Spring Boot
==========

Nov 16 Spring Boot Intro

=> 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,Indi 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)

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.

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

Spring JDBC App (Spring FrameWork App)
==========

represents
JDBC Con pol
JDBC DataSource obj
=>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

```
//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
      ...
     }
}
```

=>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.

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

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 beanf 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

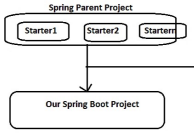
```
//DAO class (Class having persistence logic)
@Repository (or) @Component
public calss StudentDAOImpl implements IStudentDAO{
   @Autowired
   private DataSource ds;   //injects DataSource object

   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
==========

Is

Spring boot collection of parent pojects (Spring boot Starter Projects) that providees multiple Commom logics to our project (our spring Boot Project nothing 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..]

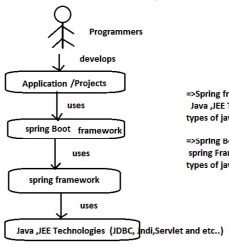
Our Spring Boot Project

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

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

Spring Boot Starters
(user-defined/pre-defined)
(Takes care of
10 Common Logics
through AutoConfiguation)

our spring boot Project
(10 application specific logics)



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

=>Spring Boot framework 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/ libraries 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

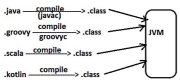
if we add "spring -boot-starter-jdbc " to the project
=> we get spring-jdbc-<ver>.jar (main jar file) | main and dependent jar files.
spring-tx-<ver>.jar (dependent jar file)
spring-context-<ver>.jar file and its dependents jar files (5 to 6 jar files)
hikaricp-<ver>.jar and its dependent jar files |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

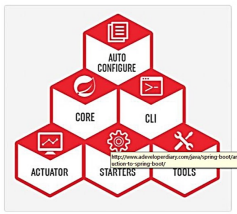
Pros of Spring Boot:

- 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.

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

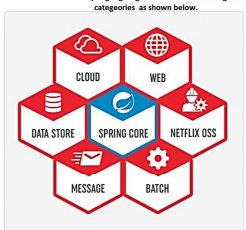
Different quick start projects to include as a dependency in your maven or gradle build file. It will have the needed dependencies for that type of application. Currently there are many starter projects (We will learn about few of them in the next section) and many more are expected to be added.

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
==========

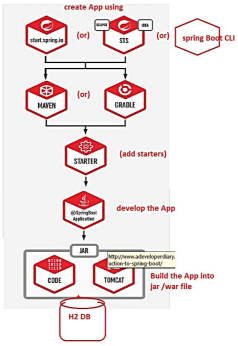
spring is giving 1000+ starters following in to different categeories as shown below.



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 IDE's
i) use STS IDE directly (Best2)
ii) use Eclipse IDE with STS plugn 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

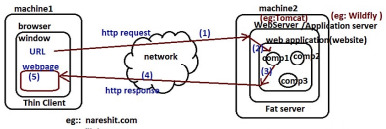
eg:: calculator app , Desktop game , Antivirus s/w , Awt/swing frame App and etc..

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 ++

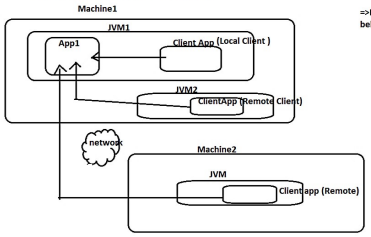
Application server = w

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

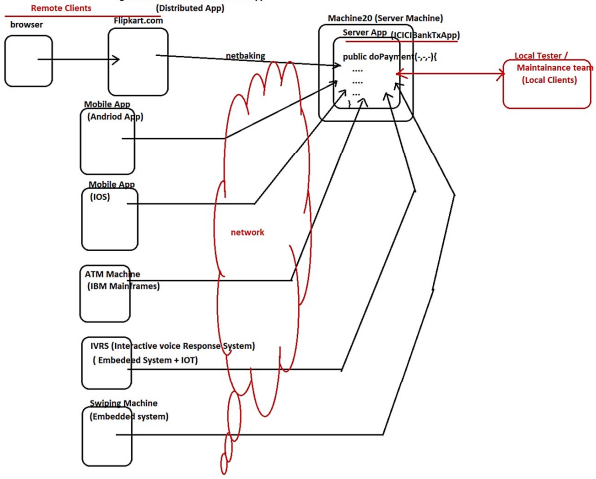
=>if App and its client are there in the same JVM then that client is Local client to App
=>if App and its client are there in two different JVMs same computer or different computer then the Client is called Remote Client to App..

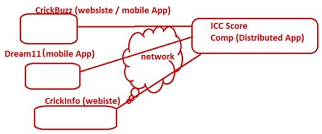


=>In Computer we can launch multiple JVMS beloging to same versions or diferent versions at a time

=> 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)

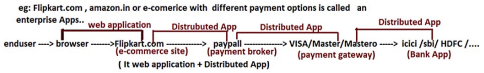
To develop Distributed Apps in java
=> RMI ,EJB, CORBA (outdated)
=> Jax-RPC, JAX-WS,Axis, Apache CXF and etc,, (SOAP based web services) (Loosing fame)
=> Jax-RS , Spring Rest, Rest Easy and etc.. (Restfull web services) (Doing well)

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..

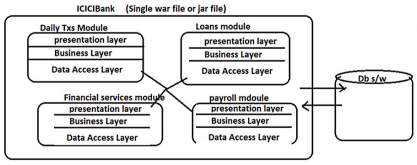


(It web application + Distributed App)

=>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



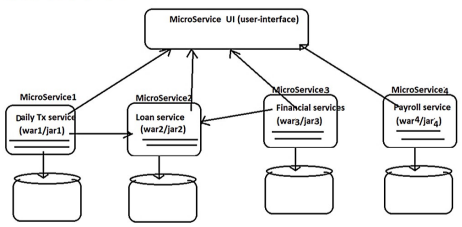
=>All these services/module of the Project should handled, deployed and mantained togather as single Project.
=>we can not controll memory management , CPU Utilization and others services seperately for each module or service (we must do for all modules/service go gather)

=> 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 taraget 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 variuls pre-defined classes given spring boot starters become spring beans and participates in neccessaryinjections.

note:: In new version @SpringBootApplcaiton 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))

Stereo type annotaiton (@Component)

target class: WishMessageGenerator
⇩ Should get sys date and time from the injected Calendar obj (GregorianCalendar obj) to generate wish messsage based on the current hour of day

Dependent class: java.util.Calendar (AC) pre-defined [use @Bean method to cfg as spring bean]
⇩ 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)

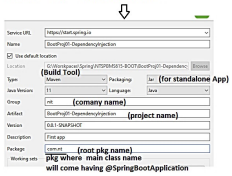
2
step) make sure that STS plugin installed to the Eclipse IDE

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)
(comsany name)
(project name)
(root pkg name)
pkg where main class name will come having @SpringBootApplication

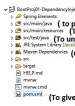
Boot
spring Latest version is : 2.6.0 and internally uses spring 5.3.13 version

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)

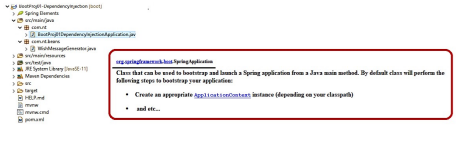
In main class

```
@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

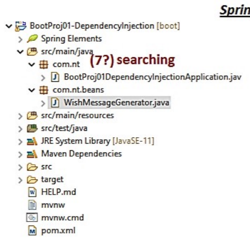
```
//main class
package com.nt;

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                                   Current class as cfg class for IOC container that is
		WishMessageGenerator generator=ctx.getBean("wmg",WishMessageGenerator.class);   created internally
		//invoke b.method
		String result=generator.generateWishMessage("raja");
		System.out.println("Wish Message is ::"+result);

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

	}

}
```

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



Nov 20 Spring Boot First App Flow

Spring Boot Application Flow

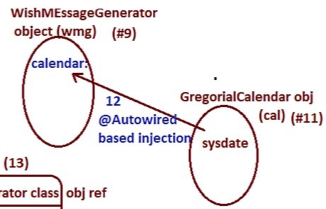


(7?) searching

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)

//target class

```
package com.nt.beans;

import java.util.Calendar;

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

@Component("wmg")      (8) finds this class
public class WishMessageGenerator {     (9) Loads the and creates the object
                                            supporting pre-instantation.
(10)      @Autowired
detects   private Calendar calendar;
@Autowired
          public WishMessageGenerator() {
           System.out.println("WishMessageGenerator:: 0-param constructor");
          }

          //b.method      (19)
          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;     (20)
               else
                    return "Good Night::"+user;
          }

}
```



Internal cache of IOC container (13)

bean ids (keys)	spring bean class obj refs
wmg (16?)	WishMesssageGenerator class obj ref
cal	GregorianCalendar object ref
bootProj01DependencyInjectionApplication	BootProj01DependencyInjectionApplication obj ref

```
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;
                                                        (1) run App
import com.nt.beans.WishMessageGenerator;                  (spring boot App/Java App)

@SpringBootApplication    (6) @ComponentScan annotation of @SpringBootApp Annotaion
public class BootProj01DependencyInjectionApplication {   looks for stereo type singleton scope spring bean
                                                         classes in the current pkg and sub pkgs
     public BootProj01DependencyInjectionApplication() {
          System.out.println("BootProj01DependencyInjectionApplication:: 0-param construtor");
     }

     @Bean(name="cal")
      public  Calendar  createCalendar() {   (12)
          System.out.println("BootProj01DependencyInjectionApplication.createCalendar()");
           return  Calendar.getInstance();  //return GregorianCalendar class(sub class of Calenar) obj
      }
                        (2)                                   (5) Checks the spring-boot-starters (jar files)
     public static void main(String[] args) {                 that added to buildpath/classpath for AutoConfiguraiton activitity
          System.out.println("main(-) method");                 (In this application No specital starters add)
gets IOC container  //Bootstrap /boot spring boot App and get IOC container ref
obj ref  (14)            (3)                                    (4)
AnnotationConfig     ApplicationContext ctx=SpringApplication.run(BootProj01DependencyInjectionApplication.class, args);
ApplicationContext obj ref)  System.out.println("IOC container  class name ::"+ctx.getClass());
          System.out.println("========================================");
          //get target class object  (17)     (15)
          WishMessageGenerator generator=ctx.getBean("wmg",WishMessageGenerator.class);
          System.out.println("============================");
          //invoke b.method                     (18)
  (21)    String result=generator.generateWishMessage("raja");
          System.out.println("Wish Message is ::"+result);

          //close container
          ((ConfigurableApplicationContext) ctx).close();  (22)
          System.out.println("end of main(-)");

     }  // (23)
        End of the App.
}
```

(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

#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-starter-jdbc we get follwing jar files and java classes spring beans automatically through AutoConfiguration process.

jar fiiles :: spring jar fiiles , 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

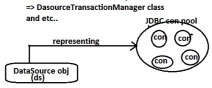
=> DasourceTransactionManager class

and etc..

javax.sql.DataSource(I)

↑ implements

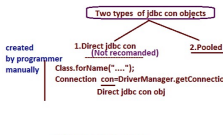
HikariDataSoruce (c)



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



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

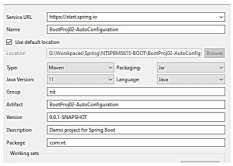
//target class DAO class(EmployeeDAO) ---------- //Dependent class DataSource (HikariDataSource)

=>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 Datasorurce 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..

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)

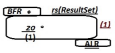
(These entries in application.properites given instruction to spring autoconfiguration process towards creating JDBC DataSource object and jdbc con objects in the JDBC con pool)

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 PreparedStatment object having SQL query
		PreparedStatement pscon.prepareStatement(GET_EMPS_COUNT);
		//excecute 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()

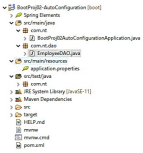
int count=rs.getInt(1);

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.

Different DataSource and Connection pool avaiable in the market

1 Standalone jdbc con pool

=>Suitable for standalone Apps

- => apache dbcp2
- => apache dbcp1
- => c3p0
- => oracle ucp
- => tomcat cp
- => hikari cp (best)
- => vibur cp
- and etc...

2 Server managed jdbc con pools

=>Suitable for those apps that are deployable in web server or Applicaiton server like web applications , spring rest Apps (Distributed Apps)

- => Tomcat server mangaed jdbc con pool
- => weblogic server mangaed jdbc con pool
- => GlassFish server mangaed jdbc con pool
- and etc..

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?

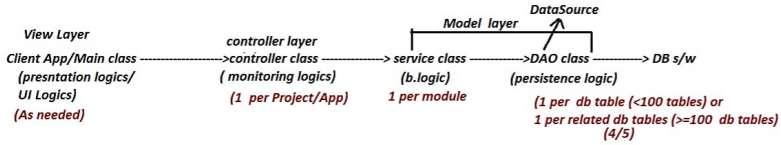
Ans) @Bean method DataSource object gets priority.

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

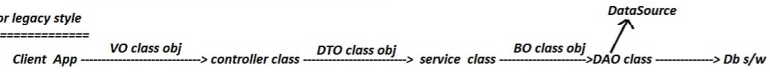


=> if 1 layer /class wants to pass more than 3 details/values to anothe layer or class then we need to pass them as Java Bean class object (java class with setter and getter methods)

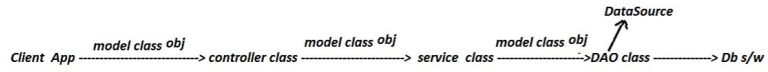
=>Earlier we use work with 3 types of Java beans to carry data across the layers
VO class (Value Object class)
DTO class (DataTransfer Object class)
BO class/Entity class/Model class (Business Object clas)

=> 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

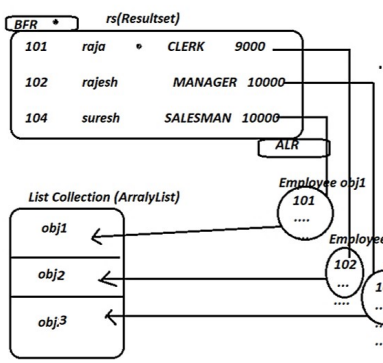
In this layered App I can spring Boot supplied IOC container
a) To give Datasource object through Autoconfiguration
b) To Inject DataSource obj to DAO class using @Autowired
c) To Inject DAO class obj to Service class using @Autowired
d) To Inject service class obj to controller class using @Autowired
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");

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

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

(f) Autoconfiguration activities 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
---------------------------------  (d)        obj (intera object)
#DataSource cfg                                 ....
spring.datasource.driver-class-name=.....  (e)  ....
spring.datasource.url=.....                     ....
spring.datasource.username=.....                ..
spring.datasource.password=.....
```

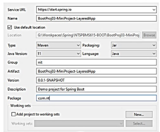
jars/starters :: spring-boot-starter-jdbc , oracle driver, lombok api

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 --->



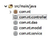
next ----> finish.

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



<u>step4) develop the Model class using lombok api annotation</u>

<u>Employee.java (model class)</u>

```java
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;

}
```

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

step5) Develop DAO Interface and DAO Impl class

DAO Interface

```java
---------------------
package com.nt.dao;

import java.util.List;

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

<u>//DAO Impl class</u>

```java
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
```

<u>step6) develop Service Interface and service impl class</u>

<u>service Interface</u>

```java
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;
}
```

<u>service Impl class</u>

```java
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;
	}

}
```

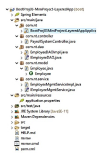
Can we cfg Controller, service , DAO calsses by just using @Component Annotation?

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.



step7) Develope the controller class

```java
//controller class
==================
package com.nt.controller;

import java.util.List;

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

import com.nt.model.Employee;
import com.nt.service.IEmployeeMgmtService;

@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;

	}

}
```

<u>step8) Write Client app code in the main(-) of main class</u>

```java
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

---

Assignment:: Display all corona patient details based on the given 3 cities
(For those students whose name starts with A to K)
(or)
Display all Tourist places details based on the given 3 states of india
(For those students whose name starts with L to Z)

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 http://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

overriing some property values thare collected parent project

depdeclares (about main jar files)

about dependent jar files info

addtional 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

---

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 issues.

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

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

(Not recomanded)

(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>
```

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

(Not recomanded to use compability issues may come)

URL to see pom.xml file "spring-boot-starter-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 plugins .. 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