STUDY MATERIAL

SPRING BOOT

NOTES BY ASHOK PATE

Enroll Now!

WWW.CODEWITHARRAYS.IN

Explore More

Subcription : Premium CDAC NOTES & MATERIAL

Contact to Join Premium Group

Click to Join Telegram Group

For More E-Notes

Join Our Community to stay Updated

TAP ON THE ICONS TO JOIN!

	codewitharrays.in freelance project available to buy contact on 8007592194	
SR.NO	Project NAME	Technology
1	Online E-Learning Platform Hub	React+Springboot+MySql
2	PG Mates / RoomSharing / Flat Mates	React+Springboot+MySql
3	Tour and Travel management System	React+Springboot+MySql
4	Election commition of India (online Voting System)	React+Springboot+MySql
5	HomeRental Booking System	React+Springboot+MySql
6	Event Management System	React+Springboot+MySql
7	Hotel Management System	React+Springboot+MySql
8	Agriculture web Project	React+Springboot+MySql
9	AirLine Reservation System / Flight booking System	React+Springboot+MySql
10	E-commerce web Project	React+Springboot+MySql
11	Hospital Management System	React+Springboot+MySql
12	E-RTO Driving licence portal	React+Springboot+MySql
13	Transpotation Services portal	React+Springboot+MySql
14	Courier Services Portal / Courier Management System	React+Springboot+MySql
15	Online Food Delivery Portal	React+Springboot+MySql
16	Muncipal Corporation Management	React+Springboot+MySql
17	Gym Management System	React+Springboot+MySql
18	Bike/Car ental System Portal	React+Springboot+MySql
19	CharityDonation web project	React+Springboot+MySql
20	Movie Booking System	React+Springboot+MySql
	freelance_Project available to buy contact on 8007592194	
21	Job Portal web project	React+Springboot+MySql
22	LIC Insurance Portal	React+Springboot+MySql
23	Employee Management System	React+Springboot+MySql
24	Payroll Management System	React+Springboot+MySql
25	RealEstate Property Project	React+Springboot+MySql
26	Marriage Hall Booking Project	React+Springboot+MySql
27	Online Student Management portal	React+Springboot+MySql
28	Resturant management System	React+Springboot+MySql
29	Solar Management Project	React+Springboot+MySql
30	OneStepService LinkLabourContractor	React+Springboot+MySql

31	Vehical Service Center Portal	React+Springboot+MySql
32	E-wallet Banking Project	React+Springboot+MySql
33	Blogg Application Project	React+Springboot+MySql
34	Car Parking booking Project	React+Springboot+MySql
35	OLA Cab Booking Portal	React+NextJs+Springboot+MySql
36	Society management Portal	React+Springboot+MySql
37	E-College Portal	React+Springboot+MySql
38	FoodWaste Management Donate System	React+Springboot+MySql
39	Sports Ground Booking	React+Springboot+MySql
40	BloodBank mangement System	React+Springboot+MySql
41	Bus Tickit Booking Project	React+Springboot+MySql
42	Fruite Delivery Project	React+Springboot+MySql
43	Woodworks Bed Shop	React+Springboot+MySql
44	Online Dairy Product sell Project	React+Springboot+MySql
45	Online E-Pharma medicine sell Project	React+Springboot+MySql
46	FarmerMarketplace Web Project	React+Springboot+MySql
47	Online Cloth Store Project	React+Springboot+MySql
48	Train Ticket Booking Project	React+Springboot+MySql
49	Quizz Application Project	JSP+Springboot+MySql
50	Hotel Room Booking Project	React+Springboot+MySql
51	Online Crime Reporting Portal Project	React+Springboot+MySql
52	Online Child Adoption Portal Project	React+Springboot+MySql
53	online Pizza Delivery System Project	React+Springboot+MySql
54	Online Social Complaint Portal Project	React+Springboot+MySql
55	Electric Vehical management system Project	React+Springboot+MySql
56	Online mess / Tiffin management System Project	React+Springboot+MySql
57	Online Examination Portal Project	React+Springboot+MySql
58	Lawyer / Advocate Appointment Booking System	React+Springboot+MySql
59	Café Management System	React+Springboot+MySql
60	Agriculture Product Rent system Portal	

Spring Boot + React JS + MySQL Project List

Sr.No	Project Name	YouTube Link
1	Online E-Learning Hub Platform Project	https://youtu.be/KMjyBaWmgzg?si=YckHuNzs7eC84-IW
2	PG Mate / Room sharing/Flat sharing	https://youtu.be/4P9clHg3wvk?si=4uEsi0962CG6Xodp
3	Tour and Travel System Project Version 1.0	https://youtu.be/-UHOBywHaP8?si=KHHfE_A0uv725f12
4	Marriage Hall Booking	https://youtu.be/VXz0kZQi5to?si=llOS-QG3TpAFP5k7
5	Ecommerce Shopping project	https://youtu.be/vJ_C6LkhrZ0?si=YhcBylSErvdn7paq
6	Bike Rental System Project	https://youtu.be/FIzsAmIBCbk?si=7ujQTJqEgkQ8ju2H
7	Multi-Restaurant management system	https://youtu.be/pvV-pM2Jf3s?si=PgvnT-yFc8ktrDxB
8	Hospital management system Project	https://youtu.be/IynIouBZvY4?si=CXzQs3BsRkjKhZCw
9	Municipal Corporation system Project	https://youtu.be/cVMx9NVyI4I?si=qX0oQt-GT-LR_5jF
10	Tour and Travel System Project version 2.0	https://youtu.be/_4u0mB9mHXE?si=gDiAhKBowi2gNUKZ

Sr.No	Project Name	YouTube Link
11	Tour and Travel System Project version 3.0	https://youtu.be/Dm7nOdpasWg?si=P_Lh2gcOFhlyudug
12	Gym Management system Project	https://youtu.be/J8_7Zrkg7ag?si=LcxV51ynfUB7OptX
13	Online Driving License system Project	https://youtu.be/3yRzsMs8TLE?si=JRI_z4FDx4Gmt7fn
14	Online Flight Booking system Project	https://youtu.be/m755rOwdk8U?si=HURvAY2VnizIyJlh
15	Employee management system project	https://youtu.be/lD1iE3W_GRw?si=Y_jv1xV_BljhrD0H
16	Online student school or college portal	https://youtu.be/4A25aEKfei0?si=RoVgZtxMk9TPdQvD
17	Online movie booking system project	https://youtu.be/Lfjv_U74SC4?si=fiDvrhhrjb4KSlSm
18	Online Pizza Delivery system project	https://youtu.be/Tp3izreZ458?si=8eWAOzA8SVdNwlyM
19	Online Crime Reporting system Project	https://youtu.be/0UlzReSk9tQ?si=6vN0e70TVY1GOwPO
20	Online Children Adoption Project	https://youtu.be/3T5HC2HKyT4?si=bntP78niYH802I7N

Sr.No	Project Name	YouTube Link
21	Online Bus ticket booking system Project	https://youtu.be/FJ0RUZfMdv8?si=auHjmNgHMrpaNZvY
22	Online Mess / Tiffin Booking System Project	https://youtu.be/NTVmHFDowyI?si=yrvClbE6fdJ0B7dQ
23		
24		
25		

TAP ON THE ICONS TO JOIN!

==========
Spring Boot & Microservices
==========

Pre-Requisites :

1) Core Java
2) Adv Java (JDBC + Servlets)
3) Oracle (SQL)
4) Hibernate (Basics of ORM)

Course Content :

1) Spring Core

2) Spring JDBC

3) Spring Boot

4) Spring Data JPA

5) Spring Web MVC

6) RESTful Services (Spring REST)

7) Spring Cloud

8) Microservices

9) Spring Security

10) Spring Boot Integrations

- Apache kafka
- Redis cache
- Docker
- Unit Testing (Junit)
- Logging

```
Programming Language : Core Java

              - Language Fundamentals
              - Syntaxes

              -> Standalone apps


Technologies : JDBC + Servlets + JSP

              -> Database communication using JDBC
              -> Servlets for web application development
              -> JSP (Presentation logic)
```

```
Java Frameworks :  Hibernate +  Spring --> Spring Boot

=================================================================

1) Programming Language  (Java)

2) Java Technologies (JDBC + Servlets + JSP)

3) Java Frameworks ( Hibernate + Spring & Spring Boot)

=================================================================
```

```
=================================================================


Core Java : To develop Standalone applications

JDBC : To develop persistence logic

Servlets : To develop web applications

JSP : To develop Presentation logic

=================================================================
```

```
Project Contains Several Logics

1) DB Logics

2) Business Logic

3) Web Logics (Request & Response)

4) Presentation Logics (User Interface)
```

1) DB Communication is mandatory

- 1.1 Load Driver
- 1.2 get connection
- 1.3 create stmt
- 1.4 excute query (it will be changed based on req)
- 1.5 close connection

Note: We have to write above steps in every program. We will end up by writing repeated code (Boiler Plate Code).

2) Web Logics (Request & Response)

- 1.1 Develop Form Display Logic
- 1.2 Form Validations
- 1.3 Capture Form data
- 1.4 Store Form data in java object

What is Framework ?
==========

=> Framework is a semi developed software

=> Frameworks will provide some common logics required for project development.

=> Frameworks provides re-usable components

=> Frameworks will reduce burden on the developers

Ex:
===

=> Capture form data and store into DB table

JDBC + Servlets : 20 lines of code

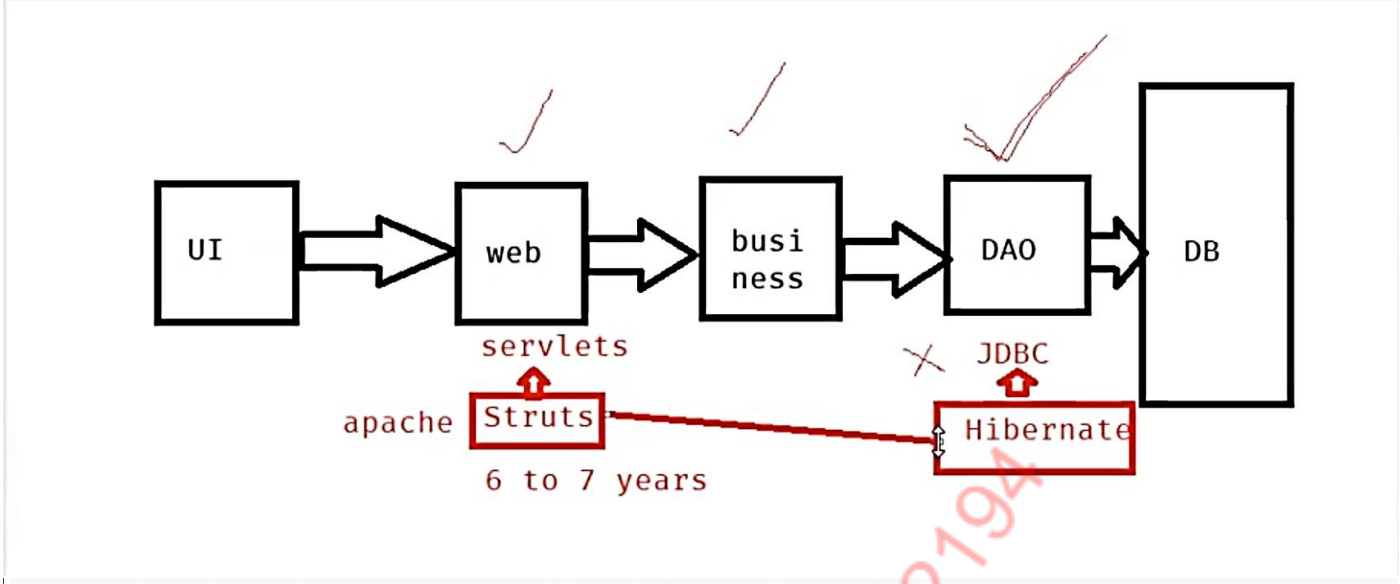
Spring : 5 lines of code

=> IN Java community we are having 2 types of frameworks

1) ORM frameworks (Ex: Hibernate)

2) Web Frameworks (Ex: Struts)

Note: Spring is called as Application development Framework.



-> Spring is a java based application development framework

-> By using spring we can develop end to end application

-> Spring is developed by using JSE & JEE

-> Spring framework developed by Rod Johnson

-> First version of spring came into market in 2004 (Spring 1.x v)

-> The current version of Spring is 6.x (2022)

-> SPring is developed in modular fashion

-> Spring Modules are loosely coupled

-> Spring is versatile framework (it can integrate with any other framework)

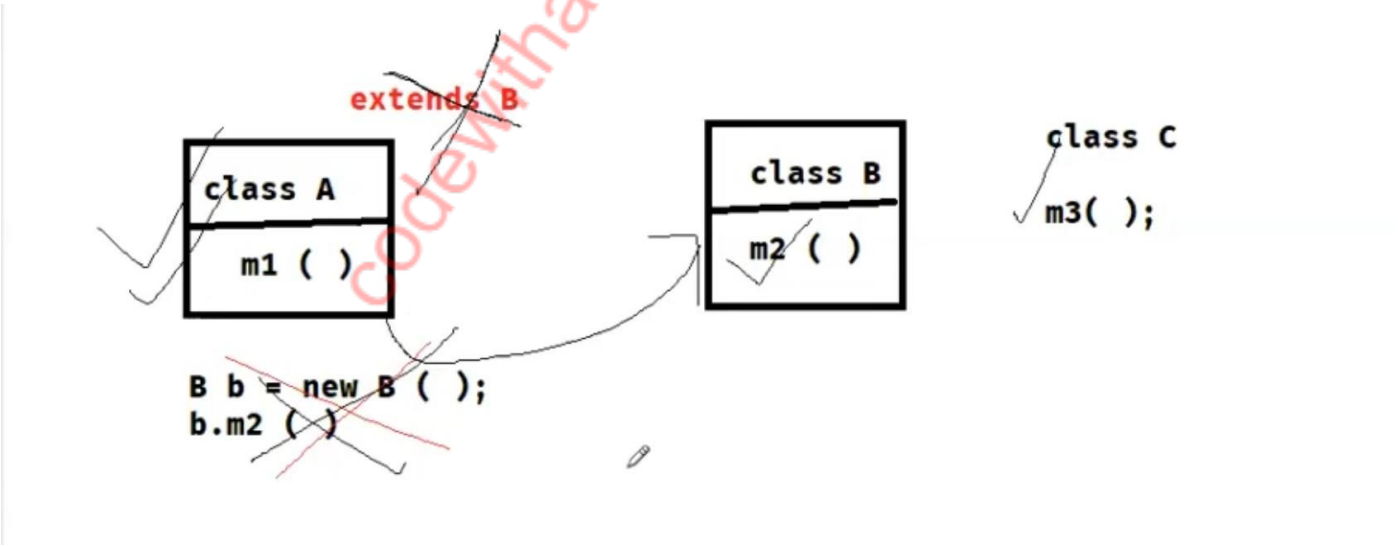
1) Spring Core
2) Spring Context
3) Spring DAO
4) Spring ORM
5) Spring AOP
6) Spring Web MVC
7) Spring REST

```
8) Spring Data JPA

9) Spring Cloud

10) Spring Security

11) Spring Social

12) Spring Batch ........
```

```
=> Spring Core module is base module in Spring Framework.
=> Spring Core Providing IOC Container & Dependency Injection
=> Using IOC & DI we can develop our classes with loosely coupling.
```

Why need of IOC & DI ?

- ➔ I have a class A and Class B and in that I have method m1() and m2() if I want to called m2() in class A I have to create an object then I can call that method on object but this is not recommended its tightly coupled suppose I have thousand of class then tomorrow if I want to change the logic from the method it difficult to change from everywhere. So we need to loosely coupled for that we need IOC and DI
- ➔ Also we can extends class to override some method of class A in Class B but I thousand class in multiple inheritance not supported in java because of that we need IOC and DI
- ➔ Because of this the Spring Core Module come in the picture.

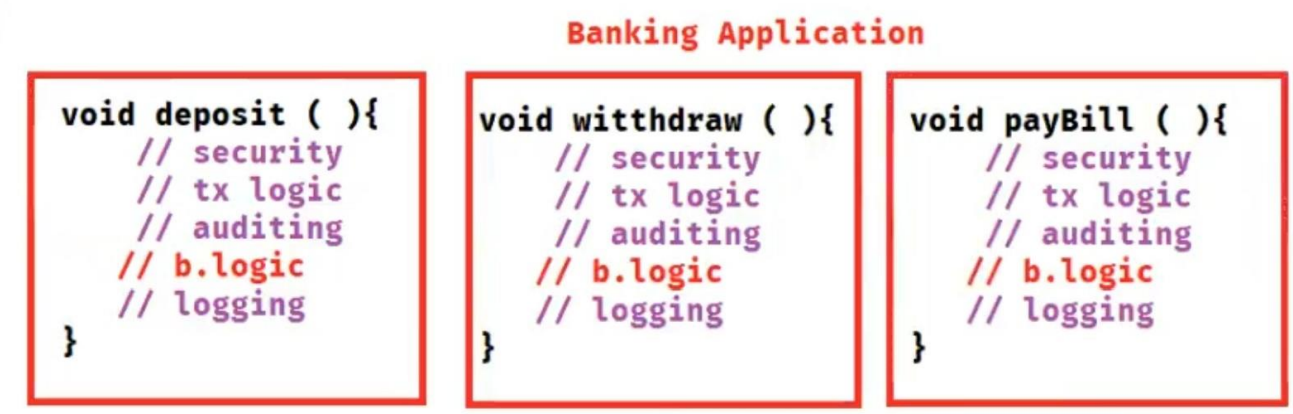


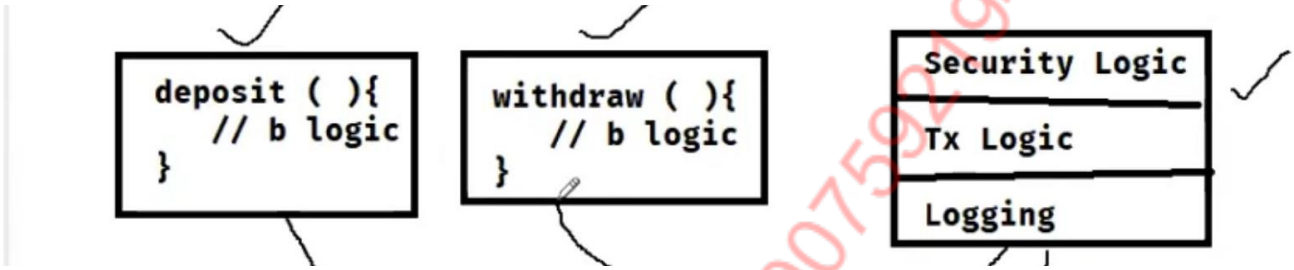
```
============
Spring Core
===========

-> Base module of spring framework

-> Core Module providing fundamental concepts of Spring those are
    IOC & DI

        IOC : Inversion Of Control
        DI : Dependency Injection

-> IOC & DI are used to develop classes with loosely coupling

-> IOC will take care of java objects life cycle (Spring Beans)
```

```
================
Spring Context
===============

-> TO manage configurations in Spring application we will use Context Module

-> It provides configuration support required for managing classes

=============
Spring AOP
=============

AOP : Aspect Oriented Programming

-> AOP is used to seperate cross cutting logics in the application


          Cross Cutting / Secondary / Helper Logics

Ex : Security, transaction, Logging, Auditing & exception handling etc...
```


Why we need Spring AOP?

➔ I have a class A and in that several methods availble like deposit(), withdraw(), paybill(), in each mehtod need security logic , then transaction logic, then auditing logic, then business logic , then logging is required suppose we have thousand method then this are repeating again and again and also code length is increase. And in future if I want to change the logic in each method I have to changes. To overcome this the Spring AOP module comes in picture. In mehtod we write only buisness logic and other we can write separate for each.






```
JDBC Logic
-----------
1. load driver
2. get connection
3. Create statement
4. Execute query
5. process result
6. Close connection

spring jdbc
-----------
1. Execute query
2. process result

=> Spring JDBC provided predefined classes to perform DB operations

        Ex: JdbcTemplate, NamedJdbcTemplate etc...
```

```
===========
Spring ORM
===========

=> SPring ORM module is extension for existing ORM frameworks

ORM - Object relational mappping

=> To suppport ORM integrations we have Spring ORM module

    Spring ORM = Spring + ORM Framework (Ex: Hibernate)
```

```
Hibernate
---------
1. Create Config obj
2. create session factory
3. create session
4. begin tx
5. execute methods
6. commit tx
7. close session
8. close sf

spring orm
-----------
HibernateTemplate.save(entityObj)
```

```
===============
Spring Web MVC
===============

-> To develop both web & distributed (webservices) applications

-> It is used to simplify web layer development in applications
```

```
==================
Spring Data JPA
=================

-> It is extension for Spring ORM

-> It is providing ready made methods to perform CRUD operations in DB
```

➔ We have these 5 options for to developed persistence layer but now days spring data JPA mostly used because we no need to write code the ready made methods available the classes and interfaces are available.

1) Jdbc
2) Spring JDBC
3) Hibernate
4) Spring ORM
5) Spring Data JPA

```
===============
Spring Security
===============

-> It is used to secure our spring based application

-> We can implement both Authentication & Authorization by using Spring
Security

Authentication : Decide who can access our application ?

Authorization : Identify logged in user having access for the functionality
or not ?
```

Spring Batch

=> Batch means bulk operation

Ex:

sending bulk email to customers

sending bulk sms to students regarding course update

sending bank statement to account holders

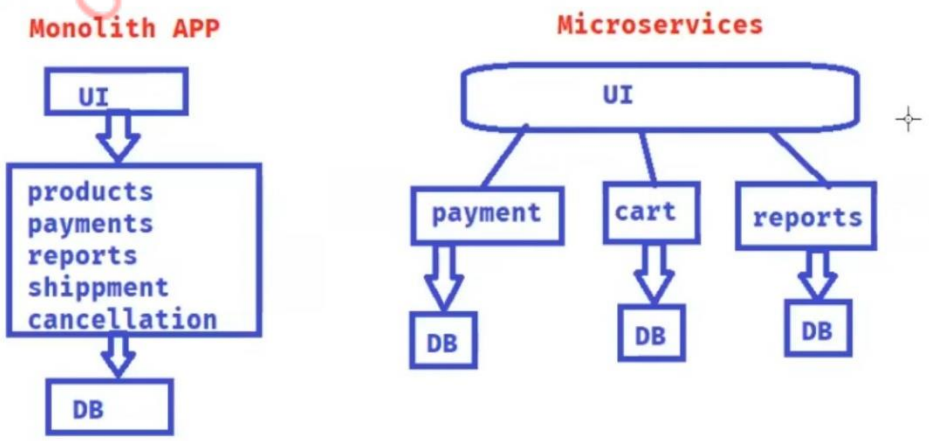
read records from file and store into DB

```
=============
Spring Cloud
=============

-> It provides configurations required for Microservices development

Service Registry
Admin Server
API Gateway
```


There are two types project in industry 1. Monolith project 2. Microservices.

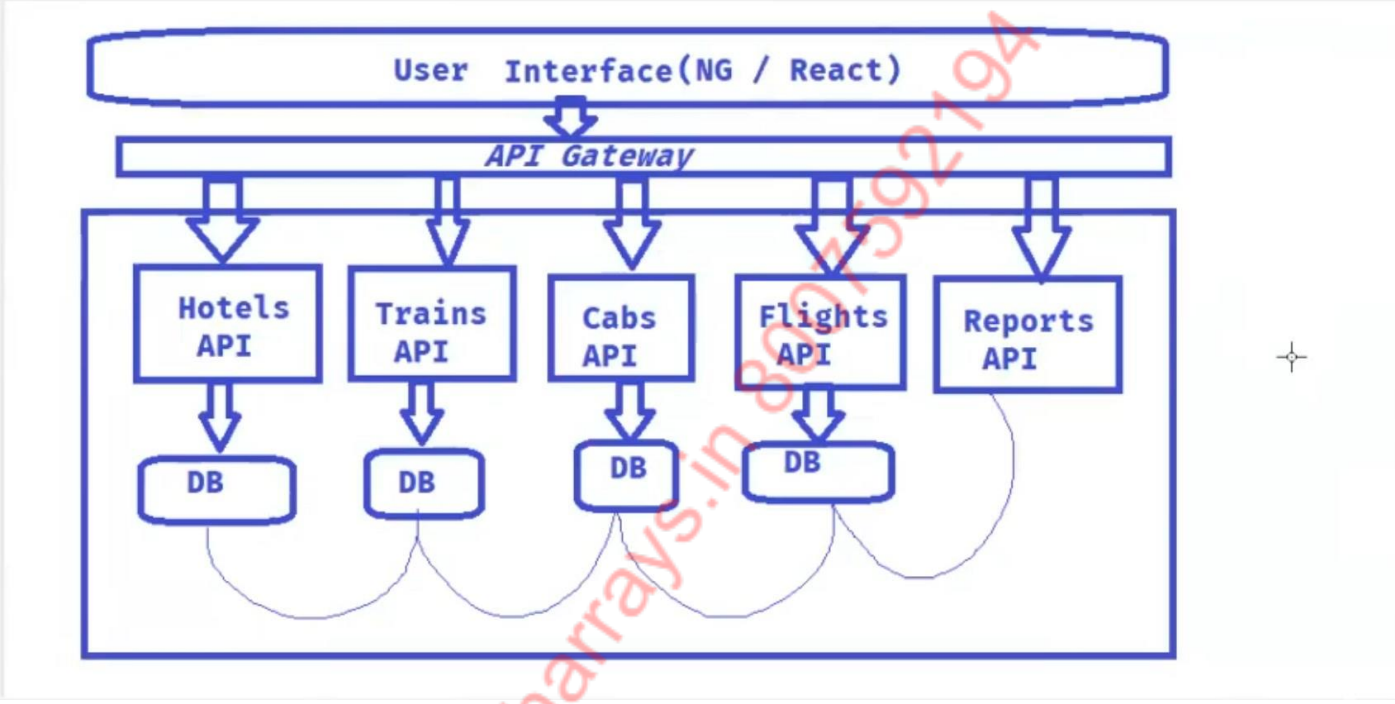
1.Monolith: means if you are developing all functionality in single project. The problem in monolith is maintainace problem. If I want to change one of the functionalities whole projects again compiled, again whole project packaged and then whole project deployed.

2.Microservices: to overcome this problem we can use the microservices. The microservices means the whole project made up in the small small project. And those small projects know as rest services. If tomorrow the maintainace occurs only that that one services we can work on that.



How to developed a full stack project?

➔ For example we have to developed a project like MakeMyTrip so first we have to made different API services like Hotel Api , Tains Api, Cabs Api, Flight Api, Reports Api and the database either separated or may be same depends on you and requirement after that we have to check all this Api services on the postman tool and if all is good and running perfect then we are going to developed a UI and after that we can integrate our UI with backend with the help of API Gateway. This Api Gateway is the mediator between frontend and backend.



Persistence layer: means which is responsible to communicate with the database is know as persistence layer.

Distributed application: means one application is communicating with another application is known as distributed application. Ex Swiggy, Zomato, Phone Pay, MakeMyTrip etc...

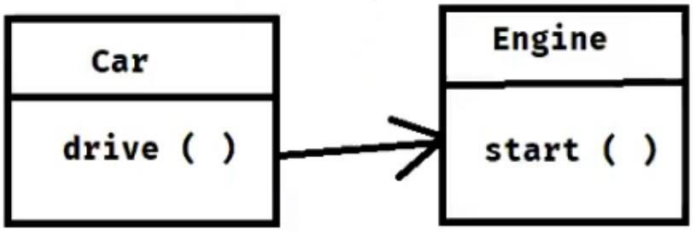
Spring Core

-> Base module of Spring Framework

-> Providing IoC & DI

-> IOC & DI are used to develop classes with loosely coupling

-> First a fall we need to understand what is tightly coupling. For that we can take one small car app example.



1) IS - A

2) HAS - A

➔ I have a car class and in that drive() method and another one class engine in that start() Method. We can make this by using IS-A relation or HAS-A relation.

```
package com.codewitharrays;

public class Car extends Engine{

        public void drive() {
            int status=super.start();
            if(status>=1) {
                System.out.println("Journey Started");
            }
            else {
                System.out.println("Engine Problem");
            }

        }
}
```

```
package com.codewitharrays;

public class Engine {
    public int start() {
        //logic
        return 1;
    }
}
```

```
package com.codewitharrays;

public class App {
    public static void main (String[]arg ) {
        Car c=new Car();
        c.drive();
    }
}
```


-> Car is extending Engine class
-> Car class cannot use inheritance in future
-> Any changes in Engine class will effect on Car class
-> Car is tightly coupled with Engine
Note: It is not recommended to develop classes with Tightly coupling.

➔ Now we can make by using Has-A Relation. Now my class is open for inheritance in future. And now I make a object and called method on object but When I do changes in future like parameter constructor make and the I get the error. So, its tightly coupled. And not recommended.

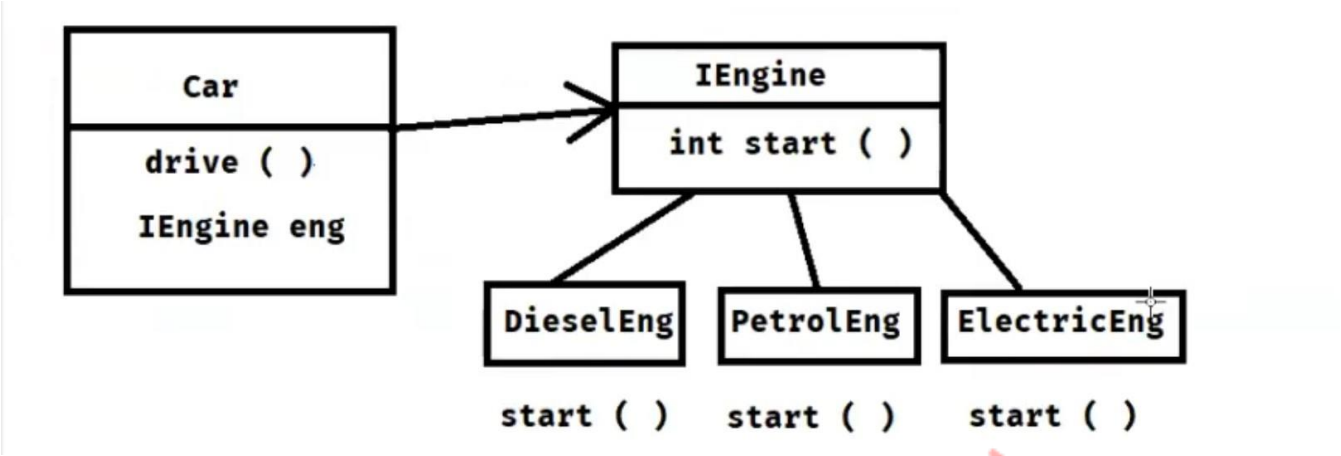
```
public class Car {

    public void drive() {
        Engine eng=new Engine();
        int status= eng.start();
        if(status>=1) {
            System.out.println("Journey Started");
        }
        else {
            System.out.println("Engine Problem");
        }

    }
}
```

```
package com.codewitharrays;

public class Engine {

    public Engine(String fuelType) {

    }

    public int start() {
        //logic
        return 1;
    }
}
```

```
package com.codewitharrays;

public class App {
    public static void main (String[]arg ) {
        Car c=new Car();
        c.drive();
    }
}
```


-> Car is directly creating Object for Engine
-> Any changes to Engine class will effect on Car class
-> Car is always talking to only one Engine
-> If i want to change from one Engine to another Engine then we should modify Car class code.
Note: Car is tightly coupled with Engine.

➔ To overcome above problem, we make engine class as Interface class and now the other class can implement engine class. Now we achieved the loosely coupled.




```java
package com.codewitharrays;

public interface IEngine {

    public int start();

}
```

```java
package com.codewitharrays;

public class PetrolEngine implements IEngine{

        public int start() {

            System.out.println("Petrol Engine Started...");
            return 1;
        }
}
```

```java
package com.codewitharrays;

public class DiselEngine implements IEngine {

    public int start() {
        System.out.println("Disel engine started");

        return 1;
    }
}
```

```java
public class Car {

    private IEngine eng;
    public Car(IEngine eng) {
        this.eng=eng;
    }

    public void drive() {

        int status=eng.start();

        if(status>=1) {
            System.out.println("Journey Started");
        }
        else {
            System.out.println("Engine Problem");
        }

    }
}
```

```
package com.codewitharrays;

public class App {

    public static void main (String[]arg ) {
        //Here we can pass the class object we can change petrol to disel
        //also without change in car class. So we achived loosely coupled.
        Car c=new Car(new PetrolEngine());
        c.drive();
        Car cb=new Car(new DiselEngine());
        cb.drive();
    }
}
```

What is Dependency Injection?

- The process of injecting dependent object into target object using target class variable / setter method / constructor is called as Dependency Injection.
- The process of injecting one class object into another class object is called dependency injection.

Dependency Injection Types

1) Field Injection (variable)
2) Setter Injection (setter method)
3) Constructor Injection (constructor)

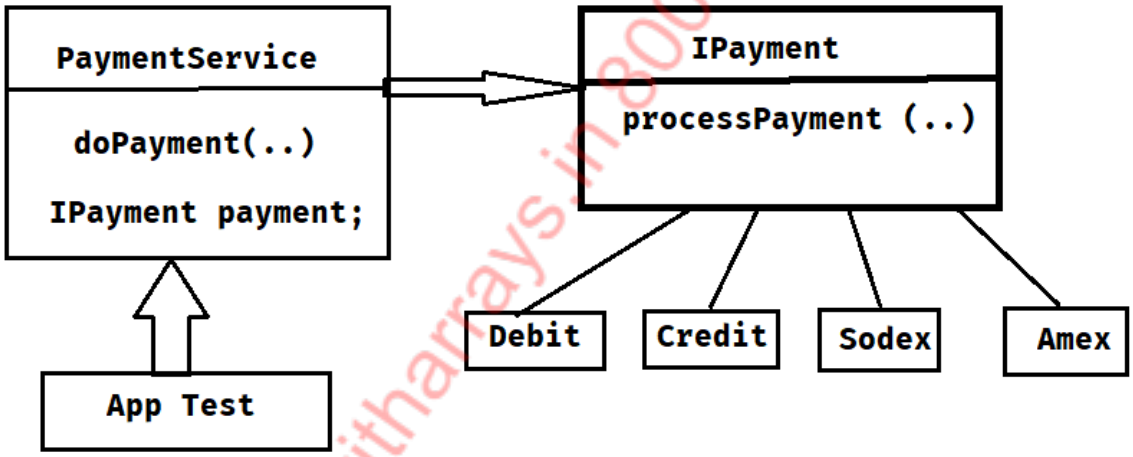
- If I'm doing injecting object by constructor that is constructor Injection.
- If I'm doing injecting object by setter method that is setter Injection.
- If I'm doing injecting object by field that is field Injection.
- Understand this concept by coding below.

```
public class Car {

//  private IEngine eng;
    IEngine eng; // field

    public Car(IEngine eng) {
        this.eng=eng;   //Constructor
    }

    public void setEng(IEngine eng) {
        this.eng=eng;  //Setter
    }

    public void drive() {

        int status=eng.start();

        if(status>=1) {
            System.out.println("Journey Started");
        }
        else {
            System.out.println("Engine Problem");
        }

    }
}
```

```
public class App {

    public static void main (String[]arg ) {

        Car c=new Car(new PetrolEngine()); //Constructor Injector
        c.drive();
        Car cb=new Car();
        cb.setEng (new DiselEngine()); //Setter Injector

        cb.eng=new PetrolEngine(); //field Injector

    }
}
```

Requirement:

Develop an application to perform bill payment. It should support for multiple Payment options (Debit card, Credit Card, Sodex & Amex....)




```
package com.codewitharrays;

public interface IPayment {

        public boolean processPayment(double billAmt);
}
```

```
package com.codewitharrays;

public class CreditCardPayment implements IPayment {

    @Override
    public boolean processPayment(double billAmt) {
        System.out.println("Creditcard payment processing..");
        return true;
    }
}
```

```java
package com.codewitharrays;

public class DebitCardPayment implements IPayment{

    @Override
    public boolean processPayment(double billAmt) {
        System.out.println("debitcard payment processing..");
        return true;
    }
}
```

```java
package com.codewitharrays;

public class SodexCardPayment implements IPayment {

    @Override
    public boolean processPayment(double billAmt) {
        System.out.println("Sodex payment processing..");
        return true;
    }
}
```

```java
package com.codewitharrays;

public class AmexCardPayment implements IPayment {

    @Override
    public boolean processPayment(double billAmt) {
        System.out.println("AmexCard payment processing..");
        return true;
    }
}
```

```java
package com.codewitharrays;

public class PaymentService {
        private IPayment payment;

        public PaymentService(IPayment payment) {
            this.payment=payment;
        }
        public void setPayment(IPayment payment) {
            this.payment=payment;
        }

        public void doPayment(double billAmt) {
            boolean status=  payment.processPayment(billAmt);

            if(status) {
                System.out.println("Printing Recipt..");
            }else {
                System.out.println("Payment Decline.");
            }
        }

}
```

```java
package com.codewitharrays;

public class App {
        public static void main1(String[]args) {
            IPayment p=new DebitCardPayment();
            PaymentService ps=new PaymentService(p); //Constructor Injector
            ps.doPayment(450.0);
        }
        public static void main(String[]args) {
            IPayment p1=new CreditCardPayment();
            IPayment p2=new DebitCardPayment();

            PaymentService ps=new PaymentService(p1); //Constructor Injector
            ps.setPayment(p2);  //Setter Injector
            ps.doPayment(100.00);
//          Ans: debitcard payment processing..      Printing Recipt..
//          First Constructor injection will happen then it will initialize the variable
//          then setter injection will happen and it will re-initialize the same variable
//          so final value be setter injection value.
//          Note: Setter Injection will override Constructor injection.

        }
}
```

➔ Right now programmer do the dependency injection but I want to IoC container inject the dependency that will saw in the next case.

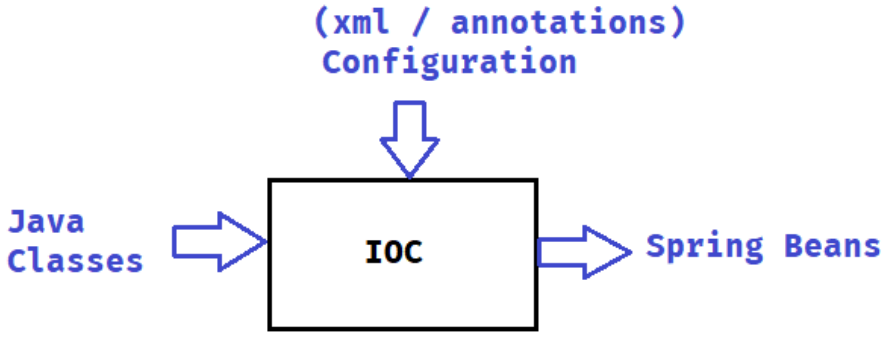
What is IoC ?

IOC: Inversion of Control

-> IOC is a principle which is used to manage and collaborate dependencies among the objects in the application.

-> In Spring, IOC is responsible for Dependency Injection.

-> What is the difference between IOC container and Dependency Injection. so many people will ask that question in the interview then tell them that the question itself is wrong we cannot compare IOC and DI. IoC is performing the DI how can we compare that.



- In Spring we have do configuration but in spring boot the auto configuration is available because of that spring boot came in the market.
- In Spring we can do configuration by using XML and annotation. But in spring boot we can do configuration by using annotation only.

Note: For IOC we need to pass Java Classes + Configuration as input then IOC will perform DI and it will produce Spring Beans.

Spring Bean: The class which is managed by IOC is called as Spring Bean.

How start IOC container?

=> We can start in 2 ways

1) BeanFactory (I) (outdated)

2) ApplicationContext (I) (recommended)

ApplicationContext ctxt = new ClassPathXmlApplicationContext(String xmlFile)

Creating First Spring Project

1) Open STS IDE

2) Create Maven Project

3) Open pom.xml file and add below dependency

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>5.3.25</version>
    </dependency>
</dependencies>
```

Note: After adding dependency verify project Maven Dependencies folder (jars should be displayed)

4) Create Required Java classes

```
package in.codewitharrays.beans;

public interface IPayment {

    public boolean processPayment(double billAmt);

}
```

```java
package in.codewitharrays.beans;

public class DebitCardPayment implements IPayment {

    public DebitCardPayment() {
        System.out.println("DebitCardPayment :: Constructor");
    }

    public boolean processPayment(double billAmt) {

        // logic

        System.out.println("DebitCard Payment successfull...");

        return true;
    }

}
```

```java
package in.codewitharrays.beans;

public class CreditCardPayment implements IPayment {

    public CreditCardPayment() {
        System.out.println("CreditCardPayment :: Constructor");
    }

    public boolean processPayment(double billAmt) {
        // logic

        System.out.println("CreditCard Payment successfull...");

        return true;
    }

}
```

```java
package in.codewitharrays.beans;

public class PaymentService {

    private IPayment payment;

    public PaymentService() {
        System.out.println("PaymentService :: 0-Constructor");
    }

    public PaymentService(IPayment payment) {
        System.out.println("PaymentService :: Param-Constructor");
        this.payment = payment;
    }

    public void setPayment(IPayment payment) {
        System.out.println("setter method called...");
        this.payment = payment;
    }

    public void doPayment(double billAmt) {

        boolean status = payment.processPayment(billAmt);

        if (status) {
            System.out.println("Recipt Printing....");
        } else {
            System.out.println("Card Declined...");
        }
    }
}
```

5) Create Bean Configuration File like below

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

    <bean id="credit" class="in.codewitharrays.beans.CreditCardPayment" />

    <bean id="debit" class="in.codewitharrays.beans.DebitCardPayment" />

    <bean id="payment" class="in.codewitharrays.beans.PaymentService">
        <constructor-arg name="payment" ref="credit" />
        <property name="payment" ref="debit" />
    </bean>

</beans>
```

-> Constructor Injection will be represented like below or we can seen in bean configuration

```
<constructor-arg name="payment" ref="credit" />

// setter injection
<property name="payment" ref="debit" />
```

Note: ref attribute represents which object should be injected.

6) Create Test Class and start IOC Container

```
package in.codewitharrays.beans;

import org.springframework.context.ApplicationContext;

public class Test {

    public static void main(String[] args) {

        ApplicationContext context =
                new ClassPathXmlApplicationContext("Beans.xml");

        PaymentService service =
                context.getBean(PaymentService.class);

        service.doPayment(199.00);

    }
}
```

Output:

```
CreditCardPayment :: Constructor
DebitCardPayment :: Constructor
PaymentService :: Param-Constructor
setter method called...
DebitCard Payment successfull...
Recipt Printing....
```

Spring Bean: Class Managed by IOC container is called Spring bean

Spring Bean Scopes (Refer page 19 code to understand this all concept)

-> Bean Scope will decide how many objects should be created for Spring Bean class

-> We have 4 types of scopes

1) singleton (default)
2) prototype
3) request
4) session

-> Singleton means only one object will be created

-> Prototype means every time new object will be created

Note: request & session scopes we will use in spring web MVC.

Example of Singleton Scope:

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

    <bean id="disel" class="in.codewitharrays.beans.DieselEngine" primary="true"/>

    <bean id="petrol" class="in.codewitharrays.beans.PetrolEngine" />

    <bean id="car" class="in.codewitharrays.beans.Car"  >
        <property name="eng" ref="petrol"></property>
    </bean>
```

Observe this carefully in this I did not mention the scope so it is a default singleton scope. And because of that the only one object is created. You can see below we created 3 objects but still give one hash code so it's only create one object. It is known as Singleton bean scope.

```
import org.springframework.context.ApplicationConte

public class Test {

    public static void main(String[] args) {

        ApplicationContext ctxt =
                new ClassPathXmlApplicationContext('

        Car c1 = ctxt.getBean(Car.class);
        System.out.println(c1.hashCode());

        Car c2 = ctxt.getBean(Car.class);
        System.out.println(c2.hashCode());

        Car c3= ctxt.getBean(Car.class);
        System.out.println(c3.hashCode());
    }
}
```

```
<terminated> Test (1) [Java Application] C:\
DieselEngine.Cons
PetrolEngine.Cons
Car::Constructor
1362546706
1362546706
1362546706
```

Example of Prototype Scope:

Observe this carefully in this I mention the scope prototype. And because of that the three object is created. You can see below we created 3 objects and complier give three different hash code so it's create three object. When every time new object will be created this known as prototype scope.

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

    <bean id="disel" class="in.codewitharrays.beans.DieselEngine" primary="true"/>

    <bean id="petrol" class="in.codewitharrays.beans.PetrolEngine" />

    <bean id="car" class="in.codewitharrays.beans.Car" scope="prototype" >
            <property name="eng" ref="petrol"></property>
    </bean>
```

```java
public class Test {

    public static void main(String[] args) {

        ApplicationContext ctxt =
                new ClassPathXmlApplicationContext('

        Car c1 = ctxt.getBean(Car.class);
        System.out.println(c1.hashCode());

        Car c2 = ctxt.getBean(Car.class);
        System.out.println(c2.hashCode());

        Car c3= ctxt.getBean(Car.class);
        System.out.println(c3.hashCode());
    }
}
```

```
<terminated> Test (1) [Java Application] C:\
DieselEngine.Cons
PetrolEngine.Cons
Car::Constructor
1362546706
Car::Constructor
1496949625
Car::Constructor
236840983
```


Autowiring (Refer page 19 code to understand this all concept)

=> Auto wiring is used to identify dependent objects and inject into target objects.

=> Autowiring works based on below 3 modes

1) byName

2) byType

3) constructor

-> byName MODE will identify dependent bean based on variable name matching with bean id.

-> byType MODE will identify dependent bean based on variable data type.

1.byName:

- In byName dependent bean base on variable name matching with bean id. So my variable is eng so it will check with eng and DieselEngine object will inject. And in output you can observe that.

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

    <bean id="eng" class="com.codewitharrays.beans.DieselEngine" />

    <bean id="eng1" class="com.codewitharrays.beans.PetrolEngine" />

    <bean id="car" class="com.codewitharrays.beans.Car" autowire="byName" >

    </bean>

</beans>
```

```
DieselEngine.Constructor....)
PetrolEngine.Constructor....
Default contructor
DieselEngine started....
Journey Started...
```

- If the both variable name is same for DieselEngine and PetrolEngine then it will give the ambiguity error.

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

    <bean id="eng" class="com.codewitharrays.beans.DieselEngine" />

    <bean id="eng" class="com.codewitharrays.beans.PetrolEngine" />

    <bean id="car" class="com.codewitharrays.beans.Car" autowire="byName" >

    </bean>

</beans>
```

```
ngframework.beans.factory.parsing.BeanDefinitionParsingException: Configuration problem: Bean name 'eng' is already used in this <beans> element
urce [Beans.xml]
```

- If bean variable is not present then the null pointer exception occurs. In below my variable is eng and in bean scope the id eng1, eng2 that is not in my class so null exception occurs.

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

    <bean id="eng1" class="com.codewitharrays.beans.DieselEngine" />

    <bean id="eng2" class="com.codewitharrays.beans.PetrolEngine" />

    <bean id="car" class="com.codewitharrays.beans.Car" autowire="byName" >

    </bean>

</beans>
```

```
DieselEngine.Constructor....)
PetrolEngine.Constructor....
Default contructor
Exception in thread "main" java.lang.NullPointerException: Cannot invoke "com.codewitharrays.beans.IEngine.start()" because "this.eng" is null
        at com.codewitharrays.beans.Car.drive(Car.java:16)
        at com.codewitharrays.beans.Test.main(Test.java:11)
```

2.byType:

- When Spring's **Autowiring byType** is enabled, Spring will search for a bean in the container that matches the type of the variable datatype or constructor parameter in the class. If it finds exactly one matching bean of that type, Spring will inject that bean. If no matching bean or more than one bean of the same type is found, Spring will throw an exception
- byType MODE will identify dependent bean based on variable data type.
- In below example the bean id only one is available and variable datatype IEngine inject with DieselEngine.

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

    <bean id="eng1" class="com.codewitharrays.beans.DieselEngine" />

    <bean id="car" class="com.codewitharrays.beans.Car" autowire="byType" >

    </bean>

</beans>
```

Output:

DieselEngine.Constructor....)
Default contructor
DieselEngine started....
Journey Started...

- **Note:** There is a chance of getting ambiguity in byType mode.
- If variable data type is interface, then we can have multiple implementation classes in this scenario IOC cannot decide which bean it has to inject.

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

    <bean id="eng1" class="com.codewitharrays.beans.DieselEngine" />

    <bean id="eng2" class="com.codewitharrays.beans.PetrolEngine" />

    <bean id="car" class="com.codewitharrays.beans.Car" autowire="byType" >

    </bean>

</beans>
```

- We can resolve byType ambiguity in 2 way

1.autowire-candidate="false":

- Agar hum autowire-candiate=”false” karte hai tho use ignore kr denga aur remaining one ko inject kr denga. Example iss mai hum ne DieselEngine ko false kiya tho PetrolEngine ka object inject ho gaya hai.
- Lekin agar 10 beans hai tho hume har jagah autowire-candidate=”false” karna padenga is liye dusra option hai.

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

    <bean id="eng1" class="com.codewitharrays.beans.DieselEngine" autowire-candidate="false"/>

    <bean id="eng2" class="com.codewitharrays.beans.PetrolEngine" />

    <bean id="car" class="com.codewitharrays.beans.Car" autowire="byType" >

    </bean>

</beans>
```


Output:

DieselEngine.Constructor....)
PetrolEngine.Constructor....
Default contructor
PetrolEngine Started...
Journey Started...

2) primary = "true":

- Jiska object inject krna hai aur jise prioprity pe rakhna hai use primary=”true” krdo tho uska object inject ho jayenga.
- Abhi iss mai DieselEngine bean mai primary=”true” kr diya issliye uska object inject ho gaya check output carefully.

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

    <bean id="eng1" class="com.codewitharrays.beans.DieselEngine" primary="true"/>

    <bean id="eng2" class="com.codewitharrays.beans.PetrolEngine" />

    <bean id="car" class="com.codewitharrays.beans.Car" autowire="byType" >

    </bean>

</beans>
```

Output:

DieselEngine.Constructor....)
PetrolEngine.Constructor....
Default contructor
DieselEngine started....
Journey Started...

Note: Why this byType is better than byName agar koi developer ne ek eng variable banaya aur bean me inject kr diya byName. Lekin In future agar kisi dusre developer ne variable name change kr diya tho. Code may gets failed. But in byType the id is anything injection is happened on variable datatype.

3. Constructor: **(Refer page 19 code to understand this all concept / only one constructor is added extra)**

- In constructor auto wire first it will check byName if the byName variable is available the bean is inject. If not then it will go for byType and the multiple class is implement by interface then the primary=”true” is necessary to give for high priority to inject the bean. Constructor is the combination of byName & byType.

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

    <bean id="eng" class="com.codewitharrays.beans.DieselEngine" />

    <bean id="eng2" class="com.codewitharrays.beans.PetrolEngine" />

    <bean id="car" class="com.codewitharrays.beans.Car" autowire="constructor" >

    </bean>

</beans>
```

```java
package com.codewitharrays.beans;

public class Car {
    private IEngine eng;
    public Car() {
      System.out.println("Default contructor");
    }

    public Car(IEngine eng) {
          System.out.println("parameter contructor");
          this.eng=eng;
        }

//  public void setEng(IEngine eng) {
//      this.eng=eng;
//  }

    public void drive() {
        int status = eng.start();
        if (status >= 1) {
            System.out.println("Journey Started...");

        } else {
            System.out.println("Engine Trouble...");
        }
    }
}
```

Output:

DieselEngine.Constructor....)
PetrolEngine.Constructor....
parameter contructor
DieselEngine started....
Journey Started...

- In above example we make one constructor and in bean scope we give autowire as constructor and because of that it will go for byName first and variable eng found and the DieselEngine bean in injected the output observed.

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

    <bean id="eng2" class="com.codewitharrays.beans.PetrolEngine" />
    <bean id="car" class="com.codewitharrays.beans.Car" autowire="constructor" >

    </bean>
</beans>
```

Output:

PetrolEngine.Constructor....
parameter contructor
PetrolEngine Started...
Journey Started...

- In above example the first check for byName and it will not found then go for byType and the variable datatype is inject PetrolEngine because other interface not mentioned here. If present then ambiguity error occurs.

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

    <bean id="eng1" class="com.codewitharrays.beans.DieselEngine" primary="true" />

    <bean id="eng2" class="com.codewitharrays.beans.PetrolEngine" />

    <bean id="car" class="com.codewitharrays.beans.Car" autowire="constructor" >

    </bean>

</beans>
```

Output:

PetrolEngine.Constructor....
parameter contructor
DieselEngine started....
Journey Started...

- In this above example the we autowire as constructor and first go for byName and not found then it will go for byType and the multiple implements by interface. So the primary=”true” is inject because of high priority.

This xml based is outdated so we go for annotation based injection and spring boot doesn’t support for xml.

Spring Annotations

- Annotation is used to provide some metadata.

1. **@Configuration:** To represent java class as config class.
   - @Configuration is used on classes that define beans. @Configuration is an analog for an XML configuration file
   - it is configured using Java classes. A Java class annotated with @Configuration is a configuration by itself and will have methods to instantiate and configure the dependencies.

2. **@Component:** To represent java class as Spring Bean class
   - @Component is used on classes to indicate a Spring component. The @Component annotation marks the Java class as a bean or component so that the component scanning mechanism of Spring can add it into the application context.

3. **@Service:** To represent java class as Spring Bean class
   - @Service marks a Java class that performs some service, such as executing business logic, performing calculations, and calling external APIs. This annotation is a specialized form of the @Component annotation intended to be used in the service layer.

4. **@Repository:** To represent java class as Spring Bean class
    - This annotation is used on Java classes that directly access the database. The @Repository annotation works as a marker for any class that fulfils the role of repository or Data Access Object.
    - This annotation has an automatic translation feature. For example, when an exception occurs in the @Repository, there is a handler for that exception and there is no need to add a try-catch block.

5. **@Scope:** To represent scope of spring bean (default: singleton)
    - @Scope is used to define the scope of a @Component class or a @Bean definition. It can be either singleton, prototype, request, session, global Session or some custom scope.

6. **@Autowired:** Inject dependent into target
   @Autowired is used to mark a dependency which Spring is going to resolve and inject automatically. We can use this annotation with a constructor, setter, or field injection.

7. **@Bean:** To customize bean object creation.
    - @Bean is a method-level annotation and a direct analog of the XML element.
    - @Bean marks a factory method which instantiates a Spring bean.
    - Spring calls these methods when a new instance of the return type is required.

8. **@Qualifier:** To identify bean based on the given name for DI
    - @Qualifier helps fine-tune annotation-based auto wiring. There may be scenarios when we create more than one bean of the same type and want to wire only one of them with a property.
    - This can be controlled using @Qualifier annotation along with the @Autowired annotation.
    - We use @Qualifier along with @Autowired to provide the bean ID or bean name we want to use in ambiguous situations.

9. **@Primary:** To give priority for the bean for auto wiring
    - We use @Primary to give higher preference to a bean when there are multiple beans of the same type.
    - When a bean is not marked with @Qualifier, a bean marked with @Primary will be served in case on ambiguity.

What is Component Scanning ?

=> It is used to identify Spring Bean classes available in the Project.

=> It will start scanning from current package.

se package name: com. codewitharrays

AppConfig
- @Configuration
- @ComponentScan

com. codewitharrays.service ----- scanned
com. codewitharrays.util ----- scanned
com. codewitharrays.dao ----- not scanned

We can understand this by the example:

```
package com.codewitharrays.beans;

import org.springframework.stereotype.Component;

@Component
public class Chip {

    public Chip() {
        System.out.println("Chip constructor");
    }

        public static String Chips() {
            return "32-bits";
        }
}
```

```
package com.codewitharrays.beans;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class Robot {

    @Autowired
    private Chip chip;

    public Robot() {
        System.out.println("Robot constructor");
    }

    public void doWork() {
                String type =chip.Chips();
                if (type.contains("32-bits")) {
                    System.out.println("performance is low with this: "+type);
                }else {
                    System.out.println("chip is not found: ");
                }
        }
}
```

```
package com.codewitharrays.config;

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan(basePackages = "com.codewitharrays")
public class AppConfig {

}
```

```java
package com.codewitharrays.config;

import org.springframework.stereotype.Component;

@Component
public class Car {

    public Car() {
        System.out.println("Car constructor");
    }

}
```

```java
package com.codewitharrays.test;

import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import com.codewitharrays.config.AppConfig;

public class App {
    public static void main(String[] args) {
        ApplicationContext ctx= new AnnotationConfigApplicationContext(AppConfig.class);
    }


}
```

With base Packages write Output:	without base Packages output:	without component Scan
Chip constructor	Car constructor	No output
Robot constructor		
Car constructor		

--ComponentScan first scan the component from the same package only and create object for component class bean. To resolve this then we can give the basePackages = ” in.codewitharrays ” then it will check for all component present in this package and create object for that component class bean. And you can give multiple package name also. Ex: { com.tcs, com.wipro, in.codewitharrays}

Right now, the all-component object is created because of singleton scope if I want to change the scope then @scope annotation is used.

@Scope: To represent scope of spring bean (default: singleton)

- @Scope is used to define the scope of a @Component class or a @Bean definition. It can be either singleton, prototype, request, session, global Session or some custom scope.

Let’s understand by following example:

Refer above 30 page no code. Only change In car class. I change the scope of car class that I prototype and now the object is not created. Until I called.

```java
package com.codewitharrays.config;

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

@Component
@Scope("prototype")
public class Car {

    public Car() {
        System.out.println("Car constructor");
    }

}
```

Output:
Chip constructor
Robot constructor

@Bean: To customize bean object creation.

- @Bean is a method-level annotation and a direct analog of the XML element.
- @Bean marks a factory method which instantiates a Spring bean.
- Spring calls these methods when a new instance of the return type is required.

-When we mentioned a class as @component IoC create the object but if I want to make the object because I have special requirement that time the @Bean annotation we can used.
Let's understand by the following example: Refer above 30-page code additional are below

```java
package com.codewitharrays.config;

public class Engine {
            public Engine() {
                System.out.println("Engine constructor");
            }
}
```

```java
package com.codewitharrays.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan(basePackages = "com.codewitharrays")
public class AppConfig {

    @Bean
    public Engine getEngine() {
        Engine engine=new Engine();
        return engine;
    }

}
```

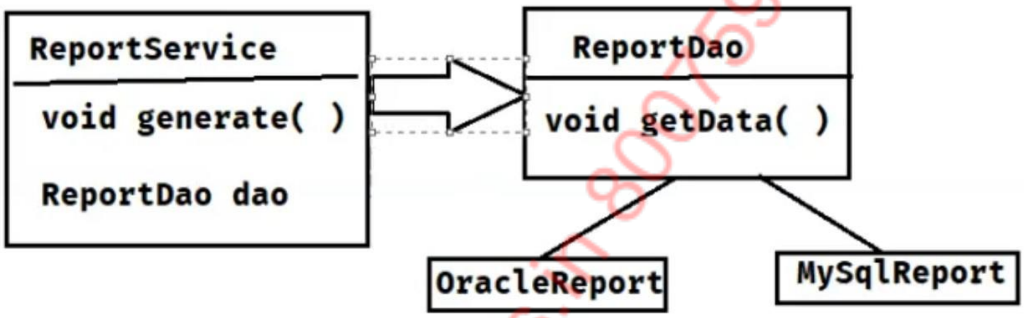
- What happened here I want to make object by me not a IoC so for that I used @Bean annotation on that getEngine() method without @Bean the object is not created you can check by compiling the code. After @bean the output is:

Output:

Chip constructor
Robot constructor
Engine constructor

Auto wiring with annotation:

--Understand some concept




```
package com.codewitharrays.bean;

public interface ReportDao {
    public void getData();
}
```

```
package com.codewitharrays.bean;

import org.springframework.stereotype.Repository;

@Repository("dao")
public class OracleReportDB implements ReportDao{

    public void getData() {
        System.out.println("Getting data from the oracle db");

    }

}
```

```
package com.codewitharrays.bean;

import org.springframework.stereotype.Repository;

@Repository("dao")
public class MySqlReportDB implements ReportDao{

    public void getData() {
        System.out.println("Getting data from the mysql db");
    }

}
```

```java
package com.codewitharrays.test;

import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import com.codewitharrays.AppConfig;
import com.codewitharrays.bean.ReportService;

public class App {
    public static void main(String[] args) {
        ApplicationContext ctxt=
                new AnnotationConfigApplicationContext(AppConfig.class);

        ReportService service=ctxt.getBean(ReportService.class);
        service.generatereport();
        //Now it will go for byType but the interface implements multiple class so
        //give me ambiguity error
    }
}
```

```java
package com.codewitharrays;

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan
public class AppConfig {

}
```

```java
package com.codewitharrays.bean;

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

@Service
public class ReportService {

        //1. Right now the autowired happened byName and it will not find so going for byType and
        //interface implement multiple class so it will gave me ambiguity error

        //1.@Autowired
        //  private ReportDao dao;


        //2. Now in oracle class i mentioned the Repository("dao") then the byName the object created
        //Getting data from the oracle db Report Generated  this output get.
        //if in mysql class i mentioned same then ambiguity error occurs.
        @Autowired
        private ReportDao dao;

        public void generatereport() {
            dao.getData();
            System.out.println("Report Generated...");
        }
}
```

-- To resolve this above code problem, we can used @Qualifier annotation

@Qualifier: To identify bean based on the given name for DI

- @Qualifier helps fine-tune annotation-based auto wiring. There may be scenarios when we create more than one bean of the same type and want to wire only one of them with a property.
- This can be controlled using @Qualifier annotation along with the @Autowired annotation.
- We use @Qualifier along with @Autowired to provide the bean ID or bean name we want to use in ambiguous situations.
- So in above code we can do small changes in Repository oracle and mysql class we give name @Repository("oracleDB") & and @Repository("MySqlDb") and with @autowire we can @Qualifier annotation. And the name of the class that I want to create the object Example in below code I create object for mysql.

```
package com.codewitharrays.bean;

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

@Service
public class ReportService {

        @Autowired
        @Qualifier("MySqlDB")
        private ReportDao dao;

        public void generatereport() {
            dao.getData();
            System.out.println("Report Generated...");
        }
}
```

```
package com.codewitharrays.bean;

import org.springframework.stereotype.Repository;

//@Repository("dao")
@Repository("MySqlDB")
public class MySqlReportDB implements ReportDao{

    public void getData() {
        System.out.println("Getting data from the mysql db");
    }
}
```

```
package com.codewitharrays.bean;

import org.springframework.stereotype.Repository;

//@Repository("dao")
@Repository("OracleDB")
public class OracleReportDB implements ReportDao{

    public void getData() {
        System.out.println("Getting data from the oracle db");

    }
```

Output:

Getting data from the mysql db

Report Generated...

- We can create object byName if its variable name not match. With @Qualifier annotation.
- But you don't want go with byName and you want go with byType then you can used @primary annotation

```
package com.codewitharrays.bean;

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

//@Repository("dao")
@Repository("MySqlDB")
@Primary
public class MySqlReportDB implements ReportDao{

    public void getData() {
        System.out.println("Getting data from the mysql db");
    }
}
```

- Right now the auto wiring is done by field. In below code we can do by using Setter.

```
package com.codewitharrays.bean;

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

@Service
public class ReportService {

        private ReportDao dao;

        @Autowired
        public void setDao(ReportDao dao) {
            System.out.println("Setter method called");
            this.dao=dao;
        }


        public void generatereport() {
            dao.getData();
            System.out.println("Report Generated...");
        }
    }
```

- when we want go for constructor injection autowired

```
package com.codewitharrays.bean;

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

@Service
public class ReportService {

        private ReportDao dao;

        //4. when we want go for constructor injection autowired
        @Autowired
        public ReportService(ReportDao dao) {
            System.out.println("report service constructor");
            this.dao = dao;
        }

        public void generatereport() {
            dao.getData();
            System.out.println("Report Generated...");
        }
}
```

- Okay now if I remove the @Autowired from the constructor it will work or not it will work because the @service is the spring bean class and for that IoC will create and object and to create object the constructor is needed and the parameterized constructor is available so here @Autowired is optional.
- But in case I have 2 constructors. One is default constructor and second is parameterized constructor then the @Autowired is given compulsory.

```
package com.codewitharrays.bean;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Service;

@Service
public class ReportService {

        private ReportDao dao;

        public ReportService() {
            System.out.println("report service : 0  constructor");
        }
//If two constructor is present then the @autowired is given compulsory else its optional
        @Autowired
        public ReportService(ReportDao dao) {
            System.out.println("report service: Param constructor");
            this.dao = dao;
        }

        public void generatereport() {
            dao.getData();
            System.out.println("Report Generated...");
        }
}
```

Spring Bean Life Cycle

=> The class which is managed by IOC is called as Spring Bean.

=> We can perform some operations when bean object created and before bean object is removed.

- post construct
- pre destory

=> To achieve above requirement we can use bean life cycle methods

=> Bean Life cycle methods we can execute in 3 ways

1) Declarative approach (xml)
2) Programmatic approach
3) Annotation approach

1. Declarative Approach

```
public class UserDao {
	public void init() {
		System.out.println("getting db connection...");
	}
	public void getData() {
		System.out.println("getting data from db....");
	}
	public void destroy() {
		System.out.println("closing db connection...");
	}
}
<bean id="dao"
		class="in.codewitharrays.UserDao"
		init-method="init"
		destroy-method="destroy"
		/>
```

2. Programmatic Approach

=> We need to implement 2 interfaces here

1) InitializingBean ----> afterPropertiesSet ( )
2) DisposableBean ---> destory ( )

```
public class UserDao implements InitializingBean, DisposableBean{
	public void afterPropertiesSet() throws Exception {
		System.out.println("init method....");
	}
	public void destroy() throws Exception {
		System.out.println("destroy method....");
	}
```

```
	public void getData() {
		System.out.println("getting data from db....");
	}
}
```

3. Annotation Approach

=> We have below 2 annotations to achieve life cycle methods execution

1) @PostConstruct

2) @PreDestroy

@DependsOn: It is used to specify one class is indirectly dependent on another class.

```
@Component
public class UserDao {
	@PostConstruct
	public void init() throws Exception {
		System.out.println("init method....");
	}

	@PreDestroy
	public void destroy() throws Exception {
		System.out.println("destroy method....");
	}
	public void getData() {
		System.out.println("getting data from db....");
	}
}

@Component("userdao")
public class UserDao implements InitializingBean {
	@Override
	public void afterPropertiesSet() throws Exception {
		System.out.println("get data from db...");
		System.out.println("store data into redis...");
	}
}
```

Project Lombok

-> Project Lombok is a third party library
-> It is used to avoid boiler-plate-code in project
-> Project Lombok will generate below things for our classes
1) setter methods
2) getter methods
3) 0-param constructor
4) param-constructor
5) toString ( ) method
6) equals ( ) method
7) hashCode ( )

Project Lombok Setup

Step-1) Add Lombok Dependency in pom.xml file

```
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.18.26</version>
</dependency>
```

Step-2) Install Lombok jar in our IDE (Eclipse / STS / IntelliJ)

-> Goto lombok jar location
-> execute below command
    $ java -jar <lombok-jar-file-name>
-> Specify IDE location (eclipse.exe / STS.exe)
-> Click on install
-> Close installer
-> Re-Start IDE

Note: Step-2 is required only first time.

Project Lombok Annotations

=> Project Lombok provided annotations to generate boiler plate code.

1) @Setter : To generate setter methods for variables
2) @Getter : To generate getter methods for variables
3) @ToString : To generate toString ( ) method
4) @NoArgsConstructor : To generate 0-param constructor

5) @AllArgsConstructor : To generate param-constructor
6) @EqualsAndHashCode : To generate equals () & hashCode ( ) methods
7) @Data =
@Setter + @Getter + @NoArgsConstructor + @ToString + @EqualsAndHashCode

```java
@Setter
@Getter
@ToString
@AllArgsConstructor
@NoArgsConstructor
public class Person {

    private Integer personId;
    private String personName;
    private String gender;
    private Long phno;
    private Date dob;
}
```

```java
import lombok.Data;

@Data
public class User {

    private Integer userid;
    private String username;
    private String userEmail;
    private String pwd;
    private String gender;
    private Long phno;

}

public class Test {

    Run | Debug
    public static void main(String[] args) {
        Person p = new Person();
        p.setPersonId(personId:101);
        p.setGender(gender:"male");
        p.setPersonName(personName:"codewitharrays");

        System.out.println(p);
    }
}
```

Spring Boot

-> Spring Boot is one approach to develop Spring Based applications with less configurations.

-> Using Spring Boot we can develop below types of applications

1) Standalone app
2) Web
3) Distributed (webservices)

-> We have below advantages with Spring Boot

1) Starter Pom (simplifies maven/Gradle build configuration)
   -> web-starter
   -> datajpa-starter
   -> security-starter
   -> mail-starter

2) Auto Configuration (boot will identify required configs for our app)
   -> Creating DB connection pool
   -> deploy web app in embedded server
   -> Start IOC container
   -> Component Scanning

First Spring Boot App: (We can learn this advatages by using simple code)

1. **Click on file -> New -> Spring Starter Project -> Artifact name -> maven -> jar -> version**
2. **Add dependency Spring Web, Spring Security etc. Finished.**

```
package com.codewitharrays;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
@RestController
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }

    @GetMapping("/")
    public String Welcome() {
        return "Welcome to codewitharrays";
    }
}
```

Now we write some code that. And instantly we created the web MVC project. By using localhost:8080 we can check on the chrome.

Output: Welcome to codewitharrays

But now directly access I want to add some security so for that spring security dependency added.

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

We cannot access directly we have to provide the username and password. In application. Properties we must set this configuration
spring.security.user.name=codewitharrays
spring.security.user.password=codewitharrays

now I created one class Report Service and make one constructor and annotated at @Service and now the IoC make an object for that class. And here the @componentScan is also not required that is the magic of Spring boot is auto configure lot of things.

```
package com.codewitharrays.beans;

import org.springframework.stereotype.Service;

@Service
public class ReportService {

    public ReportService() {
        System.out.println("Report service construtor");
    }
}
```

3) Actuators (To monitor and manage our application)

-> how many beans loaded ?
-> How many url-patterns mapped
-> What configuration properties loaded ?
-> What is health of app?
-> Heap Dump
-> Thread Dump

4) Embedded Servers: It provides server to run our boot application

-> Apache Tomcat (default)
-> Jetty
-> netty

- Spring boot default running server on 8080 port no. But if you want to change the port no then in application properties we have to configure **server.port=9090.**

Creating Spring Boot Application

1) Initializer website (start.spring.io)
2) IDE

Note: IDE will internally connect with start.spring.io website to create Spring Boot application (Spring Starter Project)

Spring Boot = Spring - xml configurations + Auto Config + Embedded Servers + Actuators

The @SpringBootApplication annotation is equal to below 3 annotations

1) @SpringBootConfiguration ----> @Configuration
2) @EnableAutoConfiguration
3) @ComponentScan

How IOC starting in Spring Boot ?

=> SpringApplication.run ( ) method contains logic to start IOC container

boot-starter :: AnnotationConfigApplicationContext
web-starter :: AnnotationConfigServletWebServerApplicationContext
starter-webflux :: AnnotationConfigReactiveWebServerApplicationContext

What is Banner in Spring Boot?

-> Spring Logo which is printing on console is called as banner

-> We have below 3 modes for banner

1) console (default) --->> prints on console
2) log ---->> prints on log file
3) off --->> don't print banner

-> We can set banner mode in application.properties file

spring.main.banner-mode=off

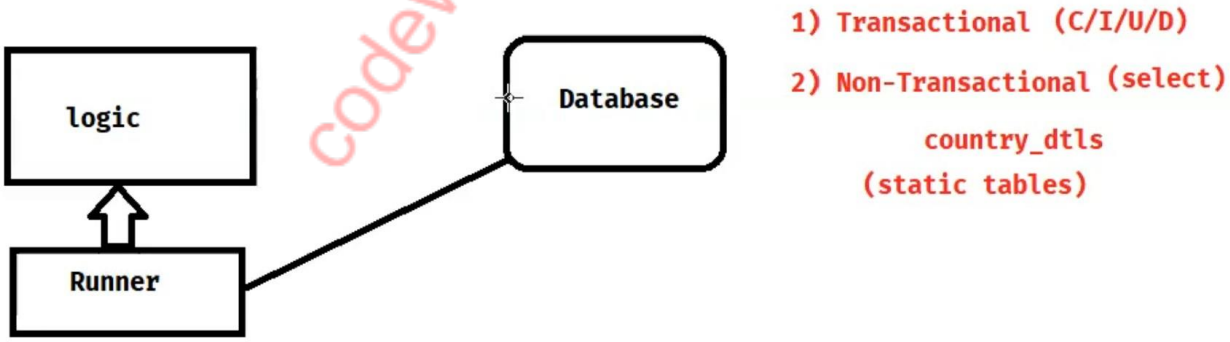
-> We can customize banner text in spring boot by creating banner.txt file under src/main/resources folder.

ASCII Text Gen URL: https://patorjk.com/software/taag/#p=display&f=Graffiti&t=SBI%20API

What is Runner in spring boot?

- Spring boot provided runner to execute logic only once when application starts.
- We have 2 types of Runners in spring boot
  a) ApplicationRunner --> run ()
  b) CommandLineRunner --> run ()

Note: Both are functional interfaces. Only one abstract method



The one is transactional data who can be modified and second one is non-transactional data who's static and cannot be modified. If in my application if I called every time static data even data is same there is no use of that. To resolve this issue we can use runner. Spring boot provided runner to execute logic only once when application starts.

```java
package com.codewitharrays.runners;

import org.springframework.boot.ApplicationArguments;

@Component
public class MyAppRunner implements ApplicationRunner{

    @Override
    public void run(ApplicationArguments args) throws Exception {
        System.out.println("Application runner run() method: ");
    }
}
```

```java
package com.codewitharrays.runners;

import org.springframework.boot.CommandLineRunner;

public class MyCmdRunner implements CommandLineRunner{

    @Override
    public void run(String... args) throws Exception {
        System.out.println("Command line runner run() method:");

    }

}
```


Use Cases in which scenario we can used Runner

1. Load Static tables data when application starts.
2. Delete data from staging tables (temporary tables)
3. Send notification regarding application startup

What is springApplication.run() method?

1. **Load Initializers**:

- Spring Boot loads any ApplicationContextInitializer classes that have been registered. These initializers are used to customize the ApplicationContext before it is refreshed.

2. **Start Listeners**:

- Any ApplicationListener beans are registered and started. These listeners are used to listen for various events during the lifecycle of the Spring Boot application (such as context refresh events, start events, etc.).

3. **Print Banner**:

- Before the application starts, Spring Boot prints a banner to the console (the banner can be customized or disabled).

4. **Create IOC Container (ApplicationContext)**:

- It creates and refreshes the ApplicationContext (which is the IoC container in Spring). The ApplicationContext is responsible for managing all the beans and dependencies of the application.

5. **Call Runners**:

- Spring Boot calls any CommandLineRunner or ApplicationRunner beans that have been defined. These beans are used to run specific logic right after the application context is loaded.

6. **Return IoC Object (ApplicationContext)**:

- Finally, the SpringApplication.run() method returns the ApplicationContext object, which represents the IoC container. This can be used to interact with the application context programmatically.

Spring Data JPA

Spring Data JPA is one module in spring framework
It is used to develop Persistence Layer (DB Logic). We can develop Persistence layer by using this 5

1. Java JDBC
2. Spring JDBC
3. Hibernate Framework
4. Spring ORM
5. **Spring Data JPA**

- Spring Data JPA is simplying CRUD operations implementation in Project. Lets understand by example: I have multiple classes in my project and for each class I have the 4 method mandatory to implement that insert record, update record, delete record, select record.
- So if I have 5000 tables then the 5000 classes in my project and for that this 4 method so the line of code is around 20000. And that is why Spring Data JPA come in picture by using the data JPA i don't have write single line of code everything take care by Data JPA.

Ex:
USER_MASTER -> UserMasterDao -> 4 methods (insert/update/delete/select)
ROLE_MASTER -> RoleMasterDao -> 4 methods (insert/update/delete/select)
PRODUCT_DTLS -> ProductDtlsDao -> 4 methods (insert/update/delete/select)
PAYMENT_DTLS -> PaymentDtlsDao -> 4 methods (insert/update/delete/select)
Like wise I have 5000 class
5000 *4 = 20000 method
But if I used Data JPA 50000 * 0= 0 methods to write.

- To avoid boiler plate code in DAO classes we can use spring Data JPA.
- Spring Data JPA will use Hibernate framework internally.
- Hibernate framework internally uses JDBC

Spring Data JPA Repositories

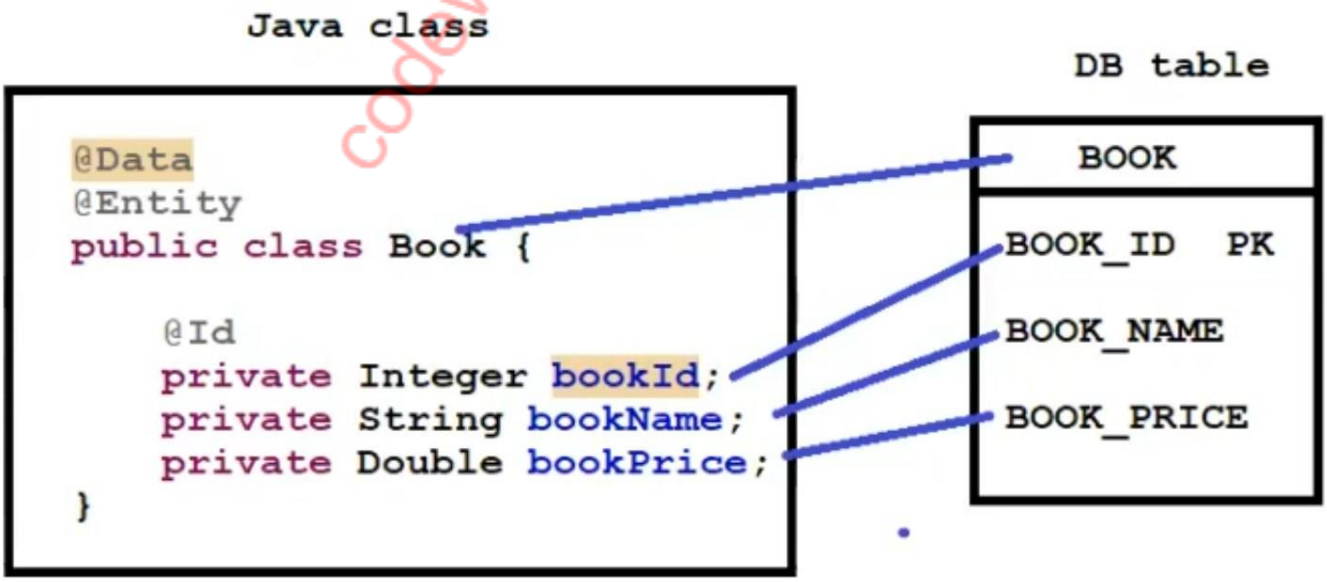
- To simplify Persistence Layer Development Data JPA provided Repository interfaces.
    1. **CrudRepository** (CRUD methods)
    2. **JpaRepository** (CRUD methods + Sorting + pagination + Query by example)

Note: To perform DB operations, we need to create an interface by extending properties from JPA repository interface.

Developing First Data JPA Application

1) Create Spring boot application with dependencies
    a) Data-jpa-starter
    b) MySQL-driver
    c) Project-Lombok
2) Configure Data source properties in application.properties file
3) Create Entity class (Java class to DB table mapping)
4) Create Repository interface by extending JPA Repository.
5) Test the application by calling Repository interface methods.

```
#This datasource is used to connection with database
spring.datasource.username=root
spring.datasource.password=cdac
spring.datasource.url=jdbc:mysql://localhost:3306/sbms
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```



```java
package com.codewitharrays.entity;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import lombok.Data;

@Data
@Entity  //To represent the class as entity class we can use entity annotation
public class Book {

    //If we want to change the table name then we can use the @Table annotation.
    //If we want to change the column name then we can use the @Column annotation
    //In database the primary key require to make the pk we can use the @Id annotation.
        @Id
        private Integer bookId;
        private String bookName;
        private Double bookPrice;
}
```

```java
package com.codewitharrays.repo;

import org.springframework.data.repository.CrudRepository;

import com.codewitharrays.entity.Book;

//In Repository the class extends by CrudRepository or JpaRepository
//the repository class is the interface class
//The two parameter can take JpaRepository class first is the entity class name and second is the primary key datatype
//For example in this the Book is the entity class name and the Integer is primary key datatype
public interface BookRepository extends CrudRepository<Book, Integer>{

}
```

```java
package com.codewitharrays;

import java.util.Optional;

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

import com.codewitharrays.entity.Book;
import com.codewitharrays.repo.BookRepository;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext ctx=
        SpringApplication.run(Application.class, args);

        BookRepository repo =ctx.getBean(BookRepository.class);
        //Here we can call the some methods from the CrudRepository
        Book b=new Book();
        b.setBookId(101);
        b.setBookName("Springboot");
        b.setBookPrice(1000.00);
        repo.save(b);


        Optional <Book> findById=repo.findById(101);
        System.out.println(findById);
    }
}
```

Methods Present in the CrudRepository

1. save (E): Saves a given entity. (To use insert and update)

- How save method execute the insert and update both the operation by example

```
package com.codewitharrays;

import java.util.Optional;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext ctx=
        SpringApplication.run(Application.class, args);

        BookRepository repo =ctx.getBean(BookRepository.class);
        //Here we can call the some methods from the CrudRepository
        Book b=new Book();
        b.setBookId(102);
        b.setBookName("Core java");
        b.setBookPrice(2000.00);
        repo.save(b);
    }
}
```

```
2024-10-23T11:23:58.071+05:30  INFO 15928 --- [10-Spring-Data_JPA-APP] [           main] com.codewitharrays.Application
Hibernate: select b1_0.book_id,b1_0.book_name,b1_0.book_price from book b1_0 where b1_0.book_id=?
Hibernate: insert into book (book_name,book_price,book_id) values (?,?,?)
2024-10-23T11:23:58.178+05:30  INFO 15928 --- [10-Spring-Data_JPA-APP] [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean
```

- Here we create the new book with id 102 and when code is compiling the hibernate generate the query first with select query check the record is available or not if not then second, he generates the insert query and add the record in database.

```
package com.codewitharrays;

import java.util.Optional;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext ctx=
        SpringApplication.run(Application.class, args);

        BookRepository repo =ctx.getBean(BookRepository.class);
        //Here we can call the some methods from the CrudRepository
        Book b=new Book();
        b.setBookId(102);
        b.setBookName("Core java");
        b.setBookPrice(4000.00);
        repo.save(b);
    }
}
```

```
2024-10-23T11:25:37.469+05:30  INFO 15160 --- [10-Spring-Data_JPA-APP] [           main] com.codewitharrays.Application
Hibernate: select b1_0.book_id,b1_0.book_name,b1_0.book_price from book b1_0 where b1_0.book_id=?
Hibernate: update book set book_name=?,book_price=? where book_id=?
2024-10-23T11:25:37.580+05:30  INFO 15160 --- [10-Spring-Data_JPA-APP] [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean
```

- Here we again create the book with same id 102 and when code is compiling the hibernate generate the query first with select query check the record and the record is found so then the hibernate generate the update query and add the record.
- This is how the save method execute both update and insert record.

2. saveAll (Iterable): Saves all given entities

```
package com.codewitharrays;

import java.util.ArrayList;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext ctx=
        SpringApplication.run(Application.class, args);

        BookRepository repo =ctx.getBean(BookRepository.class);
        //Here we can call the some methods from the CrudRepository
        Book b1=new Book();
        b1.setBookId(103);
        b1.setBookName("React js");
        b1.setBookPrice(5000.00);

        Book b2=new Book();
        b2.setBookId(104);
        b2.setBookName("HTML");
        b2.setBookPrice(6000.00);

        List<Book> books=new ArrayList<>();
        books.add(b1);
        books.add(b2);
        repo.saveAll(books);
    }
}
```


- We can make b1 record and b2 record and these two records add at a time for that we use saveAll method.
- Also we can use repo.saveAll(Arrays.asList(b1,b2) if we don't want use this List collection.

3. findById (Id): Retrieves an entity by its id. To retrieve record based on given PK

```
package com.codewitharrays;

import java.util.ArrayList;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext ctx=
        SpringApplication.run(Application.class, args);

        BookRepository repo =ctx.getBean(BookRepository.class);
        //Here we can call the some methods from the CrudRepository
//Optional is introduced in java 8 to avoid the null pointer exception and check multiple nulls
        Optional<Book>  findById =repo.findById(102);
        if (findById.isPresent()) {
            System.out.println(findById.get());
        }
    }
}
```

```
2024-10-25T12:10:56.200+05:30  INFO 19652 --- [10-Spring-Data_JPA-APP] [           main] com.codewitharrays.Application
Hibernate: select b1_0.book_id,b1_0.book_name,b1_0.book_price from book b1_0 where b1_0.book_id=?
Book(bookId=102, bookName=Core java, bookPrice=4000.0)
2024-10-25T12:10:56.247+05:30  INFO 19652 --- [10-Spring-Data_JPA-APP] [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean
```

5. findAll (): Returns all instances of the type. To retrieve all records from table

```
import com.codewitharrays.entity.Book;
import com.codewitharrays.repo.BookRepository;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext ctx=
        SpringApplication.run(Application.class, args);

        BookRepository repo =ctx.getBean(BookRepository.class);
        //Here we can call the some methods from the CrudRepository
        Iterable<Book> findAll = repo.findAll();
        for (Book b : findAll) {
            System.out.println(b);
        }
    }
}
```

```
2024-10-25T12:32:39.969+05:30  INFO 11084 --- [10-Spring-Data_JPA-APP] [           main] com.codewitharrays.Application          :
Hibernate: select b1_0.book_id,b1_0.book_name,b1_0.book_price from book b1_0
Book(bookId=101, bookName=Springboot, bookPrice=1000.0)
Book(bookId=102, bookName=Core java, bookPrice=4000.0)
Book(bookId=103, bookName=React js, bookPrice=5000.0)
Book(bookId=104, bookName=HTML, bookPrice=6000.0)
2024-10-25T12:32:40.091+05:30  INFO 11084 --- [10-Spring-Data_JPA-APP] [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean :
```

6. findAllById (Iterable Ids): Returns all instances of the type {@code T} with the given IDs. To retrieve record based on given PKs (multiple pk)

```
import com.codewitharrays.entity.Book;
import com.codewitharrays.repo.BookRepository;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext ctx=
        SpringApplication.run(Application.class, args);

        BookRepository repo =ctx.getBean(BookRepository.class);
        //Here we can call the some methods from the CrudRepository
        Iterable<Book> findAllById = repo.findAllById(Arrays.asList(102,103));

        for (Book b : findAllById) {
            System.out.println(b);
        }
    }
}
```

```
2024-10-25T12:29:04.699+05:30  INFO 10880 --- [10-Spring-Data_JPA-APP] [           main] com.codewitharrays.Application          :
Hibernate: select b1_0.book_id,b1_0.book_name,b1_0.book_price from book b1_0 where b1_0.book_id in (?,?)
Book(bookId=102, bookName=Core java, bookPrice=4000.0)
Book(bookId=103, bookName=React js, bookPrice=5000.0)
2024-10-25T12:29:04.827+05:30  INFO 10880 --- [10-Spring-Data_JPA-APP] [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean :
```

4. boolean existsById (): Returns whether an entity with the given id exists. To check Presence of record

```
package com.codewitharrays;

import java.util.ArrayList;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext ctx=
        SpringApplication.run(Application.class, args);

        BookRepository repo =ctx.getBean(BookRepository.class);
        //Here we can call the some methods from the CrudRepository
        boolean status= repo.existsById(102);
        System.out.println("Record present:: "+status);
    }
}
```

```
2024-10-25T11:50:55.502+05:30  INFO 21412 --- [10-Spring-Data_JPA-APP] [           main] com.codewitharrays.Application          :
Hibernate: select count(*) from book b1_0 where b1_0.book_id=?
Record present:: true
2024-10-25T11:50:55.778+05:30  INFO 21412 --- [10-Spring-Data_JPA-APP] [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean :
```

- If the record is present then return the true if not available then return false.

7. count (): Returns the number of entities available. To get records count in table

```
package com.codewitharrays;

import java.util.ArrayList;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext ctx=
        SpringApplication.run(Application.class, args);

        BookRepository repo =ctx.getBean(BookRepository.class);
        //Here we can call the some methods from the CrudRepository
//      It will return the count from record
        System.out.println("Records:: "+repo.count());
    }
}
```

```
2024-10-25T11:58:53.854+05:30  INFO 22412 --- [10-Spring-Data_JPA-APP] [           main] com.codewitharrays.Application
Hibernate: select count(*) from book b1_0
Records:: 4
2024-10-25T11:58:54.086+05:30  INFO 22412 --- [10-Spring-Data_JPA-APP] [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean
```

8. deleteById (Id): Deletes the entity with the given id. To delete record based on given PK (primary key)

```
import com.codewitharrays.entity.Book;
import com.codewitharrays.repo.BookRepository;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext ctx=
        SpringApplication.run(Application.class, args);

        BookRepository repo =ctx.getBean(BookRepository.class);
        //Here we can call the some methods from the CrudRepository
    if (repo.existsById(104)) {
        repo.deleteById(104);
    }else
        System.out.println("Record not found");
    }
}
```

```
2024-10-25T12:59:08.726+05:30  INFO 2272 --- [10-Spring-Data_JPA-APP] [           main] com.codewitharrays.Application          :
Hibernate: select count(*) from book b1_0 where b1_0.book_id=?
Record not found
2024-10-25T12:59:08.964+05:30  INFO 2272 --- [10-Spring-Data_JPA-APP] [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean :
```

10.deleteAllById (Iterable Ids): Deletes all instances of the type {@code T} with the given IDs. Delete records based on given PKs

11. delete (E): Deletes a given entity. Delete record based on given Entity Object

12. deleteAll (Iterable entities): Deletes a given entity. Delete records based on given entity objects

13. deleteAll (): Deletes all entities managed by the repository. Delete all the record from table.

---

1) **Find By Method are available in JPA**
2) **Custom Queries**

```
import com.codewitharrays.entity.Book;
import com.codewitharrays.repo.BookRepository;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext ctx=
        SpringApplication.run(Application.class, args);

        BookRepository repo =ctx.getBean(BookRepository.class);
        //Here we can call the some methods from the CrudRepository
        List<Book> findByBookPrice = repo.findByBookPriceGreaterThan(2000);
        for (Book b : findByBookPrice) {
            System.out.println(b);
        }
    }
}
public interface BookRepository extends CrudRepository<Book, Integer>{

    List<Book> findByBookPriceGreaterThan(double price);
}
```

```
2024-10-25T15:12:56.197+05:30  INFO 20376 --- [10-Spring-Data_JPA-APP] [           main] com.codewitharrays.Application          :
Hibernate: select b1_0.book_id,b1_0.book_name,b1_0.book_price from book b1_0 where b1_0.book_price>?
Book(bookId=102, bookName=Core java, bookPrice=4000.0)
Book(bookId=103, bookName=React js, bookPrice=5000.0)
2024-10-25T15:12:56.357+05:30  INFO 20376 --- [10-Spring-Data_JPA-APP] [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean :
```

- findBy methods are used to perform select operations
- Using non-primary key columns, we can select records
- In findBy methods method name is very important because based on method name JPA will construct the query for execution.
- Find By methods should represent entity class variables

- Here in this example we find the book price less than

```
import com.codewitharrays.entity.Book;
import com.codewitharrays.repo.BookRepository;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext ctx=
        SpringApplication.run(Application.class, args);

        BookRepository repo =ctx.getBean(BookRepository.class);
        //Here we can call the some methods from the CrudRepository

        List<Book> lessThan = repo.findByBookPriceLessThan(5000);
        for (Book b : lessThan) {
            System.out.println(b);
        }
    }
}
```

```
public interface BookRepository extends CrudRepository<Book, Integer>{

    List<Book> findByBookPriceGreaterThan(double price);

    List<Book> findByBookPriceLessThan(double price);
}
```

```
2024-10-25T15:29:34.473+05:30  INFO 2188 --- [10-Spring-Data_JPA-APP] [           main] com.codewitharrays.Application
Hibernate: select b1_0.book_id,b1_0.book_name,b1_0.book_price from book b1_0 where b1_0.book_price<?
Book(bookId=101, bookName=Springboot, bookPrice=1000.0)
Book(bookId=102, bookName=Core java, bookPrice=4000.0)
2024-10-25T15:29:34.622+05:30  INFO 2188 --- [10-Spring-Data_JPA-APP] [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean
```

- Here in these examples, we find book by name

```
import com.codewitharrays.entity.Book;
import com.codewitharrays.repo.BookRepository;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext ctx=
        SpringApplication.run(Application.class, args);

        BookRepository repo =ctx.getBean(BookRepository.class);
        //Here we can call the some methods from the CrudRepository

        List<Book> bookName = repo.findByBookName("Core java");
        for (Book b : bookName) {
            System.out.println(b);
        }
    }
}
```

```
public interface BookRepository extends CrudRepository<Book, Integer>{

    List<Book> findByBookPriceGreaterThan(double price);

    List<Book> findByBookPriceLessThan(double price);
    List<Book> findByBookName(String bookName);
}
```

```
2024-10-25T15:35:04.045+05:30  INFO 15140 --- [10-Spring-Data_JPA-APP] [           main] com.codewitharrays.Application          :
Hibernate: select b1_0.book_id,b1_0.book_name,b1_0.book_price from book b1_0 where b1_0.book_name=?
Book(bookId=102, bookName=Core java, bookPrice=4000.0)
2024-10-25T15:35:04.211+05:30  INFO 15140 --- [10-Spring-Data_JPA-APP] [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean :
```

Custom Queries

- If we want to execute our query in JPA Repo then we can go for Custom Queries. To Represent custom query we are using @Query annotation.
- Custom Queries we can write in 2 ways
- 1. HQL Queries 2. Native SQL Queries

Following example is native sql query

```
import com.codewitharrays.entity.Book;
import com.codewitharrays.repo.BookRepository;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext ctx=
        SpringApplication.run(Application.class, args);

        BookRepository repo =ctx.getBean(BookRepository.class);
        //Here we can call the some methods from the CrudRepository
        List<Book> allBooks = repo.getAllBooks();
        for (Book b : allBooks) {
            System.out.println(b);
        }
    }
}
```

```
public interface BookRepository extends CrudRepository<Book, Integer>{

    List<Book> findByBookPriceGreaterThan(double price);

    List<Book> findByBookPriceLessThan(double price);
    List<Book> findByBookName(String bookName);

    @Query(value = "Select * from book", nativeQuery = true)
    List<Book> getAllBooks();
}
```

```
2024-10-25T15:56:35.381+05:30  INFO 21368 --- [10-Spring-Data_JPA-APP] [           main] com.codewitharrays.Application          :
Hibernate: Select * from book
Book(bookId=101, bookName=Springboot, bookPrice=1000.0)
Book(bookId=102, bookName=Core java, bookPrice=4000.0)
Book(bookId=103, bookName=React js, bookPrice=5000.0)
2024-10-25T15:56:35.485+05:30  INFO 21368 --- [10-Spring-Data_JPA-APP] [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean :
```

Difference between HQL and SQL Native Queries

#1
HQL queries are DB independent queries
SQL queries are DB dependent queries
#2
In HQL query, we will use entity class name & variables
In SQL query, we will use table name & column names
#3
HQL query can't execute in DB directly (dialect class execute and then convert into sql query).
SQL query can execute in DB directly.
#4
HQL: Easy maintenance to convert one database to another database in future
SQL: you want fast performance then go for SQL query conversion not required
Note: Every HQL query should be converted to SQL query before execution that conversion will done by Dialect class

Every Database will have its own dialect class.

Ex: oracleDialect , MySqlDialect , DB2Dialect , PostgresDialect etc.

Note: Dialect class will be loaded along with DB driver class.

Example of HQL Query:

```
import java.util.ArrayList;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext ctx=
        SpringApplication.run(Application.class, args);

        BookRepository repo =ctx.getBean(BookRepository.class);
        //Here we can call the some methods from the CrudRepository
        List<Book> book = repo.getBooks();
        for (Book b : book) {
            System.out.println(b);
        }
    }
}
```

```
public interface BookRepository extends CrudRepository<Book, Integer>{

    @Query(value = "Select * from book", nativeQuery = true)
    List<Book> getAllBooks();       //Native Query

    @Query("SELECT b FROM Book b")  // HQL query
    List<Book> getBooks();
}
```

```
2024-10-25T19:04:23.510+05:30  INFO 9220 --- [10-Spring-Data_JPA-APP] [           main]
Hibernate: select b1_0.book_id,b1_0.book_name,b1_0.book_price from book b1_0
Book(bookId=101, bookName=Springboot, bookPrice=1000.0)
Book(bookId=102, bookName=Core java, bookPrice=4000.0)
Book(bookId=103, bookName=React js, bookPrice=5000.0)
2024-10-25T19:04:23.614+05:30  INFO 9220 --- [10-Spring-Data_JPA-APP] [ionShutdownHook]
```

Spring boot shortcut key for Java

1. Ctrl + Shift + R : To find all resource files, including the config XML files from the workspace

2. Ctrl + Shift + T : To find class in an application or from inside a jar

3. Ctrl + W : To close the current file

4. Ctrl + Shirt + W : To close all the open files

5. Ctrl + Shift + O : To organise the missing imports

6. Ctrl + Shift + F : For auto-formatting

7. Ctrl + / : To add/remove single line comment for selected line

8. Ctrl + Shift + Enter : To add blank line before the current line

9. Alt + Shift + S, R : To generate Setters & Getters

10. Alt + Shift + S, S : To generate toString() method

11. Alt + Shift + S, o : To add constructor with fields

12. Ctrl + 1 + Enter : To store method return value to variable

13. Ctrl + D : To delete current line

14. Ctrl + O : Display all methods of current class, Pressing Ctrl + O again will display all the inherited methods

15. Ctrl + Shift + P : Go to matching bracket

16. Alt + Shift + R : To rename

17. Alt + Shift + T : To open the context-sensitive refactoring menu

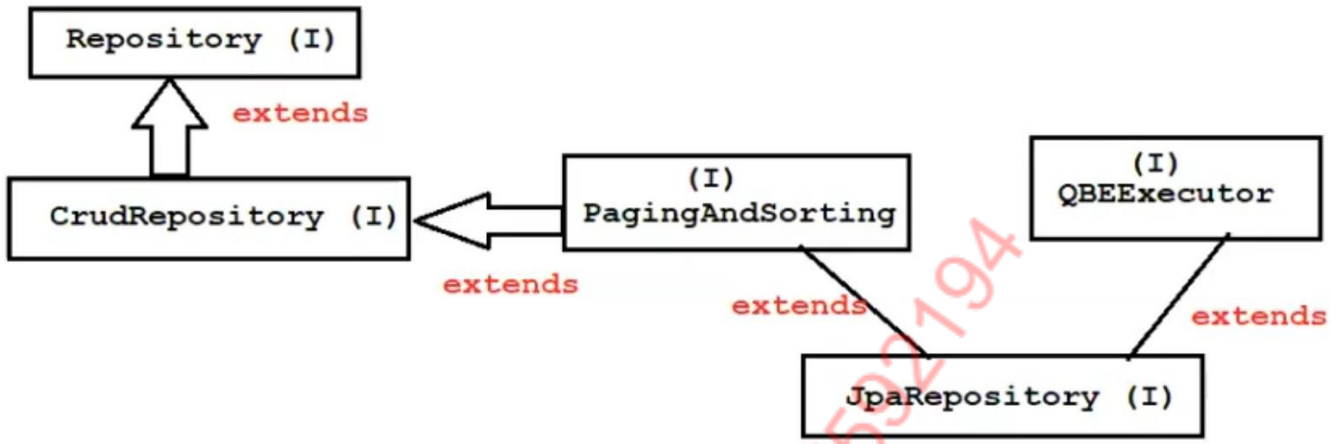
18. Alt + Shift + Z : To surround a block with try and catch

19. Ctrl + Shift + / : To comment and uncomment lines with a block comment

20. Ctrl + Q : To go to the last edited position

JpaRepository

- It is a predefined interface available in data jpa.
- Using JpaRepository also we can perform CRUD Operations
- We can perform Crud Operations + Sorting + Pagination + QBE



Understand the JPA Repository By coding Example:

```
@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "emp_tbl")
public class Employee {

        @Id
        private Integer empId;
        private String empName;
        private String gender;
        private Double salary;
        private String deptName;
}
```

```
import org.springframework.boot.SpringApplication;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext context = SpringApplication.run(Application.class, args);
        EmployeeRepository repo = context.getBean(EmployeeRepository.class);

        Employee e1=new Employee(1, "Ashok", "Male", 5000.00, "Backend");
        repo.save(e1);

    }
}
```

```
import org.springframework.data.jpa.repository.JpaRepository;

import com.codewitharrays.entity.Employee;

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}
```

- First record is inserted by e1 and we use save() method present in JpaRepository.
- All methods from crud Repository and also additional methods are available in the JPA Reposiory.
- Now we can add multiple employee records by using saveAll() methods.

```
package com.codewitharrays;

import java.util.Arrays;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext context = SpringApplication.run(Application.class, args);
        EmployeeRepository repo = context.getBean(EmployeeRepository.class);

        Employee e2=new Employee(2, "Suraj", "Male", 7000.00, "Frontend");
        Employee e3=new Employee(3, "Shubham", "Male", 8000.00, "Database");
        Employee e4=new Employee(4, "Vikas", "Male", 11000.00, "Production");
        Employee e5=new Employee(5, "Niraj", "Male", 15000.00, "Fullstack");
        Employee e6=new Employee(6, "Saurabh", "Male", 9000.00, "HR");

        repo.saveAll(Arrays.asList(e2,e3,e4,e5,e6));
    }
}
```

- Now we can find the record by using findAll() method.

```
@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext context = SpringApplication.run(Application.class, args);
        EmployeeRepository repo = context.getBean(EmployeeRepository.class);

        List<Employee> findAll = repo.findAll();
        for (Employee emp : findAll) {
            System.out.println(emp);
        }
    }
}
```

```
Hibernate: select e1_0.emp_id,e1_0.dept_name,e1_0.emp_name,e1_0.gender,e1_0.salary from emp_tbl e1_0
Employee(empId=1, empName=Ashok, gender=Male, salary=5000.0, deptName=Backend)
Employee(empId=2, empName=Suraj, gender=Male, salary=7000.0, deptName=Frontend)
Employee(empId=3, empName=Shubham, gender=Male, salary=8000.0, deptName=Database)
Employee(empId=4, empName=Vikas, gender=Male, salary=11000.0, deptName=Production)
Employee(empId=5, empName=Niraj, gender=Male, salary=15000.0, deptName=Fullstack)
Employee(empId=6, empName=Saurabh, gender=Male, salary=9000.0, deptName=HR)
2024-10-26T12:41:15.102+05:30  INFO 18872 --- [11-DataJPA_Repo-App] [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean :
```

- But if we observed carefully the record I get directly the way present in the database by I want to sort the record byName , bySalary etc. then the JpaRepository has PagingAndSorting interface class methods extending and we can use those methods.
- Iterable<T> findAll(Sort sort); For sorting …
- Page<T> findAll(Pageable pageable); for pagination…

- Sort.by(“ “) these can be used. Lets understand by coding example how to sort by ascending and descending order.

1. Ascending Order

```
@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext context = SpringApplication.run(Application.class, args);
        EmployeeRepository repo = context.getBean(EmployeeRepository.class);

        Sort ascending = Sort.by("empName").ascending();

        List<Employee> findAll = repo.findAll(ascending);
        for (Employee emp : findAll) {
            System.out.println(emp);
        }
    }
}
```

```
2024-10-26T12:53:13.331+05:30  INFO 13184 --- [11-DataJPA_Repo-App] [           main] com.codewitharrays.Application          :
Hibernate: select e1_0.emp_id,e1_0.dept_name,e1_0.emp_name,e1_0.gender,e1_0.salary from emp_tbl e1_0 order by e1_0.emp_name
Employee(empId=1, empName=Ashok, gender=Male, salary=5000.0, deptName=Backend)
Employee(empId=5, empName=Niraj, gender=Male, salary=15000.0, deptName=Fullstack)
Employee(empId=6, empName=Saurabh, gender=Male, salary=9000.0, deptName=HR)
Employee(empId=3, empName=Shubham, gender=Male, salary=8000.0, deptName=Database)
Employee(empId=2, empName=Suraj, gender=Male, salary=7000.0, deptName=Frontend)
Employee(empId=4, empName=Vikas, gender=Male, salary=11000.0, deptName=Production)
2024-10-26T12:53:13.503+05:30  INFO 13184 --- [11-DataJPA_Repo-App] [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean :
```

2. Descending order

```
@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext context = SpringApplication.run(Application.class, args);
        EmployeeRepository repo = context.getBean(EmployeeRepository.class);

        Sort descending = Sort.by("empName").descending();

        List<Employee> findAll = repo.findAll(descending);
        for (Employee emp : findAll) {
            System.out.println(emp);
        }
    }
}
```

```
2024-10-26T12:56:23.386+05:30  INFO 7460 --- [11-DataJPA_Repo-App] [           main] com.codewitharrays.Application          : S
Hibernate: select e1_0.emp_id,e1_0.dept_name,e1_0.emp_name,e1_0.gender,e1_0.salary from emp_tbl e1_0 order by e1_0.emp_name desc
Employee(empId=4, empName=Vikas, gender=Male, salary=11000.0, deptName=Production)
Employee(empId=2, empName=Suraj, gender=Male, salary=7000.0, deptName=Frontend)
Employee(empId=3, empName=Shubham, gender=Male, salary=8000.0, deptName=Database)
Employee(empId=6, empName=Saurabh, gender=Male, salary=9000.0, deptName=HR)
Employee(empId=5, empName=Niraj, gender=Male, salary=15000.0, deptName=Fullstack)
Employee(empId=1, empName=Ashok, gender=Male, salary=5000.0, deptName=Backend)
2024-10-26T12:56:23.935+05:30  INFO 7460 --- [11-DataJPA_Repo-App] [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean : C
```

Also you can do it by Salary. But I want to sort by passing the multiple properties also that can understand by coding example:

Sort sort = Sort.*by*("empName", "salary").descending();

Pagination

- Dividing total records into multiple pages is called as pagination.
- Page_Size : 2 Total Records: 6 Total page : 3 pages
- What is page number : ?
- 1st Page first 2 records
- 2nd page next 2 record
- 3rd page next 2 record
- To implement this we have to create object of PageRequest.of(pageNo, pageSize) the paramenter is pageNo and the second parameter pageSize it will take.
- In Jpa pageNo should start with 0.

```
@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext context = SpringApplication.run(Application.class, args);
        EmployeeRepository repo = context.getBean(EmployeeRepository.class);

        Sort sort = Sort.by("empName", "salary").descending();
        int pageNo=1;
        PageRequest page = PageRequest.of(pageNo-1, 2);

        Page<Employee> findAll = repo.findAll(page);
        List<Employee> content = findAll.getContent();
        for (Employee emp : content) {
            System.out.println(emp);
        }
    }
}
```

```
2024-10-26T14:33:31.654+05:30  INFO 5920 --- [11-DataJPA_Repo-App] [           main] com.codewitharrays.Application          :
Hibernate: select e1_0.emp_id,e1_0.dept_name,e1_0.emp_name,e1_0.gender,e1_0.salary from emp_tbl e1_0 limit ?,?
Hibernate: select count(e1_0.emp_id) from emp_tbl e1_0
Employee(empId=1, empName=Ashok, gender=Male, salary=5000.0, deptName=Backend)
Employee(empId=2, empName=Suraj, gender=Male, salary=7000.0, deptName=Frontend)
2024-10-26T14:33:31.795+05:30  INFO 5920 --- [11-DataJPA_Repo-App] [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean :
```

Query By Example: It is used to prepare Dynamic Query based on data available in Entity

```
@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext context = SpringApplication.run(Application.class, args);
        EmployeeRepository repo = context.getBean(EmployeeRepository.class);

        Employee e1=new Employee();
        e1.setGender("Male");
        e1.setDeptName("Backend");
        Example<Employee> example = Example.of(e1);
            List<Employee> emp = repo.findAll(example);
            for (Employee e : emp) {
                System.out.println(e);
            }
    }
}
```

```
2024-10-26T15:00:59.077+05:30  INFO 12320 --- [11-DataJPA_Repo-App] [           main] com.codewitharrays.Application           : Started Applica
Hibernate: select e1_0.emp_id,e1_0.dept_name,e1_0.emp_name,e1_0.gender,e1_0.salary from emp_tbl e1_0 where e1_0.gender=? and e1_0.dept_name=?
Employee(empId=1, empName=Ashok, gender=Male, salary=5000.0, deptName=Backend)
2024-10-26T15:00:59.217+05:30  INFO 12320 --- [11-DataJPA_Repo-App] [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean : Closing JPA Ent
2024-10-26T15:00:59.219+05:30  INFO 12320 --- [11-DataJPA_Repo-App] [ionShutdownHook] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 -
```

- Right now, the data set hardcoded but in web mvc this data set by UI.

TimeStamping In JPA

- It is used to insert CREATED_DATE & UPDATED_DATE columns values for the record
- @CreationTimeStamp: To populate record created date
- @UpdateTimeStamp: To populate record updated date
- We can understand by the coding example.

```
package com.codewitharrays.repo;

import org.springframework.data.jpa.repository.JpaRepository;

import com.codewitharrays.entity.User;

public interface UserRepository extends JpaRepository<User, Integer> {

}
```

```
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "user_tbl")
public class User {

        @Id
        private Integer userId;
        private String userName;
        private String userGender;
        private String deptName;
        private Double salary;

        @CreationTimestamp
        private LocalDate dateCreated;

        @UpdateTimestamp
        private LocalDate lastUpdated;

}
```

```
@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "user_tbl")
public class User {

	@Id
	private Integer userId;
	private String userName;
	private String userGender;
	private String deptName;
	private Double salary;

	@CreationTimestamp
	private LocalDate dateCreated;

	@UpdateTimestamp
	private LocalDate lastUpdated;
}
```

user_id	date_created	dept_name	last_updated	salary	user_gender	user_name
101	2024-10-26	Admin	2024-10-26	NULL	male	codewitharrays
NULL	NULL	NULL	NULL	NULL	NULL	NULL

- In Above code When we used this @CreationTimeStamp Annotation and @UpdateTimeStamp And insert one record in update column the date is inserted we don't want like that. When the new record inserted the timestamp create and when we update record then update column execute.
- To resolve this problem, we can use attribute in Creation Updateable=false; and in updating insertable=false;

```
@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "user_tbl")
public class User {

	@Id
	private Integer userId;
	private String userName;
	private String userGender;
	private String deptName;
	private Double salary;

	@CreationTimestamp
	@Column(name = "date_create", updatable = false)
	private LocalDate dateCreated;

	@UpdateTimestamp
	@Column(name = "last_update", insertable = false)
	private LocalDate lastUpdated;
}
```

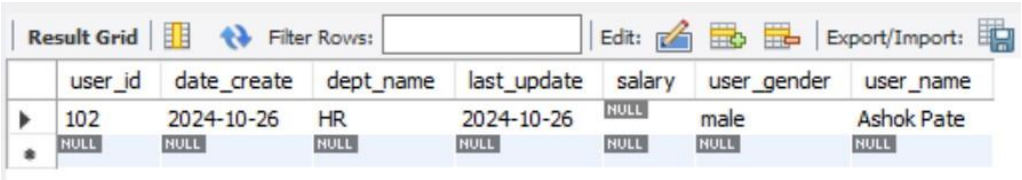
```
@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext context = SpringApplication.run(Application.class, args);

        UserRepository repo = context.getBean(UserRepository.class);

        User user=new User();
        user.setUserId(102);
        user.setDeptName("Super Admin");
        user.setUserName("Ashok Pate");
        user.setUserGender("male");
        repo.save(user);
    }
}
```

Result Grid | Filter Rows: | Edit: | Export/Import:

user_id	date_create	dept_name	last_update	salary	user_gender	user_name
102	2024-10-26	Super Admin	NULL	NULL	male	Ashok Pate
NULL	NULL	NULL	NULL	NULL	NULL	NULL

- Now the problem is resolve. We update some value now.

```
@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext context = SpringApplication.run(Application.class, args);

        UserRepository repo = context.getBean(UserRepository.class);

        User user=new User();
        user.setUserId(102);
        user.setDeptName("HR");
        user.setUserName("Ashok Pate");
        user.setUserGender("male");
        repo.save(user);
    }
}
```



Result Grid | Filter Rows: | Edit: | Export/Import:

user_id	date_create	dept_name	last_update	salary	user_gender	user_name
102	2024-10-26	HR	2024-10-26	NULL	male	Ashok Pate
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Generators

- Generators are used to generate the value for Primary key Columns
- For generators we use @GeneratedValue annotation
- Why we need generators. Right now, we passed the id but while dynamic user don't know about the primary key value so it will be auto generated and that will take care spring boot.
- Also, while defining entity variable, we used Wrapper classes instead of primitive type. Because when we use primitive data type and run the application the default value is stored in data base i.e not recommended. And when we use Wrapper class the null value is store. i.e recommended. If without @GeneratedValue run application then null value stored in primary key column and it will throw null pointer exception.
- Primary Key is a constraint in database it is used to maintain unique data in column
- Primary Key constraint is combination of below 2 constraints
  1) UNIQUE
  2) NOT NULL

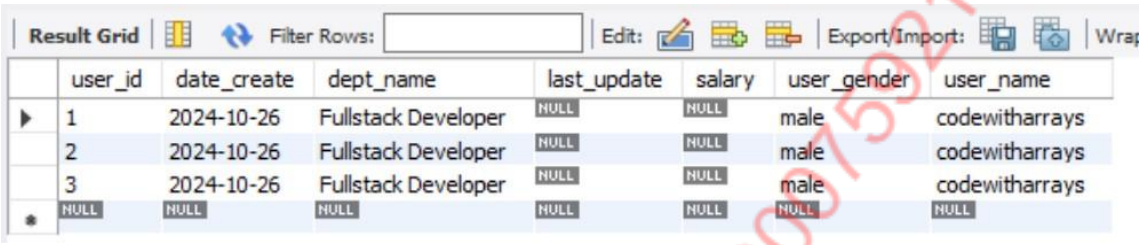
We can specify below strategies for Generator

1) AUTO: it will take automatically if its MySQL database then take IDENTITY and if oracle then take SEQUENCE.
2) IDENTITY (MySQL)
3) SEQUENCE (oracle)
4) TABLE: It will maintain another table to get primary key column values

Example:

```
@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "user_tbl")
public class User {

        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private Integer userId;
        private String userName;
        private String userGender;
        private String deptName;
        private Double salary;

        @CreationTimestamp
        @Column(name = "date_create", updatable = false)
        private LocalDate dateCreated;

        @UpdateTimestamp
        @Column(name = "last_update", insertable = false)
        private LocalDate lastUpdated;
}
```

```java
@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext context = SpringApplication.run(Application.class, args);

        UserRepository repo = context.getBean(UserRepository.class);

        User user=new User();

        user.setDeptName("Fullstack Developer");
        user.setUserName("codewitharrays");
        user.setUserGender("male");
        repo.save(user);
    }
}
```



Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap

user_id	date_create	dept_name	last_update	salary	user_gender	user_name
1	2024-10-26	Fullstack Developer	NULL	NULL	male	codewitharrays
2	2024-10-26	Fullstack Developer	NULL	NULL	male	codewitharrays
3	2024-10-26	Fullstack Developer	NULL	NULL	male	codewitharrays
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Custom Generator

- We want to generate primary key columns values as per project requirement like below.

ORDER_DTLS table: OD_1, OD_2, OD_3 …. OD_100

- To achieve this requirement, we need to develop our own generator. That is called as Custom Generator.

```
public class OrderIdGenerator implements IdentifierGenerator {

        public void generate(){
            String prefix = "OD_"
            String suffix = get from db; [1,2,3...]
            String pk = prefix+suffix;
            //set pk to entity obj
        }
}
```

Composite Primary Keys

- More than one Primary Key column in table is known as composite primary key
- Below is SQL query to create a table with multiple primary key columns

```
create table account_tbl (
        acc_num bigint not null,
        acc_type varchar(255) not null,
        branch varchar(255),
        holder_name varchar(255),
        primary key (acc_num, acc_type)
) ;
```

- To work with Composite PKs we will use below 2 annotations

1. **@Embeddable:** Class level annotation to represent primary key columns
2. **@EmbeddedId:** Variable level annotation to present Embeddable class

Note: Generators will not support for Composite Primary keys

Note: The class which is representing Composite Keys should implement Serializable interface.

```
@Data
@Entity
@Table(name = "acc_tbl")
public class Account {

    private String branchName;

    private String holderName;
    @EmbeddedId
    private AccountPK accountPK;
}
```

```
@Data
@Embeddable
public class AccountPK implements Serializable{

    private Integer accNumber;
    private String accType;
}
```

```
package com.codewitharrays.entity;

import org.springframework.data.jpa.repository.JpaRepository;

public interface AccountRepo extends JpaRepository<Account, AccountPK> {

}
```

```
import org.springframework.boot.SpringApplication;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext context = SpringApplication.run(Application.class, args);

        AccountRepo repo = context.getBean(AccountRepo.class);

        AccountPK pk=new AccountPK();
        pk.setAccNumber(389649535);
        pk.setAccType("saving");

        Account account=new Account();
        account.setHolderName("Ashok");
        account.setBranchName("cidco");
        account.setAccountPK(pk);

        repo.save(account);
    }

}
```

- To retrieve the data, we have to set the Composite key

```
@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext context = SpringApplication.run(Application.class, args);

        AccountRepo repo = context.getBean(AccountRepo.class);

        AccountPK pk=new AccountPK();
        pk.setAccNumber(389649535);
        pk.setAccType("saving");

        Optional<Account> findById = repo.findById(pk);
        if (findById.isPresent()) {
            System.out.println(findById);
        }
    }
}
```

Profiles in Spring Boot

Every Project will have multiple Environments in the Realtime.
Environments are used to test our application before delivering to client
Note: Environment means platform to run our application
(Linux VM, DB Server, Log Server etc...)

We can see below Environments in the Realtime

1) Dev Env: Developers will use it for code Integration testing
2) SIT Env: Testers will use it application end to end testing
3) UAT Env: Client / Client-Side Team will use it for acceptance testing

Note: In UAT client / client-side team will decide GO or NO-GO
GO: Approved for Production deployment
NO-GO: Denied for Production Deployment

4) PILOT Env: Pre-Prod Env for testing with live data
5) PROD Env: Live Environment (end users will access our prod app)

- Every Environment will have its own Database and every database will have separate configuration properties (uname, pwd and URL)

- If we want to deploy our code into multiple envrionments then we have to change configuration properties in application.properties file for every deployment. (It is not recommended).
    1) DB Props
    2) SMTP PRops
    3) Kafka Props
    4) Redis Props
    5) REST API Endpoint URLs
- To avoid this problem we will use Profiles in Spring Boot.
- Profiles are used to configure Environment Specific properties
    1. application-dev.properties
    2. application-sit.properties
    3. application-uat.properties
    4. application-pilot.properties
    5. application-prod.properties

In application.properties file we need to activate profile

# Activating dev profile
spring.profiles.active=dev
Note: Above represents application should load properties from Dev properties file.

Spring Web MVC

It is one module in Spring Framework to develop web applications.
Web MVC module simplified web application development process.

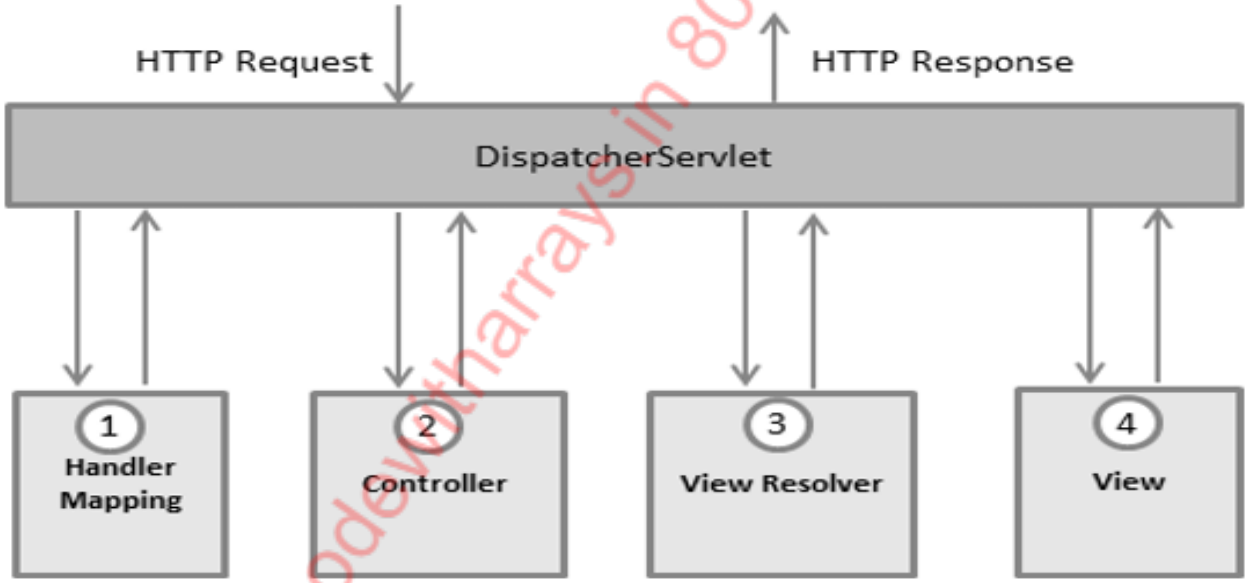
1) Form Binding (form <---> java object)
2) Flexibility in Form Binding (type conversion)
3) Multiple Presentation Technologies (JSP & Thyme leaf)
4) Form Tag Library (ready-made tags support)

Note: To develop web application using spring-boot we need to add below starter in pom.xml

spring-boot-starter-web

The above starter provides support for below things

1) MVC based web applications
2) Restful Services
3) Embedded Container (Tomcat)



- Dispatcher Servlet: Framework Servlet / Front Controller.
  ### Responsible to perform Pre-Processing and Post-Processing of request
- Handler Mapper: Responsible to identify Request Handler class (controller)
- Controller: Java class which is responsible to handle request & response
  ### Controller will return ModelAndView object to Dispatcher Servlet.
- Model: Represents data in key-value format
- View: Logical File Name
  Note: Controllers are loosely coupled with Presentation technology.

- View Resolver: To identify presentation file location and technology
- View: It is responsible to render Model data in view file.

Building First Web app with spring boot

- 1) Create Spring Starter Project with below dependencies
  - a) spring-boot-starter-web
  - b) spring-boot-devtools
  - c) tomcat-embed-jasper (mvnrepository.com)
- 2) Create controller class with required methods & map controller methods to URL pattern
- 3) Create View File with presentation logic
- 4) Configure View Resolver in application.properties file
- 5) Run the application and test it.

Observations

- devtools dependency is used to restart our server when we make code changes.
- To represent java class as controller we are using @Controller annotation
- Controller methods we need to map with HTTP methods using unique URL pattern
  - GET --> @GetMapping
  - POST --> @PostMapping
- Apache Tomcat is coming as default embedded container.
- Embedded container port number is 8080. We can change that port number using application.properties file Server.port = 9090
- Spring Boot web apps will not have context path. We can add context-path using application.properties file. server.servlet.context-path=/codewitharrays

First Web MVC Project to Display Welcome message:

```
@Controller
public class MsgController {

    @GetMapping("/welcome")
    public ModelAndView getWelcomeMsg() {
        ModelAndView mav = new ModelAndView();
        mav.addObject("msg", "Hi, Welcome to Codewitharrays..!!");
        mav.setViewName("message");
        return mav;
    }

    @GetMapping("/greet")
    public ModelAndView getGreetMsg() {
        ModelAndView mav = new ModelAndView();
        mav.addObject("msg", "Good Evening..!!");
        mav.setViewName("message");
        return mav;
    }
}
```

```
spring.mvc.view.prefix=/views/
spring.mvc.view.suffix=.jsp
server.servlet.context-path=/codewitharrays
```

Message.jsp ${msg}

02- Project using Web MVC

Requirement: Retrieve book record based on given id and display in web page.

1) Create Spring Starter Project with below dependencies
   a) web-starter
   b) data-jpa
   c) mysql-connector-j
   d) lombok
   e) devtools
   f) tomcat-embed-jasper
2) Configure View Resolver & Data Source properties in application.properties file
3) Create Entity class (table mapping)
4) Create Jpa Repository interface
5) Create Controller class with methods to handle request & response
6) Create View Page
7) Run the application and test it

@RequestParam: To capture request data using key

Model And View:

```
package com.codewitharrays.entity;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import lombok.Data;
@Data
@Entity
public class Book {

        @Id
        private Integer bookId;
        private String bookName;
        private Double bookPrice;
}


import org.springframework.data.jpa.repository.JpaRepository;

import com.codewitharrays.entity.Book;

public interface BookRepository extends JpaRepository<Book, Integer>{

}
```

```
@Controller
public class BookController {

        @Autowired
        private BookRepository repo;

        @GetMapping("/book")
        public ModelAndView getBookById(@RequestParam("id")  Integer id) {
            System.out.println("id: "+id);
            ModelAndView mav=new ModelAndView();

            Optional<Book> findById = repo.findById(id);
            if (findById.isPresent()) {
                Book bookObj = findById.get();
                System.out.println(bookObj);
                mav.addObject("book", bookObj);
            }
            mav.setViewName("index");
            return mav;
        }
```

```
<html >
<head>

</head>
<body>
        <h1>Book Details</h1>
        <form action="book">
            Book Id: <input type="text" name="id">
            <input type="submit" value="Search">
        </form>
        <hr>
        Book id:  ${book.bookId} <br>
        Book Name: ${book.bookName} <br>
        Book Price: ${book.bookPrice}
</body>
</html>
```

```
spring.application.name=14-Spring_MVC-Book-App-1
#This datasource is used to connection with database
spring.datasource.username=root
spring.datasource.password=cdac
spring.datasource.url=jdbc:mysql://localhost:3306/sbms
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true


spring.mvc.view.prefix=/views/
spring.mvc.view.suffix=.jsp
```


Output:

Book Details

Book Id: 101 Search

Book id: 101
Book Name: Springboot
Book Price: 1000.0

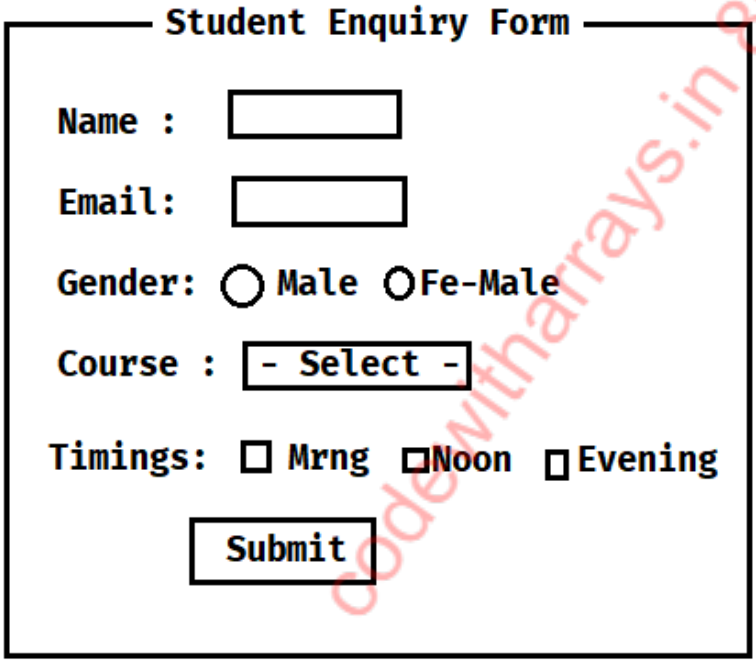
We can write the same controller method in different way. This time we take String as return type and directly return index page. Model pass as argument on model we call addAttribute to set the view

```
@GetMapping("/book")
public String getBookById(@RequestParam("id") Integer id, Model model) {
    Optional<Book> findById = repo.findById(id);
    if (findById.isPresent()) {
        Book bookObj = findById.get();
        model.addAttribute("book", bookObj);
    }
    return "index";
}
```

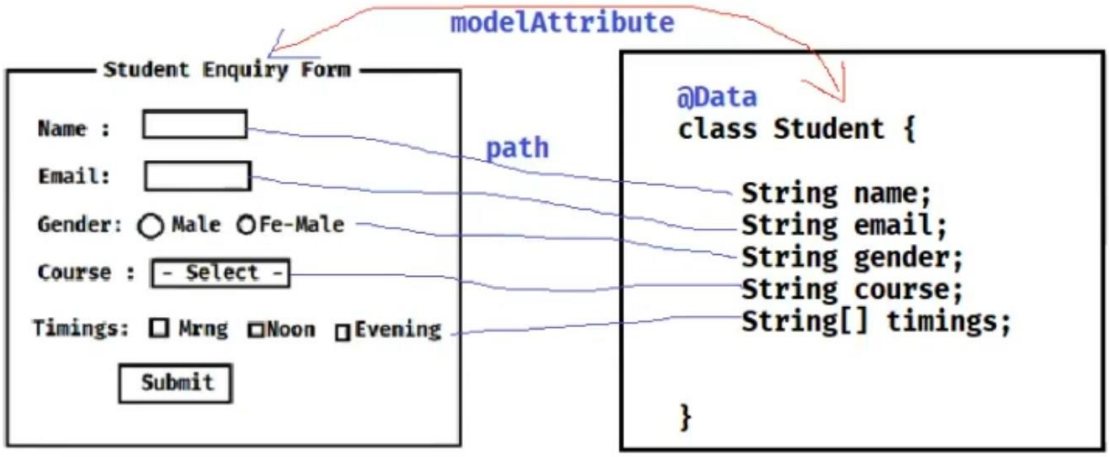
---

03 – Web Application Requirement: Develop Student Enquiry Form like below

1) Course name drop down values should come from database table
2) Timings checkboxes options should come from database table
Note: When we click on submit button record should inserted into database table (STUDENT_ENQUIRIES) and display success message on the same page.



- For above project we map student with modelAttribute and the variable and fields map with path attribute. First we take the hardcoded value for courses and timing to solve.
- Predefined tags provided to simplify Forms development
- To use Spring MVC form tag library in jsp we have to add below taglib url
- <%@ taglib uri="http://www.springframework.org/tags/form" prefix="form" %>




```
package com.codewitharrays.entity;

import lombok.Data;

@Data
public class Student {

        private String name;
        private String email;
        private String gender;
        private String courses;
        private String [] timing;
}
```

```
package com.codewitharrays.controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;

import com.codewitharrays.entity.Student;

@Controller
public class StudentController {

        @GetMapping("/")
        public String loadIndex(Model model) {
            Student std=new Student();
            model.addAttribute("student", std);

            return "index";
        }
        @PostMapping("/save")
        public String handleSaveMethod(Student s,Model model) {
            System.out.println(s);
            model.addAttribute("msg", "Data save");
            return "index";
        }
}
```

```
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form" %>
<!DOCTYPE html>
<html>
<head>
</head>
<body>
    <h1>Student Form</h1>
    <p > <font color="green">${msg}</font> </p>
    <form:form action="save" modelAttribute="student" method="POST">
        <table>
            <tr>
                <td>Name: </td>
                <td> <form:input path="name" /> </td>
            </tr>
            <tr>
                <td>Email:</td>
                <td><form:input path="email"/> </td>
            </tr>
            <tr>
                <td>Gender:</td>
                <td><form:radiobutton  path="gender" value="M"/> Male  </td>
                <td><form:radiobutton  path="gender" value="F"/> Fe-Male  </td>
            </tr>
                <td>Course</td>
                <td><form:select  path="courses">
                    <form:option value="">-Select-</form:option>
                     <!-- This is hardcoded value -->
                    <form:option value="java">Java</form:option>
                    <form:option value="springboot">SpringBoot</form:option>
                    <form:option value="react">React</form:option>
                    </form:select>
                </td>
                <td>Timing</td>
                <td>
                    <!-- This is hardcoded value -->
                <form:checkbox path="timing" value="mrng"/>Morning
                <form:checkbox path="timing" value="noon"/>Afternoon
                <form:checkbox path="timing" value="evening"/>Evening
                </td>
                <tr>
                    <td><input type="submit" value="save"></td>
                </tr>
        </table>
    </form:form>
</body>
</html>
```


Student Form

Data save

Name: ashok

Email: ashok@gmail.com

Gender: (•) Male ( ) Fe-Male

Course SpringBoot Timing [x] Morning [ ] Afternoon [ ] Evening

save

- Now we make service class and pass the course and timing dynamically.

```
package com.codewitharrays.service;

import java.util.Arrays;
import java.util.List;

import org.springframework.stereotype.Service;

@Service
public class StudentService {

    public List<String> getCourses() {

        return Arrays.asList("Java","Springboot" , "react");
    }
     public List<String> getTiming() {
        return Arrays.asList("Morning", "Afternoon", "Evening");
    }
}
```

```
@Controller
public class StudentController {

        @Autowired
        private StudentService service;

        @GetMapping("/")
        public String loadIndex(Model model) {
            Student std=new Student();
            model.addAttribute("student", std);
            List<String> courses = service.getCourses();
            model.addAttribute("course", courses);
            List<String> timing = service.getTiming();
            model.addAttribute("time", timing);
            return "index";
        }
        @PostMapping("/save")
        public String handleSaveMethod(Student s,Model model) {
            System.out.println(s);
            model.addAttribute("msg", "Data save");
            Student std=new Student();
            model.addAttribute("student", std);
            List<String> courses = service.getCourses();
            model.addAttribute("course", courses);
            List<String> timing = service.getTiming();
            model.addAttribute("time", timing);
            return "index";
        }
}
```

```
<td>Course</td>
<td><form:select  path="courses">
    <form:option value="">-Select-</form:option>
    <form:options items="${course}"/>
    <!-- Dynamically taken the values -->
    </form:select>
</td>
<td>Timing</td>
<td>
    <!-- Dynamically taken the values -->
    <form:checkboxes path="timing" items="${time}"/>
    </td>
<tr>
    <td><input type="submit" value="save"></td>
</tr>
```

- Now we can improve our controller code. That formInitBinding method again we write in handleSaveMethod otherwise we get the items null exception. Because the after data save page doesn't reload and we need that courses , timing value in that for that we write this method.

```
@Controller
public class StudentController {

        @Autowired
        private StudentService service;

        private void formInitBinding(Model model) {
            model.addAttribute("student", new Student());
            model.addAttribute("course", service.getCourses());
            model.addAttribute("time", service.getTiming());
        }

        @GetMapping("/")
        public String loadIndex(Model model) {
            formInitBinding(model);
            return "index";
        }

        @PostMapping("/save")
        public String handleSaveMethod(Student s,Model model) {
            System.out.println(s);
            model.addAttribute("msg", "Data save");
            formInitBinding(model);
            return "index";
        }
}
```

Explore More

Subcription : Premium CDAC NOTES & MATERIAL

Contact to Join Premium Group

Click to Join Telegram Group

For More E-Notes

Join Our Community to stay Updated

TAP ON THE ICONS TO JOIN!

	codewitharrays.in freelance project available to buy contact on 8007592194	
SR.NO	Project NAME	Technology
1	Online E-Learning Platform Hub	React+Springboot+MySql
2	PG Mates / RoomSharing / Flat Mates	React+Springboot+MySql
3	Tour and Travel management System	React+Springboot+MySql
4	Election commition of India (online Voting System)	React+Springboot+MySql
5	HomeRental Booking System	React+Springboot+MySql
6	Event Management System	React+Springboot+MySql
7	Hotel Management System	React+Springboot+MySql
8	Agriculture web Project	React+Springboot+MySql
9	AirLine Reservation System / Flight booking System	React+Springboot+MySql
10	E-commerce web Project	React+Springboot+MySql
11	Hospital Management System	React+Springboot+MySql
12	E-RTO Driving licence portal	React+Springboot+MySql
13	Transpotation Services portal	React+Springboot+MySql
14	Courier Services Portal / Courier Management System	React+Springboot+MySql
15	Online Food Delivery Portal	React+Springboot+MySql
16	Muncipal Corporation Management	React+Springboot+MySql
17	Gym Management System	React+Springboot+MySql
18	Bike/Car ental System Portal	React+Springboot+MySql
19	CharityDonation web project	React+Springboot+MySql
20	Movie Booking System	React+Springboot+MySql
	freelance_Project available to buy contact on 8007592194	
21	Job Portal web project	React+Springboot+MySql
22	LIC Insurance Portal	React+Springboot+MySql
23	Employee Management System	React+Springboot+MySql
24	Payroll Management System	React+Springboot+MySql
25	RealEstate Property Project	React+Springboot+MySql
26	Marriage Hall Booking Project	React+Springboot+MySql
27	Online Student Management portal	React+Springboot+MySql
28	Resturant management System	React+Springboot+MySql
29	Solar Management Project	React+Springboot+MySql
30	OneStepService LinkLabourContractor	React+Springboot+MySql

31	Vehical Service Center Portal	React+Springboot+MySql
32	E-wallet Banking Project	React+Springboot+MySql
33	Blogg Application Project	React+Springboot+MySql
34	Car Parking booking Project	React+Springboot+MySql
35	OLA Cab Booking Portal	React+NextJs+Springboot+MySql
36	Society management Portal	React+Springboot+MySql
37	E-College Portal	React+Springboot+MySql
38	FoodWaste Management Donate System	React+Springboot+MySql
39	Sports Ground Booking	React+Springboot+MySql
40	BloodBank mangement System	React+Springboot+MySql
41	Bus Tickit Booking Project	React+Springboot+MySql
42	Fruite Delivery Project	React+Springboot+MySql
43	Woodworks Bed Shop	React+Springboot+MySql
44	Online Dairy Product sell Project	React+Springboot+MySql
45	Online E-Pharma medicine sell Project	React+Springboot+MySql
46	FarmerMarketplace Web Project	React+Springboot+MySql
47	Online Cloth Store Project	React+Springboot+MySql
48	Train Ticket Booking Project	React+Springboot+MySql
49	Quizz Application Project	JSP+Springboot+MySql
50	Hotel Room Booking Project	React+Springboot+MySql
51	Online Crime Reporting Portal Project	React+Springboot+MySql
52	Online Child Adoption Portal Project	React+Springboot+MySql
53	online Pizza Delivery System Project	React+Springboot+MySql
54	Online Social Complaint Portal Project	React+Springboot+MySql
55	Electric Vehical management system Project	React+Springboot+MySql
56	Online mess / Tiffin management System Project	React+Springboot+MySql
57	Online Examination Portal Project	React+Springboot+MySql
58	Lawyer / Advocate Appointment Booking System	React+Springboot+MySql
59	Café Management System	React+Springboot+MySql
60	Agriculture Product Rent system Portal	

Spring Boot + React JS + MySQL Project List

Sr.No	Project Name	YouTube Link
1	Online E-Learning Hub Platform Project	https://youtu.be/KMjyBaWmgzg?si=YckHuNzs7eC84-IW
2	PG Mate / Room sharing/Flat sharing	https://youtu.be/4P9clHg3wvk?si=4uEsi0962CG6Xodp
3	Tour and Travel System Project Version 1.0	https://youtu.be/-UHOBywHaP8?si=KHHfE_A0uv725f12
4	Marriage Hall Booking	https://youtu.be/VXz0kZQi5to?si=llOS-QG3TpAFP5k7
5	Ecommerce Shopping project	https://youtu.be/vJ_C6LkhrZ0?si=YhcBylSErvdn7paq
6	Bike Rental System Project	https://youtu.be/FIzsAmIBCbk?si=7ujQTJqEgkQ8ju2H
7	Multi-Restaurant management system	https://youtu.be/pvV-pM2Jf3s?si=PgvnT-yFc8ktrDxB
8	Hospital management system Project	https://youtu.be/IynIouBZvY4?si=CXzQs3BsRkjKhZCw
9	Municipal Corporation system Project	https://youtu.be/cVMx9NVyI4I?si=qX0oQt-GT-LR_5jF
10	Tour and Travel System Project version 2.0	https://youtu.be/_4u0mB9mHXE?si=gDiAhKBowi2gNUKZ

Sr.No	Project Name	YouTube Link
11	Tour and Travel System Project version 3.0	https://youtu.be/Dm7nOdpasWg?si=P_Lh2gcOFhlyudug
12	Gym Management system Project	https://youtu.be/J8_7Zrkg7ag?si=LcxV51ynfUB7OptX
13	Online Driving License system Project	https://youtu.be/3yRzsMs8TLE?si=JRI_z4FDx4Gmt7fn
14	Online Flight Booking system Project	https://youtu.be/m755rOwdk8U?si=HURvAY2VnizIyJlh
15	Employee management system project	https://youtu.be/lD1iE3W_GRw?si=Y_jv1xV_BljhrD0H
16	Online student school or college portal	https://youtu.be/4A25aEKfei0?si=RoVgZtxMk9TPdQvD
17	Online movie booking system project	https://youtu.be/Lfjv_U74SC4?si=fiDvrhhrjb4KSlSm
18	Online Pizza Delivery system project	https://youtu.be/Tp3izreZ458?si=8eWAOzA8SVdNwlyM
19	Online Crime Reporting system Project	https://youtu.be/0UlzReSk9tQ?si=6vN0e70TVY1GOwPO
20	Online Children Adoption Project	https://youtu.be/3T5HC2HKyT4?si=bntP78niYH802I7N

Sr.No	Project Name	YouTube Link
21	Online Bus ticket booking system Project	https://youtu.be/FJ0RUZfMdv8?si=auHjmNgHMrpaNZvY
22	Online Mess / Tiffin Booking System Project	https://youtu.be/NTVmHFDowyI?si=yrvClbE6fdJ0B7dQ
23		
24		
25		

TAP ON THE ICONS TO JOIN!