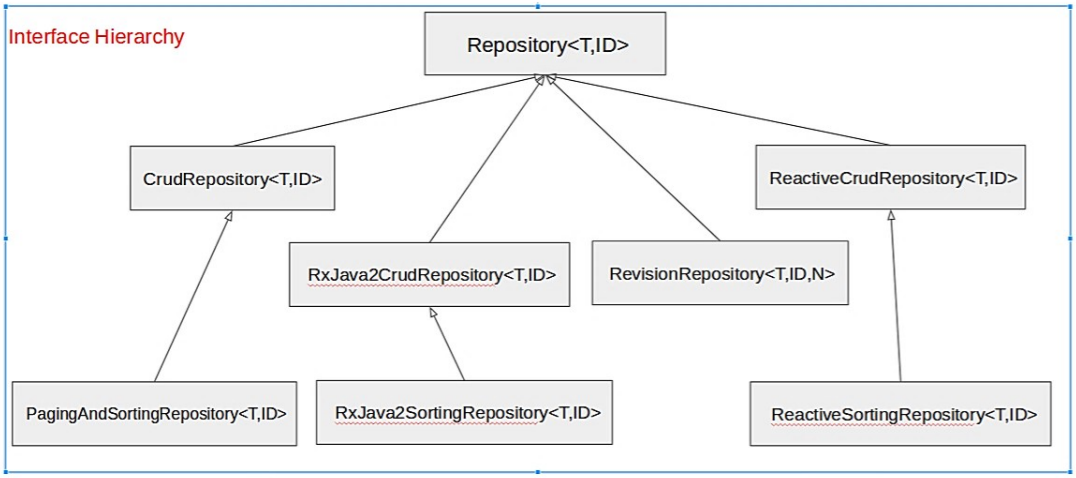
Repository Interfaces hierarchy



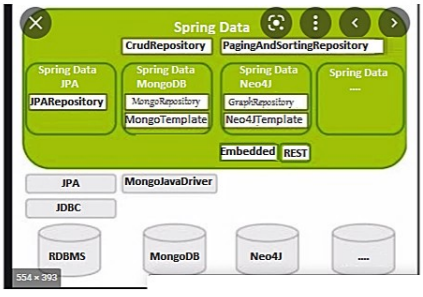
note:: spring and spring boot generates InMemory Proxy class for our Repository(I) only when that interface extends org.springframework.data.repository.Repository<T,ID> directly or indirectly.

All Repository interfaces given by spring data jpa sub modules extends from the marker Repository Interface directly or in directly to make underlying env.. to generate InMemory proxy class.

In Repository<T,ID>
=> <T> indicates Template class/ Entity class (internally generates hibernate code in Proxy class by using the given Entity class objs)
=> <ID> @Id property type i.e the property type that points to PK colum of the db table
(note: @Id Property that is mapped with PK colum of db table will be used criteria column to perform all single row CURD operation)
@Id property in entity class will be mapped with pk column of db table



All sub modules repositories like spring data jpa, springdata mogong db, spring data cassendra and etc.. will extends Repository<T,ID> (I) directly or indirectly... so they become repositories and ready to generate InMemory Proxy classes.



since the repository interfaces of both SQL and NO SQL Db s/ws are extending from the same Repository<T,ID> , So we can say we use interact with both SQL and NO SQL Db s/ws in same style.

Dec 21 Spring Data JPA First App

Procedure to develop spring boot data jpa app

==========

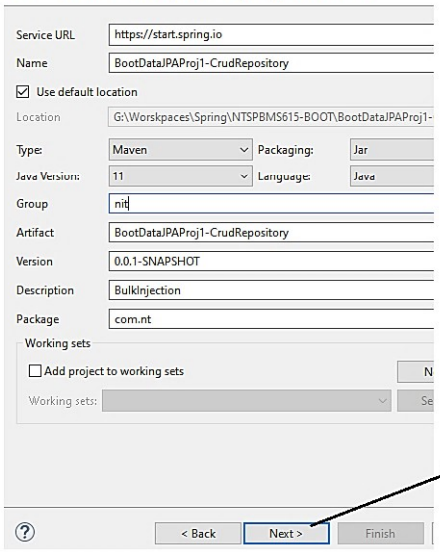
step1) create spring stater project adding the following libraries

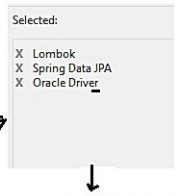
a) spring data jpa b) oracle driver c) lombok

this jar gives

hibernate jars, spring orm jars, hikari cp jars spring data jpa jars

File --> new ----> Project---> spring starter --->





next ----> finish

SQL Dialects in Hibernate

=>For connecting any hibernate application with the database, you must specify the SQL dialects.

=> There are many Dialects classes defined for RDBMS in the org.hibernate.dialect package.

RDBMS	Dialect
Oracle (any version)	org.hibernate.dialect.OracleDialect
Oracle9i	org.hibernate.dialect.Oracle9iDialect
Oracle10g	org.hibernate.dialect.Oracle10gDialect
MySQL	org.hibernate.dialect.MySQLDialect
MySQL with InnoDB	org.hibernate.dialect.MySQLInnoDBDialect
MySQL with MyISAM	org.hibernate.dialect.MySQLMyISAMDialect
DB2	org.hibernate.dialect.DB2Dialect
DB2 AS/400	org.hibernate.dialect.DB2400Dialect
DB2 OS390	org.hibernate.dialect.DB2390Dialect
Microsoft SQL Server	org.hibernate.dialect.SQLServerDialect
Sybase	org.hibernate.dialect.SybaseDialect
Sybase Anywhere	org.hibernate.dialect.SybaseAnywhereDialect
PostgreSQL	org.hibernate.dialect.PostgreSQLDialect
SAP DB	org.hibernate.dialect.SAPDBDialect
Informix	org.hibernate.dialect.InformixDialect
HypersonicSQL	org.hibernate.dialect.HSQLDialect
Ingres	org.hibernate.dialect.IngresDialect

step2) place DataSource cfg , jpa-hiberrnate cfg properties in application.properties

application.properties

```
#DataSource cfg
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

# JPA-hibernate cfgs
spring.jpa.database-platform=org.hibernate.dialect.Oracle10gDialect
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
```

Based in these details the HikariCp DataSource object will be created pointing jdbc con pool for oracle as part Autoconfiguration activities

optional to place , but recomand to place them

spring.jpa.database-platform=org.hibernate.dialect.Oracle10gDialect

=>Dialect is the comp of hiberante that generate SQL Queries internally based on the objects based persitence instructions that we are providing

=>This dialect class name optional to specify but recomand to place to give

spring.jpa.show-sql=true

=> hibernate internally generates so many SQL queries using the specified SQL Dialect .. To see those generated queries as log messages , we need to use the above property.

spring.jpa.hibernate.ddl-auto=update

=> Specifies various activties related to Dynamic schema generation, validation

The possible values are

validate , update, create , create-drop

validate :: checks wheather db table is avaiable according the details specified in the Entity classes or not , if not exception will be raised

create :: Always create new db table based on the given Entity class information if the db table is already avaiable then it deletes the db table and creates the new table

(note:: The class that is mapped with db table is called Entity calss)

update :: (best)
=>creates the new db table ,if the db table is not avaiable
=>uses the existing db table , if db table is avaiable according to Entity class
=> alters the existing db table(only by adding cols) , if db table needs the modification

create-drop:: ==>creates the db table at the begining the app (if necessary drops and creates the db table) and drop the db table at end of app .. useful in Testing env.. , UAT env..

Dec 21 Spring Data JPA First App

step3) Develop the Entity class to map with db table and its cols.

Movie.java

```
package com.nt.entity;

import java.io.Serializable;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

import lombok.Data;

@Data
@Entity
@Table(name="SP_DATA_MOVIE")
public class Movie implements Serializable {
      @Id
      @Column(name="MID")
      private Integer mid;
      @Column(name="MNAME",length = 20)
      private String mname;
      @Column(name="YEAR",length = 20)
      private String year;
      @Column(name="RATING")
      private Float rating;
}
```

=> The java bean class that is mapped with db table is called Entity class
=> @Table maps entity class with specified db table name .. if @Table is not placed then entity class will be mapped with same db table that is matching with Entity class name
=> @Column maps the property of entity class with db table column . If not specified then it maps same colum whose name is same as property

=> If spring boot data jpa is creating db table and cols dynamically then it creates it by using db table name and col names specified in @Table ,@Column annotaitons.. if those annotation not specified the db table and cols will be created with the entity class name and property names.

=>@Id annotation makes the property as Identity Property whose value or mapped col value (generally PK col)will be used as the criteria value while perfomr singler row update,delete and select opeations..
=> By Cfg the generators for @Id property we can generate the value for id property dynamically while inserting the record.

```
@Id
@Column(name="MID")
@GeneratedValue(strategy = GenerationType.AUTO)
private Integer mid;
```

uses hibernate_sequence in oracle Db s/w to generate the id value
=> the logic of hibernate_sequece is start with 1 increment by 1

=>In Every Entity we must one or another property as @Id property otherwise exception will come..we generally take the property that will be mapped with PK column as the @Id property.
=>hiberante_sequence will be created if it is not alreary avaiable otherwise existing will be used.

=>The JPA AUTO Genertor uses sequece generator intenrally incase of oracle and uses IDENTITY Generator (auto increment) in case of mysql ..

note: Using @Column annotation, we can control the length of only string type columns .. not for other type columns.

The followings are basic JPA Annotations for o-r mapping
@Entity (mandatory)
@Table (optional)
@Column (optional)
@Id (mandatory)

```
@Data
@Entity
@Table(name="SP_DATA_MOVIE")
public class Movie implements Serializable { }
```

if entity class name and DB table names are matched then placing @Table is optional, but Recommended to place (if table is not available it creates dnamically)

```
@Id
@Column(name="MID")
@GeneratedValue(strategy = GenerationType.AUTO)
private Integer mid;
```

if @Id is placed then on the property then that property will become as PK in the DB table (Nothing But Idenetity Property)

```
@Id
@Column(name = "id")
@GeneratedValue
private Integer id;
```

If You take like this means no Generator is configured..so,the default Generator is AUTO

=> writing Strategy is Optional in the generators

```
@Column(name="MNAME",length = 20)
private String mname;
```

if column name of DB Table and property name of Entity class same Placing Column is Optional, But recomended to place

STep3) Develop Repsitory interface for Entity class extending CrudRepository(I) specify Entity class name and Id type

IMovieRepo .java

```
package com.nt.repository;

import org.springframework.data.repository.CrudRepository;

import com.nt.entity.Movie;

public interface IMovieRepo extends CrudRepository<Movie,Integer> {

}
```

Entity class — @Id property type

step4) Develop Service Interface and Service Impl class having logic to save object (inserting record)

```
//service Interface
package com.nt.service;

import com.nt.entity.Movie;

public interface IMovieMgmtService {
   public String registerMovive(Movie movie);
}
```

```
// service Impl class
package com.nt.service;

import java.util.Arrays;

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

import com.nt.entity.Movie;
import com.nt.repository.IMovieRepo;

@Service("movieService")
public class MovieMgmtServiceImpl implements IMovieMgmtService {
      @Autowired   //injects the dynamically generated proxy class object
      private IMovieRepo movieRepo; //HAS-A proeprty

      @Override
      public String registerMovive(Movie movie) {
            System.out.println("!nMemory Proxu class name::"+movieRepo.getClass()+"....."+Arrays.toString(movieRepo.getClass().getInterfaces()));
              //use repo
            System.out.println("bfore saving::"+movie);
              Movie movie1=movieRepo.save(movie);
              System.out.println("after saving::"+movie1);

            return " Move is registered with the Id Value::"+movie1.getMid();
      }

}
```

```
Movie movie1=movieRepo.save(movie);
```

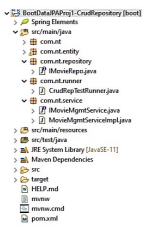
this method internally collecs given object data and generates jdbc code + INSERT Query to insert the object data as record and returns another object of Entity class having generated Id value representing the record that that is inserted.

methods of CrudRepository(I)

Modifier and Type	Method and Description
long	count() Returns the number of entities available.
void	delete(T entity) Deletes a given entity.
void	deleteAll() Deletes all entities managed by the repository.
void	deleteAll(Iterable<? extends T> entities) Deletes the given entities.
void	deleteAllById(Iterable<? extends ID> ids) Deletes all instances of the type T with the given IDs.
void	deleteById(ID id) Deletes the entity with the given id.
boolean	existsById(ID id) Returns whether an entity with the given id exists.
Iterable<T>	findAll() Returns all instances of the type.
Iterable<T>	findAllById(Iterable<ID> ids) Returns all instances of the type T with the given IDs.
Optional<T>	findById(ID id) Retrieves an entity by its id.
<S extends T> S	save(S entity) Saves a given entity.
<S extends T> Iterable<S>	saveAll(Iterable<S> entities) Saves all given entities.

step5) Develope Runner class having logic to call service class methods in run(-) method

```
//Runner class
package com.nt.runner;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

import com.nt.entity.Movie;
import com.nt.service.IMovieMgmtService;

@Component
public class CrudRepTestRunner implements CommandLineRunner {
      @Autowired
      private IMovieMgmtService service;

      @Override
      public void run(String... args) throws Exception {
          //invoke sevice method
            Movie movie=new Movie();
            movie.setMname("RRR");
            movie.setRating(4.5f);
            movie.setYear("2022");
            try {
                  System.out.println(service.registerMovie(movie));
            }
            catch(Exception e) {
                  e.printStackTrace();
            }
      }
```



Serialization

=> The Process of converting java object data into bits and bytes is called Serizaltion.. These bits and bytes can be sent over the network and can be written to file
=> We can send only Serializable object's data either to file or over the network
=> To make the java object as Serilizable object the class of that object must implement java.io.Serializable(I) (Marker interface -Empty interface)

=> The JVM makes the object of class as Serilizable object by seeing the impl of java.io.Serilizable(I) .. In that process JVM becomes ready to conver tSeriable object's into bits and bytes.

=>The marker interface (empty interface) Does not do any thing on this own directly.. by Seeing that marker interface implemetation the JVM will create object as the special object having special behaviour.

```
public class Emp1 {
  private int eno;
  private String ename;
  ...
  ..
}
```

```
Emp1 e1=new Emp1(101,"raja");
(Normal object becoz
 Emp1 class is not implenting any
 Marker interface)
```

e1 → 101 raja

```
public class Emp2   implements java.io.Serilizable{
  private int eno;
  private String ename;
  ...
  ..
}
```

```
Emp2 e2=new Emp1(101,"raja");
```

e2 → 101 raja

[Special obj (Serilizable obj) becoz the class is implementing the marker interface calls java.io.Serilziable.So this object data can be written to file or can be sent over the network in the for bits and bytes.

=>The interface that makes the undlerying JVM/Server/Container to provide special run time capabilities to impl class object is called marker interface/Tag interface

eg:: java.io.Serilziable (I)
java.lang.Cloneable(I)
java.rmi.Remote (I)
javax.servlet.SingleThreadModel(I)

These are empty interface.

Dec 22 Spring Data JPA First App Flow

Flow of execution with respect spring data jpa application

=>we run the spring boot App as java App or Spring Boot App ===> main class (@SpringBootApplication class) main(-) executes ===> In that method SpringApplication.run(-) method executes ===> this run(-) creates IOC container -->locates and reads application.propperties file content into Environment object automatically ==> Based on the starters(jar files) and the the inputs given in application.propeties the following objects will be created through AutoConfiguration Process ..

=>HikariDataSource object pointing jdbc con pool for oracle
=>Hibernate SessionFactory obj injected with dataSource obj and other hibernate properites like dialect, show_sql and etc.
=>Generates InMemory Proxy class Implmenting our IMovieRepo<Movie,Integer>(I) becoz this interface extending from standard Repository(I) of spring data jpa

InMemory Proxy class

```
@Repository
public class MovieRepo_Proxy3 implements IMovieRepo, .....{
    @Autowired
    private LocalSessionFactory sesfact;
            (injects HB's SessionFactory obj)

  //total 12 methods Inherited from CrudRepository will implemented here
    ....
    ... //having hiberate style persistence logic
    ...

    @Transactional
        @Override                (5)
        public <S extends T> S save(S entity) {

            Assert.notNull(entity, "Entity must not be null.");   --> As Log messages

            if (entityInformation.isNew(entity)) {
                em.persist(entity);   (6)   --> just inserts the record
                return entity;   (7)
            } else {
                return em.merge(entity);   --> updates the record
            }                                  if the record is already avaiable
        }
  }//class
```

note:-
it create InMemoryProxyClass only if the user defined Interface which extends from Repository(I) Interface

=> The singleton scope spring beans of root pkg(com.nt)and its sub pkgs will be pre- instantiatied (Service Impl class , runner class and etc.) ==> In this process @Autowired based Injection also takes place like

InMemory proxy class obj to Service Impl class obj
Service Impl class obj to runner class obj

=> All these singleton scope spring class objects (user-defined and came through autoconfiguration) will be kept in the Internal cache of IOC container for reusability

Internal cache of IOC container

movieService	MovieMgmtServiceImpl class obj ref
movieRepo_Proxy3	MovieRepo_Proxy3 class obj ref
...	....
...	....

=> The run(-) of Runner class will be executed automatically where service method class is called.

Runner class

```
@Component
public class CrudRepTestRunner implements CommandLineRunner {

@Autowired
private IMovieMgmtService service;

@Override        (1)
public void run(String... args) throws Exception {
   //invoke sevice method
        Movie movie=new Movie();
        movie.setMname("RRR");
        movie.setRating(4.5f);
        movie.setYear("2022");
        try {          (10)
            System.out.println(service.registerMovie(movie));   (2)
        }
        catch(Exception e) {
            e.printStackTrace();
        }

    }
}
```

Service Impl class

```
@Service("movieService")
public class MovieMgmtServiceImpl implements IMovieMgmtService {
@Autowired   //injects the dynamically generated proxy class object
private IMovieRepo movieRepo; //HAS-A proeprty


@Override          (3)
public String registerMovie(Movie movie) {
  System.out.println("InMemory Proxu class name::"+movieRepo.getClass()+"..
        ......."+Arrays.toString(movieRepo.getClass().getInterfaces()));
        //use repo
      System.out.println("bfore saving::"+movie);
   (8) Movie movie1=movieRepo.save(movie);   (4)
      System.out.println("after saving::"+movie1);

    return " Move is registered with the Id Value::"+movie1.getMid();
                                        (9)
}
}
```

long	count() Returns the number of entities available.

=>Gives count of objects or records.. by internally executing the readymade HQL/JPQL SELECT Query with count(*) aggrate function..

=>SQL Queries are DB s/w dependent queries and they will be written using db table names and col names
=>HQL/JPQL Queries are DB s/w independent queries and they will be written using Entiyt class names and their property names

HQL :: Hibernate Query Language (old name)
JPQL :: Java Persistence Query Language (new name)

note:: With the support of dialect the HQL/JPQL Queries will be converted into SQL Queries internally before sending them to DB s/w using JDBC code support..

Modifier and Type	Method and Description
long	count() Returns the number of entities available.

code in serivce interface

```
public long fetchMoviesCount();
```

code in service Impl class

```
@Override
    public long fetchMoviesCount() {
        return movieRepo.count();
    }
```

Code in run(-) of Runner class

```
System.out.println("----------------------------------------");
try {

System.out.println(" count::"+service.fetchMoviesCount());

}
catch(Exception e) {
    e.printStackTrace();
}
```

Modifier and Type	Method and Description
boolean	existsById(ID id) Returns whether an entity with the given id exists.

service interface

```
public boolean checkMovieById(Integer mid);
```

service Impl class

```
@Override
public boolean checkMovieById(Integer mid) {
    return movieRepo.existsById(mid);
}
```

code run(-) method of Runner class

```
System.out.println("----------------------------------------");
try {
System.out.println("Is availble ?"+service.checkMovieById(1));
}
catch(Exception e) {
    e.printStackTrace();
}
```

Dec 23 Spring Data JPA CrudRepository -Select

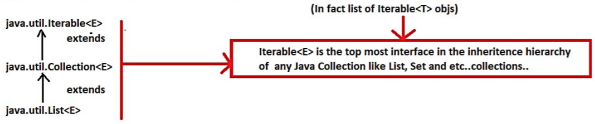
Select operations in CrudRepository

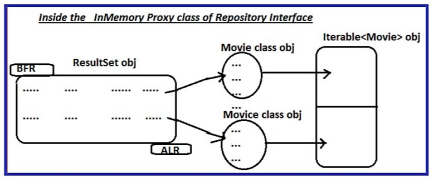
`long`	`count()` ✓ Returns the number of entities available.
`Iterable<T>`	`findAll()` Returns all instances of the type.
`Iterable<T>`	`findAllById(Iterable<ID> ids)` Returns all instances of the type `T` with the given IDs.
`Optional<T>`	`findById(ID id)` Retrieves an entity by its id.
`boolean`	`existsById(ID id)` ✓ Returns whether an entity with the given id exists.

`Iterable<T>`	`findAll()` Returns all instances of the type.

=>Gets all the records of db table by generating and execute SELECT Query with out applying any conditions and converts genrerated records of ResultSet into List of Model/Entity class objects.

(In fact list of Iterable<T> objs)





In service interface

```
public Iterable<Movie> fetchAllMovies();
```

In service Impl class

```
@Override
    public Iterable<Movie> fetchAllMovies() {
        return movieRepo.findAll();
    }
```

```
select * from sp_data_moive
```

//In runner class

```
System.out.println("--------for findAll()--method--------------");
try {
 Iterable<Movie> list=service.fetchAllMovies();
 for(Movie movie:list) { //enahaced for loop
     System.out.println(movie);
 }
 System.out.println(".........");
 list.forEach(movie->{  //forEach() method
     System.out.println(movie);
 });
 System.out.println(".........");
 list.forEach(movie-> System.out.println(movie)); //forEach() method
 System.out.println(".........");
 list.forEach(System.out::println); // forEach method + static method reference
 System.out.println(".........");
 Arrays.asList(list).stream().forEach(System.out::println); //forEach  method +streaming api+ static method reference

}
catch(Exception e) {
 e.printStackTrace();
}
```

note:: All methods of spring data jpa repositories throws unchecked exceptions so the exception propagration across the multiple layeres takes place automatically though we do not catch , handle the exception having exeption rethrowing..

`Iterable<T>`	`findAllById(Iterable<ID> ids)` Returns all instances of the type `T` with the given IDs.

In service Interface

```
public Iterable<Movie> fetchAllMoviesByIds(List<Integer> mids);
```

In service Impl class

```
@Override
    public Iterable<Movie> fetchAllMoviesByIds(List<Integer> mids) {
        return movieRepo.findAllById(mids);
    }
```

```
select * from sp_data_moive where mid in(?,?,...)
```

In runner class

```
System.out.println("--------for findAllById(-)--method--------------");
        try {
            List<Integer> idsList=new ArrayList();
            idsList.add(189); idsList.add(1);
            System.out.println("movices are ::"+service.fetchAllMoviesByIds(idsList));
            System.out.println("-----------------------------------");
          System.out.println("movies are ::"+service.fetchAllMoviesByIds(List.of(189,1)));  //java 9
          System.out.println("-----------------------------------");
          System.out.println("movies are ::"+service.fetchAllMoviesByIds(Arrays.asList(189,1)));
        }
        catch(Exception e) {
            e.printStackTrace();
        }
```

List.of(...) , Set.of(...), Map.of(...) and etc.. methods given from java 9

these are static factory methods returning Immutable (non-modifible) List,Set,Map Collections

`Optional<T>`	`findById(ID id)` Retrieves an entity by its id.

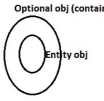
=> Returns Optional<T> obj having Enity obejct by selecting single record from db table based given id value.

=>Optional<T> object contains other object as container object and helps to check to wheather object has come or not .. if not come we can raise the exception to throw.. using multiple methods that are avaiable in Optional API

Optional API (Java8 feature)

```
Optional<Movie> opt=movieRepo.findById(id);
```

Optional obj (container obi)



To check object has come or not to Optional obj
opt.isPresent() (return true/false)
To get object from to Optional obj
Movie movice=opt.get()
e to
To throw exception if object has not com optional obj

```
opt.orElseThrows(...)
```

To return other object of object has not come to optional obj

```
opt.orElse(...)
```

=> Use the method return type as Optional<T> (optional api) if there is possiblity of mehtod returing no value or null for failed condition.. then we can return Optional<T> obj emptiness ..so the recivening need not to check null value and can avoid NullPointerException.. we can check the emptyness either using isEmpty() or isPresent() methods.

In service Interface

```
public Movie fetchMovieById(Integer mid);
```

In service Impl class

```
@Override
    public Movie fetchMovieById(Integer mid) {
      Optional<Movie> opt=movieRepo.findById(mid);
      if(opt.isPresent())
          return opt.get();
      else
          throw new IllegalArgumentException(" Record not found");
    }
```

In Client App

```
System.out.println("--------for findById(-)--method--------------");
        try {
            System.out.println("18 move is::"+service.fetchMovieById(18));
        }
        catch(Exception e) {
            e.printStackTrace();
            System.out.println(e.getMessage());
        }
```

Dec 24 Spring Data JPA CrudRepository -Select,upate,save Operations

findById(-) of CrudRepository

=> Using optional's api various methods we can return inside object if avaiable othewise we can throw exception in a singleline statement..

using orElseThrow(-) method

In service Interface

```
public Movie fetchMovieById(Integer mid);
```

In service Impl class

```
@Override
public Movie fetchMovieById(Integer mid) {
    return movieRepo.findById(mid).orElseThrow(()-> new IllegalArgumentException("record not found"));
}
```

if record is avaliable returns Entity object otherwise throws IllegalArgumentException

Note:- orElseThrow(-) method is very popular in RealTime application i.e., it is mostly used method in RealTime Applications.

In runner class

```
System.out.println("--------for findById(-)--method----------");
try {
    System.out.println("18 move is::"+service.fetchMovieById(18));
}
catch(Exception e) {
    e.printStackTrace();
    System.out.println(e.getMessage());
}
```

Example on orElse(-) mehtod of Optional API

Note:- orElseThrow(-) method is very popular in RealTime application i.e., it is mostly used method in RealTime Applications.

service Interface

```
public Movie fetchMovieById(Integer mid);
```

Service Impl class

```
@Override
public Movie fetchMovieById(Integer mid) {
    return movieRepo.findById(mid).orElse(new Movie());  // if record is avaiable returns Entiy obj with data
                                                         // otherwise empty entity object
}
```

here the new Movie() will not be inserted to db table it is just shows to the mathod caller that details are just null(not available) instead of giving NullPointerException

In client app

```
System.out.println("--------for findById(-)--method----------");
try {
    System.out.println("189 move is::"+service.fetchMovieById(189));
}
catch(Exception e) {
    e.printStackTrace();
    System.out.println(e.getMessage());
}
```

Returning Optional<T> from service class method

In service interface

```
public Optional<Movie> gatherMovieById(Integer mid);
```

In service Impl class

```
@Override
public Optional<Movie> gatherMovieById(Integer mid) {
    Optional<Movie> opt=movieRepo.findById(mid);
    return opt.isEmpty() ?Optional.empty() :opt;
}
```

In Client app

```
System.out.println("--------for findById(-)--method----------");
try {
    Optional<?> opt=service.gatherMovieById(18);
    if(!opt.isEmpty())
        System.out.println("189 movie:"+opt.get());
    else
        System.out.println("record not found");
}
catch(Exception e) {
    e.printStackTrace();
    System.out.println(e.getMessage());
}
```

---

saveAll(-) method

<S extends T> Iterable<S>	saveAll(Iterable<S> entities)
	Saves all given entities.

Parameters:
entities - must not be null nor must it contain null.

Returns:
the saved entities; will never be null. The returned Iterable will have the same size as the Iterable passed as an argument.

Throws:
IllegalArgumentException - in case the given entities or one of its entities is null.

It performs batch insertion of records /objects by generating multiple insert SQL Queries

usecase:: Group Ticket reservations.

service Interface

```
public String groupMovieRegistration(List<Movie> moviesList);
```

service Impl class

```
@Override
public String groupMovieRegistration(List<Movie> moviesList) {
    Iterable<Movie> savedMovies=movieRepo.saveAll(moviesList);
    List<Integer> listIds=new ArrayList();
    if(savedMovies!=null && ((List<Movie>) savedMovies).size()>0) {
        savedMovies.forEach(m->{
            listIds.add(m.getMid());
        });
    }//if
    return listIds.toString()+" movies are saved";
}//method
```

In runner class

```
System.out.println("--------for saveAll(-)--method----------");
try {
    String result=service.groupMovieRegistration(List.of(new Movie("RRR","2022",5.6f),
                                                          new Movie("83","2021",4.6f),
                                                          new Movie("puspha","2021",4.0f)
                                                         )); //java9 feature
    System.out.println(result);
}
catch(Exception e) {
    e.printStackTrace();
}
```

for this you've to take three constructor annotations are
@AllArgsConstructor
@NoArgsConstructor
@RequiredArgsConstructor
with the combination of @Non-null annotation for properties

---

what is the diffence beween spring data jpa persist(-) and hiberate api persit(-)?

Ans) Hibernate's(HB) persist(-) method can not work with generators to generate the id values dynamically where as spring data jpa's persist(-) can work with generators..
(used by save(-) method)
both they are given to save objects(inserting record) .. but HB's persist(-) return type void i.e does not return any thing. spring data jpa's save(-) intenrally uses JPA's persist(-) method returns saved Entity object having the generated id value.

update object operation using CrudRepository

Dec 28 Spring Data JPA CrudRepository -Select,upate,save Operations

=>There is no update(-) method in CrudRepository.. becoz the save(-) itself performs save object operation and update object operation (it internally uses persist(-) method for save object and merge(-) for update object operation)

<S extends T> S save(S entity)

=>Saves a given entity. Use the returned instance for further operations as the save operation might have changed the entity instance completely.

Parameters: entity - must not be null.

Returns: the saved entity; will never be null.

Throws: IllegalArgumentException - in case the given entity is null.

=>if given entity obj's id value is already avaiable then save(-) interanlly uses merge(-) to perform update object operation.

=>if given entity obj's id value is not already avaiable then save(-) interanlly uses persit(-) to perform save object operation

---

In Service Interface

```
public String updateMovieDetails(Integer mid, String year, Float newRating );   | For partial update of the record
public String updateMovie(Movie movie);                                          | For full object updation
```

Impl class

```
@Override
public String updateMovieDetails(Integer mid,String newYear, Float newRating) {
    Optional<Movie> opt=movieRepo.findById(mid);
    if(opt.isPresent()) {
        Movie movie=opt.get();
    movie.setYear(newYear);
    movie.setRating(newRating);
    movieRepo.save(movie);  //always calls  merge(-) to perform update operation
    return mid+" movie id movie is updated";
    }
    else                                                          For partial update..
        return " Movie Id movie is not avaiable";

}
```

(option1)

For bulk update/select/delete operatin or singlerow upate/select/delete operation with our choice type and no.of conditions we need to use other options of Spring Data JPA like findBy methods, HQL/JPQL Queries , Native SQL queries and etc..

```
@Override
public String updateMovie(Movie movie) {
        Optional<Movie> opt=movieRepo.findById(movie.getMid());
        if(opt.isPresent()) {
          movieRepo.save(movie);
          return "movie is updated";
        }
else
    return "movid is not found to update";

    }
```

(option2)

obj

For complete update

if u r not sure record will be avaiable or not

```
public String updateMovie(Movie movie) {
    movieRepo.save(movie);
    return " movied updated";
    }
```

(option3)

For full object update

if u r sure record is defintely avaiable.

obj

Partial update screen

movie id	189	(End user only should enter movie id)
new year	2022	
new rating	3.0	

use option1

Full object update screen

movie id	189	(End user only should enter movie id)
new name ::	RRR3	
new year	2023	
new rating	3.0	

use option 2

note:: other than movie id .. if any value is not given in the text boxes then NULL values will be inserted..

report screen

101	RRR	2022	4.5*	edit
102	PUSPA	2021	5*	edit
103	83	2021	4*	edit

Full object update screen

movie id	189	(End user only should enter movie id)
new name ::	RRR3	
new year	2023	
new rating	3.0	

use option3

Delete object operations using CrudRepository

void	delete(T entity) Deletes a given entity.
void	deleteAll() Deletes all entities managed by the repository.
void	deleteAll(Iterable<? extends T> entities) Deletes the given entities.
void	deleteAllById(Iterable<? extends ID> ids) Deletes all instances of the type T with the given IDs.
void	deleteById(ID id) Deletes the entity with the given id.

Examples on deleteById(-) and delete(-) methods to perform single object deletion

In service In interface

```
public String removeMovieById(Integer mid);
public String removeMovie(Movie movie);
```

In service impl class

```
@Override
public String removeMovieById(Integer mid) {
   Movie movie=movieRepo.findById(mid).orElseThrow(()->new IllegalArgumentException("movie not found"));
    movieRepo.delete(movie);
   return "record deleted";
}
```

option1

```
public String removeMovie(Movie movie) {
    Optional<Movie> opt=movieRepo.findById(movie.getMid());
    if(opt.isPresent()) {
        movieRepo.delete(movie);
        return "Movie deleted";
    }
    else {
        return "movie not found";
    }
}
```

option2

```
@Override
public String removeMovie(Movie movie) {
        movieRepo.delete(movie);
        return "Record deleted";
}
```

option3

In Runner class

```
System.out.println("--------for deleteById(-) for delete object operation--method--------------");
            try {
                System.out.println(service.removeMovieById(189));

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

```
System.out.println("--------for delete(-) for delete object operation--method--------------");
try {
Movie movie=new Movie();
movie.setMid(4);
    System.out.println(service.removeMovie(movie));
}//try
catch(Exception e) {
    e.printStackTrace();
}
```

delete movie screen

movie id	189	(movie id should be entered by enduser)

option1(recomanded) or option2

report screen

101	RRR	2022	4.5*	delete
102	PUSPA	2021	5*	delete
103	83	2021	4*	delete

use option3 (becoz record is definetly available in DB)

---

save(-) :: select Query + insert /update query
saveAll(-) :: multiple insert SQL queries
findById(-) :: Select SQL query having PK column based condition
findByAllId(-) :: Select SQL query having PK column based conditions having IN clause
findAll() :: Select SQL query with out condition
count() :: Select count(*) SQL query with out condition
existsById(-) :: Select SQL query with PK col based condition
deleteById(-) :: Select SQL query + Deleete SQL query with PK col based condition
delete(-) :: Select SQL query + Deleete SQL query with PK col based condition
deleteAll(-) :: multiple delete SQL queries having pk value based conditions to delete all records
deleteAllBId(-) :: multiple delete SQL queries having pk value based conditions based on given ids
deleteAll(-) :: multiple delete SQL queries having pk value based conditions to delete passed objs based records

Dec 29 Spring Data JPA -PagingAndSortingRepo

deleteAll

```
void deleteAll()
```

Deletes all entities managed by the repository.

=> This method generates 1 select query to get all records and their id values .. later generates muliple delete Queries having the id values as the criteria values.

Code in Service Interface

```
public String removeAllMovies();
```

Code in service Impl class

```
@Override
public String removeAllMovies() {
    long count=movieRepo.count();
    if(count!=0)
        movieRepo.deleteAll();
    return count+" no.of records are deleted";
}
```

Code in runner class

```
System.out.println("--------for deleteAll(-) for deleting all objects operation----------");
try {
    System.out.println(service.removeAllMovies());
}
catch(Exception e) {
    e.printStackTrace();
}
```

The generated SQL queries are

```
Hibernate: select count(*) as col_0_0_ from sp_data_movie movie0_  ===> for count()
Hibernate: select movie0_.mid as mid1_0_, movie0_.mname as mname2_0_, movie0_.rating as rating3_0_,
movie0_.year as year4_0_ from sp_data_movie movie0_
Hibernate: delete from sp_data_movie where mid=?
Hibernate: delete from sp_data_movie where mid=?                    =>for deleteAll() method call
Hibernate: delete from sp_data_movie where mid=?
Hibernate: delete from sp_data_movie where mid=?
```

---

deleteAllById

This should not be null and its element values should not be null

```
void deleteAllById(Iterable<? extends ID> ids)
```

=>Deletes all instances of the type T with the given IDs.

Parameters:

ids - must not be null. Must not contain null elements.

Throws:

IllegalArgumentException - in case the given ids or one of its elements is null.

code in service Interface

```
public String removeAllMoviesById(List<Integer> ids);
```

service Impl class

```
@Override
public String removeAllMoviesById(List<Integer> ids) {

    Iterable<Movie> it=movieRepo.findAllById(ids);
    long count= ((List<Movie>) it).size();
    if(ids.size()!=0 && ids.size()==count ) {
        movieRepo.deleteAllById(ids);
    return  ((List<Movie>) it).size()+" no.of records are deleted";
    }
    else
        return "No ids are given to delete or  either all or some ids are not available to delete";
}
```

In runner class

```
System.out.println("--------for deleteAllById(-) for deleting all objects operation----------");
try {
    System.out.println(service.removeAllMoviesById(List.of(11,12)));
}
catch(NullPointerException  npe) {
    System.out.println("Id(s) must not be null");
    npe.printStackTrace();
}
catch(Exception e) {
    e.printStackTrace();
}
```

---

deleteAll *(work out this method as the assignment)*

```
void deleteAll(Iterable<? extends T> entities)
```

Deletes the given entities.

Parameters:

entities - must not be null. Must not contain null elements.

Throws:

IllegalArgumentException - in case the given entities or one of its entities is null.

The service method assingment should return how many records are delete when this method is called.

PagingAndSortingRepository

=>Child Repository of CrudRepository .. having methods to select/retrieve records by applying Sorting and Pagination activities..

=>Sorting takes place in the following order (Ascending)

- *->special chars (?,_,+ and etc..)*
- *->Numbers*
- *->Uppercase Alphabets*
- *->Lowercase Alphabets.*

The code is::

```
public interface PagingAndSortingRepository<T, ID> extends CrudRepository<T, ID> {
    Iterable<T> findAll(Sort sort);
    Page<T> findAll(Pageable pageable);
}
```

Type hierarchy

- Repository<T, ID>
  - CrudRepository<T, ID>
    - PagingAndSortingRepc
      - JpaRepository<T, ID

F4 for type hierarchy

=>ctrl+shift+T :: To search and get source code given class

=>ctrl+o :: To get info about highlighed class/interface/enum

Iterable<T> findAll(Sort sort);

=>Retrives all the records of Db table either in asc or desc order specified through Sort object

Example App

Repository Interface

```
public interface IMovieRepo extends PagingAndSortingRepository<Movie,Integer> {    }
```

servcie Interface

```
// One method can have only one var arg param that to last param of the method
    // Var args are internally arrays..
public Iterable<Movie>  displayMoviesByOrder(boolean asc,String ... properites);
```

SEvice Impl class

```
@Override
public Iterable<Movie> displayMoviesByOrder(boolean asc, String... properites) {
    Sort sort=Sort.by(asc?Direction.ASC:Direction.DESC, properites);
    return movieRepo.findAll(sort);
}
```

In runner class

```
System.out.println("----------findAll(Sort sort)-------");
try {
Iterable<Movie> it=service.displayMoviesByOrder(false,"year","mname");
it.forEach(System.out::println);
}
catch(Exception e) {
    e.printStackTrace();
}
```

Pagination using PagingAndSortingRepository
==========

Dec 30 Spring Data JPA -PagingAndSortingRepo

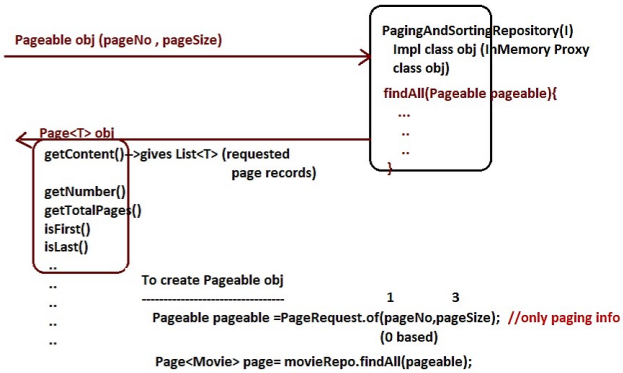
pagination
==========
=> The process of dislaying huge no.of records page by page is called pagination.It is very useful in report generation
eg:: gmail inbox , salect report , product catalogs and etc..

Page<T>	findAll(Pageable pageable) Returns a Page of entities meeting the paging restriction provided in the Pageable object.

=>Takes pageNo(0 based) ,pageSize (no.of recordss in each page) as inputs in the form of Pageable object and returns Page<T> object having multiple details like request page records, total pages count, current page number, total records and etc..

Slice<T> (I)
↑ extends
Page<T> (I)

These two interfaces contain multiple methods to get various details related to pagination like page content, pageno ,total pages and etc..
isFirst(), isLast(), getContent(), getNumber(), getSlice() , getTotalPages() and etc..



To create Pageable obj
----
1 3
Pageable pageable =PageRequest.of(pageNo,pageSize); //only paging info
(0 based)
Page<Movie> page= movieRepo.findAll(pageable);

for example ,if db table is having 20 records .. then we get 7 pages(0-6 pages) like (3,3,3,3,3,3,2 records in pages) and gives 1 page records (indirectly 2 page becoz page no is 0 based) like 4,5,6 records will come.. if ask for 6 page records(indirectly 7 page) then we get 19,20 records (only 2 are there) ..if ask for 7 or above page records then we get no records.

0 1 2 3 4 5 6

note:: we can create Pageable obj even including Sorting and paging information

eg: Pageable pageable=PageRequest.of(pageNo, pageSize ,Sort obj); //paging and sorting info
2 5 Sort.by(Direction.ASC, "mname")

Page<Movie> page= movieRepo.findAll(pageable);

=> First sorting on all records takes place and then pagination takes place

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

In service Interface

```
public Page<Movie> generateReport(int pageNo,int pageSize,boolean asc,String ...props);
```

In service Impl class

```
@Override
public Page<Movie> generateReport(int pageNo, int pageSize, boolean asc, String... props) {
        //create Pageable object
    Pageable pageable=PageRequest.of(pageNo, pageSize,Sort.by(asc?Direction.ASC:Direction.DESC, props));
    // get requested page records
    Page<Movie> page=movieRepo.findAll(pageable);
        return page;
}
```

In runner class

```
System.out.println("----------findAll(Pageable pageable)-------");
            try {
             Page<Movie> page=service.generateReport(2, 3, true, "mname");
             System.out.println("page number::"+page.getNumber());
             System.out.println("pages count::"+page.getTotalPages());
             System.out.println(" is it first page::"+page.isFirst());
             System.out.println(" is it last page::"+page.isLast());
             System.out.println("page Size::"+page.getSize());
             System.out.println("page Elements count::"+page.getNumberOfElements());
             if(!page.isEmpty()) {

               List<Movie> list=page.getContent();
               list.forEach(System.out::println);
             }
             else
                    System.out.println("no page found");
            }//try
            catch(Exception e) {
                   e.printStackTrace();
            }
```

output

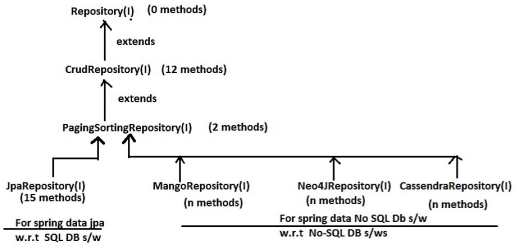
```
page number::2
pages count::3
 is it first page::false
 is it last page::true
page Size::3
page Elements count::2
Movie(mid=13, mname=jersy, year=2022, rating=4.7)
Movie(mid=513, mname=puspa, year=2022, rating=4.1)
```

It internally uses psuedo column "rownum" and sub queries concept a lot to get records through pagination

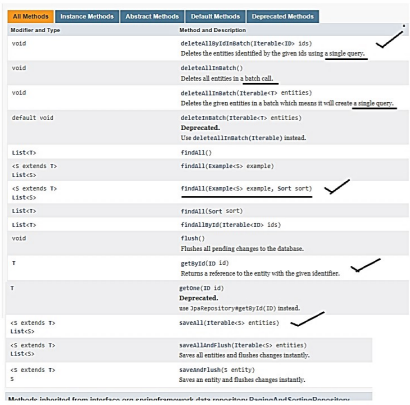
JpaRepository
==========

Dec 31 Spring Data JPA -JPARepository

=> CrudRepository , PagingSortingRepository are Commons Repository for both SQL and NO SQL Db s/ws.. if u use methods these repositories we need not to change code in service class though our app moves from SQL DB s/w to NO SQL Db s/w.



The 15 methods JPARepository(I) are
==========

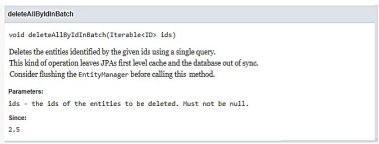


=>Most of the the methods avaiable in JpaRepository are also there in CrudRepository , PagingAndSortingRepository interfaces but they work in undelying JPA Impl (hibernate) style.. Use this JpaRepository methods only when the same methods are not there in CrudRepository , PagingSortingRepository Interfaces..

some differences here
==========

CrudRepository(I) Methods	JpaRepository (I) methods
=>saveAll() , findAll() methods return type Iterable<T> Collection	=>saveAll() , findAll() methods return type is List<T> Collection
=>Does not take Example obj, Sorting obj as the arguments in findXxx() methods	=> findXxx() methods are avaiable taking Example ,Sorting objs
=> findById(-) return type is Optional<T>	=> getById(-) return type is <T>
=> deleteXxx(-) methods performs bulk deletion by generating multiple delete queries	=> deleteXxx(-) methods performs bulk deletion by generating single delete SQL query through BatchProcessing
=> while findById(-) method no need of working with extra cfg in application.properties enabling lazy loading	=> while working with getById(-) method we need an extra cfg in application.properties enabling hibernate lazy loading
=>These method common for both SQL and NO SQL Db s/ws	=> These methods are specific SQL DB s/ws...

note:: JpaRepository is bit known for performing delete operations through batch processing.



example app
==========

Repository Interfaces

```
public interface IMovieRepo extends JpaRepository<Movie,Integer> {  }
```

In service Inteface

```
public String removieMoviesByIds(List<Integer> ids);
```

Service Impl class

```
@Override
public String removieMoviesByIds(List<Integer> ids) {
        List<Movie> list=movieRepo.findAllById(ids);
        long count= list.size();
          movieRepo.deleteAllByIdInBatch(ids);
        return count+" no.of records are deleted";
    }
```

In runner class

```
System.out.println("----------deleteAllByIdInBatch(-)-------");
    try {
      //System.out.println(service.removieMoviesByIds(List.of(513,456)));
          /*List<Integer> ids=new ArrayList();
          ids.add(567);
          ids.add(null); */
          List<Integer> ids=Arrays.asList(13,null);
          System.out.println(service.removieMoviesByIds(ids));
        }
        catch(Exception e) {
            e.printStackTrace();
        }
```

=>List.of(-,-,-,-), Set.of(-,-,-,-) , Map.of(-,-,-) are static factory methods given from Java9 to create Immutable Collection obj given data .. In this process they do not allow to have null values as the element values (Immutable collections can not have null values as the element values)

=> other pratices of creating collection like "using new operator" and using Arrays.asList(-) method and etc.. gives mutable collection where null values can be add as the element values.

What is difference deleteAllById(-) method of CrurdRepository and deleteAllByIdInBatch() method of JpaRepository

deleteAllById(-) in CrudRepository	deleteAllByIdInBatch(-) in JpaRepository
(a) delete bulk records by generating multiple delete queries (no batch processing)	(a) delete bulk records by generating single delete query with IN clause conditio (batch processing takes place)
(b) Gives ids for for deletion must not null otherwise throws IllegalStateException	b) Given ids for deletion can be null
(c) Given all ids based records must be present in Db table otherwise exception will raise	(c) Not mandatory to have all records availability in Db table
(d) Working type internally is indirecIty "and" clause	(e) IN clause based condition nothing but "or" condtion.
(f) Works in both SQL And NoSQL DB s/ws	(f) only for SQL Db s/ws.

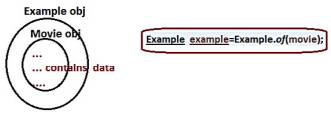
Jan 01 Spring Data JPA -JPARepository

JpaRepository methods

```
<S extends T>                    findAll(Example<S> example, Sort sort)
List<S>
```

Returns reords as List<T> objects by taking non-null values of given Example objs based Entity object and sorts the selected records accorring to sort object info. uses only non-null values of given entity obj as the criteria values *with 'and' clause conditions*

=>Example obj is Jpa supplied object containing other obejct .. It is like Optional object of Java8 ..



In service Interface

```
public List<Movie> searchMoviesByMovie(Movie movie, boolean asc, String ...props);
```

In service Impl class

```
@Override
    public List<Movie> searchMoviesByMovie(Movie movie, boolean asc, String... props) {
        Example example=Example.of(movie);
        Sort sort=Sort.by(asc?Direction.ASC:Direction.DESC,props);
        List<Movie> list=movieRepo.findAll(example, sort);
        return list;
    }
```

In Runner class

```
System.out.println("----------findAll(Example,Sort)-------");
    try {
        Movie movie=new Movie();
        //movie.setMid(100); movie.setMname("RRR");
        movie.setRating(4.1f); movie.setYear("2022"); // specify your selections for select from DB s/w
        List<Movie> list=service.searchMoviesByMovie(movie, true, "mname");
        list.forEach(System.out::println);
    }//try
    catch(Exception e) {
        e.printStackTrace();
    }
```

specify this on which basis you are going to sort your selected results

What is the difference among findAll() methods of CrurdReposiotry , PagingAndSortingRepository and JpaRepository

method	Return type	Sorting ability	Pagination Ability	Taking Example obj	used in SQL andNoSQL Db s/w
findAll() in CrudRepository	Iterable<T>	no	no	no	yes
findAll() in PASRepository	Iterable<T>	yes	yes	no	yes
findAll() in JpaRepository	List<T>	yes	no	yes	no

getById(-) (alternate to getOne(-) method of JpaRepsitory)

=>This method very similart to findById(-) of CurdRepository with few minor chanages

T	getById(ID id) Returns a reference to the entity with the given identifier.
T	getOne(ID id) Deprecated. use JpaRepository#getById(ID) instead.

getById

```
T getById(ID id)
```

Returns a reference to the entity with the given identifier.

Depending on how the JPA persistence provider is implemented this is very likely to always return an instance and throw an `EntityNotFoundException` on first access. Some of them will reject invalid identifiers immediately.

Parameters:
`id` - must not be null.
Returns:
a reference to the entity with the given identifier.

example

In service Interface

```
public Movie searchMovieById(Integer id);
```

In service Impl class

```
@Override
    public Movie searchMovieById(Integer id) {
        Movie movie=movieRepo.getById(id);
        return movie;
    }
```

In client App

```
System.out.println("----------getById(Integer id)-------");
    try {
        System.out.println("416 Movie Details are "+service.searchMovieById(419));
    }//try
    catch(Exception e) {
        e.printStackTrace();
    }
```

In application.properties

```
#To enable lazy loading of record in the underlying Hibernate f/w
spring.jpa.properties.hibernate.enable_lazy_load_no_trans=true
```

For getOne(-) ,getById(-) methods this property to make underlying Hibernate f/w to support lazy loading even though Transactional env.. is not taken.. In old versions spring data jpa just @Transacationl in service is class sufficient.

What is the difference getById(-) of JpaRepository and findById(-) of CrudRepository?

getById(-)	*findById(-)*
(a) retuns Entity obj ref representing record that is selected for given id	(a) returns Optional<T> obj having Entity obj for the same
(b) performs Lazy Loading of record/object i.e first returns proxy object and when that proxy is used then record will be retrieved from DB s/w to put into Entity class obj	(b) perform eager loading i.e gets the record/object DB table directly with out involving any proxy object
(c) if record not found we can not throw custom exception .. it gives the fixed EntityNotFoundException	(c) with the support of Optional API we can throw Custom Exception..
(d) Works only in SQL Db s/w	(d) Works in both SQL and NO SQL DB s/w
(e) Impl depends on on undelrying Hibernate framework	(e) Impl is given in spring dataJpa itself
(f) addtional property cfg in in applicaiton.properties is reuqired	(f) not required.
(g) if record not found we get exception (EntityNotFoundException)	(g) if record not found, we get Empty Optional object for which we send custom message or we can throw exception

While working with spring Data JPA choose persistence methods in the following order

(a) CrurdRepository methods
==>if not sufficient then
(b) PagingAndSortingRepository methods
==>if not sufficient then
(c) JpaRepository methods
==> if not sufficient then
(d) Custom methods in our Repository Interface.
(i) custom finder methods (only for select opertions)
(ii) @Query methods (for HQL/JPQL and Native SQL Queries custom based Select operations)
(iii) @Query + @Modifying methods (for HQL/JPQL and Native SQL Queries custom non-Select operations)

(OR)

Different ways of writing persisstence logics in spring data jpa
a) using pre-definedRepository methods
i)CrudRepository ii) PagingAndSortingRepository iii) JpaRepository
b) finder methods in our repository (select operations)
c) @Query methods (To use JPQL, SQLQueries for select operations)
d) @Query + @Modifying methods (for Non-select operations)

Jan 05 Spring Data JPA -finder Methods

finder methods in spring data jpa
==========

=>These are abstract method declarations done in our repository by following certain naming convetions
=>Using these we can perform only Select Opeations and Non-Select Operations are not possible
=> These finder methods will be implemented in the InMemory Proxy class of Repository Interface generating Select SQL queries internally ...
=> The select operations which are not possible with findXxx() methods of differnt prefined Repositories can be developed using using these custom finder methods..

syntax:: fixed
<public> <RT> findBy<propertyName(s)><Condition(s)>(<params> ....)

=>These finder methods can be used for
a) Entity operations (selecting all the col values of db table)
eg: select * from Movie (or) select mid,name ,ratings,year from Movie where year in(?,?,?)

b) Scalar opeations/Projections (selecting specific single col or multiple col values of Db table)
eg: select mid,name from Movie where mid>=? and mid<=?
eg: select mid,year from Movie where mid>=? and mid<=?
eg: select name,year from Movie where mid>=? and mid<=?

Example on Entity Queries (select queries giving 0 or more records by selecting all col values)

In Repsoitory interface

```
public interface IMovieRepo extends JpaRepository<Movie, Integer> {

    //select mid,mname,year,ratings from Movie where mname=?
    public List<Movie> findByMnameEquals(String name);
    //select mid,mname,year,ratings from Movie where mname=?
    public List<Movie> findByMnameIs(String name);
    //select mid,mname,year,ratings from Movie where mname=?
    public List<Movie> findBymname (String name);

}
```

All the 3 methods are same

note:: if condition is not specified in findBy___ methods then =(equal to) condition on the given property name.

In Runner calss

```
@Component
public class FinderMethodsTestRunner implements CommandLineRunner {
    @Autowired
    private IMovieRepo repo; // InMememory proxy class obj of our Repository(I) will be injected

    @Override
    public void run(String... args) throws Exception {
        //=====finder methods ============
        repo.findByMnameEquals("Anthim").forEach(System.out::println);
        System.out.println("----------------------------");
        repo.findByMnameIs("Don").forEach(System.out::println);
        System.out.println("----------------------------");
        repo.findBymname("RRR").forEach(System.out::println);

    }

}
```

note:: In our repository interface if the findBy___() method return type List<T> or Iterable<T> then internally generated SELECT SQL Query will performing entity Operation (select all col values of db table for given condition on given property/col

```
public List<Movie> findByMnameEquals(String name);
```

generate entity select query (select * from <Table> or select col1,col2,..(all cols) from <Table>)

generates the where clause condtion (where mname=?)

=>if List<-> or Iterrable<-> generic type is other than entity like String or user-defined interface/class then it internally generates scalar select SQL query (selecting specific colum values)

note:: The db table names, col names are not case-sensitive .. but col data/values are case-sensitive.

Jan 05.1 Spring Data JPA -finder Methods

Reference image for finder methods

Keyword	Sample	JPQL snippet
And	findByLastnameAndFirstname	… where x.lastname = ?1 and x.firstname = ?2
Or	findByLastnameOrFirstname	… where x.lastname = ?1 or x.firstname = ?2
Is,Equals	findByFirstname,findByFirstnameIs,findByFirstnameEquals	… where x.firstname = 1?
Between	findByStartDateBetween	… where x.startDate between 1? and ?2
LessThan	findByAgeLessThan	… where x.age < ?1
LessThanEqual	findByAgeLessThanEqual	… where x.age <= ?1
GreaterThan	findByAgeGreaterThan	… where x.age > ?1
GreaterThanEqual	findByAgeGreaterThanEqual	… where x.age >= ?1
After	findByStartDateAfter	… where x.startDate > ?1
Before	findByStartDateBefore	… where x.startDate < ?1
IsNull	findByAgeIsNull	… where x.age is null
IsNotNull,NotNull	findByAge(Is)NotNull	… where x.age not null
Like	findByFirstnameLike	… where x.firstname like ?1
NotLike	findByFirstnameNotLike	… where x.firstname not like ?1
StartingWith	findByFirstnameStartingWith	… where x.firstname like ?1 (parameter bound with appended %)
EndingWith	findByFirstnameEndingWith	… where x.firstname like ?1 (parameter bound with prepended %)
Containing	findByFirstnameContaining	… where x.firstname like ?1 (parameter bound wrapped in %)
OrderBy	findByAgeOrderByLastnameDesc	… where x.age = ?1 order by x.lastname desc
Not	findByLastnameNot	… where x.lastname <> ?1
In	findByAgeIn(Collection<Age> ages)	… where x.age in ?1
NotIn	findByAgeNotIn(Collection<Age> age)	… where x.age not in ?1
True	findByActiveTrue()	… where x.active = true
False	findByActiveFalse()	… where x.active = false
IgnoreCase	findByFirstnameIgnoreCase	… where UPPER(x.firstame) = UPPER(?1)

More example

Repository Interface

```
public interface IMovieRepo extends JpaRepository<Movie, Integer> {
 //select mid,mname,year,ratings  from Movie where mname=?
 public List<Movie>  findByMnameEquals(String name);
 //select mid,mname,year,ratings  from Movie where mname=?
 public List<Movie>  findByMnameIs(String name);
 //select mid,mname,year,ratings  from Movie where mname=?
 public Iterable<Movie>  findBymname (String name);
 //select mid,mname,year,ratings  from Movie where mname like 'R%'
 public Iterable<Movie>  findByMnameStartingWith(String initChars);
 //select mid,mname,year,ratings  from Movie where mname like '%n'
 public Iterable<Movie>  findByMnameEndingWith(String lastChars);
 //select mid,mname,year,ratings  from Movie where mname like '%dhe%'
 public Iterable<Movie>  findByMnameContaining(String chars);
 //select mid,mname,year,ratings  from Movie where mname like '%dhe%'
 public Iterable<Movie>  findByMnameEqualsIgnoreCase(String name);
 //select mid,mname,year,ratings  from Movie where mname like '%dhe%'
 public Iterable<Movie>  findByMnameContainingIgnoreCase(String chars);
 //select mid,mname,year,ratings  from Movie where mname like 'R%'  // movies starting with R
 //select mid,mname,year,ratings  from Movie where mname like '___'  // 3 letter movies (3 underscores)
 //select mid,mname,year,ratings  from Movie where mname like '%R%'  //Containing letter R
 //select mid,mname,year,ratings  from Movie where mname like '%R'  //ending  letter R
        public Iterable<Movie>  findByMnameLike(String chars);  // pass wild chars while calling method
 }
```

In runner class

```
//=====finder methods ============
repo.findByMnameEquals("Anthim").forEach(System.out::println);
System.out.println("----------------------------");
repo.findByMnameIs("Don").forEach(System.out::println);
System.out.println("----------------------------");
repo.findBymname("RRR").forEach(System.out::println);
System.out.println("----------------------------");
repo.findByMnameStartingWith("Ra").forEach(System.out::println);
System.out.println("----------------------------");
repo.findByMnameEndingWith("n").forEach(System.out::println);
System.out.println("----------------------------");
repo.findByMnameContaining("m").forEach(System.out::println);
System.out.println("----------------------------");
repo.findByMnameEqualsIgnoreCase("rrR").forEach(System.out::println);
System.out.println("----------------------------");
repo.findByMnameContainingIgnoreCase("r").forEach(System.out::println);
System.out.println("----------------------------");
//repo.findByMnameLike("R%").forEach(System.out::println); //or (will get Starting with R movie named records)
//repo.findByMnameLike("___").forEach(System.out::println); //or (will get 3 lettered movie named records)
//repo.findByMnameLike("%R").forEach(System.out::println);//or (will get Ending with R movie named records)
repo.findByMnameLike("%R%").forEach(System.out::println); //or(will get both starting and ending with R movie
                                                            named records)
```

Jan 06 Spring Data JPA -finder Methods

finder methods with multiple properties (working with and , or clauses)

Keyword	Sample	JPQL snippet
And	findByLastnameAndFirstname	… where x.lastname = ?1 and x.firstname = ?2
Or	findByLastnameOrFirstname	… where x.lastname = ?1 or x.firstname = ?2
Is,Equals	findByFirstname,findByFirstnameIs,findByFirstnameEquals	… where x.firstname = 1?
Between	findByStartDateBetween	… where x.startDate between 1? and ?2
LessThan	findByAgeLessThan	… where x.age < ?1
LessThanEqual	findByAgeLessThanEqual	… where x.age <= ?1
GreaterThan	findByAgeGreaterThan	… where x.age > ?1
GreaterThanEqual	findByAgeGreaterThanEqual	… where x.age >= ?1
After	findByStartDateAfter	… where x.startDate > ?1
Before	findByStartDateBefore	… where x.startDate < ?1
IsNull	findByAgeIsNull	… where x.age is null
IsNotNull,NotNull	findByAge(Is)NotNull	… where x.age not null
Like	findByFirstnameLike	… where x.firstname like ?1
NotLike	findByFirstnameNotLike	… where x.firstname not like ?1
StartingWith	findByFirstnameStartingWith	… where x.firstname like ?1 (parameter bound with appended %)
EndingWith	findByFirstnameEndingWith	… where x.firstname like ?1 (parameter bound with prepended %)
Containing	findByFirstnameContaining	… where x.firstname like ?1 (parameter bound wrapped in %)
OrderBy	findByAgeOrderByLastnameDesc	… where x.age = ?1 order by x.lastname desc
Not	findByLastnameNot	… where x.lastname <> ?1
In	findByAgeIn(Collection<Age> ages)	… where x.age in ?1
NotIn	findByAgeNotIn(Collection<Age> age)	… where x.age not in ?1
True	findByActiveTrue()	… where x.active = true
False	findByActiveFalse()	… where x.active = false
IgnoreCase	findByFirstnameIgnoreCase	… where UPPER(x.firstame) = UPPER(?1)

```
syn::
    public <RT>   findBy<Prop1><Cond1> <And> or <Or> <prop2> ....<Cond2>(params ....);
```

Examples

In Repository Interface

```
    /* finder methods  with mutliple propeties based  conditions  */
  //select mid,mname,year,ratings from Movie where mid>? and ratings<?
  public  Iterable<Movie> findByMidGreaterThanAndRatingLessThan(int startMid, float endRatings);

  //select mid,mname,year,ratings from Movie where Mname like 'R%' or  (ratings between 3.0 and 5.0)
  public Iterable<Movie> findByMnameStartingWithOrRatingBetween(String mnameChars,
                                                                float startRating,
                                                                float endRatings);

  //select mid,mname,year,ratings from Movie where year in (? , ?) or (upper(mname) like upper(?) escape ?) and (ratings between ? and ?)
  public Iterable<Movie> findByYearInOrMnameContainingIgnoreCaseAndRatingBetween(List<String> years,
                                                                                 String chars,
                                                                                 float startRatings,
                                                                                 float endRatings);
  //select mid,mname,year,ratings from Movie where (mname not like ? escape ?) and (year in (? , ?)) order by mname asc
  public Iterable<Movie> findByMnameNotLikeAndYearInOrderByMnameAsc(String letters,
                                                                    List<String> years);
```

Code in Runner class

```
    System.out.println("========finder methods with multiple properties based conditions=========");
            repo.findByMidGreaterThanAndRatingLessThan(300,5.0f).forEach(System.out::println);
                System.out.println("--------------------");
                repo.findByMnameStartingWithOrRatingBetween("R", 3.0f, 4.5f).forEach(System.out::println);
                System.out.println("--------------------");
                repo.findByYearInOrMnameContainingIgnoreCaseAndRatingBetween(List.of("2021","2022"),
                                                                             "r",
                                                                             3.0f, 5.0f).forEach(System.out::println);
                System.out.println("--------------------");
                repo.findByMnameNotLikeAndYearInOrderByMnameAsc("R%", List.of("2021","2022")).forEach(System.out::println);
```

Finder methods performing scalar opeations (selecting specific one or more column values) /Projections in spring data jpa

finder methods support two kinds of Scalar opeations

a) static Scalar/Projection opreations (Always get fixed specific column values)

b) Dynamic scalar/Projection Operations (we can get specific varying column values)

Retriving specific single or mutiple col vlaues is called Projection activities..

a) static Scalar/Projection opreations (Always get fixed specific column values)

=>Asking to hold specific our choice spefic property/col vlaues we need to make spring data jpa generating one Model/Entity class as InMemory Proxy class by Proiving Type Interface with certain properties related getter methods declaration..

For example if we expecting to InMemory Entity /Model class as property for mid, mname properties then we need to defined Type Interface as shown below..

```
interface ResultView{
  public  Integer getMid();
  public  String getMname();
 }
```

if unknow properties are placed in in the getter methods then we exceptions.

=>start using this Type interface as part of Return type for finder methods like Iterable<ResultView> or List<ResultView>

example

step1) Define Type interface having getter methods for u r choice propeties as shown below.

```
// ResultView.java
package com.nt.view;

public interface ResultView {
        public  Integer getMid();
        public String getMname();

}
```

//we can not use lombok api @Getter here becoz we need interface getter method decl .. not the definitations. more over no properties are defined here and it is interface(not the class)

step2) place finder method Repository(I) involving Type interface in the return type of the method..

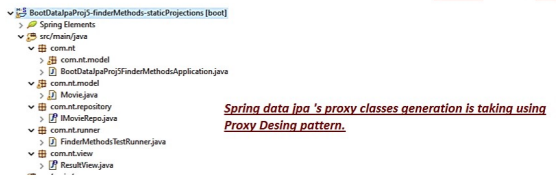
```
public interface IMovieRepo extends JpaRepository<Movie, Integer> {

    public  Iterable<ResultView>  findByMidGreaterThanEqualAndMidLessThanEqual(int startMid,int endMid);
}
```

step3) Invoke the methods in runner class

```
@Component
public class FinderMethodsTestRunner implements CommandLineRunner {
    @Autowired
    private  IMovieRepo  repo;  // InMememory proxy class obj of our Repository(I) will be injected

    @Override
    public void run(String... args) throws Exception {
        System.out.println("Reposity impl class name::"+repo.getClass());
        Iterable<ResultView> it=repo.findByMidGreaterThanEqualAndMidLessThanEqual(100,500);
        it.forEach(view->{
            System.out.println("dynamic model class name"+view.getClass());
             System.out.println(view.getMid()+"  "+view.getMname());
        });
    }
}
```

The generated query is ::

```
Hibernate: select movie0_.mid as col_0_0_, movie0_.mname as col_1_0_ from sp_data_movie movie0_
                                                                    where movie0_.mid>=? and movie0_.mid<=?
```



Spring data jpa 's proxy classes generation is taking using Proxy Desing pattern.

Jan 07 Spring Data JPA -finder Methods-Dynamic Projections

The finder methods of spring data JPA supports two types of Projections

a) static Projections
[Allows to select specific fixed column values (single or multiple))

b.) Dynamic Projections
[Allows to select specific varying column values (single or multiple))

Core Java ReCap
==========

```
class Person{
  ....
  ...
}

class Student extends Person{
  ...
  ...
  ..
}

class Employee extends Person{
  ....
  ...
  ...
}
```

option1: public Object showDetails(Object obj){
....
.... return obj;
}
=>we can pass and get not only Person class hierarchy obj type
In fact any class obj and code is not safe

→ Student st=(Student)ref.showDetails(new Student());
Employee e=(Employee)ref.showDetails(new Employee());
Here Typecastin is required i.e code is not type safe code

option2:: public Person showDetails(Person person){
....
return obj;
}
=>Here we can pass and get Only Person class hierarchy class objs and code is not type safe

→ Student st=(Student)ref.showDetails(new Student());
Employee e=(Employee)ref.showDetails(new Employee());
Here Typecastin is required i.e code is not type safe code

option3:: public <T> T showDetails(Class<T> clazz){
(generics alternate for option1)
....
....
return obj;
}
=>Here we can pass and get any class obj and code is type safe becoz of generics..

→ Student st=ref.showDetails(Student.class);
Employee emp=ref.showDetails(Employee.class);
Date date=ref.showDetails(Date.class);
Here type casting is not required..So code is type safe code..

option4: public <T extends Person> T showDetails(Class<T> clazz){
(generics alternate for option2)
....
....
return obj;
}
Here we can pass and get only Person hierarchy class obj.. and code is type safe code..

→ Student st=ref.showDetails(Student.class); ✓
Employee emp=ref.showDetails(Employee.class); ✓
Date date=ref.showDetails(Date.class); ✗

Here type casting is not required..So code is type safe code..

```
package com.nt.test;

import java.util.ArrayList;
import java.util.Date;

import com.nt.comp.Employee;
import com.nt.comp.Person;
import com.nt.comp.Student;

public class App
{

    public <T> T showDetails(Class<T> clazz) throws Exception {
        return (T) clazz.getDeclaredConstructor(new Class[] {}).newInstance();
    }                                          empty array makes the method to pickup 0-param constructor

    public <T extends Person> T showDetails1(Class<T> clazz) throws Exception {
        return (T) clazz.getDeclaredConstructor(new Class[] {}).newInstance();
    }

                                                    int a[]=new int[]{1,2,3}; //array with initial elements
                                                    int a[]=new int[]{};  //array with empty elements
                                                                          initialization

  public static void main( String[] args )throws Exception
  {
    App app=new App();
    Student st=app.showDetails(Student.class);
    System.out.println((st));
    System.out.println("-----------------");
    Employee emp=app.showDetails(Employee.class);
    System.out.println((emp));
    System.out.println("--------------------");
    ArrayList list=app.showDetails(ArrayList.class);
    System.out.println(list);
    System.out.println("--------------------");
    Date date=app.showDetails(Date.class);
    System.out.println(date);
    System.out.println("================================");

    App app1=new App();
    Student st1=app1.showDetails1(Student.class);
    System.out.println((st1));
    System.out.println("-----------------");
    Employee emp1=app1.showDetails1(Employee.class);
    System.out.println((emp1));
    System.out.println("--------------------");
    ArrayList list1=app1.showDetails1(ArrayList.class);  ✗
    System.out.println(list1);
    System.out.println("--------------------");
    Date date1=app1.showDetails1(Date.class);  ✗
    System.out.println(date1);


  }
}
```

Bound mismatch: The generic method showDetails1(Class<T>) of type App is not applicable for the arguments (Class<ArrayList>). The inferred type ArrayList is not a valid substitute for the bounded parameter <T extends Person>

Jan 07.1 Spring Data JPA -finder Methods

Dynamic Projections
==========
=>Here we can get varying specific single or multiple col values from db table using the support of finder methods. For this we need to take mutiple type interfaces having same hierarchy as show below

```
interface View{
 }

inteface ResultView1 extends View{
   public Integer getMid();
   public String getMname();
  }

interface ResultView2 extends View{
   public Integer getMid();
   public String getMname();
   public Float getRating();
  }

interface ResultView3 extends ResultView1{
   public String getYear();
  }
```

=>Use these Type View interface's super type (View(I)) in the return type of finder methdos..

```
public <T extends View> List<T> findByMnameIn(List<String> movies, Class<T> clazz);
       type declaration    RT    method name   param for finder   param for Generic (Type inference)
```

```
calls::  List<ResultView1> list =findByMnameIn(List.of("RRR","Puspha"),
                                                ResultView1.class);
         // 2 cols values (mid,mname)

         List<ResultView2> list=findByMnameIn(List.of("RRR","Puspha"),
                                                ResultView2.class);
         //gives 3 cols
            (mid,mname,rating)
```

Applicaiton devleop using Dynamic Projects in finder Methods
==========

step1) Keep spring data jpa project ready

step2) Develop multiple "Type View interfaces" having the common parent interface as shown above.

```
interface View{
 }

inteface ResultView1 extends View{
   public Integer getMid();
   public String getMname();
  }

interface ResultView2 extends View{
   public Integer getMid();
   public String getMname();
   public Float getRating();
  }

interface ResultView3 extends ResultView1{
   public String getYear();
  }
```

step3) Use the Type View interface's super type (View) in finder method as the return type In Repository Interface

```
package com.nt.repository;
import java.util.List;
import org.springframework.data.jpa.repository.JpaRepository;
import com.nt.model.Movie;
import com.nt.view.View;

public interface IMovieRepo extends JpaRepository<Movie, Integer> {

    public <T extends View> Iterable<T> findByMnameIn(List<String> movies,
                                                      Class<T> clazz);
 }
```

```
public <T extends View> Iterable<T> findByNameIn(List<String> names, Class<T> clazz);
                        Return type

Iterable<ResultView1> it = repo.findByNameIn(List.of("rrr", "achr"), ResultView1.class);
```

step4) Invoke the methods from Runner class.

```
Iterable<ResultView1> list=repo.findByMnameIn(List.of("RRR","Don"),
                                              ResultView1.class);
        list.forEach(v1->{System.out.println(v1.getMid()+" "+v1.getMname());});
        System.out.println("-------------------------");

        repo.findByMnameIn(List.of("RRR","Don"),
              ResultView2.class).forEach(v2->{System.out.println(v2.getMid()+" "+v2.getMname()+" "+v2.getRating());});

        System.out.println("-------------------------");
        repo.findByMnameIn(List.of("RRR","Don"),
      ResultView3.class).forEach(v3->{System.out.println(v3.getMid()+" "+v3.getMname()+" "+v3.getYear());});
```

Here multiple InMemory Proxy classes will be generated..

Limitations of finder methods in spring data jpa
==========

a) Only select operations are possible i.e non-select operations are not possible
b) Aggragate operations are not possible *(count(*) ,max(-),min(-) ...)*
c) GroupBy operations are not possible
d) Working with scalar operations/Projections is bit complex
e) While selecting data by using multiple properites and multiple conditions, the method names becoming really very lengthy
f) Method names must be taken by following some conventions.. That process kills the readability...
g) we can not call PL/SQL procecedure and functions..
h) conditional pagination, sorting is not possible
i) sub queries are not possible
j) joins can not used and etc..

Note:-
To Overcome all the Above problems... take the support of @Query methods using JPQL/HQL or Native SQL queries

Conclusion:-
Use Finder methods to perform single property or column coditions based select Operations

Jan 08 @Query Methods

Different ways writing custom logics in spring data jpa repository interface

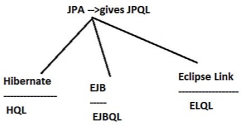
a) Using finder methods
b) Using @Query methods
c) Using @Query +@Modifying methods
and etc..

b) Using @Query methods

=> @Query method means.. the @Query annotation that is added on the top of the method declared in our Repository Interface.
=> Only @Query methods support select operations
=> @Query + @Modifying Methods support non-select operations,

Syntax:

```
@Query(" <HQL/JPQL query > (or) <Native SQL Query>")
public <Return Type> <method>(params ....)
```



JPQL :: Java Persistence Query Language
HQL :: Hibernate query language
ELQL :: Eclipse Link query language
EJBQL :: EJB Query language..

=>Native SQL /SQL Queries are developed using db tables and cols as DB dependent Queries

=>JPQL/HQL Queries are developed Using entity class names and proerty names as DB s/w indepnedent Queries.

=> The underlying ORM s/w converts every JPQL/HQL Query into DB s/w specific SQL Query intenally taking the support Dialect comp before sending and executing Query in Db s/w.

Advatages of @Query methods

=> Supports both select and non-select operations (except insert operation)
=> Methods can have flexible signatures
=>Method names can be taken with out following any naming conventions
=> Can work with both HQL/JPQL and SQL Queries
=> supports aggragate operations
=> Supports group by operations
=> Supports joins
=> Gives support to work with sub queries
and etc..

Why we can not use @Query methods having INSERT Query as HQL/JPQ or SQL Query?

Ans) HQL/JPQL or SQL INSERT Query can not work with generators to generate the id value .. but in o-r mapping of spring data jpa the id property must have the generator generated value .. So use repo.save(-) or repo.saveXxx(-) methos for insert operation who can work with the cfg generators to generate id value.

Q) How to insert partial data to record or how to perform partial save obj operation?

Ans) use rep.save(-) or repo.saveXxx(-) methods but keep partial data to the given Entity obj by making the remaining property values as NULL values or place @Transient on the top of Entity class properties which u do not want see them participating in persistence activity.

example1 *(Best)*

```
Movie movie=new Movie();
movie.setMname("RRR");
//movie.setRating(null);  | optional to place
//movie.setYear(null);    | null values explicitly
System.out.println(repo.save(movie).getMid()+"--- object is saved");
```

Hibernate: insert into sp_data_movie (mname, rating, year, mid) values (?, ?, ?, ?)

Though we setting values to few properties.. but all properties involved in the insert SQL query

=> when the spring data jpa activates hibernate f/w by creating import objs like Sessionfactory,Session and etc.. some SQL queries will be generated as pre-generated SQL queries performing CURD operations..involving all the colums. Most of times the these SQL Queries will be utilized ...

=> To make spring data jpa (internally hibernate) to generate INSERT SQL Query dynamically only by involving Non-Null values properties/cols we need to add @DynamicInseert(true) on the top of Entity class.

```
@Data
@Entity
@Table(name="SP_DATA_MOVIE")
@RequiredArgsConstructor
@NoArgsConstructor
@DynamicInsert(true)
public class Movie implements Serializable {
  ....
  ...
}
```

Now the generated SQL query is ::Hibernate: insert into sp_data_movie (mname, mid) values (?, ?)

example2 { Using @Transient}

@Transient properties of Entity class do not participate in persistence operations

```
@Data
@Entity
@Table(name="SP_DATA_MOVIE")
@RequiredArgsConstructor
@NoArgsConstructor
@DynamicInsert(true)   //optional
public class Movie implements Serializable {
@Id
@Column(name="MID")
@GeneratedValue(strategy = GenerationType.AUTO)
private Integer mid;
@Column(name="MNAME",length = 20)
@NonNull
private String mname;

@Column(name="YEAR",length = 20)
@NonNull
@Transient
private String year;

@Column(name="RATING")
@NonNull
private Float rating;
}
```

In runner class

```
Movie movie=new Movie();
movie.setMname("spiderman");
movie.setRating(5.6f);
movie.setYear("2022");
System.out.println(repo.save(movie).getMid()+"--- object is saved");
```

generated SQL query :: Hibernate: insert into sp_data_movie (mname, rating, mid) values (?, ?, ?)

year col has not come becoz of @Transient .. not becoz of @DynamicInsert(true)

note: Not a solution becoz @Trasient makes property as Non persitable property for all client apps which is not so acceptable..

---

note:: The Mapping annotations of JPA , we can add in the entity classes either on the top of fields or on the top of getter methods ..Actually it is recomanded to add on the top of getter methods for better redability and easy accessbility becoz of public modifiers. Since we are getting dynamic getters and setters using lombok api.. so we are adding on the top of properites.

Mapping annotations are :: @Column , @Id ,@GeneratedValue and etc..

Why there are only @DynamicInsert and @DynamicUpdate annotations .. why we do have not @DynamicSelect and @DynamicDelete annotations?

Ans) =>delete SQL query deletes the complete record by taking the criteria value, i.e there is not possiblity of deleting partial values of the record using delete SQL query So there is no @DynamicDelete

=>load object opeartion gets the record from db table and puts in to entity object by creating the object dynamically ... So programmer can not controll select Query generation through null values kept in the object becoz the object will be created dymically.. For this reason there is @DynamicSelect Annotation.

note :- transient keyword is used to not to participate in serialization

Jan 09 @Query Methods

HQL/JPQL Queries (HQL/JPQL Queries are objects/Entties based SQL Queries)

---

in

=>These are DB s/w idependent Quries
=>These queries will be written by using Entity class name and its properites
=> Supports both select and non-select operations
=> Allows to place both positional params(?1,?2,...) and named params (:<name>)
=> Supports aggegrations, joins, sub queries , condition clauses and etc..
=>The Learn curve of HQL/JPQL is very small (time wise) based on the knowledge of SQL
=> Every HQL/JPQL Query will be traslated to the underlying DB s/w specific SQL Query
=> HQL/JPQL Queries portable across the multiple Db s/ws and ORM frameworks
=> HQL keywords are not case-sensitive .. but the entity class names, property names used in the HQL Query are case-sensitive.

<= HQL/JPQL based insert record operation is not possbile (becoz HQL insert can not involve Generators)
<= HQL/JPQL based DDL operations(create table, drop table, alter table and etc..) are not possbile
<= HQL/JPQL based PL/SQL procedure/functions are not possbile.

SQL> SELECT * FROM SP_DATA_MOVIE;
db table name

HQL> from Movie (or)
from Movie m (or) "m" alias name and optional to given
from Movie as m (or)
SELECT m from Movie m

---

db table name
SQL> SELECT * FROM SP_DATA_MOVIE WHERE MID>=? AND MID<=?
col name col name

HQL> from Movie where mid>=?1 and mid<=?2
entity class
JPA Style ordinal positional param
property name

(or)

from Movie where mid>=:min and mid<=:max
Named Params

note:: JDBC style positional params(?) are not allowed

?,?,? are jdbc style positional params
?1,?2,?3 are jpa style ordial params
:<name1>, :<name2>,:<name3> are named params (more recomanded)

if HQL/JPQL Query is entity query i.e retrieving all col values from Db table then writing "SELECT " statement in the HQL/JPQL Query is optional.

SQL> SELECT MID,MNAME FROM SP_DATA_MOVIE WHERE MNAME IN(?,?)

HQL>SELECT mid,mname FROM Movie WHERE mname IN(?1, ?2) (or)
HQL>SELECT mid,mname FROM Movie WHERE mname IN(:movie1,:movie2) (or)
HQL>SELECT m.mid ,m.mname FROM Movie as m WHERE m.mname IN (:movie1,:movie2)

Since we are selecting specific multiple col/property values the SELECT Keyword is mandatory

---

SQL> UPDATE SP_DATA_MOVIE SET YEAR=? WHERE MID=?

HQL> UPDATE Movie SET year=?1 WHERE mid=?2 (or)
HQL> UPDATE Movie SET year=:yr WHERE mid=:id

syntax of named param is :<name>
=>named params are self descriptive

---

example
========

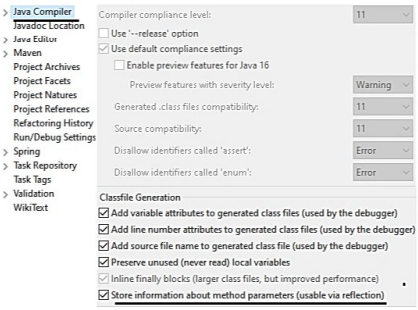
Code in repository

```
public interface IMovieRepo extends JpaRepository<Movie, Integer> {

    //@Query("from Movie where mid>=?1 and mid<=?2") (or)
    //@Query("from Movie as m where m.mid>=?1 and m.mid<=?2") (or)
    /*@Query("from Movie where mid>=:min and mid<=:max")
     public List<Movie> searchMoviesByIdRange(@Param("min") int start,@Param("max") int end);*/

    @Query("from Movie where mid>=:min and mid<=:max")
     public List<Movie> searchMoviesByIdRange( int min, int max);

}
```

=>working with named params by matching their names with method param names is quite good pratice.

if the names of Named params in HQL/JPQL Query are matching with the names java method params then there is no need of placing @Param annotation to map named params of query with method params .. For this we must ensure the following setting is enabled in eclipse IDE..

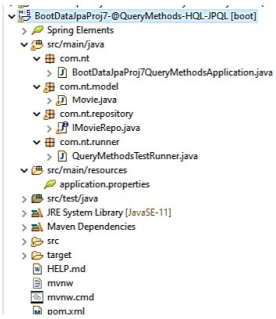
PRoject properties (right click on the Project)--->



apply --->ok

Code in runner class
==========

```
@Component
public class QueryMethodsTestRunner implements CommandLineRunner {
    @Autowired
    private IMovieRepo repo; // InMememory proxy class obj of our Repository(I) will be injected

    @Override
    public void run(String... args) throws Exception {
        System.out.println("====Query methods execution ============");
        List<Movie> list=repo.searchMoviesByIdRange(200, 500);
        list.forEach(System.out::println);

    }

}
```



Jan 10 @Query Methods using HQL-JPQL

About parameters in HQL/JPQL queries
==========

=> In HQL/JPQL Queries we can pass jpa style ordinal positional params (?1,?2,..) or Named params (:<name1>,:<name2>,..) and etc..

=> we can not palce both named and positional params in single hql/ jpql query

=> ordinal positional parameters index must begin with 1 and should be incremented by 1

=> we can use parameters only representing input values of the Query not representing HQL /JPQ keywords , Entity class name or Entity class properties.. and etc..

examples

=> from Movie where mid>=?1 and mid<=:max (invalid becoz we can not place both named ,positional params in a sigle query)

=> from Move where mid>?1 and mid<=?3 (invalid becoz there must not gap b/w ordinal params index)

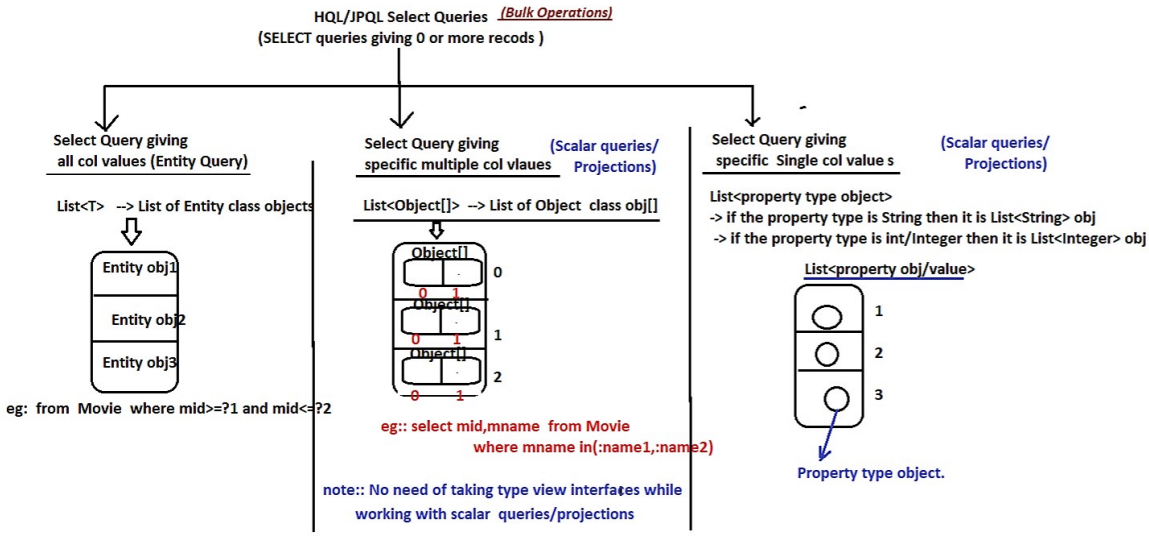
=> from Movie where mid>?0 and mid<=?1 (invalid becoz ordinal positional param must start with 1)

=> from Movie where mid>?2 and mid<=?1 (valid becoz ordinal positional param started with 1 .. and increment by 1 do not worry about placement)

=> from Movie where mid>=?2 and mid<=?4(invalid becoz ordinal parameter index must start with 1)

=> From ?1 where mid>=?2 and mid<=?3 (invalid becoz we can not Entity class name as the parameter value)

=> ?1 Movie where mid>=?2 and mid<=?3 (invalid becoz we can not HQL/JPQL keyword as the parameter value)



==========

HQL/JPQL Entity Query Example (selecting all col values of db table)

==========

Code in Repository

```
@Query("from Movie where mname In(:name1,:name2,:name3) order by mname asc")
    public List<Movie> searchMoviesByMnames(String name1,String name2,String name3); //taking array /var args /collection params
                                                                                      // multiple named params or postional params is not allowed
    @Query("from Movie where mname In(?1,?2,?3) order by mname asc")
    public List<Movie> searchMoviesByMnames1(String name1,String name2,String name3 );
```

In runner class

```
repo.searchMoviesByMnames("RRR","Don","puspha").forEach(System.out::println);
System.out.println("------");
repo.searchMoviesByMnames1("RRR","Don","puspha").forEach(System.out::println);
```

==========

Examples on Scalar Queries selecting specific multiple col values

==========

in repository

```
//================= Scalar Queries (sepecific multiple col values) =================

    @Query("select mid,mname,year from Movie where rating>=:rat and year in(:y1,:y2,:y3) ")
    public List<Object[]> fetchMovieDetailsByRatingsAndYear(float rat , String y1,String y2,String y3);
```

In runner class

```
System.out.println("==================");
        repo.fetchMovieDetailsByRatingsAndYear(4.0f, "2020","2021","2022").forEach(row->{
            for(Object val:row)
                System.out.print(val+"  ");

            System.out.println();
        });
```

==========

Examples on Scalar Queries selecting specific *single* col values

==========

code in repository

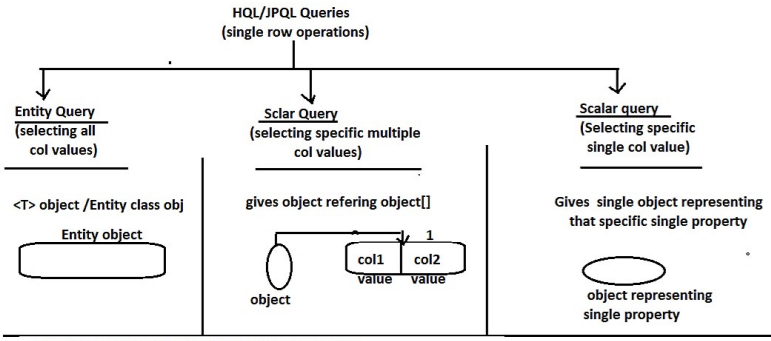
```
//================= Scalar Queries (sepecific single col values) =================

@Query("select mname from Movie where year>=:start and year<=:end order by mname desc ")
public List<String> fetchMoviesByYearRange(String start,String end);
```

Code in runner class

```
System.out.println("==================");
repo.fetchMoviesByYearRange("2020","2022").forEach(System.out::println);
```

jan 17 @Query Methods using HQL-JPQL-SingleRow

Working with HQL/JPQL Query performing single row operations.



Example on Entity Query giving single Row

In Repository Interface

```
/ Single Row Entity Query
@Query("from Movie where mname=:name") //assuming movie names are unique name
public Movie fetchMovieDataByMname(String name);
```

In runner class

```
Movie movie=repo.fetchMovieDataByMname("Anthim1");
        if(movie!=null)
         System.out.println("RRR Movie details are::"+movie);
        else
              System.out.println("Movie not found");
```

Q) when we have readymade findById(-) method why should we use @Query methods for singleRow Operations

Ans) The findById(-) method can take only id value as the criteria value... where as Query method having query for single row operations can take other unique col(s)/property(es) as the criteria values.

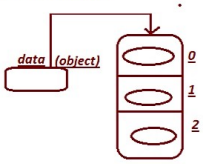
Example on getting sepecific multiple col values of a single Row

In Repository Interface

```
// Single Row Scalary query giving specific multiple col values
        @Query("select mid,mname,year from Movie where mname=:name")
         public Object fetchMoviePartialDataByMname(String name);
```

In Runner class

```
System.out.println("----------------------------------------");
        Object data=repo.fetchMoviePartialDataByMname("Anthim");
        Object result[]=(Object[])data;
        for(Object val:result) {
              System.out.print(val+" ");
        }
        System.out.println();
```



Example on getting sepecific single col value of a single Row

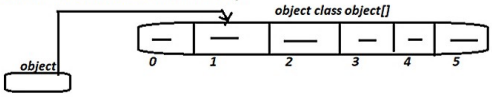
```
--- Code in Repository Interface ----------
//single Row Scalar query giving single specific col vlaue
@Query("select year from Movie where mname=:name")
 public String fetchMovieSingleDataByMname(String name);
```

```
---code in Runner class ----------
    String year=repo.fetchMovieSingleDataByMname("Anthim");
        System.out.println("Anthin movie release year ::"+year);
```

Examples on executing aggregate functions using HQL/JPQL

```
--------Code in Repository Interface----------
    @Query("select max(rating),min(rating),avg(rating),count(*),avg(year),sum(year) from Movie ")
        public Object fetchAggregateDataOnMoives();
```

```
--------Code in Runner class ----------
    System.out.println("--------------------------");
        System.out.println("highest rating for movie::"+repo.fetchMaxRatingMovie());

        System.out.println("----------------------------");
        Object results1[]=(Object[])repo.fetchAggregateDataOnMoives();
        System.out.println(" Max rating "+results1[0]);
        System.out.println(" Min rating "+results1[1]);
        System.out.println(" Avg rating "+results1[2]);
        System.out.println(" count of movies "+results1[3]);
        System.out.println(" avg of years "+results1[4]);
        System.out.println(" sum of years "+results1[5]);
```



Example on sub queries

```
---- code in repository ----------
//executing sub query
        @Query("from Movie where rating=(select max(rating) from Movie) ")
        public List<Movie> fetchMaxRatingMovies();
```

```
------ code in runner class ----------
    System.out.println("----------------------------");
        repo.fetchMaxRatingMovies().forEach(System.out::println);
```

jan 18 @Query Methods HQL-JPQL for NonSelect Operations - Native SQL

Performing Non-select Operations using HQL/JPQL

=> For this we need to take @Query + @Modifying methods in our repository Interface

=>@Modifying indicates that this @Query is having non-select HQL/JPQL Query .. not the select HQL/JPQL Query

=> We must execute non-select HQL/JPQL Queries as TRansactional env.. queries having ability to commit or rollback the DB operation .. For this we need to place @Transactional annotation.

=> if u r developing complete layed taking seperate service class then placing @Transactional on @Repository interface or on @Repository interface methods is optional becoz @Service itself takes care of Transaction Mgmt otherwise we must place @Transactional either on Repository interface or on the @Query + @Modifying methods of Repository interface.

(use repo.save(-) method)

=> Spring data JPA does not support HQL/JPQL INSERT Queries .. So we can use the above setup only executing Update , Delete Non-select HQL/JPQL Queries.

=> Generally the @Query +@Modifying methods return type will int or short or long or byte becoz the non-select HQL/JPQL queries return numeric value representing no.of records that are effected.

Example Code

----- code in Repository----------

```
package com.nt.repository;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Modifying;
import org.springframework.data.jpa.repository.Query;
import org.springframework.transaction.annotation.Transactional;
import com.nt.model.Movie;
@Transactional
public interface IMovieRepo extends JpaRepository<Movie, Integer> {

    @Query("update Movie set rating=:newRating where mname=:name")
    @Modifying
    //@Transactional
    public int modifyRatingByMoiveName(float newRating, String name);

    @Query("delete from Movie where year>=:start and year<=:end")
    @Modifying
    //@Transactional
    public int deleteMovieByYearRange(String start, String end);
}
```

note:: instead of placing @Transactional on the top of every @Query + @Modifying method it is recomanded to place only for 1 time on the top of Repository(I)

Code In Runner class

```
package com.nt.runner;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

import com.nt.repository.IMovieRepo;

@Component
public class NonSelectOperationsRunner implements CommandLineRunner {
    @Autowired
    private IMovieRepo repo;

    @Override
    public void run(String... args) throws Exception {
        System.out.println("updatting "+repo.modifyRatingByMoiveName(4.5f,"RRR")+" movies");

        System.out.println(repo.deleteMovieByYearRange("1900","2000")+" movies are deleted");

    }//run(-)

}//class
```

What is the difference b/w finder methods and @Query methods

finder methods	*@Query methods*
a) supports only select operations	*a) supports select , non-select (using @Modifying) even DDL(using native SQL) Opeations*
b) No provision to specify the queries	*b) provision is given to specify the HQL/JPQL or native SQL Queries*
c) The finder methods should be taken having fixed naming convention for method names	*c) No need of following any naming conventions for method names*
d) Can not used to call PL/SQL Procedures and functions	*d) can be used*
e) Based method name conventions the Inmemory proxy class generates SQL Queries	*e) In Memory Proxy generated code gives the specified HQ/JPQL Queries to ORM software (hibernate) which will be later converted into DB s/w specific SQL Queries.*
f) Suitable for single property based select operations	*f) suitable for all operations which are not possible with ready made methods*
g) Kill the readability becoz of the lengthy method names	*g) imprroves the readability.*

Working with Native SQL Queries

=> if certain operations are not possible using ready made methods or finder method or @Query methods then prefer using Native SQL Queries

=>For that we need to place nativeQuery="true" in @Query Annotation and these Native Queries or Native SQL Queries will be written using table names and col names.

=> We generally use NAtive SQL Queries for the following operations

a) To execute Insert SQL Query
b) To use Db s/w specific aggragate functions in the Queries
c) To use DB s/w specific date ,time functions like SysDate in oracle and now() in mysql
d) To use Db s/w sepecic psuedo colums like rownum in oracle
e) To execute DDL Queries
f) To call PL/SQL procedures and functions
g) To work with jdbc style postional params..(?)
and etc..

Native SQL Queries in spring data jpa supports positional params(?,?,?,..), ordinal positional params (?1,?2,?3,...) and Named params (:<name1>,:<name2>,:<name3>, ...

=> Native SQL Queries will be written specifying the db table names, col names .. not the entity class names and property names.

=> We need to place @Modifying and @Transactional while executing native SQL Non-select SQL Queries

In Repository Interface

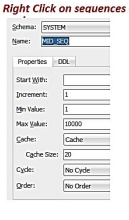
```
@Transactional
public interface IMovieRepo extends JpaRepository<Movie, Integer> {
    @Query(value="INSERT INTO SP_DATA_MOVIE VALUES(MID_SEQ.NEXTVAL,?,?,?)",nativeQuery = true)
    @Modifying
  public int registerMovie(String name,float rating, String year);
 }
```

----code in Runner class----

```
@Component
public class NativeSQLTestRunner implements CommandLineRunner {
    @Autowired
    private IMovieRepo repo;

    @Override
    public void run(String... args) throws Exception {
        int result=repo.registerMovie("raja", 4.5f, "2021");
        System.out.println(result==0?"movie not registered":"movie registered");

    }//run(-)

}//class
```

=>Sequence in oracle is given to generate PK column values dynamically

Right Click on sequences -->new



Jan 19 @Query Methods Native SQL and Calling PL-SQL Procedures and functions

Examples on Native SQL Queries using @Query methods

==========

---->Code in Repository ----------

```
@Transactional
public interface IMovieRepo extends JpaRepository<Movie, Integer> {

    @Query(value="INSERT INTO SP_DATA_MOVIE VALUES(MID_SEQ.NEXTVAL,?,?,?)",nativeQuery = true)
    @Modifying
  public  int  registerMovie(String name,float rating, String year);

    @Query(value="SELECT SYSDATE from DUAL",nativeQuery = true)
    public  String  fetchSysDate();

    @Query(value="create table temp116(col1 number(5))",nativeQuery = true)
    @Modifying
    public int  createTempTable();
}
```

----------Code in Runner class ----------

```
@Component
public class NativeSQLTestRunner implements CommandLineRunner {
      @Autowired
      private IMovieRepo  repo;

       @Override
       public void run(String... args) throws Exception {
//            int result=repo.registerMovie("raja", 4.5f, "2021");
//            System.out.println(result==0?"movie not registered":"movie registered");
             System.out.println("----------");
              System.out.println("date and time is ::"+ repo.fetchSysDate());
              System.out.println("----------");
             int result=repo.createTempTable();
             System.out.println(result<0?"Table is not created":"table is created");

      }//run(-)

}//class
```

Calling PL/SQL PRocedure or function from spring data JPA application (call PL/SQL as PL BY SEQUEL)

=> Instead of placing same SQL Queries based logics or b.logics in multiple modules of a Project or in multiple Apps of a module.. It is recomanded to write only for 1 time in Db s/w as PL/SQL procedures and functions.

=> Instead of making SQL queries as compiled SQL queries by sending to Db s/w every time we can keep the same queries as pre-compiled Queries by placing them in PL/SQL procedure and functions.

usecase1:: To get into every module we need Authenticaiton logic.. Instead of placing that Authentication logic in every module as SQL Queries .. place them in Db s/w only for 1 time as PL/SQL procedure and use it in multiple module

usecase2:: we can place Attendence calculation logics of University Project as PL/SQL procedure or function.

=> In real projects 70% Persistnece logic using SQL Queries or o-r mapping logics and 30% using PL/SQP procedure or function.

=> PL/SQL procedure does not return a value , where PL/SQL functions returns a value.

=> The params of PL/SQL procedure or function contains 3 details

a) name b) type c) mode

mode : IN , OUT , INOUT

=>IN mode provides inputs
=>OUT mode -To gather outputs/result
=>INOUT --> To provide inputs and to gather outputs

PL/SQL code/logic
==========
y:=x*x

here x should be IN param and y should be out param

PL/SQL code/logic
---
x:=x*x;
"x" should be taken as INOUT param

=>PL/SQL programming syntaxes are specific to each DB s/w .
In oracle "=" for comparision (ex:-select from movie where name="xyz";)
in this = is comparison
:= for assignment

=>if PL/SQL procedure wants to return 10 results then we need to take 10 out params
=>if PL/SQL function wants to return 10 results then we need to take 9 out params and 1 return value

Cursor in oracle's PL/SQL programming

=> Cursor is a InMemory variable of oracle's PL/SQL programming having capability to hold bunch of records given by SELECT SQL query..

=>sys_refcursor is cursor data type in oracle's PL/SQL programming

the Cursor of Oralce PL/SQL programming is like JDBC ResultSet obj (like in core list or set)

PL/SQL code
---
details sys_refcursor;

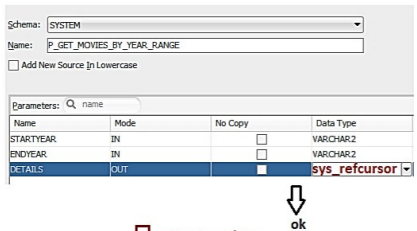
open details for
SELECT MID,MNAME,RATING,YEAR FROM SP_DATA_MOVIE;

=>The records given SELECT SQL Query will be stored in to the cursor variable "details"

Jan 19.1 @Query Methods Native SQL & Calling PL-SQL Procedures & functions

Procedure to create "Oracle PL/SQL Procuere" using SQL Developer

SQL Developer ----> right click Procedures --> new Procedure -->



ok

gives template code

```
CREATE OR REPLACE PROCEDURE P_GET_MOVIES_BY_YEAR_RANGE
(
 STARTYEAR IN VARCHAR2
, ENDYEAR IN VARCHAR2
, DETAILS OUT SYS_REFCURSOR
) AS
BEGIN

  open details for
    SELECT MID,MNAME,RATING,YEAR FROM SP_DATA_MOVIE WHERE YEAR>=STARTYEAR AND
YEAR<=ENDYEAR;

END P_GET_MOVIES_BY_YEAR_RANGE;
```

Add this code

(compile)

Calling PL/SQL PRocedure from spring data JPA Application

=>Working @Query methods of Repository Interface is legacy style (old approach)

=> working with EntityManager object is modren apporach given from hiberante 5.2 onwords so the same is continued in spring data jpa.

Using EntityManager object

=>if spring-data-jpa starter is added to Project we get EntityManager object through AutoConfiguration and it can be injected to service class or any other spring bean using Autowiring concept

=>EntityManager object encapsulates(holds) the functionality of HB SessionFactory,Session objs and provides env.. to perform multiple persistnece operations.

Example App

=> Make sure that pL/sQL procedure is ready in Oracle Db s/w

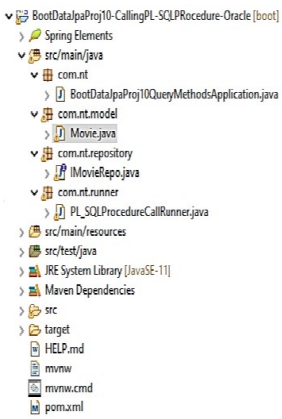
=> Create spring starter project adding spring-data jpa , ojdbc8 , lombok depndencies

=> develop user-defined Repository

```
public interface IMovieRepo extends JpaRepository<Movie, Integer> {

}
```

=> Develop model class

```
@Data
@Entity
@Table(name="SP_DATA_MOVIE")
@RequiredArgsConstructor
@NoArgsConstructor
@DynamicInsert(true)
public class Movie implements Serializable {
	@Id
	@Column(name="MID")
	@GeneratedValue(strategy = GenerationType.AUTO)
	private Integer mid;
	@Column(name="MNAME",length = 20)
	@NonNull
	private String mname;

	@Column(name="YEAR",length = 20)
	@NonNull
	//@Transient

	private String year;

	@Column(name="RATING")
	@NonNull
	private Float rating;
}
```



=> Develop service or runner class injecting EntityManager object

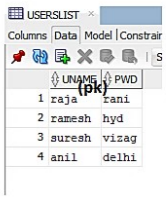
```
@Component
public class PL_SQLProcedureCallRunner implements CommandLineRunner {
	@Autowired
	 private EntityManager manager;

	@Override
	public void run(String... args) throws Exception {
		// crreate StoredProcedureQuery object representing the given PL/SQL procedure
		StoredProcedureQuery query=manager.createStoredProcedureQuery("P_GET_MOVIES_BY_YEAR_RANGE",Movie.
		//register parameters of PL/SQL procedure with java types
		query.registerStoredProcedureParameter(1, String.class, ParameterMode.IN);
		query.registerStoredProcedureParameter(2, String.class, ParameterMode.IN);
		query.registerStoredProcedureParameter(3, Movie.class, ParameterMode.REF_CURSOR);
		//set values to IN params
		query.setParameter(1, "1900");
		query.setParameter(2, "2021");

		//call PL/SQL Procedure and get thr results
		List<Movie> list=query.getResultList();
		list.forEach(System.out::println);
	}//run(-)

}//class
```

=> Run the Application..

Jan 20 Calling PL-SQL Procedures and functions

==========
Calling PL/SQL Procedure of oracle that is performing Authentication activity
==========

step1) create db table in oracle Db s/w having username, password details



step2) Place Authentication Logic in PL/SQL procedure of oracle

```
reate or replace PROCEDURE PRO_AUTHENTICATION
(
 USER IN VARCHAR2
, PASS IN VARCHAR2
, RESULT OUT VARCHAR2
) AS
  CNT NUMBER;   //Like local variable
BEGIN

  SELECT COUNT(*) INTO CNT FROM USERSLIST WHERE  UNAME=USER AND PWD=PASS;

  IF(CNT<>0) THEN
      RESULT:='VALID CREDENTIALS';
    ELSE
      RESULT:='INVALID CREDENTIALS';
     END IF ;
END PRO_AUTHENTICATION;
```

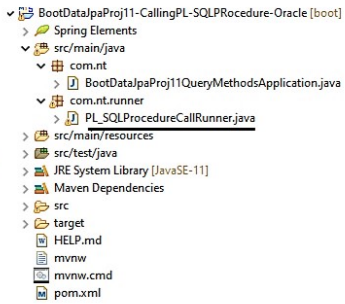
The count(*) function results will be stored into CNT variable.

<> --> means not equal to

step3) Inject EntityManager object to runner class or Service class..
write logic to call the above PL/SQL procedure .

=> Since example is not exception any record from any db table for display .. we can think about going ignoring model class development and Repository interface.

Runner class
==========

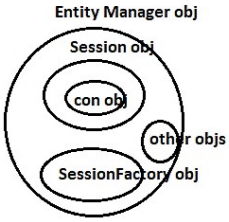



```
package com.nt.runner;

import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.ParameterMode;
import javax.persistence.StoredProcedureQuery;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

/*create or replace PROCEDURE PRO_AUTHENTICATION
(
 USER IN VARCHAR2
, PASS IN VARCHAR2
, RESULT OUT VARCHAR2
) AS
  CNT NUMBER;
BEGIN

  SELECT COUNT(*) INTO CNT FROM USERSLIST WHERE  UNAME=USER AND PWD=PASS;

  IF(CNT<>0) THEN
    RESULT:='VALID CREDENTIALS';
  ELSE
    RESULT:='INVALID CREDENTIALS';
   END IF ;
END PRO_AUTHENTICATION;
*/

@Component
public class PL_SQLProcedureCallRunner implements CommandLineRunner {

@Autowired
 private EntityManager  manager;

@Override
public void run(String... args) throws Exception {
//Create Stored ProcedureQuery object
 StoredProcedureQuery query=manager.createStoredProcedureQuery("PRO_AUTHENTICATION");
 //register parameters with  java  types
 query.registerStoredProcedureParameter(1, String.class,ParameterMode.IN);
 query.registerStoredProcedureParameter(2, String .class, ParameterMode.IN);
 query.registerStoredProcedureParameter(3, String .class, ParameterMode.OUT);
 //set values to IN params
 query.setParameter(1, "raja");
 query.setParameter(2,"rani1");
 //call PL/SQL procedure
  String result=(String)query.getOutputParameterValue(3);
  System.out.println("Authentication result::"+result);
}//run(-)

}//class
```

jan 22 Calling PL-SQL Procedures and functions

Calling PL/SQL function of oracle using Callback interfaces of Hibernate API

=> As of now there is no provision to call PL/SQL functions using Entity Manager .. But we can do indirectly with the support of Callback interface given by hibernate api.



Using EntityManager get access jdbc con obj through HB SEssion obj and use that con object to call PL/SQL function with the support CallableStatement object.

CallbackInterface

=>The interface whose methods will be called automatically is called callback interface

=>Callback method is method which will be called underlying server/container/framework automatically..

=> servlet life cycle methods are called callback methods.. becoz they will be called by servlet container automatically..

=> When the container calls callback methods automatically it will expose certain objects as the parameters of the method , So that we can use those objects to write customized logics..

=Hibernate's Callback interfaces are..

a) Work<T> (callback interface)
|----> void execute(Connection con) (callback method)
->does not return any thing — using con object write customized logics

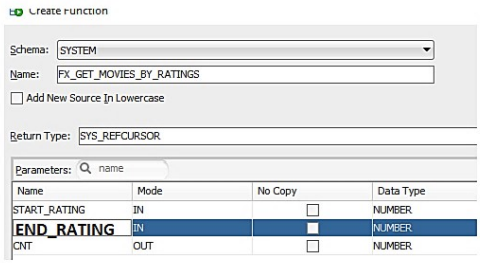
b) ReturningWork<T> (callback interface)
|---> T execute(Conneciton con) (callback method)
— using con object write customized logics
can return the output as needed.

```
@Autowired
private EntityManager manager;

Session ses=manager.unwrap(Session.class);

String result=ses.doReturningWork( new ReturningWork<String>(){

        public String execute(Connection con){
          ....
          ...  //write plain jdbc code
          ...   using the con object.
          ...
        }
    } );
```

this method is called automatically by hibernate having con object.

Example App on Calling PL/SQL Function of oracle using unwrapped jdbc con object

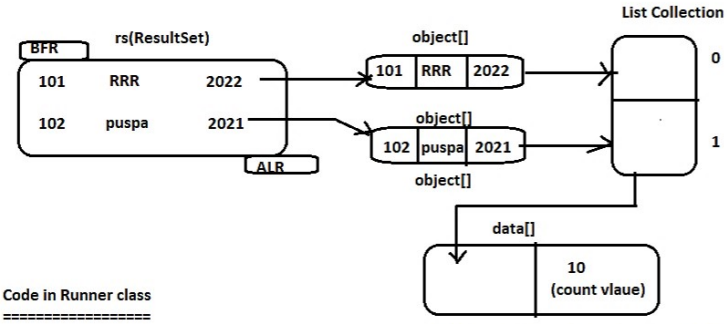
step1) place PL/SQL function is oracle Db s/w having scalar Query using SQL Developer




```
CREATE OR REPLACE FUNCTION FX_GET_MOVIES_BY_RATINGS
(
  START_RATING IN NUMBER
, END_RATING IN NUMBER          params
, CNT OUT NUMBER
) RETURN SYS_REFCURSOR AS       return type
DETAILS  SYS_REFCURSOR;  //Like local variable
BEGIN
  SELECT COUNT(*) INTO CNT FROM SP_DATA_MOVIE;   this result goes to out parameter

  OPEN DETAILS FOR
    SELECT MNAME,RATING,YEAR  FROM SP_DATA_MOVIE WHERE
    RATING>=START_RATING AND RATING<=END_RATING;

    RETURN DETAILS;   returning local varaible
                      satisfying the return type of PL/SQL function

    END FX_GET_MOVIES_BY_RATINGS;
```

Select Query results are going cursor variable

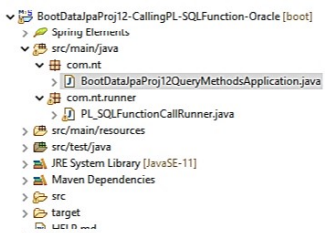
Jan 20.2 Calling PL-SQL Procedures and functions



Code in Runner class
==========

// Runner class

```
package com.nt.runner;
import java.sql.CallableStatement;
import java.sql.ResultSet;
import java.sql.Types;
import java.util.ArrayList;
import java.util.List;
import javax.persistence.EntityManager;
import org.hibernate.Session;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

 import oracle.jdbc.OracleTypes;

/*create or replace FUNCTION FX_GET_MOVIES_BY_RATINGS
(
  START_RATING IN NUMBER
, END_RATING IN NUMBER
, CNT OUT NUMBER
) RETURN SYS_REFCURSOR AS
  details SYS_REFCURSOR;
BEGIN
  SELECT COUNT(*) INTO CNT FROM SP_DATA_MOVIE;

  OPEN DETAILS FOR
    SELECT MNAME,RATING,YEAR  FROM SP_DATA_MOVIE WHERE
    RATING>=START_RATING AND RATING<=END_RATING;
 RETURN DETAILS;

END FX_GET_MOVIES_BY_RATINGS;
*/

@Component
public class PL_SQLFunctionCallRunner implements CommandLineRunner {
        @Autowired
        private EntityManager  manager;

        @Override
        public void run(String... args) throws Exception {
             //unwrap  SEsssion object from  EntityManager object
            Session ses=manager.unwrap(Session.class);
            //Work with  ReturnigWork<T> callback based Callback method
            Object results[]= ses.doReturningWork(con->{
                    //write  CallbackSTatement based logic to call PL/SQL function
             CallableStatement cs=con.prepareCall("{?=call FX_GET_MOVIES_BY_RATINGS(?,?,?)}");
              //register return(1) , out params(4) with jdbc types
              cs.registerOutParameter(1,OracleTypes.CURSOR);
              cs.registerOutParameter(4, Types.INTEGER);
             // set values to IN parameters
              cs.setFloat(2,4.0f);
              cs.setFloat(3,5.5f);
              //call  Pl/SQL function
              cs.execute();
              //gather results from output Parameters
              ResultSet rs=(ResultSet)cs.getObject(1);  // return value;
               List<Object[]> list=new ArrayList();
               while(rs.next()) {
                  Object record[]=new Object[3];
                  record[0]=rs.getString(1);
                  record[1]=rs.getString(2);
                  record[2]=rs.getString(3);
                  list.add(record);
               }//while
               Object data[]=new Object[2];
               data[0]=list;
               data[1]=cs.getInt(4);  //out param cnt value
              return data;
            });
                 //process the ressult[]
           System.out.println("movies in ratings range are::");
            List<Object[]> ratingsList=(List<Object[]>) results[0];
            ratingsList.forEach(md->{
                  for(Object val:md)
                        System.out.print(val+" ");
                  System.out.println();
            });
                  //get count of records
                  int count=(int) results[1];
                  System.out.println("records count::"+count);

              }//run(-)

      }//class
```



jan 24 Calling PL-SQL Procedure of Msyql and workign date,time values

Calling PL/SQL procedure of Mysql Db s/w from spring data jpa application

=> There is no support for Cursors in mysql PL/SQL programming.. In fact we do not need cursors in mysql PL/SQL programming.

==========
steps to create PL/SQL Procedure in MySQL Db s/w
==========

step1) Lauch mysql workbench and connect mysql DB s/w submitting username and password

step2) Keep Db table ready mysql Db s/w

Result Grid | Filter Rows

(pk) eno	ename	desg	salary
1	raja	CLERK	9000
2	ramesh	MANAGER	10000
3	suresh	CLERK	8000

step3) create PL/SQL Procedure in MySQL Db s/w

Go to logical Db ntspbms615db ---> right click stored procedure --> new procedure-->

```
CREATE PROCEDURE `P_GET_EMPS_BY_JOB` (IN  job1 varchar(20),
                                      IN  job2 varchar(20))
BEGIN
 select * FROM emp_profile where desg  in (job1,job2) order by desg;
END
```

* --added code to the template

apply

```
USE `ntspbms615db`;
DROP procedure IF EXISTS `P_GET_EMPS_BY_JOB`;

DELIMITER $$
USE `ntspbms615db`$$
CREATE PROCEDURE `P_GET_EMPS_BY_JOB` (IN  job1 varchar(20),
                                      IN  job2 varchar(20))
BEGIN
 select * FROM emp_profile where desg(job1,job2) order by desg;
END$$

DELIMITER ;
```

in mysql pl/sql programming if we are selecting all col values of db table then there is no need of working with OUT params and any cursor variables

* automatically added code

step4) create spring data jpa project adding
spring data jpa ,mysql driver , lombok starters.

step5) Add the following entries in application.properties related mysql

application.properties

```
# mysql DataSource details
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql:///ntspbms615db
spring.datasource.username=root
spring.datasource.password=root

# hibernate properties
spring.jpa.hibernate.ddl-auto=update
spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect
spring.jpa.show-sql=true
```

step6) Develop the Model or Entity class

```
@Data, @Entity
public class Employee {
	@Id
	@GeneratedValue
	private Integer eno;
	private String ename;
	private String desg;
	private float salary;
 }
```

step6) Develop Service class injecting and using EntityManager object to call PL/SQL procedure

//service Impl class

```
@Service("empService")
public class EmployeeMgmtService implements IEmployeeMgmtService {
 @Autowired
 private EntityManager manager;

 @Override
 public List<Employee> fetchEmpDetailsByDesgs(String desg1, String desg2) {
 //create StoredProcedureQuery object
 StoredProcedureQuery query=manager.createStoredProcedureQuery("P_GET_EMPS_BY_JOB",Employee.class);
 //register Parameters with JDBC types
 query.registerStoredProcedureParameter(1,String.class, ParameterMode.IN);
 query.registerStoredProcedureParameter(2, String.class, ParameterMode.IN);
 //set values to IN params
 query.setParameter(1, desg1);
 query.setParameter(2, desg2);
 //call the PL/SQL procedure and get the Result
 List<Employee> list=query.getResultList();
 return list;
 }
}
```

step7) Devleop Runner class injecting SErivce class object.

```
@Component
public class CallingPLSQLProcedureTestRunner implements CommandLineRunner {
	@Autowired
	private IEmployeeMgmtService service;

	@Override
	public void run(String... args) throws Exception {
		service.fetchEmpDetailsByDesgs("CLERK", "MANAGER").forEach(emp->{
			System.out.println(emp);
		});
	}
 }
```

jan 24.1 Calling PL-SQL Procedure of Msyql-working datetime values

Working date and time values

=> while inserting, retrieving DOB, TOB, DOJ,TOJ and etc.. we need to date,time values

DOB:: Date of Birth
TOB:: Time of Birth
DOJ:: Date of Joining
TOJ:: Time of Joining

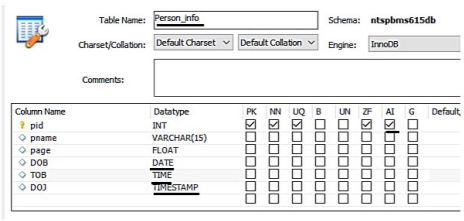
=> From Java 8 new date and time api is give
java.util.LocalDate --> To set/get date values
java.util.LocalTime --->To set/get time values
java.util.LocalDateTime --->To set/get date and time values

=>In oracle Db s/w the date,time related data types are
=> date (only date values), timestamp (date and time values)
=>In mysql Db s/w the date,time related data types are
=> date (only date values), time(only time vlaues) ,timestamp (date and time values)

In Entity class we need to take approprivate date type from date,time related property like LocalDate ,LocalDateTime, LocalTime and etc..

Example App

step1) create db table ready in mysql



step2) create spring boot project adding mysql driver, lombok, spring data jpa starters.

step3) devleop application.properties for mysql Db s/w

application.properties

```
#DataSource cfg
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql:///ntspbms615db
spring.datasource.username=root
spring.datasource.password=root

# JPA-hibernate cfgs
spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
```

step3) Develop the Entity class as java8 data ,time type properties

```
@Data, @Entity
public class PersonInfo {
    @Id
    @GeneratedValue
    private Integer pid;
    private String pname;
    private Float page;
    private LocalDate dob;
    private LocalTime tob;
    private LocalDateTime doj;
}
```

step4) Develop the Repository Interface

```
public interface PersonInfoRepo extends JpaRepository<PersonInfo,Integer> {  }
```

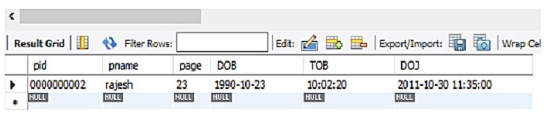
step5) Devleop the service class injecting the Repository object

```
//service Interface
public interface IPersonInfoMgmtService {

    public String registerPErson(PersonInfo info);
    public List<PersonInfo> fetchAllPersonDetails();
}
```

```
//service Impl class
----------------------------
@Service("personService")
public class PersonInfoMgmtServiceImpl implements IPersonInfoMgmtService {
    @Autowired
    private IPersonInfoRepo repo;
    @Override
    public String registerPErson(PersonInfo info) {
        return repo.save(info).getPid() +"  Person is saved";

    }
    @Override
    public List<PersonInfo> fetchAllPersonDetails() {
    return repo.findAll();
    }
}
```

step6) Develop the runner class injecting Service class obj

```
@Component
public class DateTimeValuesRunnerTest implements CommandLineRunner {
    @Autowired
    private IPersonInfoMgmtService service;
@Override
public void run(String... args) throws Exception {
    PersonInfo info=new PersonInfo();
    info.setPname("rajesh"); info.setPage(23.0f); info.setDob(LocalDate.of(1990,10,23));
    info.setTob(LocalTime.of(10, 2, 20)); info.setDoj(LocalDateTime.of(2011,10, 30, 11,35));
    System.out.println(service.registerPErson(info));
    System.out.println("-----------------------");
    service.fetchAllPersonDetails().forEach(System.out::println);

    }
}
```



jan 26 Working with Large Objects

Working with Larget object
==========

=>Large objects(LOB) mean Files like image files, audio files, video files, text file , movie files and etc..
=> UseCase:: Matrimony Apps, job portal Apps, profile mgmt Apps, Social networking Apps and etc..

=> All major Db s/w gives support for Larget obejcts
=>oracle gives BLOB, CLOB Data types suporting LOBs
=> mysql givesn TINYBLOB , BLOB , MEDIUMBLOB , LONGBLOB and
TinyText , Text , medium text , Long text

=> BLOB files means binary files internally 0,1 (images, audio files, video files, word docs and etc..)
=> CLOB files means character files internally text content (text files, ritch text fiels,csv files and etc..)

=> we need to take byte[] type property in Entity class for blob content and char[] proerty in entity
for clob content. (Both the properties in entity class must be annotated with @Lob annotation

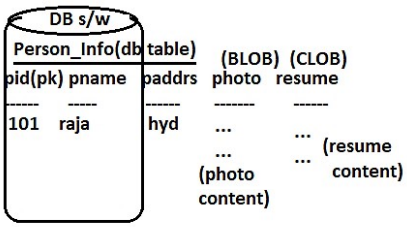
=> if we take boolean property in the Entity class.. the relavent db table col stores 0(false) , 1(true)

=> Only while devleoping standalone Apps we store LObs(files) directly in Db table cols

=> while devleoping web applications, Distributed Apps we generally avoid storing LOBS(files) in
Db table cols .. we place them in server machine file system and we place files paths as string
values in db table cols or we store them in seperate special softwares like CDN (content delivery network),
DMS (Document Management System), s3 Bucket , NAS(Network Aattched SErver) and etc.. places later we place
the generated tokesn for file from the above software in Db s/w.
each &

In Standalone Apps

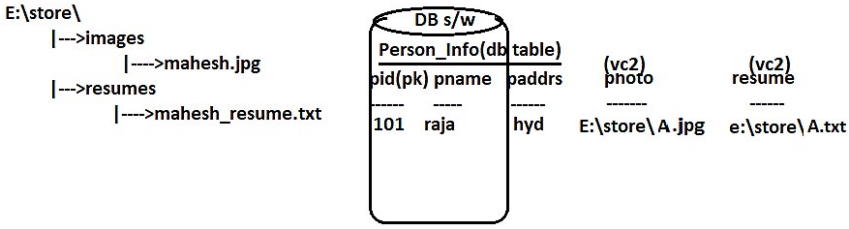
==========

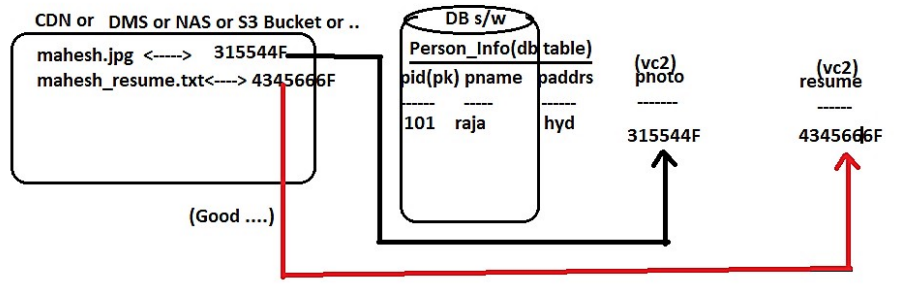


In web applications /Distributed App

==========

server machine file sytem (where web server or app server is running)





Example App (standalone App inserting LOBs)
==========

=>create project adding lombok , spring data jpa , mysql driver starters..

=> the the following entries in application.properties

```
#DataSource cfg
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql:///ntspbms615db
spring.datasource.username=root
spring.datasource.password=root

# JPA-hibernate cfgs
spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
```

jan 26.1 Working with Large Objects

=> Develop the entity class

PersonInfo.java

```
package com.nt.entity;

import java.io.Serializable;
import java.time.LocalDateTime;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Lob;
import javax.persistence.Table;

import lombok.AllArgsConstructor;
import lombok.Data;

@Entity
@Table(name="Person_Info_lob")
@Data
@AllArgsConstructor
public class PersonInfo implements Serializable{
        @Id
        @GeneratedValue
        private Integer pid;
        @Column(length=20)
        private String pname;
        private LocalDateTime dob;
        private  boolean married;
        @Lob
        private  byte[]  photo;
        @Lob
        private  char[]   resume;

 }
```

Why it is recommanded to implement java.io.Serializable Interface on Model class?

a) Nowdays we are using model/Entity class it self as DTO class i.e we are sendign data from 1 project to another project in the form of Model class obj , So Model class obj data shippable over the network we need the support of Serialization

b) While working with hibernate directly or indirectly there is possbility enabling L2 cache/Level2 cache which supports Disk Caching i.e entity classs object's data can be written to disk memory (HDD) using Serialization concept , For this we need to take Entity calss as Serialziable class by implementing java.io.Serilizable(I).

=> Develope Repository Interface

```
public interface IPersonInfoRepo extends JpaRepository<PersonInfo,Integer> {

}
```

=> Develop Service Interface and Servce Impl class

service Interface

```
public interface IPersonInfoMgmtService {

        public  String registerPErson(PersonInfo info);
        public   List<PersonInfo> fetchAllPersonDetails();

}
```

Service Impl class

```
@Service("personService")
public class PersonInfoMgmtServiceImpl implements IPersonInfoMgmtService {
        @Autowired
        private IPersonInfoRepo repo;

        @Override
        public String registerPErson(PersonInfo info) {
            return  repo.save(info).getPid() +"    Person is saved";

        }

        @Override
        public List<PersonInfo> fetchAllPersonDetails() {
       return  repo.findAll();
        }

}
```

jan 26.2 Working with Large Objects

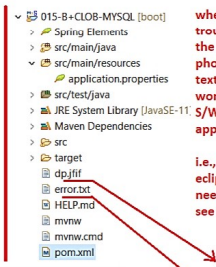
=>Deveope the Runner class

```
@Component
public class LobsTestRunner implements CommandLineRunner {
	@Autowired
	private IPersonInfoMgmtService service;

	@Override
	public void run(String... args) throws Exception {
	Scanner sc=new Scanner(System.in);
	System.out.println("enter name::");
	String name=sc.next();
	System.out.println("Is MArried::?");
	 boolean married=sc.nextBoolean();
	 System.out.println("Enter  photoPath");
	 String photoPath=sc.next();
	 System.out.println(photoPath);
	 System.out.println("Enter  resume Path");
	 String resumePath=sc.next();
	 System.out.println(resumePath);

	// create byte[]  representing photo file content
	 InputStream is=new FileInputStream(photoPath);
	byte[] photoContent=new byte[is.available()];
	photoContent=is.readAllBytes();

	// create char[]  representing resume file content
	Reader reader=new FileReader(resumePath);
	File file=new File(resumePath);
	char[] resumeContent=new char[(int)file.length()];
	reader.read(resumeContent);
	//prepare with Entity  object with       data
	PersonInfo info=new PersonInfo(-1, name, LocalDateTime.of(1990, 10, 20, 11, 23),
	                                married, photoContent, resumeContent);

	try {
		System.out.println(service.registerPErson(info));
	}

	catch(Exception e) {
		e.printStackTrace();
	}
	}
	}
```

Generated db table

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/E
pid	INT	☑	☑	☐	☐	☐	☐	☐	☐	
dob	DATETIME(6)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
married	BIT(1)	☐	☑	☐	☐	☐	☐	☐	☐	
photo	LONGBLOB	☐	☐	☐	☐	☐	☐	☐	☐	NULL
pname	VARCHAR(20)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
resume	LONGTEXT	☐	☐	☐	☐	☐	☐	☐	☐	NULL

```
My code in Runner class:-
 ========================

@Override
public void run(String... args) throws Exception {
String photo = "dp.jfif";
String resume = "error.txt";

// create byte[] representing photo file content
InputStream is = new FileInputStream(photo);
byte[] photoContent = new byte[is.available()];
photoContent = is.readAllBytes();

// create char[] representing resume content

Reader reader = new FileReader(resume);
File file = new File(resume);
char[] resumecontent = new char[(int) file.length()];
reader.read(resumecontent);

// prepare entity object with data
PersonInfo info = new PersonInfo();
// info.setPid(null);//generator is configured
info.setPname("ss");
info.setMarried(true);
info.setDob(LocalDateTime.of(1994, 06, 15, 11, 23));
info.setPhoto(photoContent);
info.setResume(resumecontent);
try {
	System.out.println(service.registerPerson(info));
} catch (Exception e) {
	e.printStackTrace();
}
```

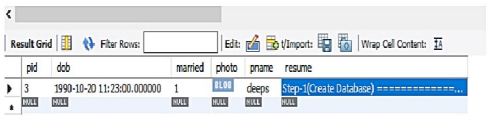


when you are getting the trouble while uploading the files like images, photos, video etc,... and text files like pdf, text, word docs etc... with DB S/W you simply use the approach

i.e., just copy the files to eclipse project that you need to upload to DB S/W see side eclipse project

its a image file

its a text doc



Result Grid | Filter Rows: | Edit: | t/Import: | Wrap Cell Content:

pid	dob	married	photo	pname	resume
3	1990-10-20 11:23:00.000000	1	BLOB	deeps	Step-1(Create Database) ==========...
NULL	NULL	NULL	NULL	NULL	NULL

==========

Retrieving LOBs from Db table

==========

=>we get byte[] from Db table through Entity class obj representing BLOB col data ,So we need to write byte[] content to file by taking the support Outputstream (stream)

=>we get char[] from Db table through Entity class obj representing CLOB col data ,So we need to write char[] content to file by taking the support Writer (stream)

code in Repository interface

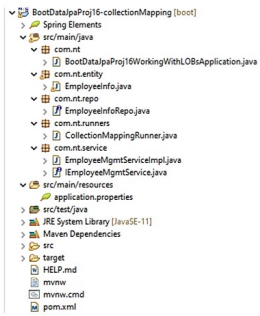
```
public interface IPersonInfoRepo extends JpaRepository<PersonInfo,Integer> {  }
```

code in service Interface and impl class

```
public interface IPersonInfoMgmtService {

	public  String registerPErson(PersonInfo info);
	public  PersonInfo fetchPersonDetailsById(int pid);
}
```

In service Impl class

```
@Service("personService")
public class PersonInfoMgmtServiceImpl implements IPersonInfoMgmtService {
	@Autowired
	private IPersonInfoRepo repo;

	@Override
	public String registerPErson(PersonInfo info) {
	    return  repo.save(info).getPid() +"   Person is saved";

	}

	@Override
	public PersonInfo fetchPersonDetailsById(int pid) {
	   Optional<PersonInfo> opt=repo.findById(pid);
	    if(opt.isPresent())
	        return  opt.get();
	    else
	        return null;
	}

}
```



Code in runner class (run(-)) method)

```
System.out.println("====================================");
		try {
		//Logic to retrieve the object
		PersonInfo info=service.fetchPersonDetailsById(3);
		 if(info!=null) {
			System.out.println(info.getPid()+" "+info.getPname()+" "+info.getDob());
			byte[] photoContent=info.getPhoto();
			OutputStream os=new FileOutputStream("retrieve_photo.gif");
			os.write(photoContent);
			os.flush();
			os.close();
			System.out.println("Photo retrieved from db table col");
			char[]  resumeContent=info.getResume();
			Writer writer=new FileWriter("retrive_resume.txt");
			writer.write(resumeContent);
			writer.flush();
			writer.close();
			System.out.println("Resume retrieved from db table col");
	}//if
		else {
			System.out.println("record not found");
		 }
		}//try
		catch(Exception e) {
			e.printStackTrace();
		}
```

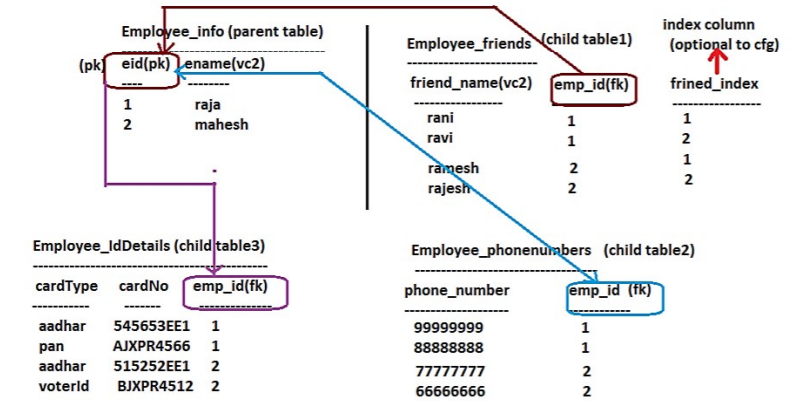
jan 29 Collection Mapping-AssociationMapping

Working Collection Mapping

=>It is all about taking collection types properties in the entity class like List ,Set , Map properties and stroing their multiple element values in child table linked with parent table mapped by Entity class by forming the one to many association using FK column

Employee
- eid (int)
- ename (String)
- friends (java.util.List)
-phoneNumbers (java.util.Set)
-idCardDetails (java.util.Map)

```
public class EmployeeInfo {

    private int eid;
    private String ename;
    private  List<String> friends;
    private Set<Long>  phoneNumbers;
    private Map<String,String> idDetails;
}
```



=>Collection mapping speaks about taking simple type element values to different Collection properties and inserting them child tables that maintains assocation with parent table mapped by Entity classs using FK column

=>Assocation mapping/relationships in entity classes is the extension of collection Mapping.

Annotationsof CollectionMapping

@ElementCollection :: enables the collection mapping i.e takes the element values of given Collection property and inserts them in chilld db table as col values.

@JoinColumn :: To specify FK colum name of child table linked pk colum of parent db table

@JoinTable :: To specify child table name

@OrderColumn :: To sepcifyIndex column while working List Collection

@ListIndexBase :: In List collection related index value insertion to index column it specifies the start/base value index

@MapKeyColumn :: Allows to specify index column while working Map Collection to store keys of the map elements to that column as index values.

Example Application:-

Entity class

```
package com.nt.entity;

import java.util.List;
import java.util.Map;
import java.util.Set;

import javax.persistence.CollectionTable;
import javax.persistence.Column;
import javax.persistence.ElementCollection;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.MapKeyColumn;
import javax.persistence.OrderColumn;
import javax.persistence.Table;

import org.hibernate.annotations.ListIndexBase;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Entity
@Table(name="EMPLOYEE_COLLECTION_INFO")
@Data
@AllArgsConstructor
@NoArgsConstructor
public class EmployeeInfo {
@Id
@GeneratedValue
private Integer eid;

@Column(length = 20)
private String ename;

   @ElementCollection
   @CollectionTable(name="EMPLOYEE_FRIENDS",joinColumns = @JoinColumn(name="EMP_ID",referencedColumnName = "EID"))
   @OrderColumn(name = "FRIEND_INDX")
   @ListIndexBase(value = 1)
   @Column(name="FRIEND_NAME", length = 20)
   private  List<String> friends;

 @ElementCollection
 @CollectionTable(name="EMPLOYEE_PHONE_NUMBERS",
                  joinColumns = @JoinColumn(name="EMP_ID",referencedColumnName = "EID"))
 @Column(name="PHONE_NUMBER", length = 11)
 private Set<Long>  phoneNumbers;

@ElementCollection
@CollectionTable(name="EMPLOYEE_ID_DETAILS",joinColumns = @JoinColumn(name="EMP_ID",referencedColumnName = "EID"))
@MapKeyColumn(name = "CARD_TYPE",length = 20)
@Column(name="CARD_NUMBER",length = 20)
private Map<String,String> idDetails;

}
```

jan 29.1 Collection Mapping-AssociationMapping

Repository Interface

```
package com.nt.repo;

import org.springframework.data.jpa.repository.JpaRepository;
import com.nt.entity.EmployeeInfo;
public interface EmployeeInfoRepo extends JpaRepository<EmployeeInfo, Integer> {   }
```

Service Interface

```
public interface IEmployeeMgmtService {
        public String registerEmployee(EmployeeInfo info);
        public  List<EmployeeInfo> getAllEmpDetails();
}
```

service Impl class

```
@Service("empService")
public class EmployeeMgmtServiceImpl implements IEmployeeMgmtService {
        @Autowired
        private  EmployeeInfoRepo repo;
        @Override
        public String registerEmployee(EmployeeInfo info) {
                return "Employee saved with"+ repo.save(info).getEid()+" id value ";
        }
        @Override
        public List<EmployeeInfo> getAllEmpDetails() {
                return repo.findAll();
        }
 }
```

application.properties

```
#DataSource cfg
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql:///ntspbms615db
spring.datasource.username=root
spring.datasource.password=root
spring.jpa.properties.hibernate.enable_lazy_load_no_trans=true

# JPA-hibernate cfgs
spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
```

In collection mapping, association mapping the HB f/w generally lods parent records eagerly and child table records lazily i.e the child table records will loaded on demand basis .. For this some Transaction mgmt support is required.. To perform same lazy loading with out Transaction management support add this property.

code in runner class

```
@Component
public class CollectionMappingRunner implements CommandLineRunner {
        @Autowired
        private IEmployeeMgmtService service;

        @Override
        public void run(String... args) throws Exception {
          //save object
/*              try {
                        //prepare object
                         EmployeeInfo info=new EmployeeInfo(123, "raja",
                                                                List.of("rani","ravi","suresh"),
                                                                Set.of(99999L,88888L),
                                                                Map.of("aadhar","5453535","voterId","45354555")
                                                                );
                         System.out.println(service.registerEmployee(info));
                }//try
                catch(Exception e) {
                        e.printStackTrace();
                } */
                System.out.println("---------------all  emp ---details are--------------");
                service.getAllEmpDetails().forEach(System.out::println);

        }//main

}//class
```

In AssociationMapping, CollectionMapping

lazy loading means:: parent records will be loaded normally/eagerly and associated child records will be loaded lazily on demand (default)

eager loading means:: parent records will be loaded normally/eagerly along with these records the associated child records will be loaded for this we need to place `@Lazy` Collection(-------.FALSE) the on collection type properties

```
@Entity
@Table(name="EMPLOYEE_COLLECTION_INFO")
@Data
@AllArgsConstructor
@NoArgsConstructor
public class EmployeeInfo {
 @Id
 @GeneratedValue
 private Integer eid;

 @Column(length = 20)
 private String ename;

 @ElementCollection
 @CollectionTable(name="EMPLOYEE_FRIENDS",joinColumns = @JoinColumn(name="EMP_ID",referencedColumnName = "EID"))
 @OrderColumn(name = "FRIEND_INDX")
 @ListIndexBase(value = 1)
 @Column(name="FRIEND_NAME", length = 20)
 @LazyCollection(LazyCollectionOption.FALSE)
 private  List<String> friends;

 @ElementCollection
 @CollectionTable(name="EMPLOYEE_PHONE_NUMBERS",
                        joinColumns = @JoinColumn(name="EMP_ID",referencedColumnName = "EID"))
 @Column(name="PHONE_NUMBER", length = 11)
 @LazyCollection(LazyCollectionOption.FALSE)
 private Set<Long>  phoneNumbers;

 @ElementCollection
 @CollectionTable(name="EMPLOYEE_ID_DETAILS",joinColumns = @JoinColumn(name="EMP_ID",referencedColumnName = "EID"))
 @MapKeyColumn(name = "CARD_TYPE",length = 20)
 @Column(name="CARD_NUMBER",length = 20)
 @LazyCollection(LazyCollectionOption.FALSE)
 private Map<String,String> idDetails;

 }
```

jan 31 Association Mapping-AssociationMapping

Association mapping in hibernate | Relationships in hibernate

Need of Associating mapping

problem:

Storing multiple entities/items data in single db table is going to the following two problems

(a) Data Redundency problem (Data Duplication problem)

(b) Data Maintainance problem

if we store Person and his phone number details in single db table .. the db table looks lik this.

PERSON_PHONE_DETAILS (single db table)

pid	pname	paddrs	phone_number	number_type	provider
101	raja	hyd			
101	raja	hyd			
102	rajesh	vizag			
102	rajesh	vizag			

Data Redundency/Duplication Problem

Solution1:: keep two entities/items data in two different db tables

(not recomanded)

PERSON_DETAILS (DB table1)

pid	pname	paddrs
101	raja	hyd
102	rajesh	vizag

PHONE_NUBER_DETAILS (DB table2)

phone_number	number_type	provider
9999999999	personal	airtel
8888888888	office	jio
7777777777	personal	VI
6666666666	office	BSNL

=> Data Redundency problem is solved... but data navigation problem is created i.e we can not access phone number details from Person _details table and vice-versa becoz there is no FK column based relatioship between the db tables.

Solution2: Keep two entities/items data in two different db tables having FK column based relationship b/w the db tables.

(Not Recomanded)

PERSON_DETAILS (parent db table)

pid	pname	paddrs	ph_regNo (Fk)
101	raja	hyd	1
101	raja	hyd	2
102	rajesh	vizag	3
102	rajesh	vizag	4

regNo(pk)
1
2
3
4

PHONE_NUBER_DETAILS (child table)

phone_number	number_type	provider
9999999999	personal	airtel
8888888888	office	jio
7777777777	personal	VI
6666666666	office	BSNL

=>This solves the data navigation problem becoz of FK column support i.e using FK column of parent db table we can access child db table reccords from parent db table and vice-versa

=>But here data Redundency is regeneated .. So to overcome this problem place FK column in child db table.

Solution3: Keep two entities/items data in two different db tables and keep in relationship by adding FK column in the child db table as shown below.

(Recomanded)

PERSON_DETAILS (parent db table)

pid (pk)	pname	paddrs
101	raja	hyd
102	rajesh	vizag

PHONE_NUBER_DETAILS (child table)

phone_number (pk)	number_type	provider	person_id(FK)
9999999999	personal	airtel	10 1
8888888888	office	jio	101
7777777777	personal	VI	102
6666666666	office	BSNL	102

=>Here there is not data redundency problem and data navigation problem.. So this is best solution..

=>The possible associations between db table are

a) 1 to 1 b) 1 to M c) M to 1 d) M to M

=>if we keep db tables in relationships the relavent entity classes of ORM should also kept in same relationship But problem is there is no FK support in programming languages (like java ,c++ and every where)... So we need to build Association b/w parent and child entity classes using HAS-A properties (Composition). We can take two types of HAS-A properties for this Associating Mapping in Entity classes.

a) Collection Type HAS-A property
(For 1 to M and M to M Association Mapping)

b) Non-Collection Type HAS-A property (Simple Reference type of other class)
(For M to 1 and 1 to 1 Association Mapping)

note:: The way we keep db tables in relationship using FK is totally different from the way we keep java entity classes in relationship using HAS-A properties .. So solve this mismatch we need special mapping called Association Mapping which gives instructions to ORM framework to map/link db tables in relationship with the java classes in relationship.

The Association Mapping annotations are

@OneToOne
@OneToMany
@ManyToOne
@ManyToMany
@JoinColum --> To specify FK column
@JoinTable ---> To join table
and etc.

feb 01 Association Mapping-OneToMany Association

we can build Associations in two modes

a) Uni-Directional
(Either parent to child or child to parent navigation possible)
b) Bi-Directional
(Both parent to child and child to parent navigation possible)

note: By keeping single FK column either in parent db table or in child db table (recomanded) we can keep db tables in both uni-directional association and bi-directional association.. But Entity classes for uni-directional association should be designed differently and the entity classes for bi-directional association should also be designed seperately.

=>For UniDirectional Association either parent class or child class contains HAS-A property to hold associated objects
=>For Bi-Directional Association both parent class and child class contain HAS-A properties to hold associated objects.

One to Many Uni-Directional Association (parent to child)
==========

=> This says 1 parent obj should have multi child objects , For that the parent class should have collection type property to hold bunch of child class objects ..

usecase :: User ==◊PhoneNumber
Person ==◊BankAccount
Person ===◊IdDetails

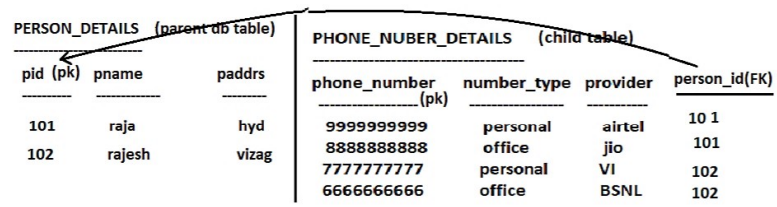
step1) create spring boot starter project addding the following dependencies
spring data jpa, lombok, mysql

step2) add entries in application.proeprties to connection mysql Db s/w

application.properties

```
#DataSource cfg
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql:///ntspbms615db
spring.datasource.username=root
spring.datasource.password=root


spring.jpa.properties.hibernate.enable_lazy_load_no_trans=true

# JPA-hibernate cfgs
spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
```

step3) develop parent and child entity classes. by keeping the following db tables in mind
(Infact they will be generated dynamically)




```
//parent class
package com.nt.entity;

import java.io.Serializable;
import java.util.Set;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.OneToMany;
import javax.persistence.SequenceGenerator;
import javax.persistence.Table;

import lombok.Getter;
import lombok.Setter;
@Entity
@Table(name="AM_PERSON_DETAILS")
@Setter
@Getter
public class Person implements Serializable {
 @Id
 @SequenceGenerator(name = "gen1",initialValue = 1000,allocationSize = 1,sequenceName = "PID_SEQ1")
 @GeneratedValue(generator = "gen1",strategy =GenerationType.SEQUENCE )
 private  Integer pid;
 @Column(length = 20)
 private String  pname;
 @Column(length = 20)
 private  String  paddrs;
 @OneToMany(targetEntity = PhoneNumber.class,cascade = CascadeType.ALL,fetch = FetchType.LAZY)
 @JoinColumn(name = "PERSON_ID",referencedColumnName = "PID")
 private Set<PhoneNumber> phonesInfo;

 @Override
 public String toString() {
       return "Person(parent) [pid=" + pid + ", pname=" + pname + ", paddrs=" + paddrs + "]";
 }

}
```

Indicates all non-select persistence operations performed on the parent obj should be propagated to associated child objs

PK column

FK coumn

indicates the parent objs should be loaded normally but the associated child objects should loaded lazily on demand basis.

note: Do not place @Data on the top of entity calsses in Association mapping becoz it gives orverridden hashCode() , equals() methods.. and they give problems towards storing objs in collections.

feb 01.2 Association Mapping-OneToMany Association

```
package com.nt.entity;
import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;
import lombok.Getter;
import lombok.Setter;

@Entity
@Table(name="AM_PHONENUMBERS_DETAILS")
@Setter
@Getter
public class PhoneNumber implements Serializable {
@Id
@GeneratedValue
private Integer regNo;
private  Long  phoneNumber;
@Column(length = 20)
private  String numberType;
@Column(length = 20)
private  String provider;

  //taking property for FK column any db table is optional
@Override
public String toString() {
return "PhoneNumber(child) [regNo=" + regNo + ", phoneNumber=" + phoneNumber + ", numberType=" + numberType
            + ", provider=" + provider + "]";
}
```

note:: Do not design Entity classes based on the location of FK column in db tables .. We generally take FK column in db tables @ child table .. But in entity classes

-> if it is parent to child uni-directional assocation then place special property in parent class
-> if it is child to parent uni-directional assocation then place special property in child class
-> if it is child to parent bi-directional assocation then place special properties in both classes.

step4) Develop Repository interfaces for both entity classes.

```
public interface PersonRepo extends JpaRepository<Person, Integer> {
}
```

```
public interface PhoneNumberRepo extends JpaRepository<PhoneNumber, Integer> {
}
```

step5) Develop ServiceInteface and ServiceImpl class.

```
//Service Interface
----------------------
package com.nt.service;

import com.nt.entity.Person;

public interface IOneToManyMappingMgmtService {
    public  String  saveDataUsingParent(Person person);
}

//service Impl lcass
--------------------------
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.nt.entity.Person;
import com.nt.repo.PersonRepo;
import com.nt.repo.PhoneNumberRepo;

@Service("otmService")
public class OneToManyMappingMgmtServiceImpl implements IOneToManyMappingMgmtService {
        @Autowired
        private  PersonRepo  personRepo;
        @Autowired
        private  PhoneNumberRepo phoneRepo;

        @Override
        public String saveDataUsingParent(Person person) {

                return personRepo.save(person).getPid()+"  Person  is saved";
        }
}
```

step6) Develop the runner class

```
@Component
public class OneToManyMappingTestRunner implements CommandLineRunner {
        @Autowired
        private  IOneToManyMappingMgmtService service;

        @Override
        public void run(String... args) throws Exception {
                //prepare child objects
                PhoneNumber ph=new PhoneNumber();
                ph.setContactNo(9999999999L); ph.setNumberType("office");
                ph.setProvider("airtel");
                PhoneNumber ph1=new PhoneNumber();
                ph1.setContactNo(88888888L); ph1.setNumberType("residence");
                ph1.setProvider("VI");
                //prpeare parent object
                Person  per=new Person();
                per.setPname("raja"); per.setPaddrs("hyd");
                per.setPhonesInfo(Set.of(ph,ph1));
                try {
                        //invoke method
                        System.out.println(service.saveDataUsingParent(per));
                }//try
                catch(Exception e) {
                        e.printStackTrace();
                }
          }
    }
```

step7) Run the Application..

feb 03 Association Mapping-Select ,Delete Operations

Loading Operation in Association Mapping (One to Many)

=> In One To many Association the parent objs will be loaded normally .. but the associated child objs will be loaded lazily. (On Demand Basis)

In service Interface

```
public interface IOneToManyMappingMgmtService {
    public  String   saveDataUsingParent(Person person);
    public  void   loadDataUsingParent();
}
```

In Service Impl class

```
@Service("otmService")
public class OneToManyMappingMgmtServiceImpl implements IOneToManyMappingMgmtService {
        @Autowired
        private  PersonRepo   personRepo;
        @Autowired
        private  PhoneNumberRepo phoneRepo;

        @Override
        public String saveDataUsingParent(Person person) {

                return personRepo.save(person).getPid()+"  Person  is saved";
        }

    @Override
    public void loadDataUsingParent() {
       List<Person> list=personRepo.findAll();
       list.forEach(per->{
            System.out.println("parent::"+per);
            //get childs of each parent
            Set<PhoneNumber>  childs=per.getPhonesInfo();
            childs.forEach(ph->{
                  System.out.println("child::"+ph);
            });
       });

    }//method

}//class
```

In runner class

```
public class OneToManyMappingTestRunner implements CommandLineRunner {
        @Autowired
        private  IOneToManyMappingMgmtService service;

        @Override
        public void run(String... args) throws Exception {
          service.loadDataUsingParent();
        }
}
```

The default fetch type in One To Many Association Lazy i.e the parent objects will laded normally but the associated child objects will be loaded lazily (one demand basis)

```
@OneToMany(targetEntity = PhoneNumber.class,cascade = CascadeType.ALL,fetch = FetchType.LAZY)
        @JoinColumn(name = "PERSON_ID",referencedColumnName = "PID")
        private Set<PhoneNumber> phonesInfo;
```

default is LAZY

The earger loading in One To Many Assoication is the child object will be loaded along with parent objs

```
@OneToMany(targetEntity = PhoneNumber.class,cascade = CascadeType.ALL,
                                                    fetch = FetchType.EAGER)
        @JoinColumn(name = "PERSON_ID",referencedColumnName = "PID")
        private Set<PhoneNumber> phonesInfo;
```

Note::

CascadeType is for propagating/cascading non-select persistence operations performed on the Parent object to the associated child objs and FetchType is related to performing select operations.

Delete Operation in One to many Association

=> if we delete parent object , the associated child obejcts will be automatically becoz we generaly take the cascadeType as ALL

code on deleting parent and its child objs

In service Interface

```
public  String   deleteParentAndItsChilds(int pid);
```

In service Impl class

```
@Override
public String  deleteParentAndItsChilds(int pid) {
 //Load parent object
Optional<Person> opt=personRepo.findById(pid);
  //Here Along with Parent object the associated child objects will be loaded.
if(opt.isPresent()) {
personRepo.delete(opt.get());  //if we delete parent obj.. the associated child object will be  deleted
return "Person  and his PhoneNumbers are deleted";

 }
 else {
        return " Person not found";
 }
 }
```

orphan cols means unrelated records which are not linked to parent

In run(-) method of Runner class

```
service.deleteParentAndItsChilds(1001);
```

Deleting all childs of a Parent

code in service Interface

```
public  String  deleteAllPhoneNumbersOfAPerson(int pid);
```

code in service Impl class

```
@Override
        public String deleteAllPhoneNumbersOfAPerson(int pid) {
                //Load Parent object
                Optional<Person> opt=personRepo.findById(pid);
                if(opt.isPresent()) {
                        //get childs of a parent
                        Set<PhoneNumber> childs=opt.get().getPhonesInfo();
                          /*  childs.forEach(ph->{
                                  phoneRepo.delete(ph);
                            });*/                    (or)
                        phoneRepo.deleteAll(childs);
                        return  "All childs(PhoneNumbers) of a Person are deleted";
                }
                else {
                        return  "Person not found";
                }
        }
```

In run(-) method Runner class

```
service.deleteAllPhoneNumbersOfAPerson(1000);
```

feb 04 HQL-JPQL Joins

HQL/JPQL Joins in spring data JPA (hibernate)

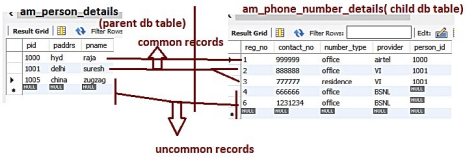
=>Joins are given as implicit conditions to get common and /or un common data from the db tables of relationship

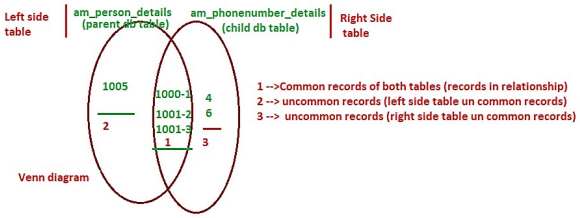
=>To use HQL/JPQL JOINS the db tables and the respective entity classes must in relationship..

=> On the top of join type implicit conditions , we can add more other explicit conditions if needed.

=> HQL/JPQL support 4 types of joins

a) INNER JOIN
b) LEFT OUTER JOIN / LEFT JOIN
c) RIGHT OUTER JOIN / RIGHT JOIN
d) FULL JOIN / RIGHT JOIN





a) INNER JOIN

Gives common records of both left side and right side db tables (---1-- area records)

b) LEFT OUTER JOIN / LEFT JOIN

Gives common records of both db tables and also gives uncommon records of left side table
(--- 1 --- + ---2 --- area records)

c) RIGHT OUTER JOIN / RIGHT JOIN

Gives common records of both db tables and also gives uncommon records of right side table
(--- 1 --- + ---3 --- area records)

d) FULL JOIN / RIGHT JOIN

Gives common and uncommon records of both db tables
(--- 1 --- + ---3 --- + ----2 --- area records)

Syntax of HQL JOIN Query (parent to child)

```
SELECT [properties of parent class, child class] from
<Parent class> as <alias name> <join type> <parent class HAS-A property><alias name> <addtional condition>
```

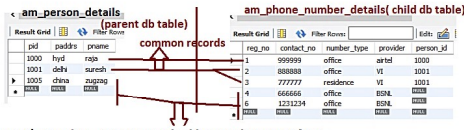
- `<Parent class>` : represents parent db table as left side db table
- `<join type>` : inner join / right join / left join / full join
- `<parent class HAS-A property>` : represents child db table as right side db table

=> As of now there is no propper support for SQL full join mysql Db s/w.. So we can not work with HQL full join in mysql DB s/w.. (full joins are possible in oracle Db s/w)

Example App

step1) keep any Association Mapping example App ready

step2) make sure that the parent and child db table are having both common records and uncommon records



=> Entity classes are same as previous App

step3) Develop @Query method in repository Interface.

```
public interface PersonRepo extends JpaRepository<Person, Integer> {
//@Query("SELECT p.pid,p.pname,p.paddrs,ph.regNo,ph.contactNo,ph.numberType,ph.provider
                                        from Person p  inner join  p.phonesInfo ph")
//@Query("SELECT p.pid,p.pname,p.paddrs,ph.regNo,ph.contactNo,ph.numberType,ph.provider
                                        from Person p  right join  p.phonesInfo ph")
 @Query("SELECT p.pid,p.pname,p.paddrs,ph.regNo,ph.contactNo,ph.numberType,ph.provider
                                        from Person p  left join  p.phonesInfo ph")
//@Query("SELECT p.pid,p.pname,p.paddrs,ph.regNo,ph.contactNo,ph.numberType,ph.provider
                                        from Person p  full  join  p.phonesInfo ph")
public List<Object[]> getDataByJoins();
                  ↳ To hold bunch of multiple col values from records of both parent and child tables.
}
```

step4) develop service interface and service Impl class

```
//service Interface
public interface IOneToManyMappingMgmtService {
      public List<Object[]> fetchDataByJoins();

}

//Service Impl class

@Service("otmService")
public class OneToManyMappingMgmtServiceImpl implements IOneToManyMappingMgmtService {
      @Autowired
      private PersonRepo  personRepo;
      @Autowired
      private PhoneNumberRepo phoneRepo;

      @Override
      public List<Object[]> fetchDataByJoins() {
            return personRepo.getDataByJoins();
      }
```

step4) develop the Runner class

```
public class OneToManyMappingTestRunner implements CommandLineRunner {
      @Autowired
      private IOneToManyMappingMgmtService service;

      @Override
      public void run(String... args) throws Exception {
            service.fetchDataByJoins().forEach(row->{
                  for(Object val:row) {
                        System.out.print(val+"  ");
                  }
                  System.out.println();
            });

      }//run(-)

}//class
```

feb 04 Spring Data JPA -- Introduction to MangoDB

Spring Data MangoDB
==========

(spring data jpa is given to interact with SQL Dbs/w.. and
and spring data provides seperate sub module to interact with each NO SQL Dbs/w)

=> Spring Data provides unified model to work with both SQL and NoSQL Db s/ws...

=> SQL Db s/ws are oracle, postgreSQL, mysql and etc.. (Db tables based)
=> No SQL Db s/ws are MangoDB , cassendra, couchbase, neo4j and etc.. (key-value based, document based, graph based and etc..)
=> if data having fixed format and schema the go for SQL Db s/ws
=> if data the is dynamically growing and no fixed format then go for NoSQL Db s/w

Difference between SQL and NoSQL

Parameter	SQL	NOSQL
Definition	SQL databases are primarily called RDBMS or Relational Databases	NoSQL databases are primarily called as Non-relational or distributed database
Design for	Traditional RDBMS uses SQL syntax and queries to analyze and get the data for further insights. They are used for OLAP systems.	NoSQL database system consists of various kind of database technologies.These databases were developed in response to the demands presented for the development of the modern application.
Query Language	Structured query language (SQL)	No declarative query language
Type	SQL databases are table based databases	NoSQL databases can be document based, key-value pairs, graph databases
Schema	SQL databases have a predefined schema	NoSQL databases use dynamic schema for unstructured data.
Ability to scale	SQL databases are vertically scalable	NoSQL databases are horizontally scalable
Examples	Oracle, Postgres, and MS-SQL.	MongoDB, Redis, , Neo4j, Cassandra, Hbase.
Best suited for	An ideal choice for the complex query intensive environment.	It is not good fit complex queries.
Hierarchical data storage	SQL databases are not suitable for hierarchical data storage.	More suitable for the hierarchical data store as it supports key-value pair method.
Variations	One type with minor variations.	Many different types which include key-value stores, document databases, and graph databases.
Development Year	It was developed in the 1970s to deal with issues with flat file storage	Developed in the late 2000s to overcome issues and limitations of SQL databases.
Open-source	A mix of open-source like Postgres & MySQL, and commercial like Oracle Database.	Open-source
Consistency	It should be configured for strong consistency.	It depends on DBMS as some offers strong consistency like MongoDB, whereas others offer only offers eventual consistency, like Cassandra.
Best Used for	RDBMS database is the right option for solving ACID problems.	NoSQL is a best used for solving data availability problems
Importance	It should be used when data validity is super important	Use when it's more important to have fast data than correct data
Best option	When you need to support dynamic queries	Use when you need to scale based on changing requirements
Hardware	Specialized DB hardware (Oracle Exadata, etc.)	Commodity hardware
Network	Highly available network (Infiniband, Fabric Path, etc.)	Commodity network (Ethernet, etc.)
Storage Type	Highly Available Storage (SAN, RAID, etc.)	Commodity drives storage (standard HDDs, JBOD)
Best features	Cross-platform support, Secure and free	Easy to use, High performance, and Flexible tool.
top companies using	Hootsuite, CircleCI, Gauges	Airbnb, Uber, Kickstarter
ACID Vs BASE Model	ACID(Atomicity, Consistency, Isolation, and Durability) is a standard for RDBMS	Base (Basically Available, Soft state, Eventually Consistent) is a model of many NoSQL systems

MongoDB
=========

=> It is Docucment based NO SQL Db s/w. It internally maintains the data in the form of JSON documents

JSON :: Java Script Object Notation..

Examples of JSON Data

```
{
    "Name": "Kim",
    "address": {
    "streetAddress": "1st Street",
    "city": "Seoul",
    "postalCode": 1112
    },
    "phoneNumbers": [
    "111-222-333",
    "555-666-777"
    ]
}
```

```
{
    "emp_details": [
        {
            "emp_name": "Shubhan",
            "email": "ksingh.shubh@gmail.com",
            "job_profile": "intern"
        },
        {
            "emp_name": "Gaurav",
            "email": "gaurav.singh@gmail.com",
            "job_profile": "developer"
        },
        {
            "emp_name": "Nikhil",
            "email": "nikhil@geeksforgeeks.org'
            "job_profile": "Full Time"
        }
    ]
}
```

JSON (Java Script Object Notation) is format defining text data as key=value pairs alternate to the tag based xml format

---

=>Xml docs are tag based docs and they are cross platform docs holding the text data .. but in xml docs compare to data ...the tags decoration more..

xml document
==========

```
<students>
  <student>
     <sno>101</sno>
     <sname>raja</sname>
     <avg> 45.66 </avg>
  </student>
  <student>
     <sno>102</sno>
     <sname>rajesh</sname>
     <avg> 46.66 </avg>
  </student>
</students>
```

(Data is less, but tags decoration is more)

Json Document
==========

```
{
  {
    "sno" : 101,
    "sname" : "raja",
    "avg" : 45.66
  }
 {
   "sno" : 102,
   "sname" : "rakesh",
   "addrs": "hyd",
  }
}
```

Here data and docoration are equally managed.

MangoDB s/w installation

MongoDB download::
https://www.mongodb.com/try/download/community --> select current version (4.4.3) --> select widows --> select msi ---> download.

⇩

Install like any other windows software..

note:: we have multiple MongoDB gui Tools or clients s/ws like compass, Robot3T and etc..

To start MongoDB software ::
Go to C:\Program Files\MongoDB\Server\4.4\bin a and use mango.exe file...

SQL terminlogies

Physical DB s/w (like oracle)
|---->Logical DB1 (SID)
| |----> db table1 (cols and rows)
| |----> db table2 (cols and rows)
| |----> db table3 (cols and rows)
|---> Logical DB2

Db table and cols maintain records/rows

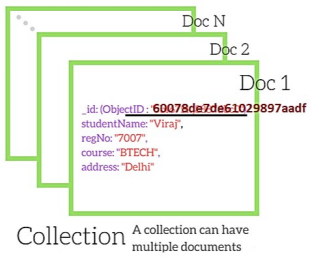
=>When we insert document to collection the mongoDb dynamically generateds unique ID as Hexadecimal number

Binary --> (Base2) :: 0 1
Decimal --> (Base10) :: 0-9
Octal --> (Base8) :: 0-7
Hexa ---> (Base16) --> 0-9 , a-f

NoSQL MongoDB terminologies

Physical Db s/w (like MongoDB)
|-----> Logical DB1
| |---->Collection1 (document1, document2,.. (json docs)
| |---->Collection2 (document1, document2,.. (json docs)
|-----> Logical DB2

collections maintain documents (equals to recrods)

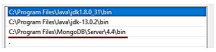


feb 04.1 spring data jpa Mango DB

Mongodb Shell commands

1) add <MongoDB_home>\bin folder to PATH env... variable

This PC --> properties --> advanced system setttings --> env.. variables -->system variable --> name :: PATH value::



--> ok --> ok -->ok

2) Open MongoDB Shell using mongo.exe file from any command prompt..

cmd> mongo (from any location)

3) To list all logical Dbs..

cmd> show dbs

4) To create new Logical DB

```
cmd> use ntsp713DB
     switched to db ntsp713DB
```

Unless and untill we add collections to Logical Db , the newly created Logical DB will not appear in the list of Logical Dbs

5) To know current logical DB

```
cmd> db
ntsp713DB
```

6)To create collection with one document..

```
cmd> db.customer.insertOne({cno:1001,cname:"raja",cadd:"hyd",billAmt:89000.0})
{
    "acknowledged" : true,
    "insertedId" : ObjectId("60078de7de61029897aadf28")
}
```

dynamically generated PK column kind of property with unique value.

```
cmd> db.customer.insertOne({cno:1002,cname:"rajesh"})
{
    "acknowledged" : true,
    "insertedId" : ObjectId("60078eb8de61029897aadf29")
}
cmd> db.customer.insertOne({cno:1003,cname:"rajesh",mobileNo:987677445})
{
    "acknowledged" : true,
    "insertedId" : ObjectId("60078edfde61029897aadf2a")
}
```

7) To list out documments of the collection

```
cmd> db.customer.find()
{ "_id" : ObjectId("60078de7de61029897aadf28"), "cno" : 1001, "cname" : "raja", "cadd" : "hyd", "billAmt" : 89

cmd> db.customer.find().pretty()
{
    "_id" : ObjectId("60078de7de61029897aadf28"),
    "cno" : 1001,
    "cname" : "raja",
    "cadd" : "hyd",
    "billAmt" : 89000
}
```

8) To insert many documents at once to a Collection

```
> db.customer.insertMany([{cno:567,cname:"ramesh",cadd:"hyd"},{cno:789,cname:"rakesh"}])
{
    "acknowledged" : true,
    "insertedIds" : [
        ObjectId("60078fb1de61029897aadf2b"),
        ObjectId("60078fb1de61029897aadf2c")
    ]
}
```

doc1 doc2

for reference :: https://docs.mongodb.com/manual/tutorial/insert-documents/

9) To insert document to a collection with array values..

```
> db.customer.insertOne({cno:1003,cname:"rajesh",mobileNo:[987671245,6788886777]})
{
    "acknowledged" : true,
    "insertedId" : ObjectId("6007911ade61029897aadf2d")
}
```

10) To find docs of a collection with condition

```
> db.customer.find({cadd:"hyd"}).pretty()
{
    "_id" : ObjectId("60078de7de61029897aadf28"),
    "cno" : 1001,
    "cname" : "raja",
    "cadd" : "hyd",
    "billAmt" : 89000
}
{
    "_id" : ObjectId("60078fb1de61029897aadf2b"),
    "cno" : 567,
    "cname" : "ramesh",
    "cadd" : "hyd"
}
```

```
> db.customer.find({cadd:"hyd",cname:"raja"}).pretty()
{
    "_id" : ObjectId("60078de7de61029897aadf28"),
    "cno" : 1001,
    "cname" : "raja",
    "cadd" : "hyd",
    "billAmt" : 89000
}
```

To update the document

where cluase cond set clause modifications

```
> db.student.updateOne({sadd:"hyd"},{$set:{sadd:"mumbai",avg:67.88}});
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```

11) To delete mongodb document

```
> db.customer.remove({cno:1001 })
WriteResult({ "nRemoved" : 1 })
```

To Remove multiple docs

removes only 1st matching

```
> db.customer.deleteOne({cno:1002})
{ "acknowledged" : true, "deletedCount" : 1 }
```

_id key holds dynamically generated unique value as pk VALUE.

```
> db.customer.deleteMany({cno:1003})
{ "acknowledged" : true, "deletedCount" : 2 }
> db.customer.find().pretty()
{
    "_id" : ObjectId("60078fb1de61029897aadf2b"),
    "cno" : 567,
    "cname" : "ramesh",
    "cadd" : "hyd"
}
{
    "_id" : ObjectId("60078fb1de61029897aadf2c"),
    "cno" : 789,
    "cname" : "rakesh"
}
```

_id is fixed property for documents added to a collection ... That propery values (hexa decimal values) are unique values and they will be used to indentify the docs..

For MongoDB shell commands
https://docs.mongodb.com/manual/reference/mongo-shell/

12) To remove collection

```
> db.customer.drop()
true
```

13) To list all collections

```
>> show collections
```

14) To switch to other logical DB

```
>use CollegeDB
```

16) To drop logical DB

```
> db.dropDatabase();
{ "dropped" : "ntspbms615db", "ok" : 1 }
> db
ntspbms615db
> show dbs;
NTSPBMS714DB1  0.001GB
admin          0.000GB
config         0.000GB
local          0.000GB
ntsp713DB      0.000GB
```

=>In spring Data JPA , we need to devleop @Entity class to map with db table and we have generators to generate id property value dynamically. if do not use Generators we can manually assign values to id proepery .

=> In spring data MogoDB ,we ened to devleop @Document class to map with Collection of MongoDb Db s/w.. Here there are Generators to generte the values for id property.. So either we need to assign unique or random value id property manually or we need to allow to store the MongoDB generated hexa decimal unique value to Id property.

Version1::: (Storing mongoDB generated _id property value (hexadecimal value) to @id property of @Document class) **Best Approach**

```
@Document
@Data
public class Student{
  @Id
  private String id;
  private Integer sno;
  private String sname;
  private String sadd;
  ....
  ...
}
```

property name must be "id" and type must be "String" to hold value generated for the _id property of JSON doc to this "id" property

example inserted JSON document

```
{
    "_id" : ObjectId("62012bfb4a76b1a1ead13923"),
    "sno" : 678,
    "sname" : "mahesh",
    "sadd" : "vizag"
}
```

```
Student st=new Student();
st.setSno(678);
st.setName("mahesh");
st.setSadd("vizag");
```

Version2: (Make MondoDB to use the programmer assigned /generated value to @Id property of Document obj as id value for each inserted json Document)

```
@Document
@Data
public class Student{
  @Id
  private Integer sno;
  private String sname;
  private String sadd;
  ...
  ...
}
```

example inserted JSON document

```
{
"sno" : 678 ==> this becomes id value
"sname" : "mahesh",
"sadd" : "vizag"
}
```

Now duplicate "sno" will give error.

```
Student st=new Student();
st.setSno(678);
st.setName("mahesh");
st.setSadd("vizag");
```

=>we can use Custom Generator or Third party Generator to store the generated id value to id property.

feb 08 NoSQL MangoDB-Robo 3T

Different GUI Tools to work with MongoDB
a) Compass (Bit heavy s/w)
b) Robo3T (Light weight)

Working with Robo3T
==========

=>donwload s/w from :: https://robomongo.org/download
=>install s/w like any other window s/w

Procedure to create Logical DB with with collection in MongoDB using Robo3T
==========

step1) launch robo3t and connect to MongoDB by creating new connection

step2) create Logica DB

Right click connection ---> create Data base ---> name:: NTSPBMS615DB

step3) create Collection and insert document

expand logical DB NTSPBMS714DB1 ---> right Click on Collection ---> new collection :: customer

right click on customer ---> insert document -->

```
{
  cno:101,
  cname:"raja",
  cadd:"hyd"
}
```

-->save

step4) To update the document

option1: View the document ---> edit document and modify attribute vlaues

(or)

right click on customer (document) --> update document -->

```
db.getCollection('customer').update(
// query
{
  cno : 101      | creteria
},

// update

{$set: {cadd:"mumbai"} }   | set new values..

);
```

step4) perform other operations on the documents and collection ..

=>remove document
=>rename cllection
=> drop collection

Add username, password to Logical DB for Authentication using Robot3T

Expand LogicaDB ---->users --> add user --->
username :: testuser
password :: testuser
select read , readwrite checkboxes..

Disconnect from MongoDB ---> connect ----> delete the existing connection ---> create new connection
name : con1 ---> Authentication tab ---> select checkbox --> db name :: (NTSPBMS615DB)
username: testuser , password: testuser --> connect.

Developing Document class whose objects represent documents of a collection in MongoDB

collection ::
customer
|-->cno
|-->cname | Doc1 attributes
|--->cadd

customer
|--->cno
|-->cname | Doc2 attributes
|--->cadd
|--> billAmt

=>Like Hibernate /JPA we do not have readymade generators cfg for Id property in MongoDB document class.

In MongoDB officially there is no PK , FK constraits .. but we can bring that effect indirectly..

PK constriant using _id attribute
FK constraint using documents nesting or chanining.

if want to represent the documents of a collection using the objects of java class.. then that class must be designed having highest possible properties representing the attributes of the Document.

The java class whose objects represent documents of Collection is called Document class (@Document class)

version1::
==========

```
@Document
@Data
public class Customer{
  @Id
  private String id;
  private Integer cno;
  private String cname;
  private String cadd;
  private Double billAmt;
}
```

// if u want to use MongoDB generated id value as the id value of Document object then take String property having @Id as the id property as shown here.
// if u r not intrested to use MongoDB generated id value as the id value of the Document then we can take support one or another generator support to generate the id value (like UUIDGenerator class)

Version2:
==========

```
@Document
@Data
public class Customer{
  @Id
  private Integer cno;
  private String cname;
  private String cadd;
  private Double billAmt;
}
```

// if u feel that u can store unique value in "cno" property then make "cno" property as the id property.. otherwise follow "version1".

conclusion::

The java Document object representing the MongoDB document of a collection can have
dynamically generated string value as the id value
or custom String value as the id value (given by IdGenerator) | (version1) (Best)
or our choice property value as the id value | (version2)

feb 08.1 NoSQL MangoDB-RoboBT

Procedure to develop First Springdata MongoDB Application using MongoRepository

step1) create Spring Boot starter project

selecting spring data MongoDB , Lombok starters..

step2) add the following properties in application.properties file.

application.properties

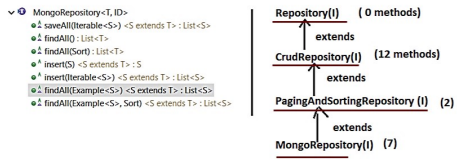
```
# mongoDB connection properties
spring.data.mongodb.host=localhost
spring.data.mongodb.port=27017
spring.data.mongodb.database=NTSPBMS615DB
spring.data.mongodb.username=testuser
spring.data.mongodb.password=testuser
```

step3) Develop the Document class

```
package com.nt.document;
import java.io.Serializable;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;
import lombok.Data;
@Document
@Data
public class Tourist implements Serializable {
	@Id
	private String id; //fixed name
	private String touristName;
	private String nativePlace;
	private Float age;
	private String passportNo;
	private Long contactNo;
	private boolean isVaccinated;
}
```

step4) develop Repository Interface extending from MongoRepository(I)

```
package com.nt.repo;

import org.springframework.data.mongodb.repository.MongoRepository;

import com.nt.document.Tourist;

public interface ITouristRepo extends MongoRepository<Tourist, String> {
}
```



step5) Develop the service interface and service Impl class

//Service Interface

```
package com.nt.service;

import com.nt.document.Tourist;

public interface ITouristMgmtService {
	public String registerTourist(Tourist tourist);	}
```

Service Impl class

```
@Service("touristService")
public class TouristMgmtServiceImpl implements ITouristMgmtService {
	@Autowired
	private ITouristRepo touristRepo;

	@Override
	public String registerTourist(Tourist tourist) {
		return "Documennt is saved with id value::"+ touristRepo.insert(tourist).getId();
	}
}
```

step6) Devleop the runner class invoking the service class b.method

```
@Component
public class MongoDBTestRunner implements CommandLineRunner {
	@Autowired
	private ITouristMgmtService service;

	@Override
	public void run(String... args) throws Exception {
		try {
			Tourist tourist=new Tourist(null,"ramesh", "USA",45.0f,"3543543L54AS5",9999999L ,true);
			String result=service.registerTourist(tourist);
			System.out.println(result);
		}
		catch(Exception e) {
			e.printStackTrace();
		}

	}//run(-)
```

```
db.getCollection('tourist').find({})
```

tourist 0.002 sec.

	_id	touristName	nativePlace	age	passportNo	contactNo	isVaccinated	_class	visaDuration	visaExpirityTime
1	ObjectId("62028434…")	ramesh	USA	45.0	3543543L54AS5	9999999	true	com.nt.…		
2	ObjectId("62028643…")	ramesh	nepal				false	com.nt.…		
3	ObjectId("62028686…")	suraj	nepal					com.nt.…		
4	ObjectId("62028756…")	ramesh	USA	45.0	3543543L54AS5	9999999	true	com.nt.…	100	2022-10-20…

We can observ dynamic schema based documents management.

```
==================================
To fetch All document of a Collection
==================================
```

In service Interface

```
public List<Tourist> fetchAllToursits();
```

In service Impl class

```
@Override
public List<Tourist> fetchAllToursits() {
	return touristRepo.findAll();
	}
```

In Runner class

```
//============= findAll Documents method=================
	service.fetchAllToursits().forEach(System.out::println);
```

Batch insertion of documents using saveAll(-) method

In service Interface

```
public String saveTouristGroup(List<Tourist> tourists);
```

In service Impl class

```
@Override
public String saveTouristGroup(List<Tourist> tourists) {
	int count=touristRepo.saveAll(tourists).size();
	return count+" no.of tourirsts are saved";
	}
```

In runner class

```
//=================saveAll(-) method==========================
		try {
		Tourist tourist1=new Tourist(null,"chari", "USA",45.0f,"3543543L54AS5",9999999L
,true,100,LocalDateTime.of(2022,10,20,11,34));
		Tourist tourist2=new Tourist();
		tourist2.setTouristName("suraj");
		tourist2.setNativePlace("nepal"); // with 3 fileds
		Tourist tourist3=new Tourist(null,"ramana", "canada",45.0f,"35435435LL6S5",888888L ,true,null,null);
		List<Tourist> touristList=List.of(tourist1, tourist2, tourist3);
		String msg=service.saveTouristGroup(touristList);
		System.out.println(msg);
		}
		catch(Exception e) {
			e.printStackTrace();
		}
```

feb 09 Spring Data --MangoDB - CURD Operations

fetching all document Using Sort object

In service Interface

```
public List<Tourist> fetchAllToursits(boolean asc, String ...properties);
```

In service Impl class

```
@Override
    public List<Tourist> fetchAllToursits(boolean asc, String... properties) {
        Sort sort=Sort.by(asc?Direction.ASC:Direction.DESC, properties);
        return touristRepo.findAll(sort);
    }
```

In Runner class

```
service.fetchAllToursits(true,"touristName","contactNo").forEach(System.out::println);
```

---

fetching document by Id

In service Interface

```
public Optional<Tourist> fetchTouristById(String id);
```

In service Impl class

```
@Override
    public Optional<Tourist> fetchTouristById(String id) {
        Optional<Tourist> opt=touristRepo.findById(id);
        return opt;
    }
```

In runner class

```
Optional<Tourist> opt=service.fetchTouristById("620284349c941c58ccc6d608");
        if(opt.isPresent())
            System.out.println("Document is ::"+opt.get());
        else
            System.out.println("Document not found ");
```

---

updating the document

In service interface

```
public String modifyTouristById(String id , long contactNo);
```

In serivce Impl class

```
@Override
public String modifyTouristById(String id, long contactNo) {
    Optional<Tourist> opt=touristRepo.findById(id);
    if(opt.isPresent()) {
        Tourist tourist=opt.get();
        tourist.setContactNo(contactNo);
        return touristRepo.save(tourist).getId()+" is updated with new contactNo";
    }
    else {
        return "Document is not found";
    }
}
```

In runner class

```
System.out.println(service.modifyTouristById("620284349c941c58ccc6d608", 565646464776L));
```

Removing Document

In service Interface

```
public String removeToursitById(String id);
```

In service Impl class

```
@Override
public String removeToursitById(String id) {
    Optional<Tourist> opt=touristRepo.findById(id);
    if(opt.isPresent()) {
        Tourist tourist=opt.get();
  touristRepo.delete(tourist);
  return "Document is found and deleted";
 }
    else {
        return "Document is not found";
    }
}//nethod
```

In runner class

```
System.out.println(service.removeToursitById("620284349c941c58ccc6d608"));
```

---

feb 09.1 Spring Data --MangoDB - CURD Operations

=>In order to control on String id value generation as hexa decimal number we need to use third party supplied classes UUID , GUID and etc.. which generated 32 letters hexa decimal number as id value.

These value are unique beczo they are generated based multiple parameters
a) System date and time b) current Process Id c) System Id address and etc..

Example
========

Document class

```
@Document
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Tourist implements Serializable {
        @Id
        private String id; //fixed name
        private String touristName;
        private String nativePlace;
        private Float age;
        private String passportNo;
        private Long contactNo;
        private Boolean isVaccinated;
        private Integer visaDuration;
        private LocalDateTime visaExpirtyTime;
}
```

Serivce Interface and SErvice Impl class

In SErvice clas

```
public String registerTourist(Tourist tourist);
```

In serivce Impl class

```
@Override
public String registerTourist(Tourist tourist) {
        return "Documennt is saved with id value::"+ touristRepo.insert(tourist).getId();
}
```

In Runner class

```
//===== save document ===============
try {
String id=UUID.randomUUID().toString();
System.out.println(id);
Tourist tourist=new Tourist(
        id,"ramesh", "USA",45.0f,"3543543L54AS5",9999999L ,true,null,null); //with 7 fields
String result=service.registerTourist(tourist);
System.out.println(result);
 }
 catch(Exception e) {
        e.printStackTrace();
 }
```

Taking Numeric Property as @Id property

=> For this MongoDB does not generated id value dynamically .. So we need to use one or another technique or some third party classes to generated id value for each document seperatley..

Document class
==========

```
@Document(collection = "TOURIST1")
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Tourist implements Serializable {
        @Id
        private Integer touristId;
        private String touristName;
        private String nativePlace;
        private Float age;
        private String passportNo;
        private Long contactNo;
        private Boolean isVaccinated;
        private Integer visaDuration;
        private LocalDateTime visaExpirtyTime;
}
```

service Interface

```
public interface ITouristMgmtService {
        public String registerTourist(Tourist tourist);
}
```

service Impl class

```
@Override
public String registerTourist(Tourist tourist) {
        return "Documennt is saved with id value::"+ touristRepo.insert(tourist).getTouristId();
}
```

In Runner class

```
//===== save document ===============
try {
 Integer idVal=new Random().nextInt(100000);
 Tourist tourist=new Tourist(
 idVal,"subhash", "USA",45.0f,"3543543L54AS5",9999999L ,true,10,LocalDateTime.now()); //with 7 fields

String result=service.registerTourist(tourist);
System.out.println(result);
}
catch(Exception e) {
        e.printStackTrace();
}
```

feb 10 MangoDB - finder methods and special types

finder method spring data mongoDB
==========
=>we need to add these methods in Repository Interface Performing select operations on MongoDB document.
=>These methods must be developed by following one or another naming convention

syntax :: public <ReturnType> findBy<Property><Condition>(Params)
==>These finder methods are very much similar to spring data jpa finder mehtods..

=>finder methods spring data mongodb supports only select opeartion .. in both Entity , scalar operations mode.
Entity mode means ::selecting all col/property values
scalar mode means :: selecting specific multiple col/property values.

Code in Repository
==========

```
public interface ITouristRepo extends MongoRepository<Tourist,String> {
        public List<Tourist>  findByNativePlaceOrderByTouristNameAsc(String nativePlace);
        public Tourist   findByContactNo(long contactNo);

}
```

Code in Service Interface

```
public interface ITouristMgmtService {
        public List<Tourist>  fetchTouristsByNativePlaceAscOrder(String nativePlace);
        public  Tourist   fetchTouristByContactNo(long contactNo);
}
```

Code in SErvice Impl class

```
@Service("touristService")
public class TouristMgmtServiceImpl implements ITouristMgmtService {
        @Autowired
        private ITouristRepo touristRepo;

        @Override
        public List<Tourist> fetchTouristsByNativePlaceAscOrder(String nativePlace) {
                return  touristRepo.findByNativePlaceOrderByTouristNameAsc(nativePlace);
        }

        @Override
        public Tourist fetchTouristByContactNo(long contactNo) {
           return touristRepo.findByContactNo(contactNo);
        }

}//class
```

Code in Runner class

```
@Component
public class MongoDBFinderMethodsTest implements CommandLineRunner {
        @Autowired
        private ITouristMgmtService service;

        @Override
        public void run(String... args) throws Exception {

                //=====  call finder methods ===============
        //      service.fetchTouristsByNativePlaceAscOrder("USA").forEach(System.out::println);
                System.out.println("------------------------------");
                Tourist tourist=service.fetchTouristByContactNo(888888L);
                if(tourist==null)
                        System.out.println("tourist not found");
                else
                        System.out.println("tourist details :"+tourist);
        }// main(-)

}// class
```

feb 10.1 MangoDB - findder methods and special types

Special types in MongoDB

=>array type properties
=> java.util.List type properties
=> java.util.Set type properties
=> java.util.Map type properties
=> java.util.Properties type properties
=> HAS-A property (non-collection type)
=> HAS- A property (collection type)

In MongoDb array , list ,set, property will be treated as same type properties
i.e at java level they may be different.. but a mongo db level all those properties will be saved in same style.

["value1","value2","value3","value4",....]

InMongoDB , the Map type , Properties type properties will be treated as same properties towards stroing their values in db table.

{ "key1":"val1","key2:val2","key3:val3" ,.....}

=>Collection with simple values or wrapper values will be used in Entity /Document calss for collection mapping
=>Collection with Associated class objects will be used in Entity /Document calss for Association mapping

=> In spring data MongoDB there are no seperate terminologies called collection mapping and assoicaiton mapping .. they both are always treated as special mapping.

example App
==========
//Document class

```
package com.nt.document;
import java.io.Serializable;
import java.util.List;
import java.util.Map;
import java.util.Properties;
import java.util.Set;

import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.n
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Document(collection = "Person_Info")
@Data
@AllArgsConstructor
@NoArgsConstructor
public class PersonInfo implements Serializable {
@Id
private Integer pid;
private String pname;
private String paddrs;
private String[] favColors;
private  List<String> friends;
private  Set<Long> contactNumbers;
private Map<String,Long>  bankAccountDetails;
private Properties  idDetails;

}
```

Repository Interace
==========

```
public interface IPersonInfoRepo extends MongoRepository<PersonInfo,Integer> {
}
```

Serivce Intefae, Service Impl class
==========

```
public interface IPersonInfoMgmtService {
        public String registerPerson(PersonInfo info);

}
```

in serviceImpl

```
    @Override
    public String registerPerson(PersonInfo info) {
            return "Person is saved with:"+ personRepo.save(info).getPid();
    }
```

Rnnner class

```
@Override
public void run(String... args) throws Exception {
try {
 Properties props=new Properties();
 props.put("aadhar", 543535435);
 props.put("voterId", 354355435);
 PersonInfo info=new PersonInfo(new Random().nextInt(10000),"raja","hyd",
                                new String[] {"red","green","yellow"},
                                List.of("srinu","ramesh"),
                                Set.of(9999999L,8888888L),
                                Map.of("sbi",454354354L,"icici",54353543L),
                                props);
 System.out.println(service.registerPerson(info));
}//try
catch(Exception e) {
        e.printStackTrace();
}
```

feb 11 Spring Data Association Mapping in MongoDB

HAS-A properties in spring data MongoDB
==========

=>To build One to One , many to One Assoication b/w two documents we need HAS-Property which is non-collection type

=>To build One to Many , many to Many Assoication b/w two documents we need HAS-Property which is collection type

=>All collections and Has-A propeerties in spring data mongoDB are called Specical Types.

Building One to One Association Mapping (Bi-Directional) beweeen the documents of MongoDB
==========

Person ----------DrivingLicense (1 Peron contains 1 Driving License)
Person ---------- PassPort (1 person contains 1 Passport)
Person ---------> panCard (1 peron contains 1 pan card)

```
//parent Document
==========
@Document
@Setter
@Getter
public class Person{
  @Id
  private String id;
  private String name;
  private String addrs;
  private DrivingLicense licenseDetails;
  //toStirng()
    ....
    .....
  }
```

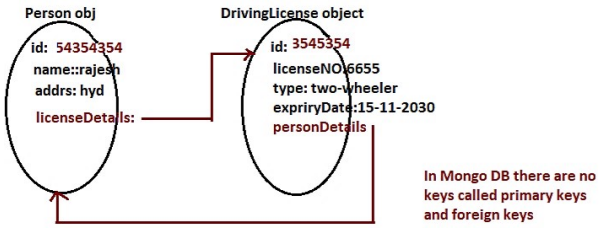
special property

HAS-A property to hold one associated child obj

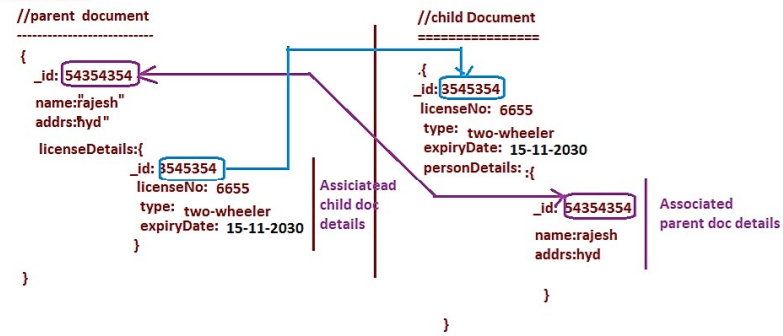
```
//Child Document

@Document
@Setter
@Getter
public class DrivingLicense{
  @Id
  private Strng id;
  private Long licenseNo;
  private String type;
  private Date expirtyDate;
  private Person personDetails;
  //toString()
    ....
    ...
}
```

HAS-A property to hold one associated parent obj

JAva Objects level 1 to 1 Assoication (Bidirectional Association)



In MongoDB DB s/w
==========



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

//Parent Document

```
package com.nt.document;

import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.
                        core.mapping.Document;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.NonNull;
import lombok.RequiredArgsConstructor;
import lombok.Setter;

@Document
@Setter
@Getter
@NoArgsConstructor
@AllArgsConstructor
@RequiredArgsConstructor
public class Person {
@Id
@NonNull
private Integer pid;
@NonNull
private String name;
@NonNull
private String addrs;
private DrivingLicense licenseDetails;
@Override
public String toString() {
        return "Person [pid=" + pid + ", name="
                + name + ", addrs=" + addrs + "]";
}
}
```

//child Document

```
package com.nt.document;
import java.time.LocalDate;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.NonNull;
import lombok.RequiredArgsConstructor;
import lombok.Setter;

@NoArgsConstructor
@AllArgsConstructor
@RequiredArgsConstructor
@Document
@Setter
@Getter
public class DrivingLicense {
@NonNull
@Id
private Long licenseNo;
@NonNull
private String type;
@NonNull
private LocalDate expiryTime;
private  Person personDetails;
 @Override
 public String toString() {
        return "DrivingLicense [ licenseNo=" + licenseNo +
        ", type=" + type + ", expiryTime=" + expiryTime + "]";
 }
 }
```

applicaiton.properites
==========

```
spring.data.mongodb.host=localhost
spring.data.mongodb.port=27017
spring.data.mongodb.database=NTSPBMS615DB
spring.data.mongodb.username=testuser
spring.data.mongodb.password=testuser
```

feb 11.1 Spring Data-Association Mapping in MongoDB

Repositoruy1
==========

```
public interface IDrivingLicenseRepo extends MongoRepository<DrivingLicense,Long> {   }
```

```
//Repository2
==========
public interface IPersonRepo extends  MongoRepository<Person, Integer>{   }
```

in service interface:-

```
public interface IRTOMgmtService {
    public  String   registerPersonWithDrivingLicense(Person person);
    public  String   registerLicenseWithPerson(DrivingLicense license);
    public    List<Person>   fetchAllPersonDetails();
    public    List<DrivingLicense>   fetchAllLicenseDetails();
}
```

in service implementations:

```
@Service("rtoService")
public class RTOMgmtServiceImpl implements IRTOMgmtService {
        @Autowired
        private IDrivingLicenseRepo licenseRepo;
        @Autowired
        private  IPersonRepo  personRepo;

        @Override
        public String registerPersonWithDrivingLicense(Person person) {
                return "Person with DrivingLicense is saved with id value ::"+ personRepo.save(person).getPid();
        }
        @Override
        public String registerLicenseWithPerson(DrivingLicense license) {
                return "Person with DrivingLicense is saved with id value ::"+ licenseRepo.save(license).getLicenseNo();
        }
        @Override
        public List<Person> fetchAllPersonDetails() {
                return personRepo.findAll();
        }
        @Override
        public List<DrivingLicense> fetchAllLicenseDetails() {
                return licenseRepo.findAll();
        }
}
```

in runner class:-

```
@Component
public class OneOneAssociationTestRunner implements CommandLineRunner {
        @Autowired
        private  IRTOMgmtService service;
        @Override
        public void run(String... args) throws Exception {
        /*  //parent to child
        try {
                Person per=new Person("raja","hyd");
                DrivingLicense license=new DrivingLicense(5654645L,"2-wheeler",LocalDate.now());
                 //child to parent
                per.setLicenseDetails(license);
               System.out.println(service.registerPersonWithDrivingLicense(per));
        }//try
        catch(Exception e) {
              e.printStackTrace();
        }*/

        /*          System.out.println("==========parent===to child================");
                    service.fetchAllPersonDetails().forEach(per->{
                            System.out.println("parent:: "+per);
                            DrivingLicense license=per.getLicenseDetails();
                            System.out.println("child::"+license);
                    });
        */
     //child to parent
    try {
            Person per=new Person(new Random().nextInt(1000),"suresh1","vizag1");
            DrivingLicense license=new DrivingLicense(new Random().nextLong(),"4-
                                                    wheeler",LocalDate.now());
             // parent to child
        license.setPersonDetails(per);
        System.out.println(service.registerLicenseWithPerson(license));
    }//try
    catch(Exception e) {
          e.printStackTrace();
    }
    System.out.println("==========child to parent================");
    service.fetchAllLicenseDetails().forEach(lic->{
          System.out.println("child:: "+lic);
          Person  per=lic.getPersonDetails();
          System.out.println("parent:: "+per);
    });
   }
}
```

feb 15 MangoDB - Association Mapping 1 to Many

HAS-A properties (special type) in Association mapping (Collection type)

=> To build one to many and many to many association , we need to take collection type HAS- A properties in Document classes.

=> Player will pay multiple sports , So the relation b/w Player and Sport is one to many association

=> Player will get multiple medals , So relation b/w Player and Medal is one to many Association

```
//parent class
@Document
@Setter @Getter
public class Player{
   @Id
  private Integer pid;
   private String pname;
   private String paddrs;
   private Set<Sport> sports;
   private Map<String,Medal> medals;

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

```
//child class1
@Document
 @Setter @Getter
public class Sport{
 @Id
 private Integer sid;
 private String name;
 private String[] kitItems;

 //toString()
}
```

```
//child class2
@Document
 @Setter @Getter
public class Medal{
 @Id
 private String mid;
 private String name;
 private String event;
 private String place;
 //toString()
}
```

sample json doc representing the above one to many association

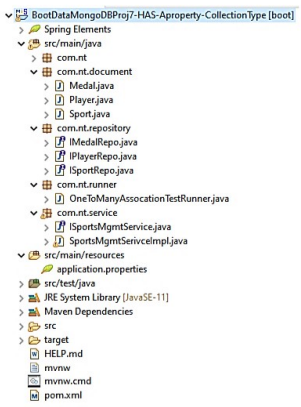
```
{
  _id: ....
  pname: "p.vsindhu"
  paddrs: ..hyd"
  sports : [
            {
             _id:___
              sname: "tennis"
               kitItems: [ "rocket","ball", "net"]
            }
            {
              _id:___
              sname: " batmintton"
               kitItems: [ "s.cock","bat ", "net"]
            }
          ]
  medals: {
            "olympic":{
                 _id: ___
                 mname: "RIO-silver",
                 event: "olympic-2010",
                 place :: "RIO",
                }
            "asianGames":{
                 _id: ____
                 mname: " gold"
                 event: "asin-2020"
                 place : "japan"
                }
          }
```

Set of child documents

Map of documents

example App

```
//Player (parent class)
package com.nt.document;
import java.util.Map;
import java.util.Set;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@Document
@Setter
@Getter
@AllArgsConstructor
@NoArgsConstructor
public class Player {
      @Id
      private Integer pid;
      private String pname;
      private String paddrs;
      private Set<Sport> sports;
      private Map<String,Medal> medals;

      @Override
      public String toString() {
            return "Player [pid=" + pid + ", pname=" + pname + ", paddrs=" + paddrs + "]";
      }
}
```




```
//child1 class
package com.nt.document;
import java.util.Arrays;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
@Document
@Setter
@Getter
@AllArgsConstructor
@NoArgsConstructor
public class Sport {
      @Id
      private Integer sid;
      private String sname;
      private String[] kitItems;
      @Override
      public String toString() {
            return "Sport [sid=" + sid + ", sname=" + sname + ",
                  kitItems=" + Arrays.toString(kitItems) + "]";
   }
}
```

```
//Child2 class
package com.nt.document;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
@Document
@Setter
@Getter
@AllArgsConstructor
@NoArgsConstructor
public class Medal {
      @Id
      private String mid;
      private String mname;
      private String event;
      private String place;
      @Override
      public String toString() {
            return "Medal [mid=" + mid + ", mname=" + mname + ",
                  event=" + event + ", place=" + place + "]";
   }
}
```

feb 15.1 MangoDB - Association Mapping 1 to Many

Repossitories

```
public interface ISportRepo extends MongoRepository<Sport, Integer> {  }

public interface IMedalRepo extends MongoRepository<Medal,String> {  }

public interface IPlayerRepo extends MongoRepository<Player, Integer> {  }
```

Service interface and service impl class

```
public interface ISportsMgmtService {
   public  String registerPlayerWithSportsAndMedals(Player player);
   public  List<Player> fetchAllPlayersInfo();
}
```

code in service impl

```
@Service("sportService")
public class SportsMgmtSerivceImpl implements ISportsMgmtService {
@Autowired
private IPlayerRepo playerRepo;
@Autowired
private  ISportRepo  sportRepo;
@Autowired
private  IMedalRepo  medalRepo;

@Override
public String registerPlayerWithSportsAndMedals(Player player) {
      return "Player with sports and medal info is saved with "
                  +playerRepo.save(player).getPid()+" id value ";
}
@Override
public List<Player> fetchAllPlayersInfo() {
      return  playerRepo.findAll();
}

}
```

Generated document in MongoDB

```
/* 1 */
{
  "_id" : 130,
  "pname" : "sindhu",
  "paddrs" : "hyd",
  "sports" : [
    {
      "_id" : 308,
      "sname" : "tennis",
      "kitItems" : [
        "racket",
        "t.ball",
        "net"
      ]
    },
    {
      "_id" : 226,
      "sname" : "badmintton",
      "kitItems" : [
        "racket",
        "scock",
        "net"
      ]
    }
  ],
  "medals" : {
    "medal2" : {
      "_id" : "cwg-silver",
      "mname" : "silver",
      "event" : "cwg-japan",
      "place" : "tokyo"
    },
    "medal1" : {
      "_id" : "olympic-gold",
      "mname" : "Gold",
      "event" : "Tokyo -olympic",
      "place" : "tokyo"
    }
  },
  "_class" : "com.nt.document.Player"
}
```

Runner class

```
@Component
public class OneToManyAssocationTestRunner implements CommandLineRunner {
      @Autowired
      private  ISportsMgmtService service;

      @Override
      public void run(String... args) throws Exception {
            try {
                  // child class1 objs
                  Sport sport1=new Sport(new Random().nextInt(1000), "badmintton",new String[] {"racket","scock","net"});
                  Sport sport2=new Sport(new Random().nextInt(1000), "tennis",new String[] {"racket","t.ball","net"});

                  //child class2 objs
                  Medal medal1=new Medal("olympic-gold","Gold","Tokyo -olympic","tokyo");
                  Medal medal2=new Medal("cwg-silver","silver","cwg-japan","tokyo");

                  //parent class obj
                  Player player=new Player(new Random().nextInt(1000),
                                           "sindhu",
                                           "hyd",
                                           Set.of(sport1,sport2),
                                           Map.of("medal1",medal1,"medal2",medal2));
                  //invoke the method
                  System.out.println(service.registerPlayerWithSportsAndMedals(player));
            }//try
            catch(Exception e) {
                  e.printStackTrace();
            }
            System.out.println("==============findAll() operation==================");
             service.fetchAllPlayersInfo().forEach(per->{
                  System.out.println("parent::"+per);
                  //access child1 objs
                  Set<Sport>  sports=per.getSports();
                  sports.forEach(sport->{
                        System.out.println("child1::"+sport);
                  });
                  //access child2 objs
                  Map<String,Medal> medals=per.getMedals();
                  medals.forEach((type,medal)->{
                        System.out.println("child2::"+type+"....."+medal);
                  });
             });
      }//main
}//class
```

application.properties

```
spring.data.mongodb.host=localhost
spring.data.mongodb.port=27017
spring.data.mongodb.database=NTSPBMS615DB
spring.data.mongodb.username=testuser
spring.data.mongodb.password=testuser
```

feb 16 Spring Data MangoDB -@Query methods

MongoDB @Query method with Projections (Queries in MongoDB)

=>By default @Query placed in MongoDB app selects all the fields of the Document class
=> Use "fields" attribute of @Query annotation having field names with 0 or 1 value to select/deselect a specific field
syntax:: @Query(fields="{property:0/1, property:0/1 }")
=> 1 indicates involve the field/variable/property in the select query.
=> 0 indicates do not involve the field/variable/property in the select query.
For all properties default value is 0 , but for @Id property the default value is 1.

=>"value" attribute of @Query is useful to specify the where condition clauses..

=>Projections in Querying means selecting either specific single col or multiple column values.

=>Entity operation in querying means selecting all col values of db table

eg1: @Query(value="{cadd:?0}",
fields="{cname:1,billAmt:1}")
cno is id property (@Id)

is equal to "SELECT CNO,CNAME,BILLAMT FROM CUSTOMER WHERE CADD=?" (SQL)
(By Default @Id property will be seleted becoz its default value is 1)

eg2: @Query(value="{cadd:?0}",
fields="{cno:0,cname:1,billAmt:1}")

is equal to "SELECT CNAME,BILLAMT FROM CUSTOMER WHERE CADD=?" (SQL)

Only for @Id property we can use 0 or 1 in Projection opertaions
for remaning only "1" is allowed (i.e involve the property) ..
if u do not want to involve the property do not take "0" .. just
ignore to place that propery.

@Query(fields = "{id:0,cno:1,cname:1,cadd:1,billAmt:1}" ,value = "{cadd:?0}")
valid

@Query(fields = "{id:0,cno:0,cname:1,cadd:1,billAmt:1}" ,value = "{cadd:?0}")
invalid

eg3:: To get all fileds/property values

@Query(value="{cadd:?0}",
fields="{ }")
or
@Query(value="{cadd:?0}")
(special case)

is equal to "SELECT * FROM CUSTOMER WHERE CADD=?" (SQL)

=>In MongoDB Queries params position starts with 0
where as in JPQL/HQL the params position starts with 1
=> In fileds attribute of @Query 0 or 1 is allowed for id proeprty
and remaining popertoes only 1 is allowed

=> In value attribute @Query we can ?<n> with any number like
?0 ,?1,?2,?3 and etc..

eg4:: To use multiple fields in where clause..
@Query(value="{cadd:?0,cname:?1}")

is equal to "SELECT * FROM CUSTOMER WHERE CADD=? AND CNAME=?" (SQL)

Example

Document class

```
@Document
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Tourist implements Serializable {
    @Id
    private String id; //fixed name
    private String touristName;
    private String nativePlace;
    private Float age;
    private String passportNo;
    private Long contactNo;
    private Boolean isVaccinated;
    private Integer visaDuration;
    private LocalDateTime visaExpirtyTime;

}
```

Repository Interface

```
public interface ITouristRepo extends MongoRepository<Tourist, String> {

    //================ Entity Queries ====================
    //@Query(fields ="{}", value= "{nativePlace:?0}") //to select all filed values (or)
    @Query( value= "{nativePlace:?0}") //to select all filed values (or)
    public List<Tourist> getAllTouristsByNativePlace(String nativePlace);

    @Query( value= "{nativePlace:?0,age:{$gte:?1}}") // applies nativePlace=? and age>=?
    public List<Tourist> getAllTouristsByNativePlaceAndAge(String nativePlace,int minAge);

    //@Query( value= "{age:{$gte:?0,$lte:?1}}") // applies age>=? and age<=? (or)
    @Query( value= "{age:{$gte:?0},age:{$lte:?1}}") // applies age>=? and age<=?
    public List<Tourist> getAllTouristsByAgeRange(int startAge,int endPage);

    @Query( value= "{$or:[{nativePlace:?0},{nativePlace:?1}]}") // applies nativePlace>=? or nativePlace<=?
    public List<Tourist> getAllTouristsByNativePlaces(String nativePlace1,String nativePlace2);

    //============================ Projections(selecting specific single column or multiple col values ============:=====
    @Query(fields="{touristName:1,nativePlace:1,age:1}",value= "{nativePlace:?0}")
      //gives select id,touristName,nativePlace,age from Tourist where nativePlace=?
    public List<Object[]> getTouristDataByNativePlace(String nativePlace);

    @Query(fields="{id:0,touristName:1,nativePlace:1,age:1,contactNo:1}",value= "{nativePlace:{$in:[?0,?1,?2]}}")
      //gives select id,touristName,nativePlace,age from Tourist where nativePlace=?
    public List<Object[]> getTouristDataByNativePlace(String nativePlace1,String nativePlace2,String nativePlace3);
}
```

Serivce Inteface

```
public interface ITouristMgmtService {

    public List<Tourist> fetchAllTouristByNativePlace(String nativePlace);
    public List<Tourist> fetchAllTouristByNativePlaceAndAge(String nativePlace,int minAge);
    public List<Tourist> fetchAllTouristByAgeRange(int minAge,int maxAge);
    public List<Tourist> fetchAllTouristsByNativePlaces(String nativePlace1,String nativePlace2);
    public List<Object[]> fetchTouristDataByNativePlace(String nativePlace);
    public List<Object[]> fetchTouristDataByNativePlaces(String nativePlace1,String nativePlace2,String nativePlace3);
}
```

Service Impl

```
@Service("touristService")
public class TouristMgmtServiceImpl implements ITouristMgmtService {
    @Autowired
    private ITouristRepo touristRepo;
    @Override
    public List<Tourist> fetchAllTouristByNativePlace(String nativePlace) {
        return touristRepo.getAllTouristsByNativePlace(nativePlace);
    }
    @Override
    public List<Tourist> fetchAllTouristByNativePlaceAndAge(String nativePlace, int minAge) {
        return touristRepo.getAllTouristsByNativePlaceAndAge(nativePlace, minAge);
    }
    @Override
    public List<Tourist> fetchAllTouristByAgeRange(int minAge, int maxAge) {
        return touristRepo.getAllTouristsByAgeRange(minAge, maxAge);
    }
    @Override
    public List<Tourist> fetchAllTouristsByNativePlaces(String nativePlace1, String nativePlace2) {
        return touristRepo.getAllTouristsByNativePlaces(nativePlace1, nativePlace2);
    }
    @Override
    public List<Object[]> fetchTouristDataByNativePlace(String nativePlace) {
      return touristRepo.getTouristDataByNativePlace(nativePlace);
    }
    @Override
    public List<Object[]> fetchTouristDataByNativePlaces(String nativePlace1, String nativePlace2,
            String nativePlace3) {
        return touristRepo.getTouristDataByNativePlace(nativePlace1,nativePlace2,nativePlace3);
    }
}
```

Runner class

```
@Component
public class MongoDBQueryTestRunner implements CommandLineRunner {
 @Autowired
 private ITouristMgmtService service;
 @Override
 public void run(String... args) throws Exception {
  //service.fetchAllTouristByNativePlace("nepal").forEach(System.out::println);
  System.out.println("-----------------------------------------------");
  //service.fetchAllTouristByNativePlaceAndAge("USA", 20).forEach(System.out::println);
  System.out.println("-----------------------------------------------");
  //service.fetchAllTouristByAgeRange(20, 48).forEach(System.out::println);
  System.out.println("-----------------------------------------------");
  //service.fetchAllTouristsByNativePlaces("nepal","USA").forEach(System.out::println);
  System.out.println("-----------------------------------------------");
  /*service.fetchTouristDataByNativePlace("USA").forEach(row->{
        System.out.println(Arrays.toString(row)); //(or)
        /*for(Object val:row) {
            System.out.print(val);
        }
        System.out.println(); (or) */

        /*for(int i=0;i<row.length;++i) {
            System.out.print(row[i]);
        }
        System.out.println();*/

  //}); */
  service.fetchTouristDataByNativePlaces("nepal","USA","canada").forEach(row->{
        System.out.println(Arrays.toString(row));
  });
 }
}// class
```

feb 17 Spring Data MangoDB -@Query methods

Regular expression In MongoDB @Query methods

	In SQL	In MongoDB
Starting with given data	data%	^data
Ending with given data	%data	data$
Having given data	%data%	data or ^data$ (It is not working in few versions of MongoDB)

For Documentation..
https://docs.mongodb.com/manual/tutorial/query-documents/

Example1 :: gettting using by placing regular expression in the input data

In Repository

```
public interface ITouristRepo extends MongoRepository<Tourist, String> {
    @Query(value="{nativePlace:{$regex:?0} }")
    public List<Tourist> getTouristsByNativePlaceRegEx(String nativePlace);
}
```

In Service Interface

```
public interface ITouristMgmtService {
    public List<Tourist> fetchTouristsByNativePlaceWithRegEx(String nativePlace);
}
```

In service Impl class

```
@Override
public List<Tourist> fetchTouristsByNativePlaceWithRegEx(String nativePlace) {
    return touristRepo.getTouristsByNativePlaceRegEx(nativePlace);
}
```

In runner class

```
//service.fetchTouristsByNativePlaceWithRegEx("^U").forEach(System.out::println);
        [ gives those docs whose nativePlace starts with U]
//service.fetchTouristsByNativePlaceWithRegEx("a$").forEach(System.out::println);
        [ gives those docs whose nativePlace endswith a]
service.fetchTouristsByNativePlaceWithRegEx("a").forEach(System.out::println);
        [ gives those docs whose nativePlace contains a]
```

delete attribute of @Query

using delete=true parame in @Query we can delete the selected document based on the condition placed in "value" param

code in Repository

```
@Query(value="{contactNo:null}", delete = true)
public int deleteTouristsWithNoContactNumber();
```

code in service impl class

```
@Override
public int deleteTouristsWithNoContactNumber() {
    return touristRepo.deleteTouristsWithNoContactNumber();
}
```

code in Runner class

```
System.out.println("delete docs count is ::"
+service.deleteTouristsWithNoContactNumber());
```

Working with Exists

using exists=true param of @Query annotation we can check wheather docs avaiable with given condition or not if avaiable method return true otherwise returns flase.

code in repository

```
@Query(value="{isVaccinated:false}",exists = true)
public boolean isTouristsExistWithOutVaccination();
```

code in serivce Impl class

```
@Override
public boolean IsThereTouristsWithOutVaccination() {
    return touristRepo.isTouristsExistWithOutVaccination();
}
```

code in runner class

```
System.out.println("====================");
System.out.println("Does the Tourist exist without vaccination::"
+service.IsThereTouristsWithOutVaccination());
```

@Query annotation is having multiple attributes like count,sort , exists and etc... to perform some aggregate operations

Working count attribute of @Query

The "count" param of @Query Annotation returns numeric value representing the count of document for the given condition..

code in repository

```
@Query(value="{age:{$gte:?0,$lte:?1} }", count = true)
public int getTouristsCountByAgeRange(int minAge, int maxAge);
```

code Service impl class

```
@Override
public int fetchTouristsCountByAgeRange(int minAge, int maxAge) {
    return touristRepo.getTouristsCountByAgeRange(minAge, maxAge);
}
```

code in runner class

```
System.out.println("Tourists count ::"+service.fetchTouristsCountByAgeRange(40,50));
```

=>The Query in MongoDB whose inputs are not hardcoded.. and passed through parameters like ?0,?1 and etc.. is called Dynamic Queries and other queries are called static queries

=> The MongoDB queries do not support named params they support only JPA style ordinal params (?0,?1,...)

"sort" attribute of @Query param

"sort" param of @Query annoation can sort the data (documents or selected fields) based given filed indicating "-1" for descending order and "1" for ascending order
note:: The input values "1", "-1" must be hardcoded and can not be taken through parameters /arguments.

code in repository

indidcates no condition

```
@Query(value="{}", sort = "{touristName:1}")
public List<Tourist> getAllTouristsSortedByName();
```

code service Impl class

```
@Override
public List<Tourist> {
    return touristRepo.getAllTouristsSortedByName();
}
```

code in runner class

```
service.fetchTouristsSortedByName() .forEach(System.out::println);
```