

# SPRING BOOT AND MICROSERVICE

BY  
MR. RAGHU SIR



An ISO 9001 : 2008 Certified Company

# SPRING BOOT

## 1. Spring Boot :--

=>Spring boot is a spring based framework which is open source and developed by Pivotal Team.

=>Available versions of Spring Boot are

a>Spring Boot 1.x.

b>Spring Boot 2.x.

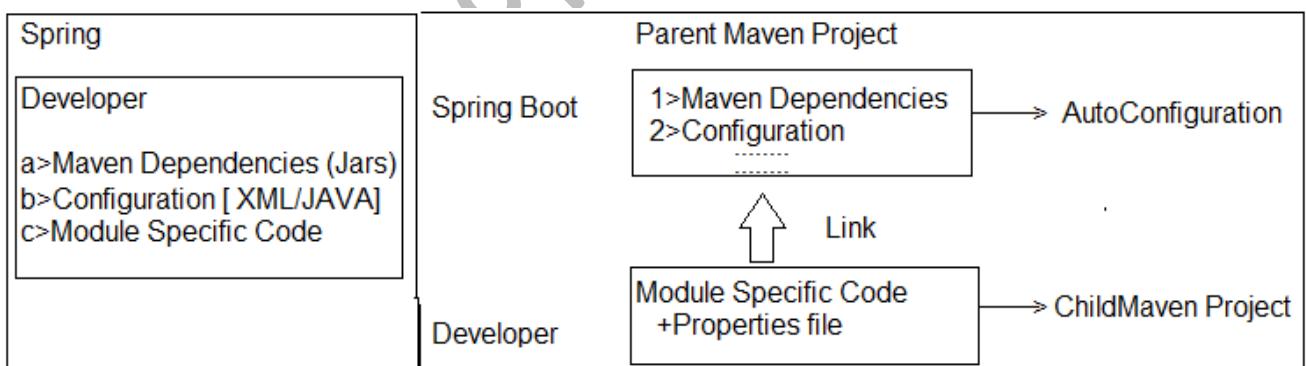
=>Spring Boot provides **AutoConfiguration** which means reduce Common lines of code in Application which is written by Programmers and handles Jars with version management. (i.e. Providing Configuration code XML/Java and maintaining all jars required for Project **Parent Jars + Child Jars**)

=>Spring Boot is an Abstract Maven project also called as Parent Maven Project (A Project with partial code and jars)

=>Here Programmer will not write configuration code but need to give input data using

a>Properties File (application.properties).

b>YAMAL File (application.yml).



Example:--

### 1>Spring JDBC and Spring Boot JDBC [Sample code Comparison]:--

#### 1>Spring JDBC:--

##### a>XML configuration:--

<beans...>

```

<bean name="dsObj" class=org.sf...DriverManagerDataSource>
<property name="driverClassName" value="oracle.jdbc.OracleDriver"/>
<property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>

```

```
<property name="username" value="system"/>
<property name="password" value="system"/>
</bean></beans>
```

**b>Maven Dependencies with version jar:--**

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-jdbc</artifactId>
  <version>5.1.3.RELEASE</version>
</dependency>
```

**2>Spring Boot:--**

**a>application.properties:--**

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

**b>Starter Dependency (which gives config code and Jars):--**

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

=>Spring Boot supports end to end process that is called.

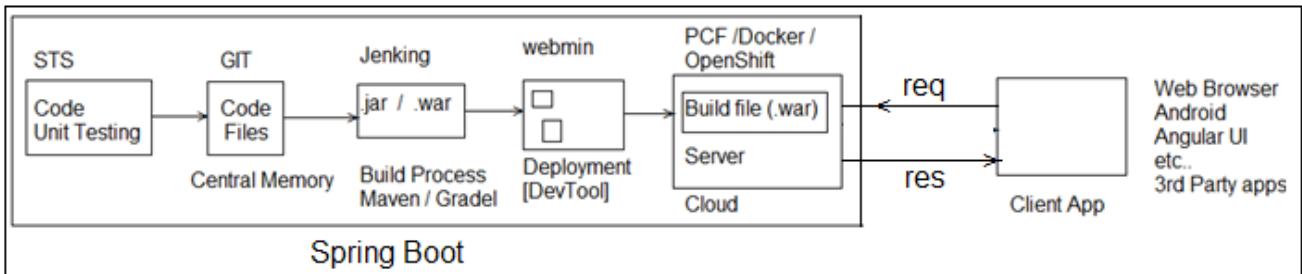
=>Coding => Unit testing => Version control => Build => Deployment => Client Integration.

a.>GIT (github.com) is used to store our code files. It is called as **Central Repository or version Control Tool.**

b.>.Java is converted to .class (Compile) .class + (other files .xml, .html...) converted to .jar/.war finally (build process).

c.>Place .jar/.war in server and start server is called as Deployment.

d.>Spring Boot Application is a service provider app which can be integrated with any UI client like Android, Angular UI, RFID (Swiping Machine), Any 3<sup>rd</sup> party Apps, Web Apps using Rest and JMS.

**NOTE:--**

- a>Spring Boot supports two build tools **Maven** and **Gradle**.
- b>Spring Boot supports 3 embedded servers and 3 embedded databases. These are not required to download and install.

**i>Embedded Servers:--**

- 1>Apache Tomcat (default)
- 2>JBoos Jetty
- 3>Undertow

**ii>Embedded DataBase:--**

- 1>H2
- 2>HSQL DB
- 3>Apache Derby

**c>Spring Boot supports cloud apps with micro services pattern. ["Both coding and Deployment"].**

- =>Coding is done using Netflix Tools
- =>Deployment is done using PCF Tools (or its equal...)

**d>Spring Boot supports basic Operations:--**

- 1>WebMVC and WebServices (Rest).
- 2>JDBC and ORM (Hibernate with JPA).
- 3>Email, Scheduling, JMS, Security.
- 4>Cache and Connection Pooling.
- 5>DevTools, Swagger UI, Actuator and Profiles.
- 6>UI Design using HTML, JSP, Thymeleaf ...etc.

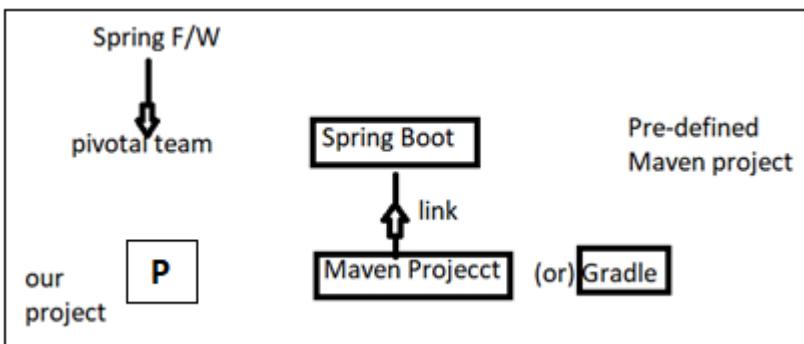
**e> Supports Input Data (Key = val) Using (for AutoConfiguration code):--**

- =>Properties file or YAML files.

### 1.1 Spring Boot Application Folder System:--

=>We can write spring Boot application either using Maven or using Gradle (one of build tool).

=>Our project contains one parent project of spring boot which is internally maven project (hold version of parent).



=>Application should contain 3 major and required files.

Those are

1. **SpringBootStarter class**
2. **application.properties /application.yml**
3. **pom.xml/build.gradle**

**1. SpringBootStarter class:--** It is a main method class used to start our app. It is entry point in execution. Even for both stand alone and web this file used.

**2. application.properties/application.yml:--** This is input file for Spring boot (Spring container). It holds data in key=value format.

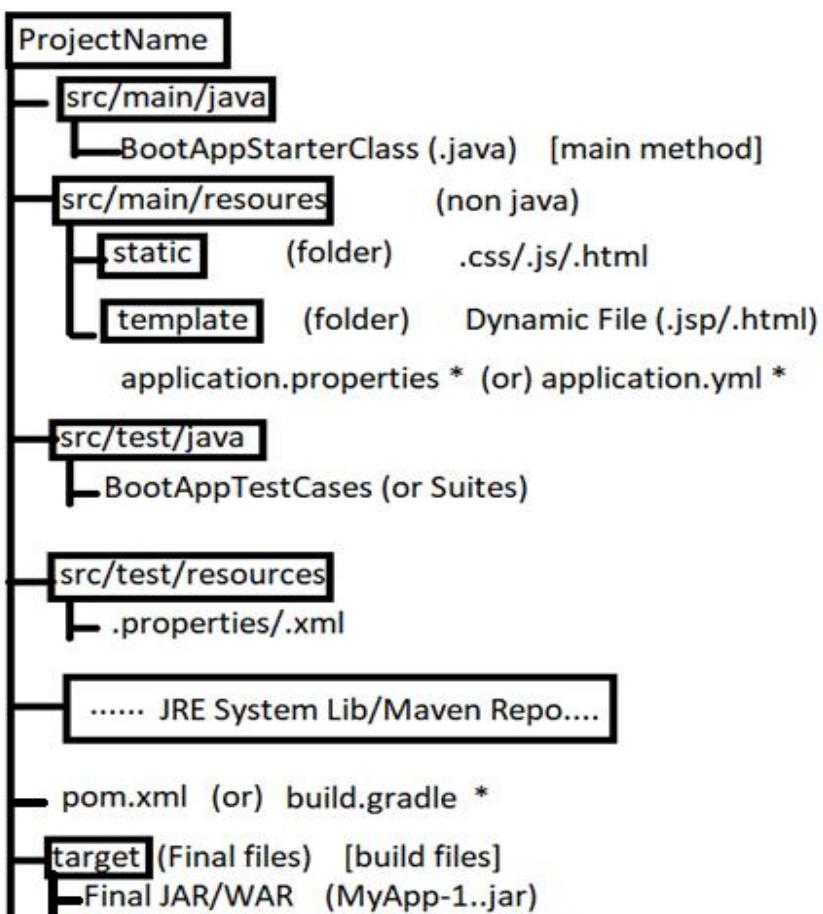
\*\* File name must be “application” or its extended type.

\*\* Even .yml (YAML) file is finally converted to .properties only using SnakeYaml API

\*\* yml is better approach to write length properties code.

**3. pom.xml (or) build.gradle:--** This file holds all information about

- a. Parent boot project version
- b. App properties (JDK version/maven/cloud versions....)
- c. Dependencies (JARS Details)
- d. Plugins (Compiler/WAR...etc)

Application Folder System

# CHAPTER#1 SPRING BOOT CORE

## 1. Spring Boot Runners:--

- =>A Runner is an auto-executable component which is called by container on application startup only once.
- =>In simple this concept is used to execute any logic (code) one time when application is started.

## Types of Runners(2):--

- 1.1 CommandLineRunner** :-- This is legacy runner (old one) which is provided in Spring boot 1.0 version.
- =>It has only one abstract method “run(String... args) : void”.
  - =>It is a Functional Interface (having only one abstract method).
  - =>Add one **Stereotype Annotation** over Implementation class level (Ex:- @Component). So that container can detect the class and create object to it.

## Code Setup:--

#Setup : JDK 1.8 and Eclipse / STS.

## #1. Create Maven Project (simple one):--

- =>File=>new=>Maven Project (\*\*Click check box [ v ])=> Create Simple Project
- =>Next =>Enter Details (example)
- Group Id : com.app
- ArtifactId : SpringBootRunners
- Version : 1.0
- =>Finish

## #2. Open pom.xml and add parent, Properties, dependencies with plugins:--

Add details in pom.xml (Project Object Model). This file should contain bellow details in same order. It is Automatic Created with project.

- 1.>Parent Project Details.
- 2.>Properties (with java version).
- 3.>Dependencies (jar file details).
- 4.>Build Plugin.

pom.xml:--

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.app</groupId>
  <artifactId>SpringBootRunner</artifactId>
  <version>1.0</version>
  <!-- a. Parent Project details -->
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.2.RELEASE</version>
  </parent>
  <!-- b. Versions/properties -->
  <properties>
    <java.version>1.8</java.version>
  </properties>
  <!-- c. dependencies/jars -->
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter</artifactId>
    </dependency>
  </dependencies>
  <!-- d. build plugins -->
  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>
</project>
```

**#3. Create Properties file under src/main/resources folder:--**

=>Right click on “src/main/resources”=>new =>other=>Search and choose “File”

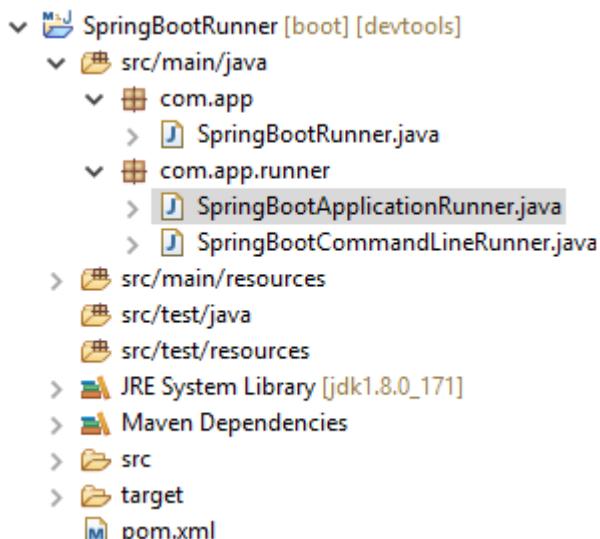
=>next=>Enter name Ex:- application.properties => Finish

**#4. Write Spring Boot starter class under src/main/java folder:--**

=>Right click on “src/main/java”> new => class => Enter details, like:

Package Name : com.app

Name : MyAppStarter > Finish

**#5. Create one or more Runner classes under src/main/java folder with package “com.app”:--****#1. Folder Structure of CommandLineRunner & ApplicationRunner with Ordered interface implementations:-**

Code:--

**1>SpringBootRunner.java (Spring Boot Starter class):--**

```
package com.app;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringBootRunner {
    public static void main(String[] args) {
        SpringApplication.run(SpringBootRunner.class, args);
        System.out.println("Hello Uday");
    }
}
```

## #Runner #1: SpringBootCommandLineRunner.java

```
package com.app.runner;
import org.springframework.boot.CommandLineRunner;
import org.springframework.core.Ordered;
import org.springframework.stereotype.Component;

/*CommandLineRunner with Ordered implementations Manual Approach*/
@Component
public class SpringBootCommandLineRunner implements CommandLineRunner,
Ordered {
    @Override
    public void run(String... args) throws Exception {
        System.out.println("Hii CommandLine Runner");
    }
    @Override
    public int getOrder() {
        return 50;
    }
}
```

## #Runner #2: SpringBootApplicationRunner.java

```
package com.app.runner;
import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.ApplicationRunner;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.annotation.Order;

/*ApplicationRunner with Ordered implications with Annotations*/
@Configuration
@Order(50)
public class SpringBootApplicationRunner implements ApplicationRunner {
    @Override
    public void run(ApplicationArguments args) throws Exception {
        System.out.println("Hello Application Runner");
    }
}
```

**Output:--**

```

  \ \ / \ \ 
  (( )) [ ] [ ] 
  \| \| [ ] [ ] 
  | | | | | | | | 
  ======| | | | | | 
  :: Spring Boot ::      (v2.1.2.RELEASE)

2019-02-14 18:08:47.050 INFO 9424 --- [ restartedMain] com.app.SpringBootRunner      : Starting SpringBootRunner on DESKTOP-CH8LPUH with PID 9424 (started
2019-02-14 18:08:47.058 INFO 9424 --- [ restartedMain] com.app.SpringBootRunner      : No active profile set, falling back to default profiles: default
2019-02-14 18:08:47.195 INFO 9424 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'spring.devtools.add-propert
2019-02-14 18:08:48.749 INFO 9424 --- [ restartedMain] o.s.b.d.a.OptionalLocalReloadServer   : LiveReload server is running on port 35729
2019-02-14 18:08:48.814 INFO 9424 --- [ restartedMain] com.app.SpringBootRunner      : Started SpringBootRunner in 2.924 seconds (JVM running for 6.501)
Hello Application Runner
Hi Commandline Runner
Hello Uday

```

**NOTE:--**

a>Boot Application can have multiple runners Ex:-- EmailRunner, JmsRunner, SecurityRunner, CloudEnvRunner, DevOpsRunner, DatabaseRunner etc...

b>Boot provides default execution order.

=>To specify programmer defined order use

i>Interface : Ordered or else

ii>Annotation : @Order

=>If we are configures both Runner but not implements Ordered then by default **Annotation based Configuration** will be executed first.

**1. Input Data to Runners using (CommandLineArguments):--**

Programmer can pass one time setup data using Command Line Arguments, in two formats.

a>Option Arguments

b>NonOption Arguments

Syntax : --key =val [Option Arguments]

Ex:-- --db=MySQL --db=Oracle

--env=prod --server.port=9876

etc...

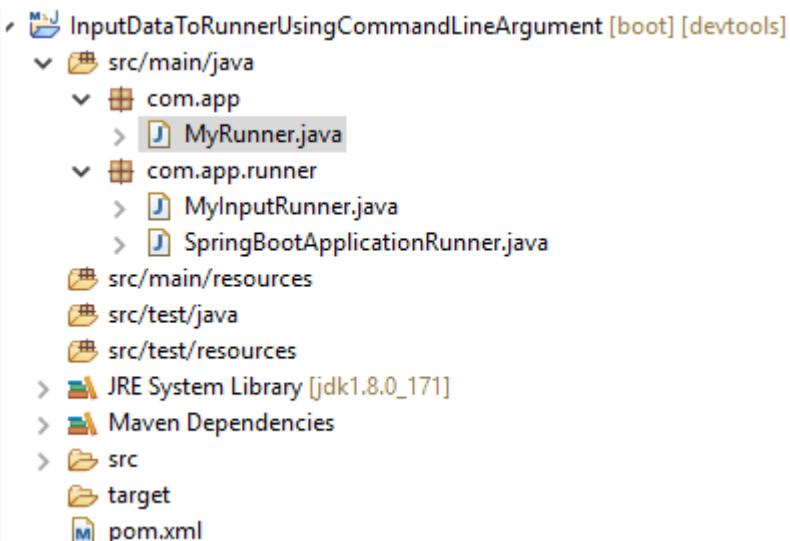
Syntax : data [NonOption Argument]

Test clean package execute rollnone etc...

\*\*>Data is converted into String[ ] (String...) [var-args] and send to Runner class.

\*\*>Read data based on index or all.

## #2. Folder Structure of Reading Input Data Using CommandLine Arguments:--



### 1>Starter class (MyRunner.java):--

```
package com.app;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
@SpringBootApplication
public class MyRunner {
    public static void main(String[] args) {
        SpringApplication.run(MyRunner.class, args);
        System.out.println("Starter class Called");
    }
}
```

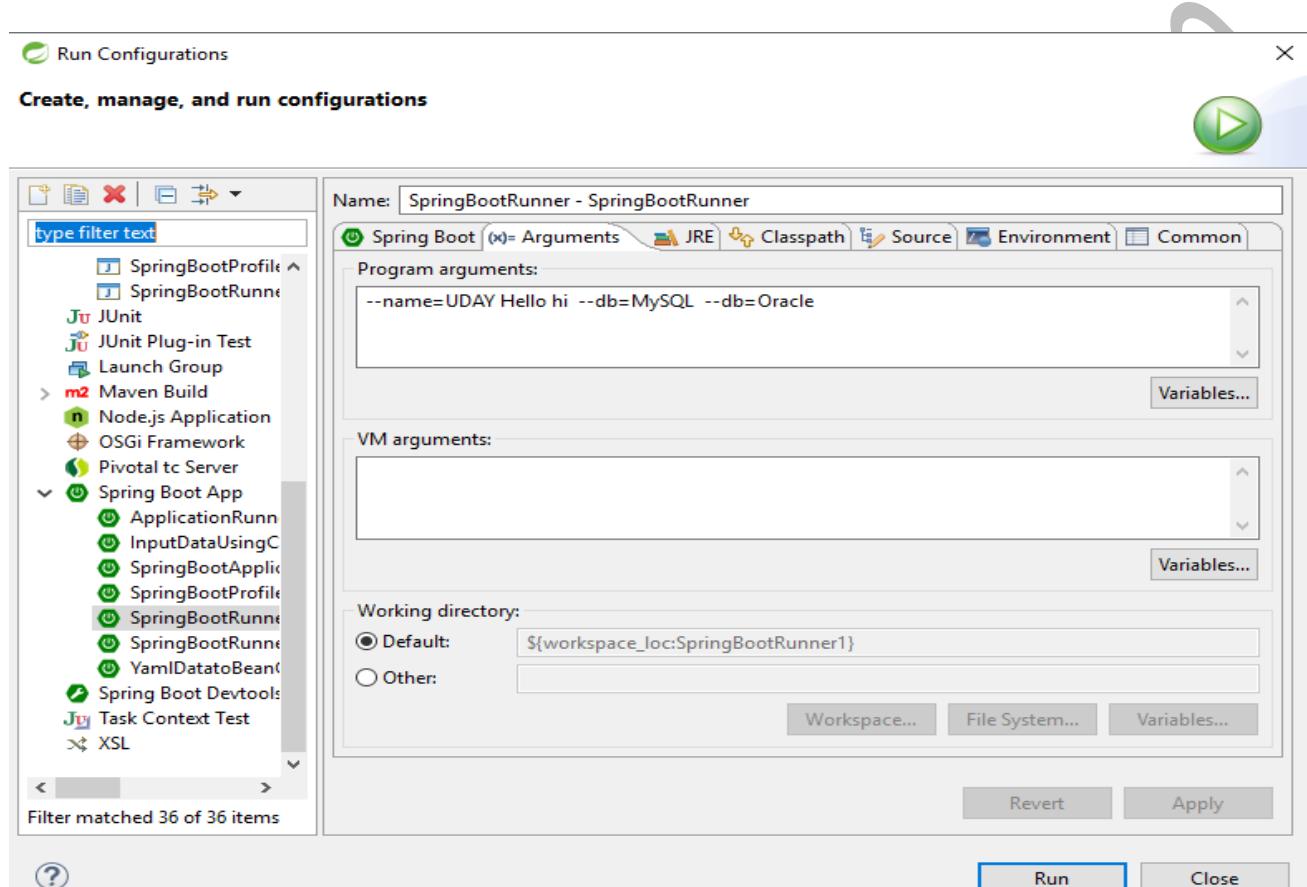
### 2>Runner #1 MyInputRunner.java:--

```
package com.app.runner;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

@Component
public class MyInputRunner implements CommandLineRunner {
    public void run (String... args) throws Exception {
        System.out.println("Hello CommandLineRunner");
        System.out.println(args[1]);
        System.out.println(Arrays.asList(args));
        System.out.println("End of CommandLineRunner");
    }
}
```

**Execution:-**

- =>Right Click on starter class code (main)
- =>Run As => Run Configuration
- =>Choose “Arguments” tab
- =>Enter data in Program arguments (with space)  
--name=UDAY Hello hi --db=MySQL --db=Oracle



- =>Click on Apply and Run
- =>It is internally converted to  
String [] (String... args)

String[]	
--Hello=a	0
hi	1
ok	2
-db=mysql	3
--db=xyz	4

**Anonymous Inner class:--**

=>A nameless class and object created for an interface having abstract methods.  
=>In simple create one class without name and create object at same time without name. Used only one time.

Syntax:--

```
new InterfaceName() {  
    //Override all methods  
}
```

**Example#1:--**

```
interface Sample {  
    void show();  
}
```

**-----Anonymous Inner class--**

```
new Sample() {  
    public void show() {  
        System.out.println("Hello");  
    }  
}
```

**Example#2:--**

```
interface CommandLineRunner {  
    void run(String... args) throws Exception;  
}
```

**Anonymous Inner class:--**

```
new CommandLineRunner() {  
    public void run (String... args) throws Exception {  
        System.out.println("HI");  
    } };
```

**\*\*Java Style Configuration for CommandLineRunner:--**

**Code:-**

```

package com.app;
//Ctrl+shift+O
@Configuration
public class AppConfig {
    //JDK 1.7 or before (Inner class style)
    @Bean
    public CommandLineRunner cob () {
        return new CommandLineRunner() {
            public void run (String... args) throws Exception{
                System.out.println(Arrays.asList(args));
            } };
    }
    //JDK 1.8 or higher (lambda)
    @Bean
    public CommandLineRunner cob2() {
        return (args) -> {
            System.out.println(Arrays.asList(args));
        };
    }
}

```

**Q>How CommandLineRunner works?**

A>CommandLine arguments which are passed to application which will be given to Spring Boot starter main(..) method. Those are stored as “string Array” (String[]). SpringApplication.run(...) reads this input and internally calls run(..) methods of RunnerImpl classes and pass same data.

CommandLine Input	SpringBootStarter class <b>(1)</b>  main(String[] args) { SpringApplication.run(----, args) <b>(2)</b>	<b>CommandLineRunner (I)</b>  <b>(3)</b> run(String... args)										
--Hello=a hi ok --db=mysql --bd=xyz -	String[] <table border="1"> <tr><td>--Hello=a</td><td>0</td></tr> <tr><td>hi</td><td>1</td></tr> <tr><td>ok</td><td>2</td></tr> <tr><td>-db=mysql</td><td>3</td></tr> <tr><td>--db=xyz</td><td>4</td></tr> </table>	--Hello=a	0	hi	1	ok	2	-db=mysql	3	--db=xyz	4	
--Hello=a	0											
hi	1											
ok	2											
-db=mysql	3											
--db=xyz	4											

**1.2 ApplicationRunner(I) :-** It is new type runner added in Spring boot 1.3 which makes easy to access arguments.

=>This will separate Option Arguments (as Map<String, List<String>>) and Non-Option Arguments (<List<String>)

=>This Data Stored in Object of “ApplicationArguments” as given below.

CommandLine Input	SpringBootStarter class (1)	ApplicationRunner																		
--Hello=a hi ok --db=mySQL --bd=xyz	<pre>main(String[] args) {     SpringApplication.run(..., args) (2) } String[]</pre> <table border="1"> <tr><td>--Hello=a</td><td>0</td></tr> <tr><td>hi</td><td>1</td></tr> <tr><td>ok</td><td>2</td></tr> <tr><td>-db=mySQL</td><td>3</td></tr> <tr><td>--db=xyz</td><td>4</td></tr> </table>	--Hello=a	0	hi	1	ok	2	-db=mySQL	3	--db=xyz	4	<b>ApplicationArguments</b> <b>List (non-option-args)</b> <table border="1"> <tr><td>hi</td><td>ok</td></tr> </table> <b>Map &lt;String, List&lt;String&gt;&gt;</b> <table border="1"> <thead> <tr><th>name</th><th>val</th></tr> </thead> <tbody> <tr><td>Hello</td><td>a</td></tr> <tr><td>db</td><td>mySQL xyz</td></tr> </tbody> </table>	hi	ok	name	val	Hello	a	db	mySQL xyz
--Hello=a	0																			
hi	1																			
ok	2																			
-db=mySQL	3																			
--db=xyz	4																			
hi	ok																			
name	val																			
Hello	a																			
db	mySQL xyz																			

### 3>Runner#2 (SpringBootApplicationRunner.java):-

```
package com.app.runner;
import java.util.Arrays;
import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.ApplicationRunner;
import org.springframework.stereotype.Component;

@Component
public class SpringBootApplicationRunner implements ApplicationRunner {
    public void run(ApplicationArguments args) throws Exception {
        System.out.println("hello Application Runner");
        System.out.println(Arrays.asList(args.getSourceArgs()));
        System.out.println(args.getNonOptionArgs());
        System.out.println(args.getOptionNames());
        System.out.println(args.getOptionValues("db"));
        System.out.println(args.containsOption("bye"));
        System.out.println("End of Application Runner");
    }
}
```

## **Output:-**

**Q> What is the difference between CommandLineArgument and ApplicationRunner?**

A> Working process of CommandLineRunner and ApplicationRunner are same, but CommandLineRunner (CLR) holds data in String[] format where as Application (AR) holds data as ApplicationArguments as Option/Non-Option format.

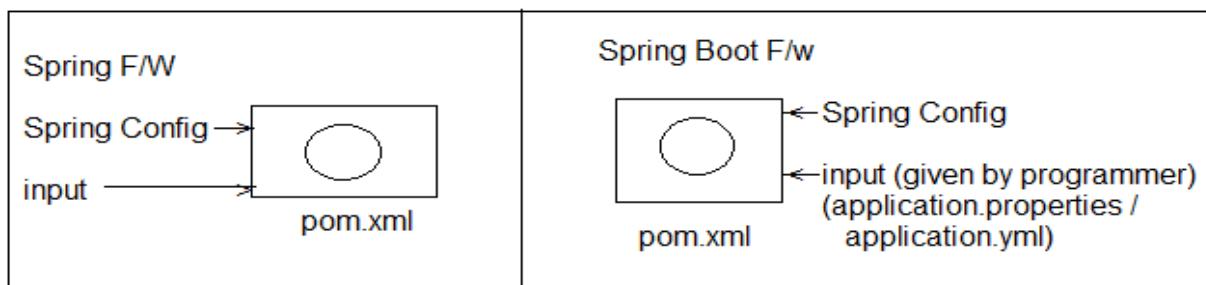
## 2. Spring Boot Input Data (Using application.properties):--

=>application.properties or application.yml is a primary source input to spring boot (Spring Container).

=>Spring Boot F/W writes Configuration code (XML/Java Config) for programmer.

=>Here we are not required to write (@Bean or <bean..>) configuration for common application setup like JDBC Connection, Hibernate Properties, DispatcherServlet Config, Security Beans etc..

=>But Programmer has to provide input to the above beans (Objects) using Properties or YAML File (any one).



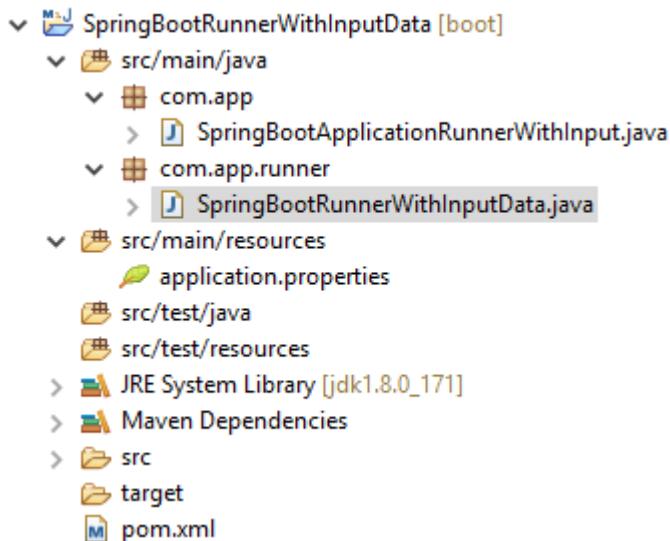
**application.properties:--**

1>It holds in key=value format

2>Keys are two types

a>Spring Boot defined (Predefined) Reference Link: <https://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html>)

b>Programmer defined

**#3. Folder Structure of Spring Boot Input Data using application.properties:--****Code:--**

**#1.** Create maven project and provide pom.xml and starter class

**#2.** application.properties (src/main/resources)

=>Right click on src/main/resource folder=>new =>other=>search and Select

“File”=>enter name “application.properties” => finish

**application.properties:--**

my.info.product.id=999A

my.info.product.code=xyz

my.info.product.model-version=44.44

my.info.product.release\_dtl\_enable=false

my.info.product.start-key=N

**NOTE:--**

a> Allowed special symbol are **dot(.)**, **dash(-)** and **underscore (\_)**.

b> Key=value both are String type, Spring supports both are String type, Spring supports type conversation (ex String->int) automatically.

c> To read one key-value in code use Legacy syntax : @Value("\${key}")

#3. Starter class same as above.

#4. Runner with key data class (**SpringBootRunnerWithInputData.java**)---

```
package com.app.runner;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

@Component
public class SpringBootRunnerWithInputData implements CommandLineRunner {
    @Value("${my.info.product.id}")
    private int proId;
    @Value("${my.info.product.code}")
    private String prodCode;
    @Value("${my.info.product.model-version}")
    private double modelver;
    @Value("${my.info.product.release_dtl_enable}")
    private boolean isDetEnable;
    @Value("${my.info.product.start-key}")
    private char startKey;

    //Constructor methods
    //Setters and Getters method
    //toString method
    @Override
    public String toString() {
        return "SpringBootRunnerWithInputData [proId=" + proId + ", prodCode=" + prodCode + ", modelver=" + modelver + ", isDetEnable=" + isDetEnable + ", startKey=" + startKey + "]";
    }
    //Overridden run method
    public void run(String... args) throws Exception {
        System.out.println(this);
        //System.out.println(this.toString());
    }
}
```

**Output:--**

```

Console Progress Problems
<terminated> SpringBootRunnerWithInputData - SpringBootApplicationWithInput [Spring Boot App] C:\Program Files\Java\jdk1.8.0_171\bin\javaw.exe (Feb 24, 2019, 01:17:13.094)
.
.
.
:: Spring Boot ::      (v2.1.2.RELEASE)

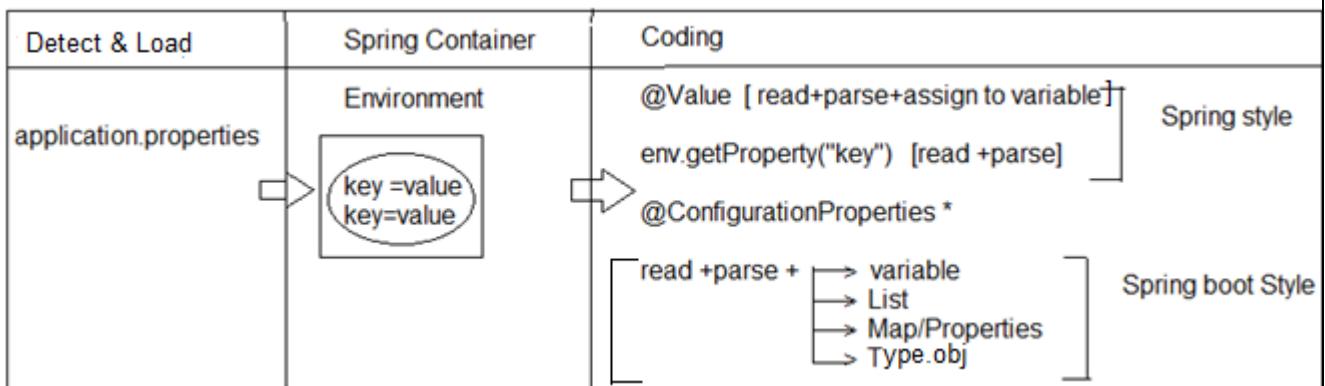
2019-02-24 01:17:13.094  INFO 16588 --- [           main] c.a.SpringApplicationWithInput : Starting SpringBootApp
2019-02-24 01:17:13.094  INFO 16588 --- [           main] c.a.SpringApplicationWithInput : No active profile set, falling back to default profiles: default
2019-02-24 01:17:14.484  INFO 16588 --- [           main] c.a.SpringApplicationWithInput : Started SpringBootApp in 1.397 seconds (JVM: 1.397s)
SpringBootRunnerWithInputData [prodId=999, prodCode=xyzs, modelver=44.48, isDetEnable=false, startKey=N]

```

**NOTE:--** If key data is mismatched with variable data type, then Spring Container throws Exception : TypeMismatchException : Failed to convert value.

**Internal flow:--**

Spring Boot will search for file “application.properties” in project (4 different locations)  
=>Once found (detected) then load into Container and store as “Environment” obj.  
2>We can read data in legacy style @Value or 2>env.getProperty(..).  
3>Boot Style (Bulk Loading) can be done using Annotation. \*\*\*  
4>@ConfigurationProperties

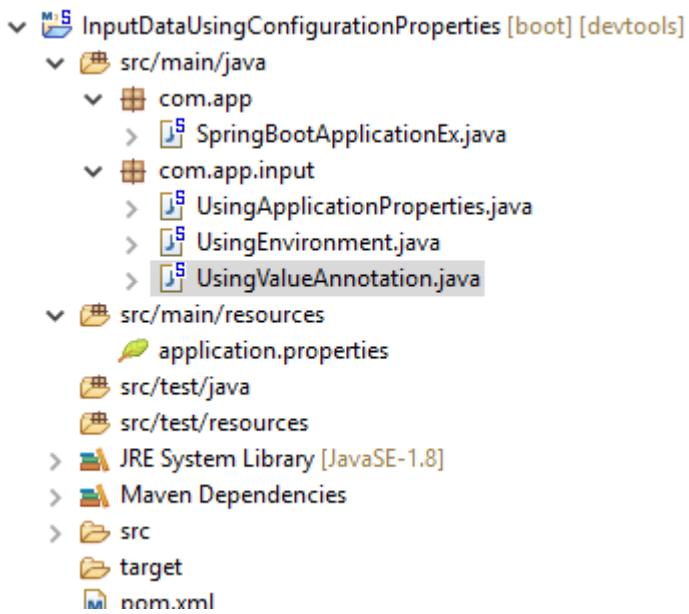
**4>@ConfigurationProperties:--**

=>This Annotation is used to perform bulk data reading (multiple keys at a time) and parsing into one class type (Stores in project).

=>Possible conversions are.

- a>1key = 1 variable
- b>Multiple keys with index = List/Set/Array
- c>Multiple keys with key-value format = Map or Properties
- d>Multiple keys with common type = Class Object (Has-A)

## #4. Input Data Using ConfigurationProperties:--



Example:--

**1. Starter class (SpringBootApplicationEx.java):--**

```
package com.app;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringBootStarter {
    public static void main(String[] args) {
        SpringApplication.run(SpringBootStarter.class, args);
        System.out.println("Hello Spring Boot");
    }
}
```

**2. application.properties:--**

#One variable data

```
my.prod.ID=999
my.prod.co-de=ABC
my.prod.Ty_pe=true
my.prod.MOD-E_L=p
```

#List&lt;DT&gt;/Set&lt;DT&gt;/DT[]

my.prod.prjnm[0]=P1

my.prod.prjnm[1]=**P2**

my.prod.prjnm[2]=**P3**

**#Map or Properties**

my.prod.mdata.s1=**55**

my.prod.mdata.s2=**66**

my.prod.mdata.s3=**88**

**#One class Object**

my.prod.dpt.dname=**AAA**

my.prod.dpt.did=**8987**

**#Random data generator**

my.random.stringval= **\${random.value}**

my.random.num= **\${random.int}**

my.random.bignum= **\${random.long}**

my.random.num-range= **\${random.int[10]}**

my.random.num-from-to= **\${random.int[10,100]}**

my.random.uuid-type= **\${random.uuid}**

### **3. Runner #1 class (UsingEnvironment.java):--**

```
package com.app.input;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.core.env.Environment;
import org.springframework.stereotype.Component;
```

**@Component**

```
public class UsingEnvironment implements CommandLineRunner{
```

**@Autowired**

```
private Environment env;
```

**@Override**

```
public void run(String... args) throws Exception {
```

```
    System.out.println(env.getProperty("my.prod.ID"));}
```

```
        System.out.println(env.getProperty("my.prod.code"));
        System.out.println(env.getProperty("my.prod.Ty_pe"));
        System.out.println(env.getProperty("my.prod.MOD-E_L"));
    }
}
```

**Example #2:**-- Load all key-value based on common prefix

\*\* Do not provide any prefix at annotation level Make sure create variable with first level prefix before first will)

**Runner #2 class (UsingApplicationProperties):--**

```
package com.app.input;
import java.util.Arrays;
import java.util.List;
import java.util.Set;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;

@ConfigurationProperties("my.prod")
@Component
public class UsingApplicationProperties implements CommandLineRunner {

    private int id;
    private String code;
    private boolean type;
    private char model;
    private List<String> projname;
    private Set<String> projname1;
    private String[] projname2;

    public UsingApplicationProperties() {
        super();
    }

    public int getId() {
        return id;
    }
}
```

```
public void setId(int id) {  
    this.id = id;  
}  
public String getCode() {  
    return code;  
}  
public void setCode(String code) {  
    this.code = code;  
}  
public boolean isType() {  
    return type;  
}  
public void setType(boolean type) {  
    this.type = type;  
}  
public char getModel() {  
    return model;  
}  
public void setModel(char model) {  
    this.model = model;  
}  
public List<String> getProjname() {  
    return projname;  
}  
public void setProjname(List<String> projname) {  
    this.projname = projname;  
}  
public Set<String> getProjname1() {  
    return projname1;  
}  
public void setProjname1(Set<String> projname1) {  
    this.projname1 = projname1;  
}  
public String[] getProjname2() {  
    return projname2;  
}
```

```

    public void setProjname2(String[] projname2) {
        this.projname2 = projname2;
    }
    @Override
    public String toString() {
        return "UsingApplicationProperties [id=" + id + ", code=" + code + ", type=" + type + ",
model=" + model+ ", projname=" + projname + ", projname1=" + projname1 + ",
projname2=" + Arrays.toString(projname2)+ "]";
    }
    @Override
    public void run(String... args) throws Exception {
        System.out.println(this.toString());
    }
}

```

**Example #3 Generating Random Values:**--We can use a direct expression

@Value("{random.--}") in java code or in properties file.

=>Possible random data is

- a>Hexa Decimal Value
- b>int or long type
- c>int or long with range
- d>UUID (Universal Unique Identifier)

**Example #3:**--

1>**application.properties**--

```

#Random data generator
my.random.stringval=${random.value}
my.random.num=${random.int}
my.random.bignum=${random.long}
my.random.num-range=${random.int[10]}
my.random.num-from-to=${random.int[10,100]}
my.random.uuid-type=${random.uuid}

```

**2>Model class with Runner (UsingValueAnnotation):--**

```
package com.app.input;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;

@Component
public class UsingValueAnnotation implements CommandLineRunner {

    //@Value("${my.random.stringval}")
    //@Value("${my.random.stringval}")
    //@Value("${random..value}")
    @Value("${my.random.uuid-type}")
    private String code;

    @Value("${my.random.num}")
    //@Value("${my.random.num-rang}")
    //@Value("${my.random.num-rang-from-to}")
    private int num;

    @Value("${my.random.bignum}")
    private long numbig;

    @Override
    public void run(String... args) throws Exception {
        System.out.println(this);
    }
    public UsingValueAnnotation() {
        super();
    }
    public UsingValueAnnotation(String code, int num, long numbig) {
        super();
        this.code = code;
        this.num = num;
        this.numbig = numbig;
    }
}
```

```
    }

    @Override
    public String toString() {
        return "UsingValueAnnotation [code=" + code + ", num=" + num +",
numbig=" + numbig + "]";
    }
}
```

## **Output:-**

### **3. Possible Locations for Properties (YAML) file:-**

=>Spring Boot supports 4 default and priority order locations, which are loaded by container for key=val data.

1> Under Project:-- Under Config folder:--

Project/config/application.properties (file:./config/application.properties)

2> Under Project (Only):--

## Project/application.properties (file: ./application.properties)

3> Under Project (Under resources/config)---

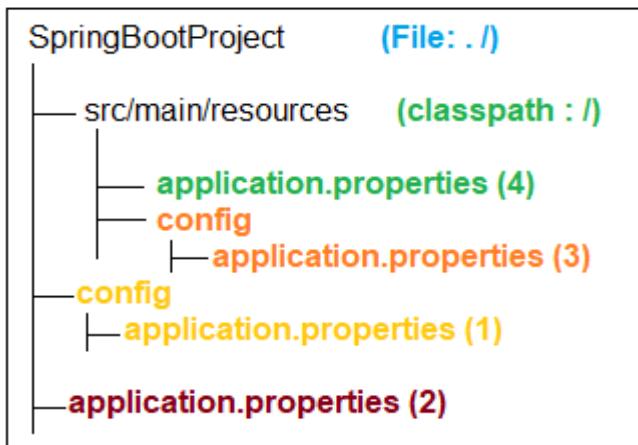
Project/src/main/resources/config/application.properties

(classpath:/config/application.properties)

3> Under “Project” folder:-

## Project/src/main/resource/application.properties

(classpath:/application.properties)



Specify Programmer File name for application.properties (even YAML)

=>Spring Boot supports programmer defined Properties (YAML) file name.

=>Which can be placed in any of 4 locations given as before (priority order is applicable if same file exist in all places).

**Step#1:-** Create your file under one location

Ex:-- src/main/resources

  |-mydata.properties

(or any other location also valid)

**Step#2:-** Use Run Configuration and provide option argument => apply and Run

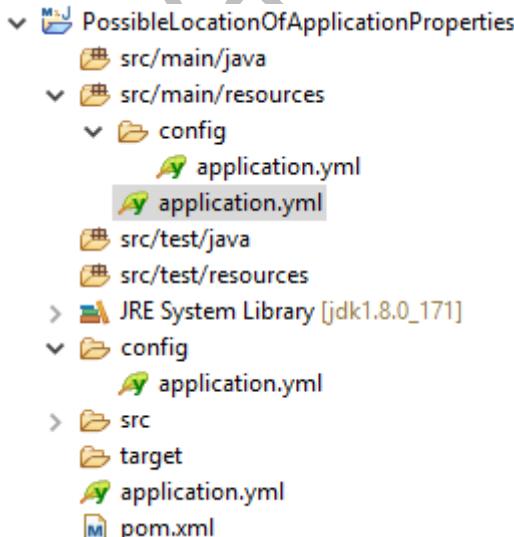
Ex:--     s--spring.config.name=mydata (user defined)

NOTE:-- To avoid default location select priority order and select exact Properties file use option argument:

Ex#1:-- --spring.config.location=classpath:/config/mydata.properties

Ex#2:-- --spring.config.location=file:./config/mydata.properties

## #5. Folder Structure of Possible application.properties/application.yml Locations:--



#### 4. YAML (YAMLian Language):--

=>It is representation style of key=val without duplicate levels in keys if they are lengthy and having common levels.

=>File extension is “.yml”.

=>It will hold data in below format

key : <space> value

=>Default name used in Spring boot is application.yml.

=>At least one space must be used but same should be maintaining under same level.

=>Spring Boot System converts .yml to .properties using SnakeYaml API.

=>Snake YAML will

a>Check for space and prefix levels

b.>Trace keys for data find.

c>Convert .yml to .properties internally system is while loading.

=>Consider below example properties file

-=-= “application.properties” -=-=

Ex#1:-

my.data.id-num=10

my.data.core\_val=AB

my.data.enabled.costBit=3.6

my.data.enabled.valid=true

Ex#2:--

my.code.id=56

my.code.mdn.type=big

my.code.str.service=ALL

my.code.str.service.info=true

my.code.mdn.obj=CRL

my.code.cost=3.67

my.code.mdn.sale=YES

=>Its equal YAML file looks as

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

my:

  code:

    id: 56

cost: 3.67

mdn:

type: big

obj: CRL

sale: YES

str:

service: ALL

info: true

=>Key=value format List<DataType>/Set<DataType>/Array(<DataType>[]) Style:--

=>In properties file we can use from zero.

=>In yml file use just dash (-) with <space> value under same level.

Ex:-- application.proeprties---

my.code.version[0]=V1

my.code.version[1]=V2

my.code.version[2]=V3

---application.yml:---

my:

code:

version:

-V1

-V2

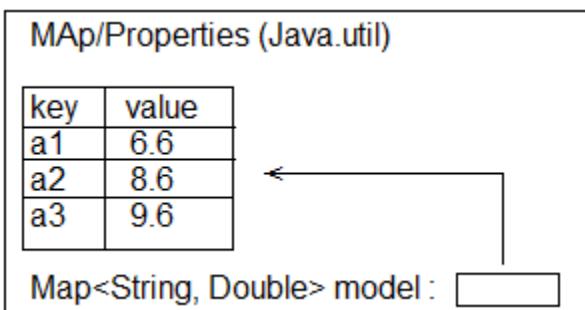
-V3

=>final meaning is:

V1	0 List<String>
V2	1 Set<String>
V3	2 String []

=>Key=value format Map/Properties Style:--

=>Consider below example:--



=> Its equal properties file will be

Ex:-- application.properties

my.data.model.a1=6.6

my.data.model.a2=8.6

my.data.model.a3=9.6

=> Its equal : application.yml file

my:

  data:

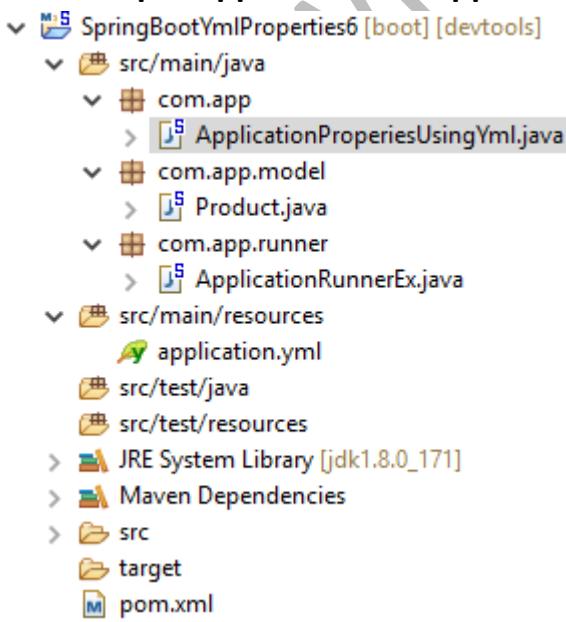
    model:

      a1: 6.6

      a2: 8.6

      a3: 9.6

## #6. Example Application for application.yml properties:--



**1>pom.xml**

=>Add one extra dependency for auto detection of keys in properties/yml file.

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

**2.>Write starter class (same as before) [main method]**

```
package com.app;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
@SpringBootApplication
public class ApplicationPropertiesUsingYml {
    public static void main(String[] args) {
        SpringApplication.run(ApplicationPropertiesUsingYml.class, args);
    }
}
```

**3.>application.yml:--**

#Normal Data

my:

```
prod:
  id: 5
  code: AB
  cost: 4.5
```

#List Data

```
version:
  -V1
  -V2
  -V3
```

#Map Data

```
model:
  a1: 6.6
  a2: 8.6
  a3: 9.6
```

4>Write Model class (Product.java):--

```
package com.app.model;
import java.util.List;
import java.util.Map;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;

@ConfigurationProperties("my.prod")
@Component
public class Product {
    private int id;
    private String code;
    private double cost;
    private List<String> version;
    private Map<String, Double> model;

    public Product() {
        super();
    }
    public Product(int id) {
        super();
        this.id = id;
    }
    public Product(int id, String code, double cost, List<String> version,
Map<String, Double> model) {
        super();
        this.id = id;
        this.code = code;
        this.cost = cost;
        this.version = version;
        this.model = model;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
```

```
        this.id = id;
    }
    public String getCode() {
        return code;
    }
    public void setCode(String code) {
        this.code = code;
    }
    public double getCost() {
        return cost;
    }
    public void setCost(double cost) {
        this.cost = cost;
    }
    public List<String> getVersion() {
        return version;
    }
    public void setVersion(List<String> version) {
        this.version = version;
    }
    public Map<String, Double> getModel() {
        return model;
    }
    public void setModel(Map<String, Double> model) {
        this.model = model;
    }
    @Override
    public String toString() {
        return "Product [id=" + id + ", code=" + code + ", cost=" + cost + ", version=" + version + ", model=" + model+ "]";
    }
}
```

##### 5. Runner class (ApplicationRunnerEx implements.java):--

```
package com.app.runner;
import java.util.Arrays;
import org.springframework.beans.factory.annotation.Autowired;
```

```

import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.ApplicationRunner;
import org.springframework.stereotype.Component;
import com.app.model.Product;

@Component
public class ApplicationRunnerEx implements ApplicationRunner {
    @Autowired
    private Product prod;
    @Override
    public void run(ApplicationArguments args) throws Exception {
        System.out.println(Arrays.asList(prod));
    }
}

```

**Output:-**

```

Console Progress Problems
<terminated> SpringBootApplicationProperties - ApplicationPropertiesUsingYml [Spring Boot App] C:\Program Files\Java\jdk1.8.0_111
.
.
.
:: Spring Boot ::      (v2.1.2.RELEASE)

2019-02-24 01:51:45.530  INFO 13548 --- [ restartedMain] com.app.ApplicationPropertiesUsingYml
2019-02-24 01:51:45.546  INFO 13548 --- [ restartedMain] com.app.ApplicationPropertiesUsingYml
2019-02-24 01:51:45.687  INFO 13548 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor
2019-02-24 01:51:47.124  INFO 13548 --- [ restartedMain] o.s.b.d.a.OptionalAllLiveReloadServer
2019-02-24 01:51:47.182  INFO 13548 --- [ restartedMain] com.app.ApplicationPropertiesUsingYml
[Product [id=5, code=AB, cost=4.5, version=[-V1 -V2 -V3], model={a1=6.6, a2=8.6, a3=9.6}]]

```

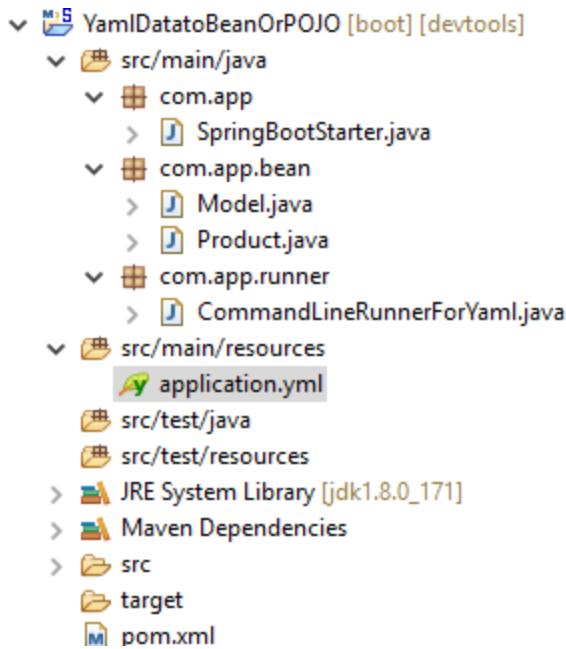
**Yaml Data to Bean/POJO:-**

=>At a time multiple values (key pair) can be converted to one POJO (Java Bean/Spring Bean) using @ConfigurationProperties annotation.

**POJO Rules :-**

- 1>Class, variable \*\*default constructor with set/get methods.
- 2>Java Bean :-- POJO Rules + Inheritance + Special methods + Param constructor.
- 3>Spring Bean:-- OJO Rules + Inheritance (Spring API) + Annotations (Spring API) + Special methods (Object class and Spring API) [toString()...]

## #7. Folder Structure for providing Yaml data to Bean / POJO:--



Example Code:--

**#1 Starter class + pom.xml:--** Same as before

@SpringBootApplication

**public class** SpringBootStarter {

**public static void** main(String[] args) {

        SpringApplication.run(SpringBootStarter.class, args);

        System.out.println("Hello Spring Boot");

    }

}

**#2. application.yml:--**

my:

    dt:

        pid: 55

        mo:

            mid: 67

            mcode: ABC

        colors:

            -RED

            -GREEN

            -YELLOW

## #3. Model class #1(Child ) Model--

```
package com.app.bean;
import java.util.List;
import org.springframework.stereotype.Component;
@Component
public class Model {
    private int mid;
    private String mcode;
    private List<String> colors;

    public Model() {
        super();
    }
    public int getMid() {
        return mid;
    }
    public void setMid(int mid) {
        this.mid = mid;
    }
    public String getMcode() {
        return mcode;
    }
    public void setMcode(String mcode) {
        this.mcode = mcode;
    }
    public List<String> getColors() {
        return colors;
    }
    public void setColors(List<String> colors) {
        this.colors = colors;
    }
    @Override
    public String toString() {
        return "Model [mid=" + mid + ", mcode=" + mcode + ", colors=" + colors + "]";
    }
}
```

## #4. Model class(Parent) Product :--

```
package com.app.bean;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;

@ConfigurationProperties("my.dt")
@Component
public class Product {
    private int pid;

    @Autowired
    private Model mo; //HAS-A

    public Product() {
        super();
    }

    public int getPid() {
        return pid;
    }

    public void setPid(int pid) {
        this.pid = pid;
    }

    public Model getMo() {
        return mo;
    }

    public void setMo(Model mo) {
        this.mo = mo;
    }

    @Override
    public String toString() {
        return "Product [pid=" + pid + ", mo=" + mo + "]";
    }
}
```

## #5. Runner class (CommandLineRunnerForYaml):--

```

package com.app.runner;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;
import com.app.bean.Product;

@Component
public class CommandLineRunnerForYaml implements CommandLineRunner {
    @Autowired
    private Product pob;
    public void run(String... args) throws Exception {
        System.out.println(pob);
        System.out.println("Hello Application Runner");
    }
}

```

## Output:--

```

:: Spring Boot ::      (v2.1.2.RELEASE)

2019-02-24 02:01:59.755  INFO 7156 --- [ restartedMain] com.app.SpringBootStarter
2019-02-24 02:01:59.770  INFO 7156 --- [ restartedMain] com.app.SpringBootStarter
2019-02-24 02:01:59.927  INFO 7156 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor
2019-02-24 02:02:01.800  INFO 7156 --- [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer
2019-02-24 02:02:01.866  INFO 7156 --- [ restartedMain] com.app.SpringBootStarter
Product [pid=55, mo=Model [mid=67, mcode=ABC, colors=[-RED -GREEN -YELLOW]]]
Hello Application Runner

```

## Place Holder Process in yml or properties:--

=>Internal place holders are used to re-use existed key value for another key as a part or full.

=>Read as \${fullPathKey} (it must be properties style even in yml file)

**application.properties:**--

```
my.dt.pid=68  
my.dt.mid=${my.dt.pid}
```

**application.yml:**--

```
my:  
  dt:  
    pid: 68  
    mid: ${my.dt.pid}
```

**NOTE:**--

- 1.>Symbol '#' indicates a comment line in yml file.
- 2.>Using 3 dash (---) symbols in yml is divided into multiple files internally (mainly used for profiles\*\*)

Ex:-- application.yml:--

```
#Hello data to Product
```

```
my:  
  dt  
    pid: 57  
---  
my:  
  dt:  
    do:  
      mid: 98
```

**NOTE:**--Priority order for .yml is same as .properties file.

3.>Search locations in order

```
a>file:./config  
b>file:./  
c>classpath:/config  
d>classpath:/
```

\*\*file:./ = Under project folder

classpath:/ = Under src/main/resources

### Priority Order for key Search:-

=>Spring boot has provided default priority to “Option Arguments” (Command Line Args).

- 1>With format –key=value
- 2>If not found, next level is .properties.
- 3>else finally chosen .yml
- 4>No-where found default value

Priority Order	Search Index
1	1. OptionArguments
2	2. Properties File a. file:./config
3	b. file:./
4	c. classpath:/config
5	d. classpath:/
6	3. YAML File
7	a. file:./config
8	b. file:./
9	c. classpath:/config
	d. classpath:/

**NOTE:**-- If no key is matched then it will give default value(Int/long=0, double=0.0, String=null), but not given any Exceptions.

### Q>What are the Difference between @ConfigurationProperties & @Value:-

Type	Spring Boot	Spring F/W
Cases	@ConfigurationProfile	@Value
1. Kebab case Ex: std-name std_name STD-NAME.. STD-name	✓	✗
2. Camel Case Ex:-- stdName stdNameModel	✓	✗
3. SpEL (Expression Language) #{Language} Ex:-- #{2+3} #{'hello'.toUpperCase()}	✗	✓

## 5. Spring Boot Profiles:--

=>In RealTime, Application is

- ❖ Developed in =>Dev Environment
- ❖ Tested in =>Quality Analyst (QA) Environment
- ❖ Maintained in =>PROD Environment
- ❖ Client tested in =>UAT Environment
- ❖ Go live in =>Cloud / prod Environment

\*\*\*>Environment is place where our application is running or what is current state of application

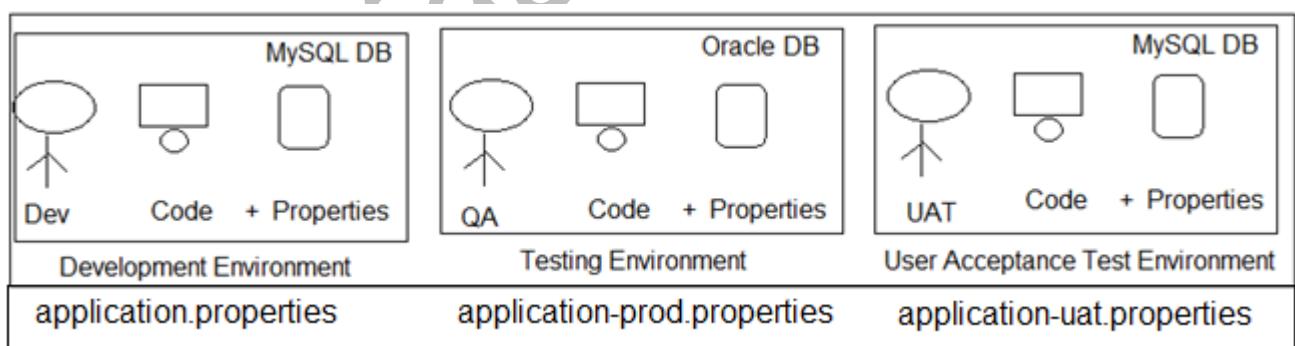
Example:-- dev= development

- ❖ QA = Quality Analysis,
- ❖ MS = Management Service,
- ❖ PROD = Production
- ❖ UAT = User acceptance Testing
- ❖ Cloud = Cloud Environment

=>In this case we should not modify existed properties file, use new one with key=val data. File naming rule is:

**application-{profile}.properties**

**application-{profile}.yml (or 3 dash)**



### Profile Specific Tasks:--

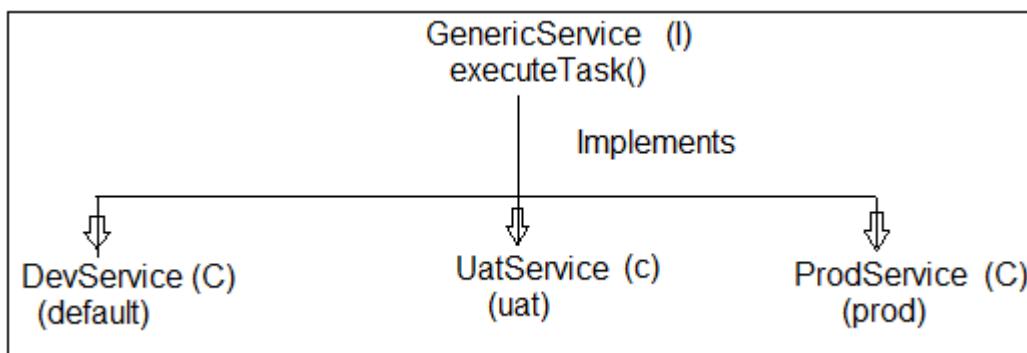
=>Profile supports Environment based Code (Task) selection, not only Properties (yaml).

=>But in this case class should have `@Profile("...")` with `@Configuration` or `@Component` (or its equal StereoType).

=>StereoType Annotations are:-- @Component, @Repository, @Service, @Controller, @RestController.

=>Consider example Profiles default Production (prod), User Acceptance Test (uat) then.

Profile Code	Properties File	Class level Annotation
Default	application.properties	@Profile("default")
Prod	application-prod.properties	@Profile("prod")
Uat	application-uat.properties	@Profile("uat")
Qa	application-qa.properties	@Profile("qa")
Cloud	application-cloud.properties	@Profile("cloud")



### Example:--

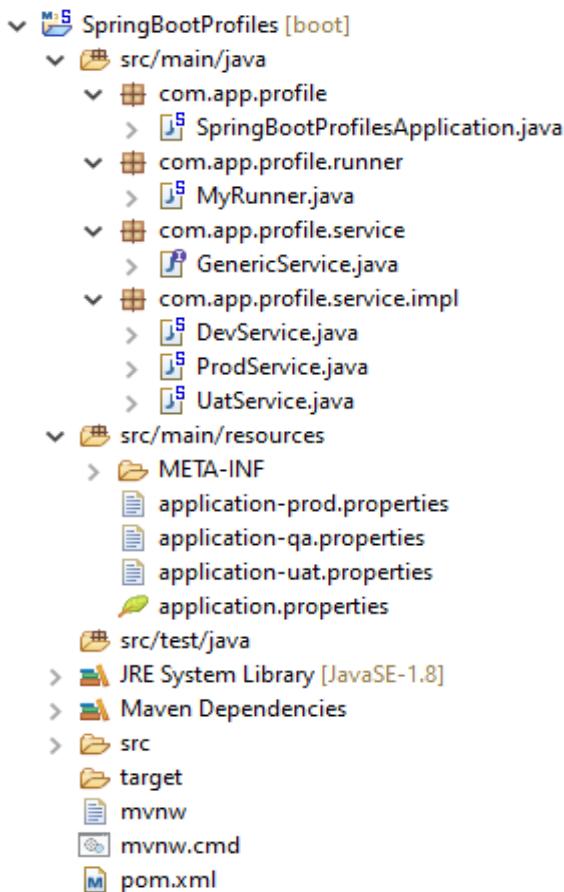
=>File => new => Spring Starter Project => Enter Details

Project Name : SpringBootProfiles and also

GroupId : org.sathyatech  
 ArtifactId : SpringBootProfiles  
 Version : 1.0

=> next => next => finish.

## #8. Folder structure of Spring Boot Profiles using application.properties:--



## 1&gt;Starter class (SpringBootProfilesApplication.java):--

```

package com.app.profile;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringBootProfilesApplication {
    public static void main(String[] args) {
        SpringApplication.run(SpringBootProfilesApplication.class, args);
        System.out.println("/**Starter class Executed**");
    }
}

```

## 2&gt;Properties files:--

a>application.properties

Ex:-- my.profile.code=Hello from default

b> application-prod.properties

Ex:-- my.profile.code=Hello from PROD

c> application-uat.properties  
Ex:-- my.profile.code=Hello from UAT  
d> application-qa.properties  
Ex:-- my.profile.code=Hello from QA

**3>service interface:--****#1 Create an Interface (GenericService.java):--**

```
package com.app.profile.service;
```

```
public interface GenericService {  
    public void executeTask();  
}
```

**4>Create Multiple classes like and implements GenericService interface:--****1. DevService.java:--**

```
package com.app.profile.service.impl;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.context.annotation.Profile;  
import org.springframework.stereotype.Component;  
import com.app.profile.service.GenericService;
```

```
@Component  
@Profile("default")  
public class DevService implements GenericService {  
    @Value("${my.profile.code}")  
    private String code;  
  
    public DevService(String code) {  
        super();  
        this.code = code;  
    }  
    public String getCode() {  
        return code;  
    }  
    public void setCode(String code) {  
        this.code = code;  
    }
```

```
@Override  
public String toString() {  
    return "DevService [code=" + code + "]";  
}  
  
@Override  
public void executeTask() {  
    System.out.println("From Dev Profiles");  
    System.out.println("code is "+code);  
}  
}
```

## 2. ProdService.java:--

```
package com.app.profile.service.impl;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.context.annotation.Profile;  
import org.springframework.stereotype.Component;  
import com.app.profile.service.GenericService;  
  
@Component  
@Profile("prod")  
public class ProdService implements GenericService {  
  
    @Value("${my.profile.code}")  
    private String code;  
  
    public ProdService() {  
        super();  
    }  
    public ProdService(String code) {  
        super();  
        this.code = code;  
    }  
    public String getCode() {  
        return code;  
    }  
    public void setCode(String code) {  
        this.code = code;  
    }
```

```
    }
    @Override
    public String toString() {
        return "ProdService [code=" + code + "]";
    }
    public void executeTask() {
        System.out.println("From Prod Profile");
        System.out.println("code is "+code);
    }
}
```

### 3. UatService class:--

```
package com.app.profile.service.impl;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Profile;
import org.springframework.stereotype.Component;
import com.app.profile.service.GenericService;
```

```
@Component
@Profile("uat")
public class UatService implements GenericService {
    @Value("${my.profile.code}")
    private String code;

    public String getCode() {
        return code;
    }
    public void setCode(String code) {
        this.code = code;
    }
    @Override
    public String toString() {
        return "UatService [code=" + code + "]";
    }
    public void executeTask() {
        System.out.println(this);
        System.out.println("From Uat Profiles");
    }
}
```

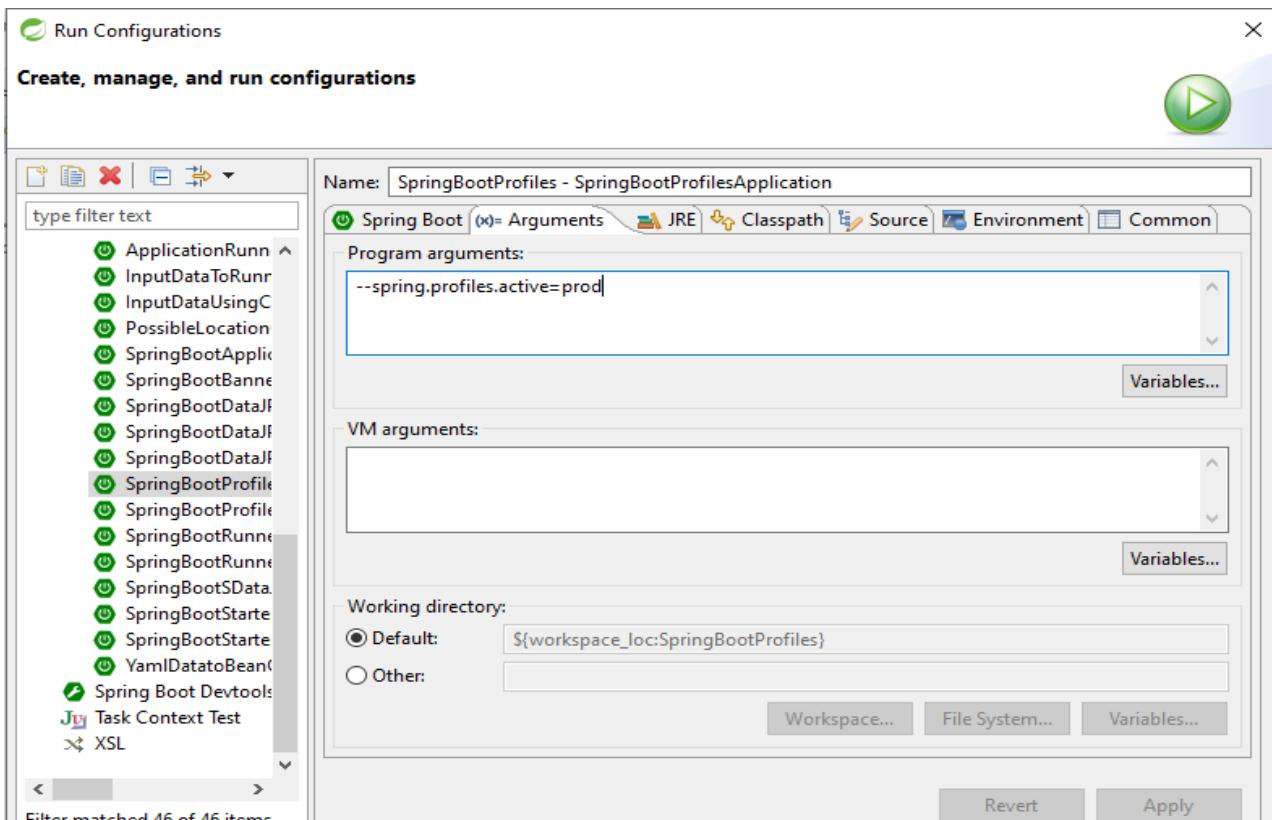
```

        System.out.println("code is "+code);
    }
}

```

### Execution of Program:--

=>Run As => Run configurations... and provide below details in Argument tab field.



### Output:--

```

Console Progress Problems Servers
<terminated> SpringBootProfiles - SpringBootProfilesApplication [Spring Boot App] C:\Program Files\Java\jdk1.8.0_171\bin\java
'   |__| .__|_|_|_\|_,| / / / /
=====|_|=====|/_=/\_/_/
:: Spring Boot ::      (v2.1.2.RELEASE)

2019-03-14 00:45:32.461  INFO 10340 --- [           main] c.a.p.SpringBootTestApplication
2019-03-14 00:45:32.477  INFO 10340 --- [           main] c.a.p.SpringBootTestApplication
2019-03-14 00:45:34.352  INFO 10340 --- [           main] c.a.p.SpringBootTestApplication
MyRunner [service=UatService [code=Hello from UAT]]
**Hello Application Runner**
Hello[--spring.output.ansi.enabled=always, --spring.profiles.active=uat]
**Starter class Executed**

```

### NOTE:--

- 1>Use key `spring.profiles.active=profile`
  - =>We can provide using 3 ways. Those are
  - a> Command Line Arguments (Option Arguments)

Ex:-- --spring.profiles.active=prod

b>In application.properties

Ex:-- spring.profiles.active=prod

b> VM (JVM/System) Argument:--

Ex:-- -Dspring.profiles.active=prod

=>Right click on Starter class => Run As=>

Run Config =>Choose Arguments

=>Enter below format in VM Arguments

-Dspring.profiles.active=prod

=>apply and Run

2>Key Search highest priority is given in same profile properties file, if key is not found in current profile properties file then goes to default properties file.

=>If no where key is found then

a>@Value generate Exception (IllegalArgumentException)

b>@ConfigurationProperties :- Default value of DataType is chosen by container.

application.properties	application-prod.properties	application-uat.properties
A=10	A=15	A=30
B=7	B=20	
C=6		

**Case#1** spring.profiles.active=prod then

Key	Value
A	15
B	20
C	6
D	0 ( for int type default value)

**Case#2** spring.profiles.active=uat

Key	Value
A	30
B	7
C	6
D	0 (for int type default value)

**Including Child Profiles:--**

=>In spring Boot applications active profile can be specified using key

Ex: --spring.profiles.active=[----]

=>In this case one properties file is loaded into memory which may have more set of key=value pairs.

=>These can be divided into multiple child properties file and loaded through active profile, also called as “Profiles Include”.

=>This can be done using key spring.profiles.include=-,-,-,-

spring.profiles.active=prod

main	child#1
application-prod.properties my.id=10	application-pemail.properties host=a
spring.profiles.include=pemail,db	application-db.properties dcn=x host=b

child#2

**Spring Container will load:--**

=>Parent (main) Profiles first (all its key=value pairs)

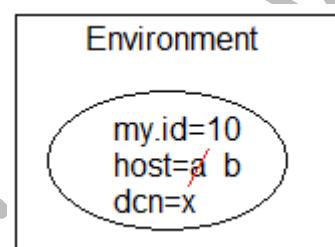
=>Then child profiles in given order will be loaded.

=>For above example, priority for loading (loading order is)

a>application-prod.properties

b>application-pemail.properties

c>application-db.properties

**Profiles using YAML:--**

=>YAML Files also works same as Properties file for both “active and include” profiles.

=>File Naming Rule:-

application-{profile}.yml

**Example:--**

application.yml (default)

application-prod.yml (prod)

application-uat.yml (uat)

**Multiple Profiles using one YAML:--**

=>YAML File supports using writing multiple profiles in one file using symbol 3 dash.

**application.yml:--**

```
my:  
  profile:  
    id: 666  
---  
my:  
  profile:  
    id: 999  
spring:  
  profiles: prod  
---  
my:  
  profiles:  
    id: 888  
spring:  
  profiles: uat
```

**NOTE:--**

#1 To specify active and include profiles use

**a>Option Arguments:--**

```
--spring.profiles.active=prod  
--spring.profiles.include=prodemail
```

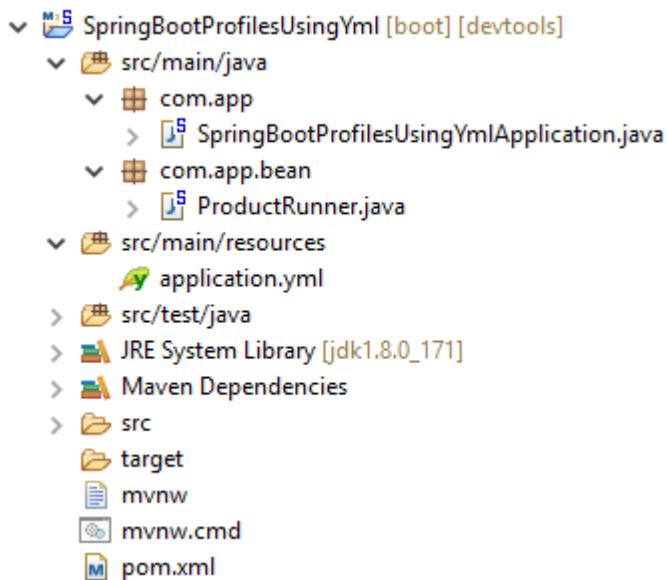
**b>use Properties file:--**

```
spring.profiles.active=prod  
spring.profiles.include=prodemail
```

**c>use YAML file:--**

```
spring:  
  profiles:  
    active: prod  
    include:  
      -prodemail
```

## #9. Folder Structure of include &amp; active properties in Profiles using Yml:--

**1>Starter and pom.xml same as before:--**

```
package com.app;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
@SpringBootApplication
public class SpringBootProfilesUsingYmlApplication {
    public static void main(String[] args) {
        SpringApplication.run(SpringBootProfilesUsingYmlApplication.class, args);
    }
}
```

**2>application.yml:--**

```
my:
  profile:
    id: 666
spring:
  profiles:
    active: prod
    include:
      -prodemail
```

**#Production Profiles****---**

```
my:
  profile:
```

```
    id: 999
spring:
  profiles: prod
#Uat profiles
---
my:
  profile:
    id: 888
spring:
  profiles: uat
#Prodemail profile
---
my:
  profile:
    email: udaykumar0023@gmail.com
spring:
  profile: prodemail
```

### 3.>Bean and Runner class (ProductRunner.java):--

```
package com.app.bean;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;

@Component
@ConfigurationProperties("my.profile")
public class ProductRunner implements CommandLineRunner {

    private int id;
    private String email;
    public ProductRunner() {
        super();
    }
    public int getId() {
        return id;
    }
}
```

```
public void setId(int id) {  
    this.id = id;  
}  
public String getEmail() {  
    return email;  
}  
public void setEmail(String email) {  
    this.email = email;  
}  
@Override  
public String toString() {  
    return "ProductRunner [id=" + id + ", email=" + email + "]";  
}  
public void run (String... args)throws Exception{  
    System.out.println(this);  
}  
}
```

## **Output:-**

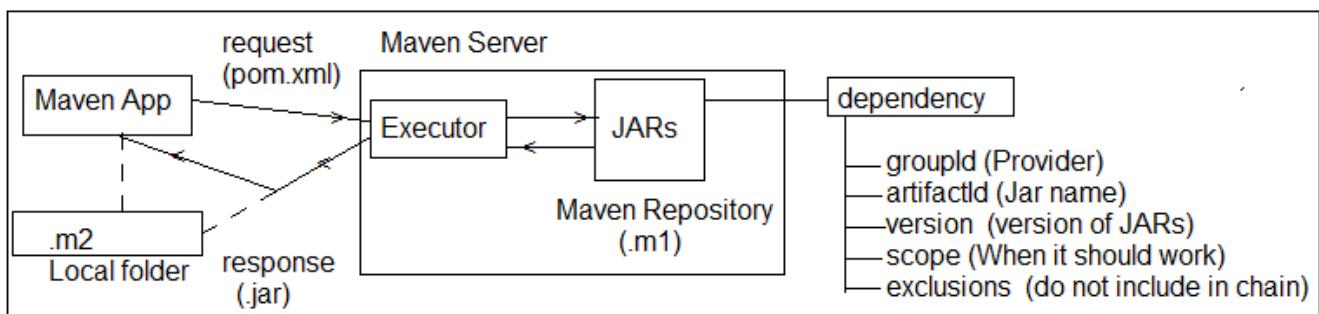
## 6. pom.xml (Maven Process) : --

=> Maven is Dependency management and build tool used to handle both stand alone and Archetype (web, restful...) applications.

## **Dependency Management:-**

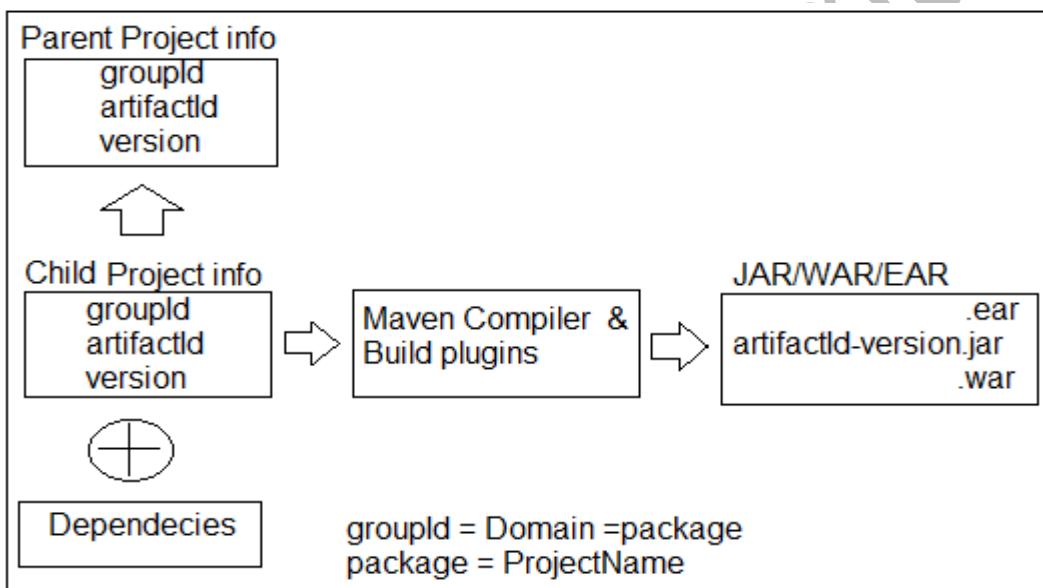
Getting Main Jars and its child jars with version support (without conflicts) into project workspace is called as Dependency Management.

**Build**-- Converting our application into final JAVA executable format i.e .jar/.war/.ear.



### Major components of pom.xml:--

- 1>Current Project
- 2>Parent Project
- 3>Dependencies
- 4>Build plugins



### pom.xml format:--

```

<project....>
<modelVersion>4.0.0</modelVersion>
<!-- CUrrent Project info -->
  <groupId>a.l</groupId>
  <artifactId>HelloApp</artifactId>
<version>1.0</version>

```

```
<!-- Parent Project info -->
<parent>
    <groupId>a.a</groupId>
    <artifactId>DBApp</artifactId>
    <version>6.1</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>
<!-- Current Project info -->
<properties>
    <java.version>1.8</java.version>
</properties>
<!-- Project JARs Details -->
<dependencies>
    <dependency>
        <groupId>a.b</groupId>
        <artifactId>Web-Test</artifactId>
        <version>5.6</version>
        <scope>compile</scope>
        <exclusions>
            <exclusion>
                <groupId>xyz.com</groupId>
                <artifactId>Web.abc</artifactId>
            </exclusion>
        </exclusions>
    </dependency>
</dependencies>
<!-- Plugins Details -->
<build>
    <plugins>
        <plugin>
            <groupId>org.maven..</groupId>
            <artifactId>maven-compiler</artifactId>
        </plugin>
    </plugins>
</build>
</project>
```

**Dependency exclusions:--**

=>When <dependency> tag is added in pom.xml then it will download Parent jars and all its child jars also.

=>To avoid any one or more child jars from chain, use concept called exclusion.

Syntax:--

```
<dependencies>
  <dependency>
    <groupId>..</groupId>
    <artifactId>..</artifactId>
    <version>..</version>
    <exclusions>
      <exclusion>
        <groupId>..</groupId>
        <artifactId>..</artifactId>
      </exclusion>
    </exclusions>
  </dependency>
</dependencies>
```

Ex:--

```
<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.1.5</version>
    <exclusions>
      <exclusion>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
      </exclusion>
    </exclusions>
  </dependency>
</dependencies>
```

**Scope (<scope> </scope> in dependency):--**

=>For every dependency one scope is given by maven i.e. default scope : **compile**.

=>This tag is optional and indicates when a JAR should be used/loaded.

**POM format:--**

```
<dependency>
    <groupId>...</groupId>
    <artifactId>...</artifactId>
    <scope>....</scope>
</dependency>
```

**Possible Maven dependency scopes are (5) :--**

**1>compile**:-- A jar Required from compilation time onwards. It is only default scope.

**2>runtime** :-- A jar required when we are running an Application, not before that.

**3>test** :-- A Jar required only for UnitTesting time.

**4>provided** :-- A jar provided by servers or Frameworks (Container....).

**5>system**:-- A Jar loaded from File System (like D:/abc/myjar/...)

=>In this case we should also give <SystemPath> with location of JAR.

Ex:-- <systemPath>D:/ASF/lib/</systemPath>

**NOTE:--** There is a dependency jar which not existing in the maven centre but locally. After mvn clean install, this dependency jar can't be found from the fat jar. Is this an known-issue? the workaround is have to install it into local maven repo with command:  
`mvn install:install-file -Dfile=lib/routines.jar -DgroupId=org.talend -  
-DartifactId=routines -Dversion=1.0 -Dpackaging=jar`

=>Then using normal dependency format in the pom.xml like this:

```
<dependency>
    <groupId>org.talend</groupId>
    <artifactId>routines</artifactId>
    <version>1.0</version>
</dependency>
```

**Format of Scope:--**

```
<dependencies>
  <!-- Compiler time (Default) Execution -->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter</artifactId>
    <scope>compile</scope> <!-- Optional -->
  </dependency>
  <!-- Runtime Execution -->
  <dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
  </dependency>
  <!-- Test time Execution -->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
  <!-- framework or container time Execution -->
  <dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>4.0.0</version>
    <scope>provided</scope>
  </dependency>
  <!-- From local system Execution -->
  <dependency>
    <groupId>routines</groupId>
    <artifactId>routines</artifactId>
    <version>1.0</version>
    <scope>system</scope>
    <systemPath>${basedir}/lib/routines.jar</systemPath>
  </dependency>
</dependencies>
```

**Maven Goals Execution:--**

**1>Maven clean:--** It is used to clear target folder in maven project. i.e delete all old files from target.

**2>Maven install :--** It will download all required plugins and also  
=>compile the source files.  
=>load required properties.  
=>Execute JUnit Test cases.  
=>Create final build (.jar/.war).

**#1. clean :--**

=>right click on project => Run As => Maven clean

**#2. install:--**

=>right click on project => Run As => Maven install.

**#3. Build:--**

=>right click on project => Run As => maven build... =>provide goals like clean install  
=> Also choose skipTests =>Apply and Run.

=>Update JDK to project before install or build else “BUILD FAILED” Error will be displayed.

=>A final jar will be created with same format “**artifactId-version.jar**”

=>Maven Build Plugin (integrated with Spring Boot) must be provided in pom.xml.

Ex:--

```
<plugin>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-maven-plugin</artifactId>
</plugin>
```

**Version Management in Spring Boot Application:--**

=>For all required dependencies (mostly used Spring Boot Parent Project provided fixed and stable version management.

=>To see all jars provided with what version,

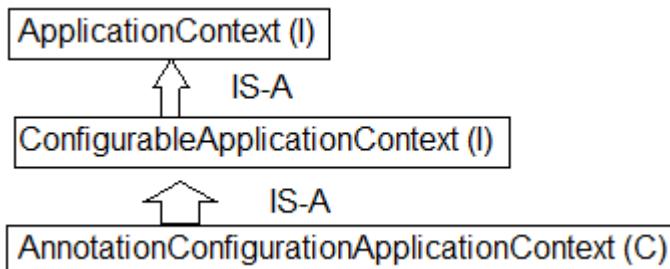
- a.>Goto pom.xml
- b.>Ctrl + Mouse over <artifactId> then click
- c.>Search for tag properties

## 7. Spring Boot Starter class Concepts:--

=>A revolving period of upto two years subject to certain amortization events.

=>Spring Boot Starter class uses run(..) method from class "**SpringApplication**"

defined in package : **org.springframework.boot** which creates Spring container using "**AnnotationConfigApplicationContext (C)**"



=>In case of Web/WebServices, Container is created using classes

**"AnnotationServletWebServerApplicationContext (C)"**.

### NOTE:--

1>ApplicationContext can be customized even using its supportive methods and API Types.

Ex:--

a.>For banner use Banner.Mode.Property, Banner.Mode.OFF (to turn off banner)

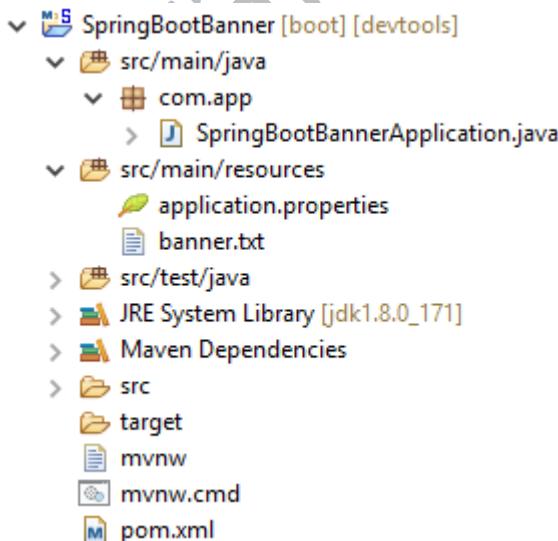
b.>Mode is Inner enum defined in functional Interface "Banner".

c.>We can provide our own banner using file : banner.txt (Create under classpath)

i.e : src/main/resource/banner.txt (file)

(<https://devops.datenkollektiv.de/banner.txt/index.html>)

## 10. Folder Structure of Banner in Spring Boot:--



**1>SpringBootBannerApplication.java:--**

```

package com.app;
import org.springframework.boot.Banner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ConfigurableApplicationContext;

@SpringBootApplication
public class SpringBootBannerApplication
{
    public static void main(String[] args)
    {
        SpringApplication sa = new SpringApplication (SpringBootBannerApplication.class);
        //sa.setBannerMode(Banner.Mode.OFF); //to Disable the banner
        //sa.setBannerMode(Banner.Mode.CONSOLE); //to Disable the banner on console
        sa.setBannerMode(Banner.Mode.LOG); //to Display the banner in Log file
        //some other configuration
        ConfigurableApplicationContext c = sa.run(args);
        System.out.println(c.getClass().getName().toString());
    }
}

```

**2>Banner.txt file:--**
**Output with Banner:--**

```

2019-02-28 01:06:19.113  INFO 18596 --- [ restartedMain] o.s.boot.SpringApplication : Starting SpringBootBannerApplication on DESKTOP-CH8LPUH with PID 18596 (started by user)
2019-02-28 01:06:19.113  INFO 18596 --- [ restartedMain] o.s.boot.SpringApplication : No active profile set, falling back to default profiles: default
2019-02-28 01:06:19.645  INFO 18596 --- [ restartedMain] e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'spring.devtools.add-properties' to 'false' to disable
2019-02-28 01:06:21.785  INFO 18596 --- [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port 35729
2019-02-28 01:06:21.916  INFO 18596 --- [ restartedMain] com.app.SpringBootBannerApplication : Started SpringBootBannerApplication in 3.818 seconds (JVM running for 7.648)
org.springframework.context.annotation.AnnotationConfigApplicationContext

```

**Use of Starter class:--**

1>Define Spring Container.

=>Spring container holds all required beans (Objects), this is created using Impl class.

**AnnotationConfigApplicationContext (C)** for simple (non-server) based application.

=>For server based Application, Impl class is :

**AnnotationConfigServletWebServerApplicationContext (C).**

2> Here “Starter class Package” behaves as basePackage, if nothing is provided by programmer.

=>If Programmer writes externally @ComponentScan then Starter class package never taken as basePackage.

=>Spring Boot starter class package behaves as base package for componentScan of all classes having annotated with @Component [or its equal].

=>Annotations are : (class should have any one)

- a>@Component
- b>@Repository
- c>@Service
- d>@Controller
- e>@RestController
- f>@Configuration

**Consider below starter class:--**

```
package com.app;
@SpringBootApplication
//{@ComponentScan("com.app") -->Added by Spring Boot
public class MyStarter {.....}
```

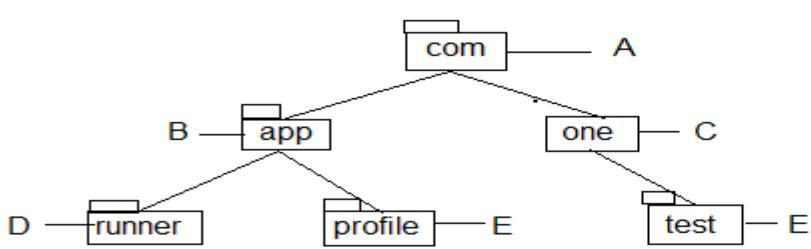
=>In this case only classes under package “app” and its sub package classes are detected by Spring (Boot) Container by default.

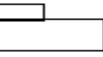
=>Programmer can provide externally basePackage using @ComponentScan

a>Avoid Starter class package and provide our own.

Ex:-- @ComponentScan (“com.app”)

=>In this case Starter package classes are not included.



package => 

Starter class pack = com.app

Selected classes B,D,E

b>Provide our own starter package (even other packages) using array style {,-,-,-,-}

Ex:-- @ComponentScan({"com.app","com.one","com"})

c>We can provide one common package name which covers all sub-levels.

Ex:-- @ComponentScan("com")

### Example:--

```
package com.app;
```

```
@SpringBootApplication
//@ComponentScan("com.one")
@ComponentScan("com")
//@ComponentScan({"com.one", "com", "com.app"})
public class MyStarter {
    public static void main (String[] args)
    {
        SpringApplication s = new SpringApplication(AppStarter.class);
        ConfigurableApplicationContext ac = s.run(args);
        System.out.println(ac.getClass().getName());
        System.out.println(ac.getBean("Product"));
        System.out.println(ac.getBean("Info"));
    }
}
```

3.> Every Spring Boot Application Starter class itself Component. i.e

@SpringBootApplication is having @Component annotation internally.

=>It is only highest Priority component by default, if app has multiple components.

Ex:- We can convert starter even as Runner.

```

package com.app;
import org.springframework.boot.CommandLineRunner;

@SpringBootApplication
public class MyStarter implements CommandLineRunner {
    public void run (String... args) throws Exception {
        System.out.println("From Starter");
    }
    public static void main(String[] args) {
        SpringApplication.run(MyStarter.class, args);
        System.out.println("***Starter class Executed***");
    }
}

```

4> Auto-Detect and Execute Configuration classes [Auto-Load Configuration files]

=>Every Spring Boot starter class itself A configuration class (@Configuration) which auto detects other Config Classes even without @Import annotation.

i.e. We can define @Bean (Objects creation in Starter class).

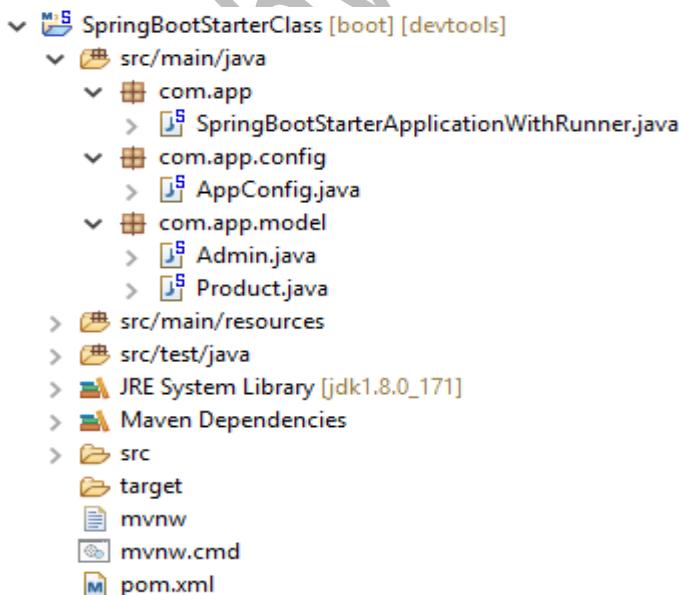
=>All Spring (Java based) Configuration files are loaded into container with

**@Configuration.**

=>All Spring (java based) Configuration files are loaded into container by spring boot if classes are annotated with @Configuration.

=>Not required to pass as ApplicationContext input (as like Spring f/w).

## 11. Folder Structure of Starter class with AutoConfiguration, import:-



**1>Model classes****a>Admin.java:--**

```
package com.app.model;

public class Admin
{
    private int adminId;
    private String adminName;

    public Admin() {
        super();
    }
    public int getAdminId() {
        return adminId;
    }
    public void setAdminId(int adminId) {
        this.adminId = adminId;
    }
    public String getAdminName() {
        return adminName;
    }
    public void setAdminName(String adminName) {
        this.adminName = adminName;
    }
    @Override
    public String toString() {
        return "Admin [adminId=" + adminId + ", adminName=" + adminName + "]";
    }
}
```

**b>Product.java:--**

```
package com.app.model;

public class Product {

    private int proId;
    private String prodName;
```

```
public Product() {
    super();
}
public int getProId() {
    return proId;
}
public void setProId(int proId) {
    this.proId = proId;
}
public String getProdName() {
    return prodName;
}
public void setProdName(String prodName) {
    this.prodName = prodName;
}
@Override
public String toString() {
    return "Product [proId=" + proId + ", prodName=" + prodName + "]";
}
}
```

**2> AppConfig.java:--**

```
package com.app.config;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import com.app.model.Admin;

@Configuration
public class AppConfig {
    @Bean
    public Admin aobj() {
        Admin a = new Admin();
        a.setAdminId(100);
        a.setAdminName("Uday");
        return a;
    }
}
```

**3>Starter class:--**

```
package com.app;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Import;
import com.app.config.AppConfig;
import com.app.model.Admin;
import com.app.model.Product;

@SpringBootApplication
//@Import (AppConfig.class) //Its not required
public class SpringBootStarterApplicationWithRunner implements
CommandLineRunner
{
    @Autowired
    private Product p;
    @Autowired
    private Admin a;

    public void run(String... args)throws Exception {
        System.out.println("From starter class :" + p);
        System.out.println("From starter class :" + a);
    }
    public static void main(String[] args) {
        SpringApplication.run(SpringBootStarterApplicationWithRunner.class, args);
        System.out.println("***Starter class main method executed");
    }
    @Bean
    public Product proj() {
        Product p = new Product();
        p.setProdId(300);
        p.setProdName("Mobile");
        return p;
    }
}
```

}

## **Output:-**

## 8. Spring Initializer:--

Link : <https://start.spring.io/>

=>This web site is used to generate one Maven (or Grade Project) for Spring Boot Apps with all configuration and setup.

Like starter class, application.properties, pom.xml, folder system etc.

=>By using this we can Create Boot App which can be imported to normal Eclipse IDE or any other equal (No STS Required).

=>Even STS (or Manual Approaches) uses internally **SPRING INITIALIZER** only.

**Step#1:-** Open Browser and type URL <https://start.spring.io/>

**Step#2:-** Provide all details and click on generate Project.

**Step#3:-** It will be downloaded as .zip, Extract this to one Folder.

**Step#4:-** Open Eclipse (or any IDE), then

>Right click on Project Explorer

>Choose Import => type maven

>select Existed Maven Project

>\*\*\*Enter/browse location of extracted folder where pom.xml is available

>Click enter => choose next/finish

# CHAPTER#2: SPRING BOOT DATA JPA

## 1. Introduction about Data-JPA:--

#1:- Data JPA provides **@NoRepositoryBean** (S) which is auto configured and self logic implemented for basic operations i.e : Programmer not required to write any logic for basic operations (No Implementation class and method).

=>Configuration for DataSource (I), SessionFactory (I), HibernateTemplate (C) Hibernate TransactionManger (C) all are not required.

=>When we add below dependency in pom.xml it will download Jars and above Config code given from parent Project.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>Spring-boot-starter-data-jpa</artifactId>
</dependency>
```

#2:- Data JPA provides “**Embedded Database Support**”. It means Database provided in application itself.

=>It is not required to download and Install, not even properties required (like driver class, url, user, password).

=>Spring Boot supports 3 Embedded DBs. Those are : H2, **HSQLDB, Apache Derby**.

=>We can use any one Embedded Database which runs in RAM (Temp memory).

=>It uses hbm2ddl.auto=create-drop i.e Tables created when App starts and deleted before App Stops.

=>These DBs are used in both Development and Testing Environment, but not in Production.

#3:- Spring Boot also supports Both SQL (MySQL, Oracle) and NoSQL (MongoDB) Database etc.. also.

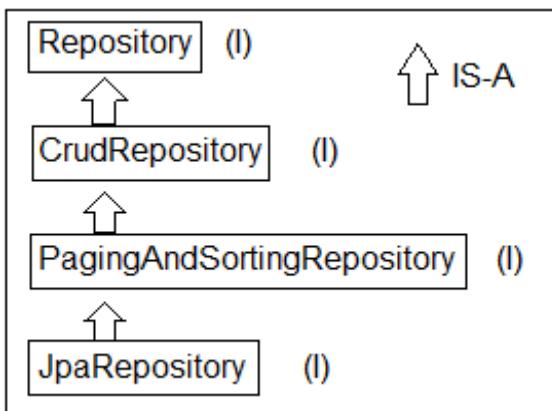
#4:- Data JPA Supports Special concept “Query Methods an easy way to code and fetch data from DB” (ex : findBy, @Query).

#5:- Data JPA supports Easy Connection Pooling (Auto Config) concept.

#6:- Data JPA supports Cache Management (AutoConfig).

### Repository Interfaces:-

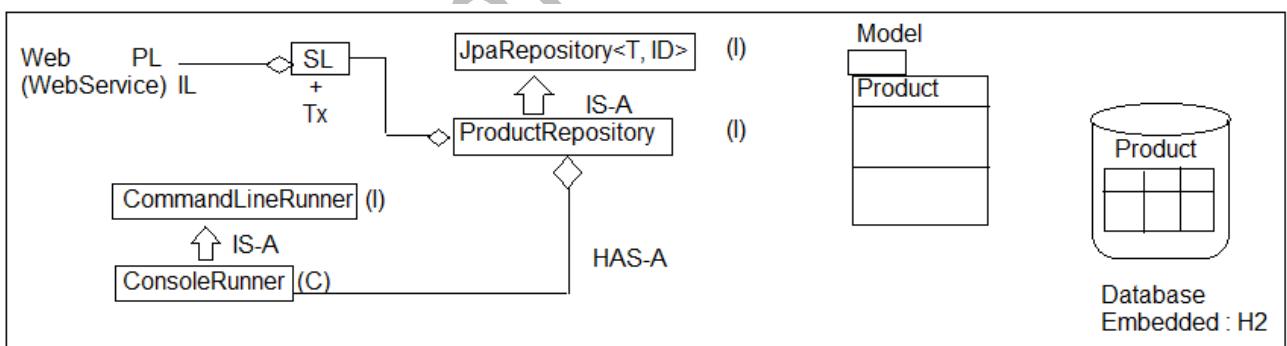
=>Data JPA has provided Repository Interfaces in package "org.springframework.data.repository".



### Spring Boot Data JPA Module Design:-

Required:

- 1> Database (Using Embedded : H2)
- 2> Model class : Product (C)
- 3> Repository : ProductRepository
- 4> Runner : ConsoleRunner



T = ? = Model class Name

ID = ? = Pk DataType = Integer

```

com.app.model
+Product
- prodId : Integer
- prodName : String
- prodCost : double
- prodColor : String
//def const
//get, set
//toString

```

PK	Prod_Tab	pName	pCost	color
pld				

=>Primary key data Type must be Wrapper class or any other classes which implements **java.io.Serializable**.

=>**Primitive Types** are not accepted as PK DataType for model & for Repository Coding.

### Eclipse Shortcuts:--

F3 => Goto code

F3 => Overview (Press again for super type also)

Ctrl +alt +DownArrow => Copy current line, paste

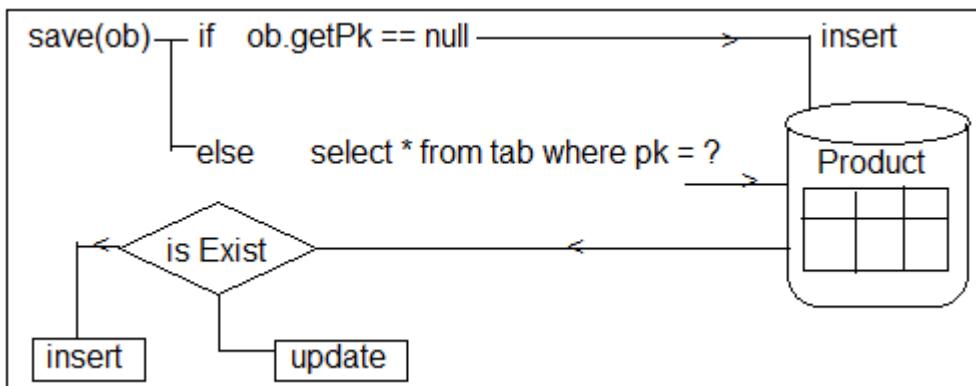
Select Lines

+ctrl + shift + / =>comment lines

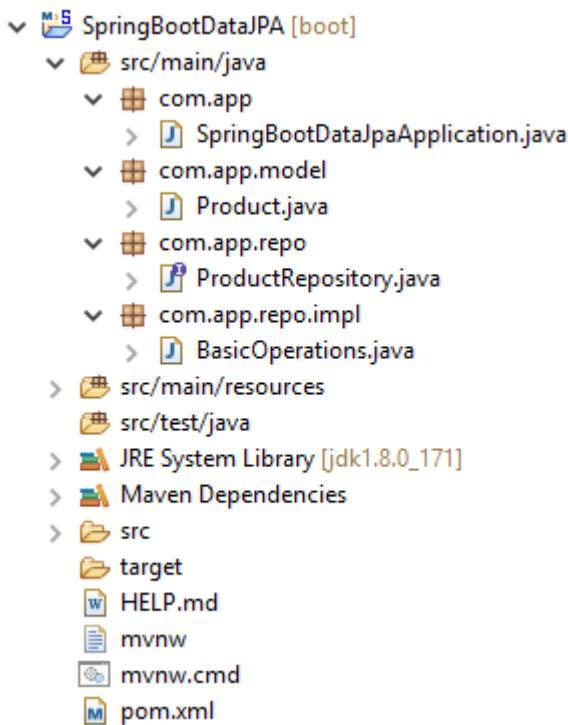
+ctrl +shift + \ =>Uncomment lines

**Save(T ob) : T** :-- This method is from CrudRepository (I) which is used to perform save or update operation.

=>If Primary Key value is Null or not exist in DB then perform insert operations, else record found in DB based on PK then performs update operation.



## #12. Folder Structure of Spring Boot Data JPA with Embedded DB H2 (Basic Operations):--



**Setup :- Create Project with dependencies H2, JPA, WEB > Finish**

**pom.xml:--**

```

<!-- spring-boot-starter-data-jpa -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

<!-- H2 Database -->
<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
  <scope>runtime</scope>
</dependency>

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

**Step#1:- Add in application.properties:--**

```
server.port=2019  
spring.jpa.show-sql=true  
spring.h2.console.enabled=true  
spring.h2.console.path=/h2
```

**Step#2:- Define model class (Product.java):--**

```
package com.app.model;  
import javax.persistence.Entity;  
import javax.persistence.GeneratedValue;  
import javax.persistence.Id;  
  
@Entity //Mandatory  
public class Product {  
  
    @Id //Mandatory  
    @GeneratedValue  
    private Integer proId;  
    private String prodName;  
    private double prodCost;  
    private String prodColor;  
  
    //super constructor  
    public Product() {  
        super();  
    }  
    //Id (PK) based constructor  
    public Product(Integer proId) {  
        super();  
        this.proId = proId;  
    }  
    //Parameterized constructor without Id(PK)  
    public Product(String prodName, double prodCost, String prodColor) {  
        super();  
        this.prodName = prodName;  
        this.prodCost = prodCost;
```

```
        this.prodColor = prodColor;
    }
    //Parameterized Constructor with Id (PK)
public Product(Integer proId, String prodName, double prodCost, String prodColor)
{
    super();
    this.proId = proId;
    this.prodName = prodName;
    this.prodCost = prodCost;
    this.prodColor = prodColor;
}
//setters & getters method
public Integer getProId() {
    return proId;
}
public void setProId(Integer proId) {
    this.proId = proId;
}
public String getProdName() {
    return prodName;
}
public void setProdName(String prodName) {
    this.prodName = prodName;
}
public double getProdCost() {
    return prodCost;
}
public void setProdCost(double prodCost) {
    this.prodCost = prodCost;
}
public String getProdColor() {
    return prodColor;
}
public void setProdColor(String prodColor) {
    this.prodColor = prodColor;
}
```

```

@Override
public String toString() {
    return "Product [prodId=" + prodId + ", prodName=" + prodName +",
prodCost=" + prodCost + ", prodColor=" + prodColor + "]";
}
}

```

**Step#3:- Write Repository Interface (ProductRepository.java):--**

```

package com.app.repo;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import com.app.model.Product;

```

```

@Repository //Optional
public interface ProductRepository extends JpaRepository<Product, Integer>
{ }

```

**Step#4:- CommandLine Runner for testing (BasicOperations.java):--**

```

package com.app.repo.impl;
import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;
import com.app.model.Product;
import com.app.repo.ProductRepository;

```

```

@Component
public class BasicOperations implements CommandLineRunner{

```

```

    @Autowired
    private ProductRepository repo;

```

```

    @Override
    public void run(String... args) throws Exception {

```

```

        /*1.*****Save******/

```

```

        //1. Method

```

```

        repo.save(new Product("PEN", 6.8, "BLUE"));

```

```

repo.save(new Product("PENCIAL", 5.8, "RED"));
repo.save(new Product("MOBILE", 5000.8, "BLACK"));
repo.save(new Product("LAPTOP", 2000.8, "GRAY"));

/*2. *****Find*****/
//2.1 method.
Optional<Product> p = repo.findById(3);
if(p.isPresent())
{
    System.out.println(p.get());
} else {
    System.out.println("No Data found");
}

//2.2 Method.
repo.findAll().forEach((System.out::println));
/*3. *****Delete*****/
//3.1 Delete by specific Id
repo.deleteById(3);
//3.2 Delete all Rows one by one in (Sequence order)
repo.deleteAll(); //Multiple Query fired No of record = no of Query
//3.3 Delete all rows in Batch (Single Query fired)
repo.deleteAllInBatch();
}
}

```

**Output:-**

```

2019-03-05 15:35:23.724 INFO 14000 --- [           main] o.hibernate.annotations.common.Version : HCANN000001: Hibernate Commons Annotations [5.0.4.Final]
2019-03-05 15:35:24.068 INFO 14000 --- [           main] org.hibernate.dialect.Dialect      : HHH000476: Executing import script 'org.hibernate.tool.schema.internal.exec.Scrip
Hibernate: drop table product if exists
Hibernate: drop sequence if exists hibernate_sequence
Hibernate: create sequence hibernate_sequence start with 1 increment by 1
Hibernate: create table product (prod_id integer not null, prod_color varchar(255), prod_cost double not null, prod_name varchar(255), primary key (prod_id))
2019-03-05 15:35:25.560 INFO 14000 --- [           main] o.h.t.schema.internal.SchemaCreatorImpl : HHH000476: Executing import script 'org.hibernate.tool.schema.internal.exec.Scrip
2019-03-05 15:35:25.576 INFO 14000 --- [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2019-03-05 15:35:26.755 INFO 14000 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
2019-03-05 15:35:26.858 WARN 14000 --- [           main] aWebConfiguration$JpaWebMvcConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be
2019-03-05 15:35:27.327 INFO 14000 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 2019 (http) with context path ''
2019-03-05 15:35:27.343 INFO 14000 --- [           main] com.app.SpringBootDataJpaApplication : Started SpringBootDataJpaApplication in 10.884 seconds (JVM running for 13.869)
Hibernate: call next value for hibernate_sequence
Hibernate: insert into product (prod_color, prod_cost, prod_name, prod_id) values (?, ?, ?, ?)
Hibernate: call next value for hibernate_sequence
Hibernate: insert into product (prod_color, prod_cost, prod_name, prod_id) values (?, ?, ?, ?)
Hibernate: call next value for hibernate_sequence
Hibernate: insert into product (prod_color, prod_cost, prod_name, prod_id) values (?, ?, ?, ?)
Hibernate: call next value for hibernate_sequence
Hibernate: insert into product (prod_color, prod_cost, prod_name, prod_id) values (?, ?, ?, ?)
Hibernate: select product0_.prod_id as prod_id1_0_0_, product0_.prod_color as prod_col2_0_0_, product0_.prod_cost as prod_cos3_0_0_, product0_.prod_name as prod_nam4_0_0_ from product
Product [prodId=3, prodName=MOBILE, prodCost=5000.8, prodColor=BLACK]
2019-03-05 15:35:27.655 INFO 14000 --- [           main] o.h.h.i.QueryTranslatorFactoryInitiator : HHH000397: Using ASTQueryTranslatorFactory
Hibernate: select product0_.prod_id as prod_id1_0_0_, product0_.prod_color as prod_col2_0_0_, product0_.prod_cost as prod_cos3_0_0_, product0_.prod_name as prod_nam4_0_0_ from product
Product [prodId=1, prodName=PEN, prodCost=6.8, prodColor=BLUE]
Product [prodId=2, prodName=PENCIAL, prodCost=5.8, prodColor=RED]
Product [prodId=3, prodName=MOBILE, prodCost=5000.8, prodColor=BLACK]
Product [prodId=4, prodName=LAPTOP, prodCost=2000.8, prodColor=GRAY]
Hibernate: select product0_.prod_id as prod_id1_0_0_, product0_.prod_color as prod_col2_0_0_, product0_.prod_cost as prod_cos3_0_0_, product0_.prod_name as prod_nam4_0_0_ from product
Hibernate: delete from product where prod_id=?
Hibernate: select product0_.prod_id as prod_id1_0_0_, product0_.prod_color as prod_col2_0_0_, product0_.prod_cost as prod_cos3_0_0_, product0_.prod_name as prod_nam4_0_0_ from product
Hibernate: delete from product where prod_id=?
Hibernate: delete from product where prod_id=?
Hibernate: delete from product where prod_id=?
Main Method call

```

Activate Windows

**Method Descriptions:--**

**1>save (obj)** :-- Behaves like save or update, If PK exist in DB table then “UPDATE” else “INSERT”.

**2>findById(ID): Optional<T>** :-- It will return one row as one Object based on Primary key in Optional <T> format.

=>use methods isPresent() to check record is exist or not? If exist use method get() : T to read object.

**3>findAll ()** :-- It returns Collection of Objects (=no Of rows in DB Table)

=>In simple select \* from tableName;

**4>deleteById(ID)** :-- To delete one Row based on PK.

**5>deleteAll()** :-- To delete all Rows [One by one row]

**6>deleteAllInBatch ()** :-- To delete All rows at a time ex: delete from <tableName>

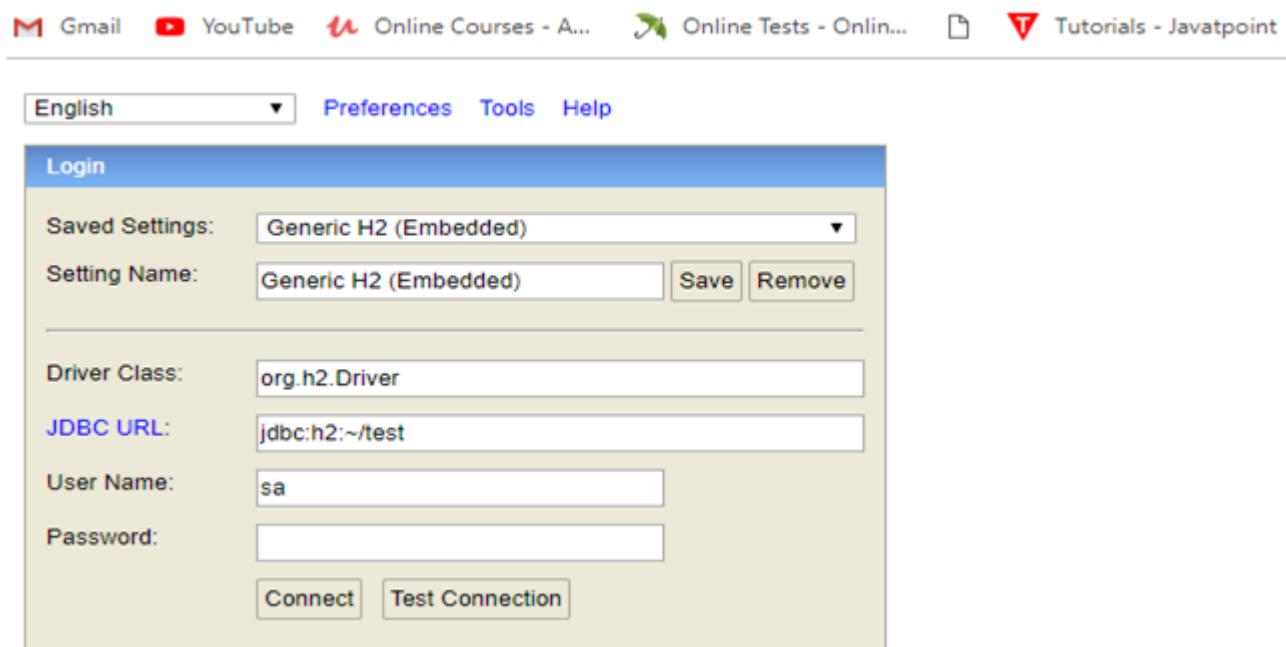
=>H2 is an Embedded database provided by SpringBoot which uses “hbm2ddl.auto=create-drop”, which means create table when server/app starts and drop all tables when server/app stopped.

=>H2 console works only if use WebApp and default path is : /h2-console, default port : 8080

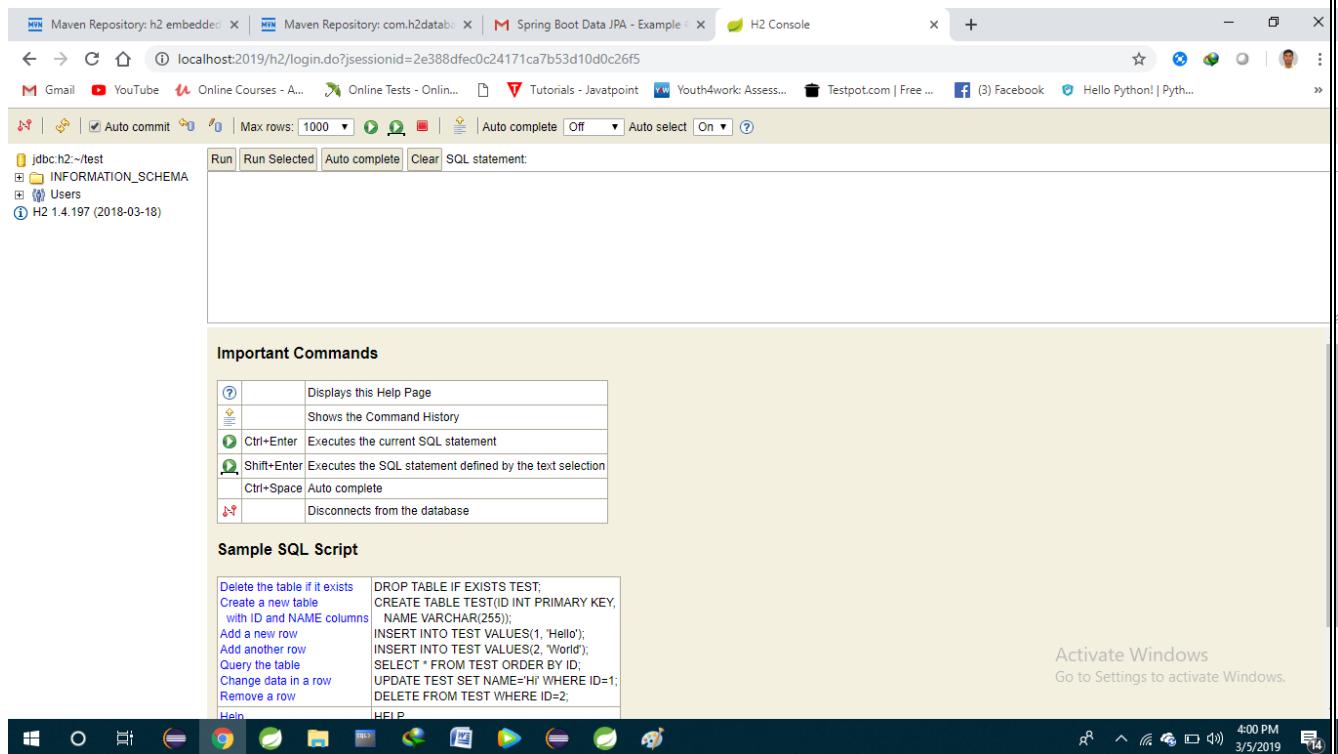
**How to Show Data in H2 DataBase:--**

**Step#1:- Execute Program:--**

**Step#2:- Type Url in Browser (<http://localhost:2019/h2>)**



=>Click on connect.

**Step #3:-****2.2 Query Method in Spring Boot Data:--**

=>Spring Data generates a query based on method written in Repository by Programmer.

**Types of Query Methods (3):--**

1>findBy

2>@Query (manual Query)

3>Special Parameters/ReturnTypes

=>These are used to specify our columns (Projections) and rows (restrictions) details.

**1>findBy :--** It will generate select query based on abstract method given by programmer. We can provide columns and rows details.

=>It will be converted to equal SQL query based on Database at runtime.

Syntax:--

RT findBy\_\_\_\_\_ (Parameters ...);

Here, RT = ReturnType, ex: List<T>, T, Object, Page<T>, Slice<T>, Object[], Specific Projection etc.

## Spring Boot Data JPA findBy methods (where clause):--

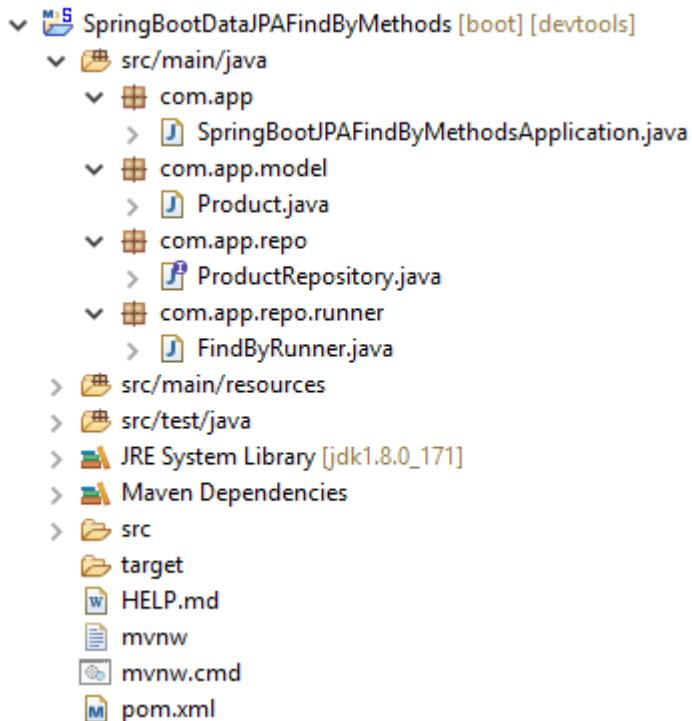
Keyword	Sample	JPQL snippet
And	findByLastnameAndFirstname	... where x.lastname = ?1 and x.firstname = ?2
Or	findByLastnameOrFirstname findByFirstname, findByFirstnamels,	... where x.lastname = ?1 or x.firstname = ?2
Is, Equals	findByFirstnameEquals	... where x.firstname = ?1 ... where x.startDate
Between	findByStartDateBetween	between ?1 and ?2
LessThan	findByAgeLessThan	... where x.age < ?1
LessThanEqual	findByAgeLessThanEqual	... where x.age <= ?1
GreaterThan	findByAgeGreaterThan	... where x.age > ?1
GreaterThanOrEqual	findByAgeGreaterThanOrEqual	... where x.age >= ?1
After	findByStartDateAfter	... where x.startDate > ?1
Before	findByStartDateBefore	... where x.startDate < ?1
IsNull	findByAgeIsNull	... where x.age is null
IsNotNull, NotNull	findByAge(Is)NotNull	... where x.age not null
Like	findByFirstnameLike	... where x.firstname like ?1
NotLike	findByFirstnameNotLike	... where x.firstname not like ?1 ... where x.firstname like ?1 (parameter bound with appended %)
StartingWith	findByFirstnameStartingWith	... where x.firstname like ?1 (parameter bound with preended %)
EndingWith	findByFirstnameEndingWith	... where x.firstname like ?1 (parameter bound wrapped in %)
Containing	findByFirstnameContaining	... where x.age = ?1 order by x.lastname desc
OrderBy	findByAgeOrderByLastnameDesc	x.lastname desc
Not	findByLastnameNot	... where x.lastname <> ?1
In	findByAgeIn(Collection<Age> ages)	... where x.age in ?1
NotIn	findByAgeNotIn(Collection<Age> ages)	... where x.age not in ?1
True	findByActiveTrue()	... where x.active = true

[Raghu Sir]

[NareshIT, Hyd]

False	findByActiveFalse()	... where x.active = false
IgnoreCase	findByFirstnameIgnoreCase	... where UPPER(x.firstname) = UPPER(?1)

### 13. Folder Structure of findBy Methods of Data JPA:--



#### 1>Starter class (SpringBootSjpaFindByMethodsApplication.java):--

```
package com.app;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringBootSjpaFindByMethodsApplication {
    public static void main(String[] args) {
        SpringApplication.run(SpringBootSjpaFindByMethodsApplication.class, args);
        System.out.println("Main Method Executed");
    }
}
```

#### 2>Model class (Product.class):--

```
package com.app.model;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
```

```
import javax.persistence.Table;

@Entity
@Table(name="product")
public class Product {
    @Id
    @GeneratedValue
    private Integer proId;
    private String prodCode;
    private String prodName;
    private double prodCost;

    //super constructor
    public Product() {
        super();
    }
    //Id (PK) based constructor
    public Product(Integer proId) {
        super();
        this.proId = proId;
    }
    //Parameterized constructor
    public Product(String prodName, double prodCost, String prodCode) {
        super();
        this.prodName = prodName;
        this.prodCost = prodCost;
        this.prodCode = prodCode;
    }
    public Product(Integer proId, String prodName, double prodCost, String prodCode)
    {
        super();
        this.proId = proId;
        this.prodName = prodName;
        this.prodCost = prodCost;
        this.prodCode = prodCode;
    }
}
```

```
//setters & getters method
public Integer getProdId() {
    return prodId;
}
public void setProdId(Integer prodId) {
    this.prodId = prodId;
}
public String getProdName() {
    return prodName;
}
public void setProdName(String prodName) {
    this.prodName = prodName;
}
public double getProdCost() {
    return prodCost;
}
public void setProdCost(double prodCost) {
    this.prodCost = prodCost;
}
public String getProdCode() {
    return prodCode;
}
public void setProdCode(String prodCode) {
    this.prodCode = prodCode;
}
@Override
public String toString() {
    return "Product [prodId=" + prodId + ", prodCode=" + prodCode + ",
prodName=" + prodName + ", prodCost=" + prodCost + "]";
}
}
```

### 3>Repository Interface (ProductRepository.class):--

=>Add below methods in Repository

```
package com.app.repo;
import java.util.Collection;
import java.util.List;
```

```
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import com.app.model.Product;

@Repository
public interface ProductRepository extends JpaRepository<Product, Integer> {

    //select * from prodtab where prod_code=prodCode;
    Product findByProdCode(String prodCode);

    //select * from prodtab where prod_code like prodCode
    List<Product> findByProdCodeLike (String pc);

    //select * from prodtab where prod_code is null
    List<Product> findByProdCodeIsNull();

    //select * from prodtab where prod_cost=cost
    List<Product> findByProdCostGreaterThanOrEqual(Double cost);

    //select * from prodtab where prod_cost in (cost)
    List<Product> findByProdCostIn(Collection<Double> costs);

    //select * from prodTab where pid=? Or pcost=?
    List<Product> findByProdIdOrProdCost(Integer pid, Double cost);

    //select * from prodtab where pid between pid1 and pid2
    List<Product> findByProdIdBetween (Integer pid1, Integer pid2);

    //select * from prodtab where p_cost=? Order by prod_code asc
    List<Product> findByProdCostLessThanOrderByProdCode(Double cost);

    //Select * from prodtab where pid<=? And pcost>=? And vendor is not null order by
    pcode;

    //List<Product>
    findByProdIdLessThanAndProdCostGreaterThanOrEqualAndVendorNotNullOrderByProdCode
    (Integer prodId, Double prodCost);
}
```

#### 4>Runner class (FindByRunner.java):--

```

package com.app.repo.runner;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Service;
import com.app.model.Product;
import com.app.repo.ProductRepository;

@Service
public class FindByRunner implements CommandLineRunner
{
    @Autowired
    private ProductRepository repo;

    @Override
    public void run(String... args) throws Exception {
        Product p = repo.findByProdCode("A");
        System.out.println(p);
        repo.findByProdCodeIsNull().forEach((System.out::println));
    }
}

```

#### Output:-

```

2019-06-23 23:56:55.191 INFO 2228 --- [ restartedMain] org.hibernate.dialect.Dialect      : HHH000400: Using dialect: org.hibernate.dialect.Oracle10gDialect
2019-06-23 23:56:56.535 INFO 2228 --- [ restartedMain] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2019-06-23 23:56:56.625 INFO 2228 --- [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer   : LiveReload server is running on port 35729
2019-06-23 23:56:57.015 INFO 2228 --- [ restartedMain] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
2019-06-23 23:56:57.062 INFO 2228 --- [ restartedMain] a.webConfiguration$JpaWebMvcConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be
2019-06-23 23:56:57.718 INFO 2228 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 2019 (http) with context path ''
2019-06-23 23:56:57.723 INFO 2228 --- [ restartedMain] a.SpringBootJPAFindByMethodsApplication : Started SpringBootJPAFindByMethodsApplication in 4.027 seconds (JVM running for 16
2019-06-23 23:56:57.724 INFO 2228 --- [ restartedMain] o.a.c.c.QueryTranslatorFactoryInitiator : HHH000397: Using ASTQueryTranslatorFactory
2019-06-23 23:56:57.755 INFO 2228 --- [nio-2019-exec-1] o.a.c.c.C.[Tomcat-1].localhost.[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2019-06-23 23:56:57.755 INFO 2228 --- [nio-2019-exec-1] o.s.web.servlet.DispatcherServlet     : Initializing Servlet 'dispatcherServlet'
2019-06-23 23:56:57.787 INFO 2228 --- [nio-2019-exec-1] o.s.web.servlet.DispatcherServlet     : Completed initialization in 32 ms
Hibernate: select product0_.prod_id as prod_id1_0_, product0_.prod_code as prod_code2_0_, product0_.prod_cost as prod_cost3_0_, product0_.prod_name as prod_name4_0_ from product prod
null
Hibernate: select product0_.prod_id as prod_id1_0_, product0_.prod_code as prod_code2_0_, product0_.prod_cost as prod_cost3_0_, product0_.prod_name as prod_name4_0_ from product prod
Product [prodId=10, prodCode=null, prodName=PEN, prodCost=10.5]
Product [prodId=20, prodCode=null, prodName=PENCIL, prodCost=5.0]
Product [prodId=30, prodCode=null, prodName=MOBILE, prodCost=700.5]
Product [prodId=40, prodCode=null, prodName=LAPTOP, prodCost=1000.5]
Product [prodId=50, prodCode=null, prodName=MOUSE, prodCost=500.5]
2019-06-23 23:56:57.833 INFO 2228 --- [ restartedMain] .ConditionEvaluationDeltaLoggingListener : Condition evaluation unchanged
Main Method Executed

```

**2>@Query("hql"):**-- This is used to perform (Hibernate Query Language) operations, works for both select and non-select operations.

=>To pass where clause inputs use positional parameters in style ?1, ?2, ?3....

Or

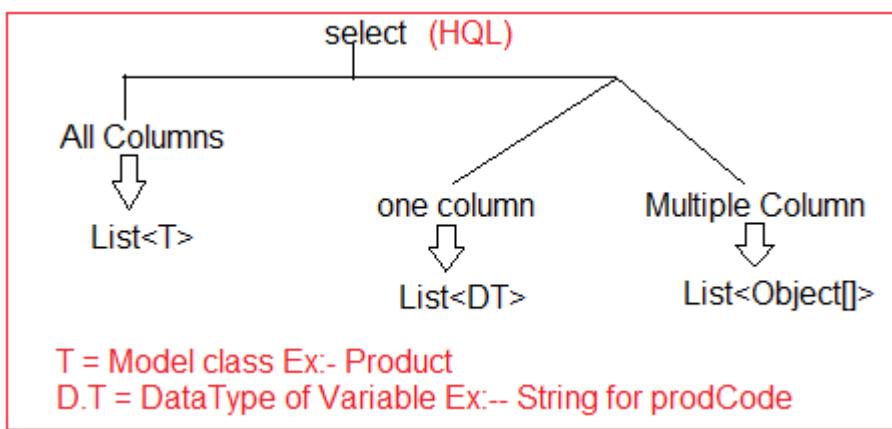
Named parameters in style [:name]

:a, :b, :c, :hello, : mydata

=>But variable name must be same as named parameter.

=>Providing package name for Model class in @Query is optional.

i.e. select p from com.app.model.Product, select p from Product (are same)



#### 14. Folder Structure of @Query param in Spring Boot:--

```

└─ SpringBootDataJPAQueryParam [boot] [devtools]
    └─ src/main/java
        └─ com.app
            ├─ Application.java
            └─ model
                └─ Product.java
        └─ repo
            └─ ProductRepository.java
        └─ repo.runner
            └─ NonSelectOperationUsingQueryParam.java
            └─ SelectOperationUsingQueryParam.java
    └─ src/main/resources
        └─ application.properties
    └─ src/test/java
    └─ JRE System Library [JavaSE-1.8]
    └─ Maven Dependencies
    └─ src
        └─ target
            └─ HELP.md
            └─ mvnw
            └─ mvnw.cmd
    └─ pom.xml

```

#### application.properties:--

```

server.port=2019
spring.datasource.driverClassName=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe
spring.datasource.username=system
spring.datasource.password=system
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.database-platform=org.hibernate.dialect.Oracle10gDialect

```

**1>Starter class (SpringBootDataJpaQueryParamApplication):--**

```
package com.app;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringBootDataJpaQueryParamApplication {
    public static void main(String[] args) {
        SpringApplication.run(SpringBootDataJpaQueryParamApplication.class, args);
        System.out.println("Main Method Executed ");
    }
}
```

**2.> Model class(Product.java):--**

```
package com.app.model;
import javax.persistence.Entity;
import javax.persistence.Id;
```

```
@Entity
public class Product {

    @Id
    private Integer proId;
    private String vendorCode;
    private String prodName;
    private Double prodCost;

    public Product() {
        super();
    }

    public Product(Integer proId) {
        super();
        this.proId = proId;
    }

    public Product(Integer proId, String vendorCode, String prodName, Double prodCost)
    {
        super();
    }
```

```
        this.prodId = prodId;
        this.vendorCode = vendorCode;
        this.prodName = prodName;
        this.prodCost = prodCost;
    }
//Setters and Getters method
public Integer getProdId() {
    return prodId;
}
public void setProdId(Integer prodId) {
    this.prodId = prodId;
}
public String getVendorCode() {
    return vendorCode;
}
public void setVendorCode(String vendorCode) {
    this.vendorCode = vendorCode;
}
public String getProdName() {
    return prodName;
}
public void setProdName(String prodName) {
    this.prodName = prodName;
}
public Double getProdCost() {
    return prodCost;
}
public void setProdCost(Double prodCost) {
    this.prodCost = prodCost;
}
@Override
public String toString() {
    return "Product [prodId=" + prodId + ", vendorCode=" + vendorCode + ",
prodName=" + prodName + ", prodCost=" + prodCost + "]";
}
}
```

**3>ProductRepository**-- Add below method in repository:--

```

package com.app.repo;
import java.util.List;
import javax.transaction.Transactional;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Modifying;
import org.springframework.data.jpa.repository.Query;
import org.springframework.stereotype.Repository;
import com.app.model.Product;

@Repository
public interface ProductRepository extends JpaRepository<Product, Integer> {

    //select * from product where ven_code=?  

    //or prodCost>?  

    @Query("select p from Product p where p.vendorCode=:a or p.prodCost>:b")  

    List<Product> getAllProducts(String a, Double b);

    //select prodCode from product where vendor vendorCode=? or prodCost=?  

    @Query("select p.vendorCode from Product p where p.vendorCode=?1 or  

    p.prodCost=?2")  

    List<String> getAllProductsCodes(String a, Double b);

    //select prodCode, prodCost from product  

    @Query("select p.vendorCode, p.prodCost from com.app.model.Product p")  

    List<Object[]> getAllProductData();

    /*****NON-Select Operation*****/  

    @Modifying //non-select operation  

    @Transactional //service Layer  

    @Query("update Product p set p.vendorCode=:a, p.prodCost=:c where  

    p.productId=:b")  

    void updateMyData(String a, Double c, Integer b);

    @Modifying //non-select operation  

    @Transactional //service Layer  

    @Query("delete from Product p where p.productId=:productId")  

    void deleteMyData(Integer productId);
}

```

#### 4.1 Select Operation Using QueryParam:--

=>CommandLineRunner code for getAllProductData method

```

package com.app.repo.runner;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;
import com.app.model.Product;
import com.app.repo.ProductRepository;

@Component
public class SelectOperationUsingQueryParam implements CommandLineRunner {

    @Autowired
    private ProductRepository repo;

    @Override
    public void run(String... args) throws Exception {
        //Normal Method
        /*repo.save(new Product(10, "A", "PEN", 10.5));
        repo.save(new Product(20, "B", "PENCIL", 50.5));
        repo.save(new Product(30, "C", "MOBILE", 700.5));
        repo.save(new Product(40, "D", "LAPTOP", 1000.5));
        repo.save(new Product(50, "E", "MOUSE", 500.5));*/

        List<Product> p = repo.getAllProducts("A", 56.98);
        for(Product p1:p) {
            System.out.println(p1);
        }
        //JDK 1.5
        List<Object[]> obs=repo.getAllProductData();
        for(Object[] ob:obs) {
            System.out.println(ob[0]+","+ob[1]);
        }
        System.out.println("For Each loop");*/
        //JDK 1.8 (Streams + Lambda+Method Reference)
        repo.getAllProductData().stream().map((ob)->
        ob[0]+","+ob[1]).forEach(System.out::println);
    }
}

```

**Output:--**

```

2019-06-24 00:11:33.709 INFO 1456 --- [ restartedMain] org.hibernate.Version : HHH000412: Hibernate Core {5.3.7.Final}
2019-06-24 00:11:33.709 INFO 1456 --- [ restartedMain] org.hibernate.cfg.Environment : HHH000206: hibernate.properties not found
2019-06-24 00:11:34.177 INFO 1456 --- [ restartedMain] org.hibernate.common.Version : HCANN000001: Hibernate Commons Annotations {5.0.4.Final}
2019-06-24 00:11:34.490 INFO 1456 --- [ restartedMain] org.hibernate.dialect.Dialect : HHH000400: Using dialect: org.hibernate.dialect.Oracle10gDialect
2019-06-24 00:11:34.490 INFO 1456 --- [ restartedMain] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2019-06-24 00:11:37.092 INFO 1456 --- [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : HHH000397: Using ASTQueryTranslatorFactory
2019-06-24 00:11:37.749 INFO 1456 --- [ restartedMain] o.h.h.i.QueryTranslatorFactoryInitiator : LiveReload server is running on port 35729
2019-06-24 00:11:38.483 INFO 1456 --- [ restartedMain] o.s.s.concurrent.ThreadPoolTaskExecutor : Tomcat started on port(s): 2019 (http) with context path ''
2019-06-24 00:11:38.593 WARN 1456 --- [ restartedMain] a.webConfiguration5JpaWebMvcConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be
Hibernate: select product0_.vendor_code as col_0_0_, product0_.prod_cost as col_1_0_ from product0_
A,10.5
B,50.5
C,700.5
D,1800.5
E,500.5
Main Method Executed
2019-06-24 00:11:39.359 INFO 1456 --- [nio-2019-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2019-06-24 00:11:39.359 INFO 1456 --- [nio-2019-exec-1] o.s.web.servlet.DispatcherServlet : : Initializing Servlet 'dispatcherServlet'
2019-06-24 00:11:39.390 INFO 1456 --- [nio-2019-exec-1] o.s.web.servlet.DispatcherServlet : : Completed initialization in 31 ms

```

Activate Windows

**@Query("") non-select operations:--**

=>To perform non-select operation define HQL(Query) and apply @Modifying and @Transactional over @Query method.

=>@Transactional can be applied over repository method or in service layer method.

**ProductRepository code:--**

@Modifying //non-select operation

@Transactional //service Layer

@Query("update Product p set p.prodCode=:a, p.prodCost=:c where p.prodid=:b")

```
void updateMyData(String a, Double c, Integer b);
```

@Modifying //non-select operation

@Transactional //service Layer

@Query("delet from Product p where p.prodid=:prodid")

```
void deleteMyData(Integer prodid);
```

**4.2 Non-Select Operation Using QueryParam:--**

=>Add respected method in controller/CommandLineRunner implemented class class.

```
repo.updateMyData("E", 900.0, 80);
repo.deleteMyData(10);
```

**3. PROJECTIONS:--**

=>By default every Query method (findBy\_) return all columns (full loading).

=>Here Projections are used to select few columns (variables).

=> Projections are two types

- ❖ Static Projections
- ❖ Dynamic Projections

**1. Static Projections:**-- This concept is used for always fixed types of columns selection for **multiple** runs (or calls).

**Steps to implements static Projections:**--

**Step#1:-** Define one child interface (inner interface) in Repository interface with any name.

(OR)

Create one public interface & use that insideRepositoryInterface as DataType

**Step#2:-** Copy variable equal getter method (getMethods()) from model class to child interface.

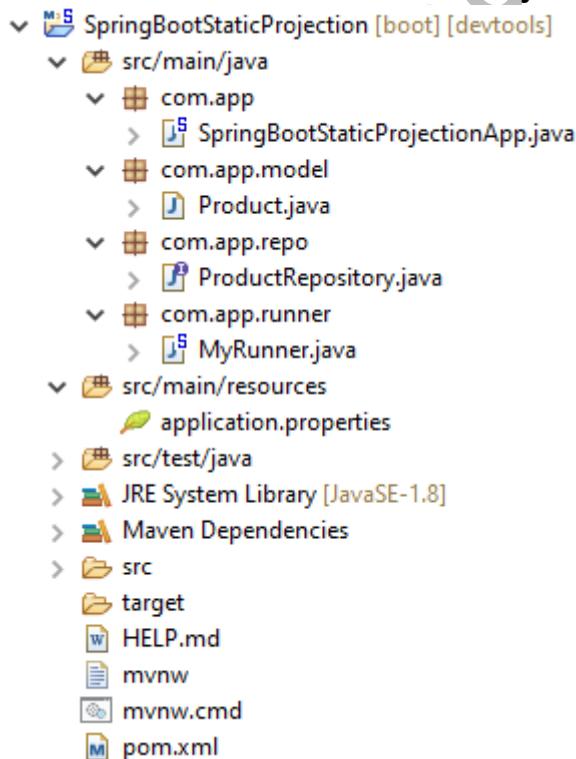
**Step#3:-** Use that child Interface as ReturnType for findBy() findBy methods.

**Format:**

**SYNTAX:**

```
Interface _____Repository extends JpaRepository<...>{
    Interface <childType> {
        DataType getVariable();
        DataType getVariable();
    }
    List<childType> findBy____(...);
}
```

## #15. Folder Structure of Static Projection:--



**1. Model class:-**

```
package com.app.model;
import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
public class Product {

    @Id
    private Integer proId;
    private String vendorCode;
    private String prodName;
    private Double prodCost;

    public Product() {
        super();
    }
    public Product(Integer proId) {
        super();
        this.proId = proId;
    }
    public Product(Integer proId, String vendorCode, String prodName, Double
prodCost) {
        super();
        this.proId = proId;
        this.vendorCode = vendorCode;
        this.prodName = prodName;
        this.prodCost = prodCost;
    }
    public Integer getProId() {
        return proId;
    }
    public void setProId(Integer proId) {
        this.proId = proId;
    }
    public String getVendorCode() {
        return vendorCode;
    }
    public void setVendorCode(String vendorCode) {
        this.vendorCode = vendorCode;
    }
}
```

```

        public String getProdName() {
            return prodName;
        }
        public void setProdName(String prodName) {
            this.prodName = prodName;
        }
        public Double getProdCost() {
            return prodCost;
        }
        public void setProdCost(Double prodCost) {
            this.prodCost = prodCost;
        }
    }

```

**2. Repository code:--**

```

package com.app.repo;
import java.util.List;
import org.springframework.data.jpa.repository.JpaRepository;
import com.app.model.Product;

public interface ProductRepository extends JpaRepository<Product, Integer> {

    interface MyView {
        //Copy from getter method (RT and methodName)
        String getVendorCode();
        String getProdName();
        Double getProdCost();
    }
    //select code, cost from prodtab where ven_code=?
    List<MyView> findByVendorCode(String vc);
}

```

**3. CommandLineRunner code:--**

```

package com.app.runner;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

```

```

import com.app.model.Product;
import com.app.repo.ProductRepository;
import com.app.repo.ProductRepository.MyView;

@Component
public class MyRunner implements CommandLineRunner{

    @Autowired
    private ProductRepository repo;

    @Override
    public void run(String... args) throws Exception {
        repo.save(new Product(10, "A", "PEN", 10.5));
        repo.save(new Product(20, "B", "PENCIL", 50.5));
        repo.save(new Product(30, "C", "LAPTOP", 700.5));
        repo.save(new Product(40, "D", "MOBILE", 800.5));

        List<MyView> p = repo.findByVendorCode("B");
        for(MyView p1:p)
        {
            System.out.println(p1.getVendorCode()+" , "+p1.getProdName()+" , "+p1.getProdCost());
        }
    }
}

```

#### 4. application.properties:--

```

server.port=2019
spring.datasource.driverClassName=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe
spring.datasource.username=system
spring.datasource.password=system
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.database-platform=org.hibernate.dialect.Oracle10gDialect

```

#### Output:--

```

2019-06-25 00:34:52.933 INFO 15220 --- [ restartedMain] org.hibernate.Version : HH000412: Hibernate Core {5.3.7.Final}
2019-06-25 00:34:52.945 INFO 15220 --- [ restartedMain] org.hibernate.cfg.Environment : HH000206: hibernate.properties not found
2019-06-25 00:34:53.476 INFO 15220 --- [ restartedMain] o.hibernate.annotations.common.Version : HCANN00001: Hibernate Commons Annotations {5.0.4.Final}
2019-06-25 00:34:53.827 INFO 15220 --- [ restartedMain] org.hibernate.dialect.Dialect : HH000400: Using dialect: org.hibernate.dialect.Oracle10gDialect
2019-06-25 00:34:56.967 INFO 15220 --- [ restartedMain] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2019-06-25 00:34:57.108 INFO 15220 --- [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port 35729
2019-06-25 00:34:58.286 INFO 15220 --- [ restartedMain] com.app.SpringBootStaticProjectionApp : Started SpringBootStaticProjectionApp in 11.204 seconds (JVM running for 14.909)
Hibernate: select product0_.prod_id as prod_id1_0_0_, product0_.prod_cost as prod_cost2_0_0_, product0_.prod_name as prod_name3_0_0_, product0_.vendor_code as vendor_code4_0_0_ from
Hibernate: select product0_.prod_id as prod_id1_0_0_, product0_.prod_cost as prod_cost2_0_0_, product0_.prod_name as prod_name3_0_0_, product0_.vendor_code as vendor_code4_0_0_ from
Hibernate: select product0_.prod_id as prod_id1_0_0_, product0_.prod_cost as prod_cost2_0_0_, product0_.prod_name as prod_name3_0_0_, product0_.vendor_code as vendor_code4_0_0_ from
Hibernate: select product0_.prod_id as prod_id1_0_0_, product0_.prod_cost as prod_cost2_0_0_, product0_.prod_name as prod_name3_0_0_, product0_.vendor_code as vendor_code4_0_0_ from
2019-06-25 00:34:58.547 INFO 15220 --- [ restartedMain] o.h.h.i.QueryTranslatorFactorInitiator : HH000397: Using ASTQueryTranslatorFactory
Hibernate: select product0_.vendor_code as col_0_0_, product0_.prod_name as col_1_0_, product0_.prod_cost as col_2_0_ from product product0_ where product0_.vendor_code=? B,PENCIL,50.5
B,MOBILE,800.5
2019-06-25 00:34:58.859 INFO 15220 --- [ Thread-11] j.LocalContainerEntityManagerFactoryBean : Closing JPA EntityManagerFactory for persistence unit 'default'
2019-06-25 00:34:58.883 INFO 15220 --- [ Thread-11] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown initiated...
2019-06-25 00:34:58.915 INFO 15220 --- [ Thread-11] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown completed.

```

Activate Windows

**b. Dynamic Projections:--**

In this case `findBy(____)` method return type is decided at runtime (i.e It is Generics)  
Format & Syntax:

```
<T> List<T> findBy____(...,Class<T>clz); //fixed Line
```

Note:-- Here "T" is child interface type, provide at runtime.

**Code Changes:--****1. Repository code:--**

```
package com.app.repo;
import java.util.List;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import com.app.model.Product;
```

```
@Repository
```

```
public interface ProductRepository extends JpaRepository<Product, Integer> {
```

**//2. Dynamic Projection Code**

```
interface MyViewTwo {
    //Copy from getter method (RT and methodName)
    Integer getProdId();
    Double getProdCost();
}
```

```
interface MyView {
    //Copy from getter method (RT and methodName)
    String getProdCode();
    Double getProdCost();
}
```

```
<T> List<T> findByVendorCode(String vc, Class<T>clz);
}
```

**2. CommandLineRunner code:--**

```
package com.app.runner;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;
import com.app.model.Product;
import com.app.repo.ProductRepository;
import com.app.repo.ProductRepository.MyViewTwo;
```

```
@Component
public class MyRunner implements CommandLineRunner {

    @Autowired
    private ProductRepository repo;
```

```
@Override
public void run(String... args) throws Exception {
```

```
    repo.save(new Product(10, "A", "V1", "PEN", 10.5));
    repo.save(new Product(20, "B", "V2", "PENCIL", 50.5));
    repo.save(new Product(30, "C", "V3", "MOBILE", 700.5));
    repo.save(new Product(40, "B", "V3", "MOBILE", 800.5));
```

## //2. Dynamic Projection

```
    repo.findByVendorCode("V3", MyViewTwo.class)
        .stream()
        .map((ob) -> ob.getProdId() + "," + ob.getProdCost())
        .forEach(System.out::println);
    //or
    repo.findByVendorCode("V2", MyView.class)
        .stream()
        .map((a) -> a.getProdCode() + "," + a.getProdCost())
        .forEach(System.out::println);
}
```

### Output:--

```
2019-06-26 00:39:01.739  INFO 6488 --- [ restartedMain] o.h.h.i.QueryTranslatorFactoryInitiator : HHH000397: Using ASTQueryTranslatorFactory
Hibernate: select product0_.prod_id as col_0_0_, product0_.prod_cost as col_1_0_ from product product0_ where product0_.vendor_code=?
30,700.5
40,800.5
Hibernate: select product0_.prod_code as col_0_0_, product0_.prod_cost as col_1_0_ from product product0_ where product0_.vendor_code=?
8,50.5
Spring Boot Starter Executed :
2019-06-26 00:39:02.164  INFO 6488 --- [      Thread-11] j.LocalContainerEntityManagerFactoryBean : Closing JPA EntityManagerFactory for persistence unit 'default'
2019-06-26 00:39:02.183  INFO 6488 --- [      Thread-11] com.zaxxer.hikari.HikariDataSource      : HikariPool-1 - Shutdown initiated...
2019-06-26 00:39:02.202  INFO 6488 --- [      Thread-11] com.zaxxer.hikari.HikariDataSource      : HikariPool-1 - Shutdown completed.
```

Activate Win

### Note:--

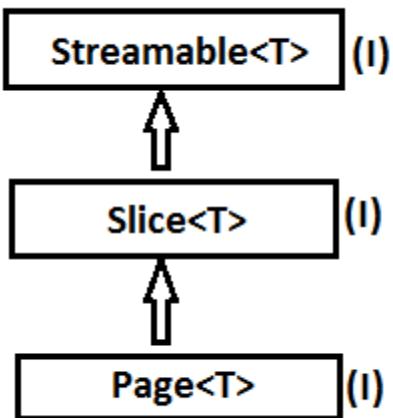
=>Dynamic Projections are slow compare to static Projections.

=>For dynamic Projections “Type selection, validate and execution” done at Runtime, where as for static select and validate done at compile time, only execution done at runtime.

=>Static Projections also called as compile time (selected) Projections and Dynamic Projections also called as Runtime (selected) Projections.

### Special Types in Query methods:-

1. Page<T>
2. Slice<T>
3. Stream<T>



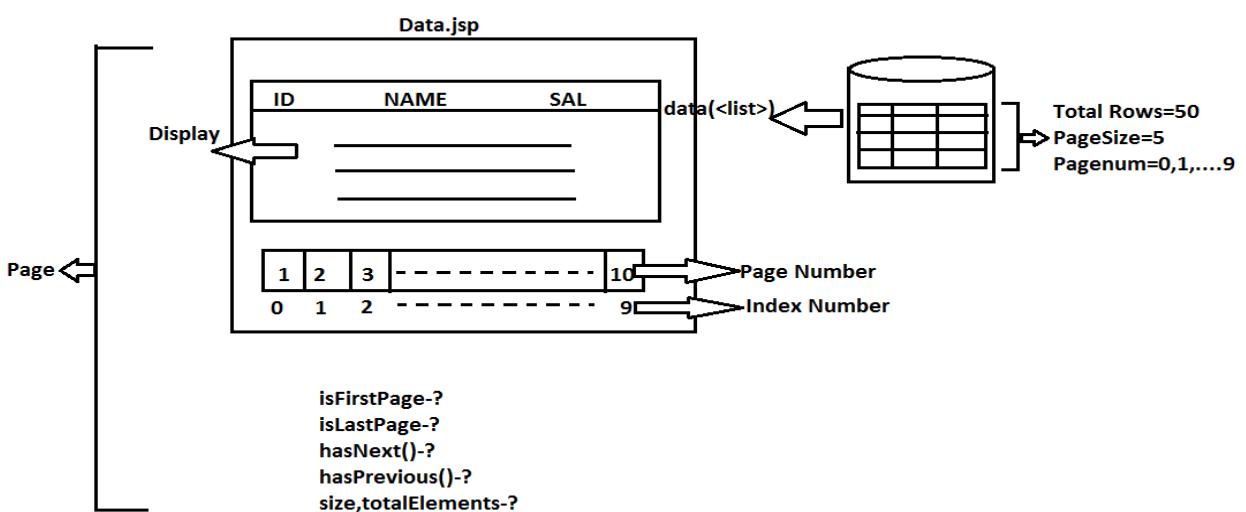
=>These are Special type outputs for Query methods to get “Pagination data”.

**1. Streamable(I):--** It will help to return Collection data (one page only) in Stream<T> (java.util) format, where we can apply functions like filter, sort, map and collect...etc

**2. Slice(I):--** It is used to provide current page data and links with previous and next page details

\*\* It never holds all pages details like total no.of pages and total Elements.

**3. Page(I):--** It is extension to Slice and holds even total no.of pages (links to other pages) and total Elements (total rows details).



**Page(and Slice):--**

**Step#1:-** Insert few records in db (ex; 20 Rows).

**Step#2:-** Define a Repository method with parameter must be Pageable (as last param).

Ex:-Page<Product> findByVendorCode (String vc, Pageable pageable);

**Step#3:-** In Runner class call method and print details like, isEmpty?, first?, last?....etc

**Streaming:**-- Streaming is process of

Read → filter → sort → map → collect

=>Over collection data which uses Streams concept given in JDK 1.8.

=>Streams are used to execute series of operations in one or less Iterations, which reduces execution time to normal programmer iterations.

#### 4. Connection Pooling in SpringBootData:--

=>SpringBoot 1.X works on Tomcat Connection pooling as default and Boot2.X works on Hikari connection pooling which is very fast in service.

\*\*This is not required to be configured by developer. By default spring boot configures using class input “HikariConfig” with all default values (milli sec time).

\*\*To provide Programmer specific values use key `spring.datasource.hikari.*` in application.properties.

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

```
spring.datasource.hikari.connection-timeout=20000
spring.datasource.hikari.minimum-idle=5
spring.datasource.hikari.maximum-pool-size =12
spring.datasource.hikari.idle-timeout =300000
spring.datasource.hikari.max-lifetime =12000
spring.datasource.hikari.auto-commit =true
```

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

```
spring:
  datasource:
    hikari:
      connection-timeout:20000
      minimum-idle: 5
      maximum-pool-size: 12
      idle-timeout: 300000
      max-lifetime: 12000
      auto-commit: true
```

## **CHAPTER#3 MONGODB**

### **1. Introduction:--**

Spring boot supports working with NoSQL database Ex:- MongoDB.

=>MongoDB is a simple and open-source **document** database and leading NoSQL database and lightweight Database which stores data in JSON format.

=>JSON (Java Script Object Notation) it is object in java script language, but used in all programming, web services & DBs... etc.

=>Compare to programming object, XML or other Data formats JSON is light weight.

=>JSON format is : {"key" : value,...}.

=>JACKSON is a converter API used to convert java object <=> JSON.

Another example for converter is GSON.

Java	Java Script
int a =10; String b = "Hi";	var a = 10; var b = "Hi";
Employee e = new Employee(); e.setEmplId(10); e.setEmpName("A"); e.setEmpSal(3.5); //Java Notation	var = { "emplId" : 10, "empName" : "A", "empSal" : 3.2 }

### **Download and Install MongoDB (NoSQL):--**

**Step#1:-** Goto MongoDB Download Center.

<https://www.mongodb.com/download-center/community>

**Step#2:-** Choose “Server” option and select details OS, Version and package (MSI) then click on download.

Details:

Version : 3.6 or 3.4

Type : MSI (Do not use ZIP)

The screenshot shows the MongoDB Download Center interface. At the top, there are tabs for Cloud, Server, and Tools, with 'Server' being the active tab. Below this, a section titled 'Select the server you would like to run:' offers two options: 'MongoDB Community Server' (FEATURE RICH. DEVELOPER READY) and 'MongoDB Enterprise Server' (ADVANCED FEATURES. PERFORMANCE GRADE). The Community Server is selected. A dropdown menu for 'Version' shows '3.6.11 (previous release)', and another for 'OS' shows 'Windows 64-bit x64'. Under 'Package', 'MSI' is chosen. A large green 'Download' button is prominently displayed. To the right of the download button is a list of links: Release notes, Changelog, All version binaries, Installation instructions, and Download source (tgz).

**Step#3:- Install DB in system (double click => next...=>finish).**

=>While Installing Full (Complete) Type.

The screenshot shows the MongoDB Compass Community interface. On the left, there's a sidebar with 'CREATE FREE ATLAS' (Includes 512 MB of storage), 'New Connection', 'Favorites', and 'RECENTS'. The main area is titled 'Welcome to MongoDB Compass Community'. It displays a 'QUERY PERFORMANCE SUMMARY' for a query involving 'last\_login' and 'state'. The 'EXPLAIN PLAN' tab is active, showing a 'VISUAL TREE' diagram of the query execution stages: SHARD\_MERGE, SHARD01, and SHARD02, each with its own execution time and clock details. A callout points to the 'Execution Time' field in the summary. Another callout points to the 'Clocks' in the tree diagram. Navigation arrows for 'Previous' and 'Next' are visible at the bottom right.

The screenshot shows the MongoDB Compass Community interface. The sidebar remains the same. The main area is titled 'Welcome to MongoDB Compass Community'. It displays an 'INDEX MANAGEMENT' section. The 'INDEXES' tab is active, showing a table of indexes with columns: Name and Definition, Type, Size, Usage, and Properties. Examples include '\_id\_1\_gender\_1' (REGULAR), 'email\_1\_favorite\_features\_1' (REGULAR), '\_id\_1' (REGULAR), 'last\_login\_indexed' (HASHED), and 'name\_text\_address\_city\_text' (TEXT). A note explains that the index type is shown in the 'Type' column and provides more info via an info circle. Another note discusses 'Index Usage'. Navigation arrows for 'Previous' and 'Next' are visible at the bottom right.

Welcome to MongoDB Compass Community

Values are now validated inline based on their type.

```

1 _id: ObjectId('50317ff7fe4dce3731b54ab0')
2 favorite_feature : "Document Validation"
3 membership_status : "PENDING"
4 name : "Elle Harrison"
5 gender : "female"
6
7 age : 999999999 ◀ Value 999999999 is outside the valid Int32 range
8 phone_no : "+177232403234"
9 last_login : July 29 2017
10 address : Object
11 last_position : Object
12 email : "sidewalkenforcer4@aim.com"

```

Improved type conversion: pick any type from the dropdown to convert a value to the chosen type.

Flexible date inputs: Compass tries to interpret the entered value as a date.

**Update not permitted while document contains errors.**

CANCEL UPDATE

**Improved CRUD**

Better editing with validation of individual BSON types.

&lt; Previous

Next &gt;

Welcome to MongoDB Compass Community

To connect to a Replica Set, enter the Replica Set name in the connect dialog.

Replica Set Name: repset-us-east

Read Preference: ✓ Primary  
Primary Preferred  
Secondary  
**Secondary Preferred**  
Nearest

SSL: Nearest

Choose a read preference from the dropdown menu. On fail-over, Compass will automatically connect you to a new member of the Replica Set.

**Deployment Awareness**

Replica set aware connections allow for continued use during replica set configuration changes and provides additional information of the connected cluster.

&lt; Previous

Next &gt;

Connect View Help

CREATE FREE ATLAS Includes 512 MB of Learn more

New Connection

Favorites

RECENTS

**Welcome to MongoDB Compass Community**

Open the query history with this icon.

POST QUERIES RECENT PREFERENCES

Fri Oct 13 2017 17:15:12 GMT-0400 (EDT)

POST
 {
 "start": 3
 }
 SORT
 {
 "business\_id": 1
 }
 SKIP
 3
 LIMIT
 10

Fri Oct 13 2017 17:14:24 GMT-0400 (EDT)

POST
 {
 "text": "This is the place"
 }

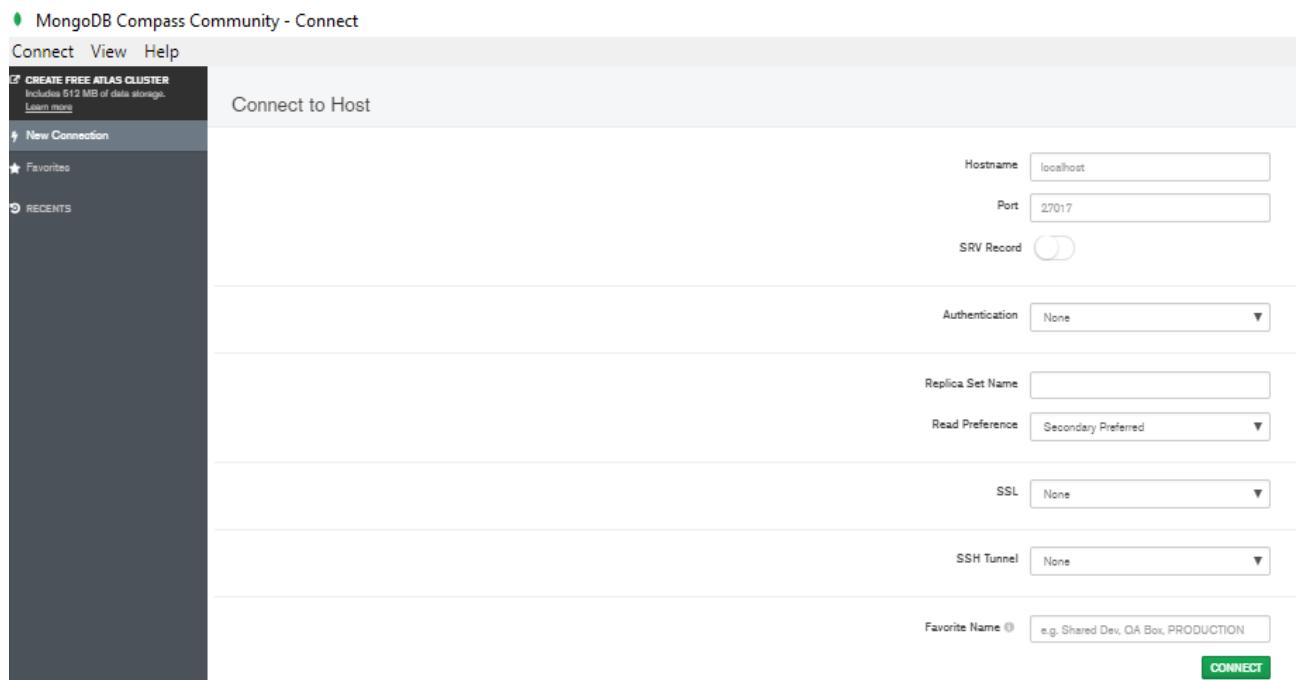
Recent and favorite queries are stored in the query history panel

**Query History**

Easily access and manage executed queries and save favorites for often executed queries.

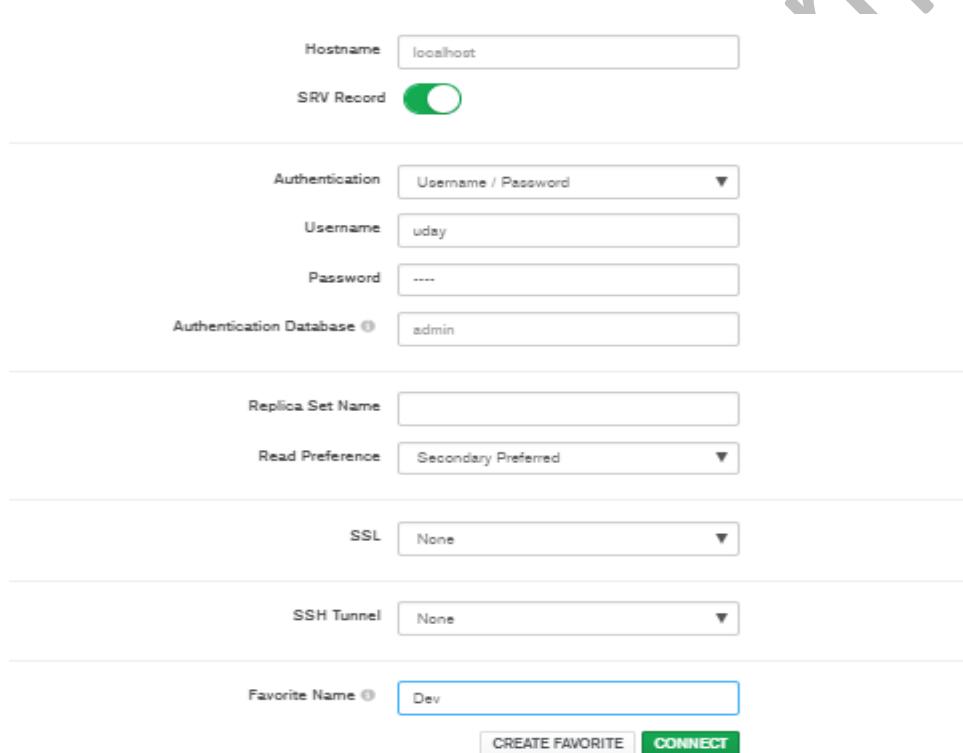
&lt; Previous

Get Started



The screenshot shows the MongoDB Compass interface for connecting to a host. The connection parameters are set as follows:

- Hostname:** localhost
- Port:** 27017
- SRV Record:** On
- Authentication:** None
- Replica Set Name:** (empty)
- Read Preference:** Secondary Preferred
- SSL:** None
- SSH Tunnel:** None
- Favorite Name:** e.g. Shared Dev, QA Box, PRODUCTION
- CONNECT** button

The screenshot shows the MongoDB Compass interface for connecting to a host, with authentication details filled in:

- Hostname:** localhost
- SRV Record:** On
- Authentication:** Username / Password
- Username:** uday
- Password:** ----
- Authentication Database:** admin
- Replica Set Name:** (empty)
- Read Preference:** Secondary Preferred
- SSL:** None
- SSH Tunnel:** None
- Favorite Name:** Dev
- CREATE FAVORITE** button
- CONNECT** button

=>Open Notepad and Type below command

->mongod > save with .bat

Ex:-- "mongo-server.bat"

->mongo > save with .bat

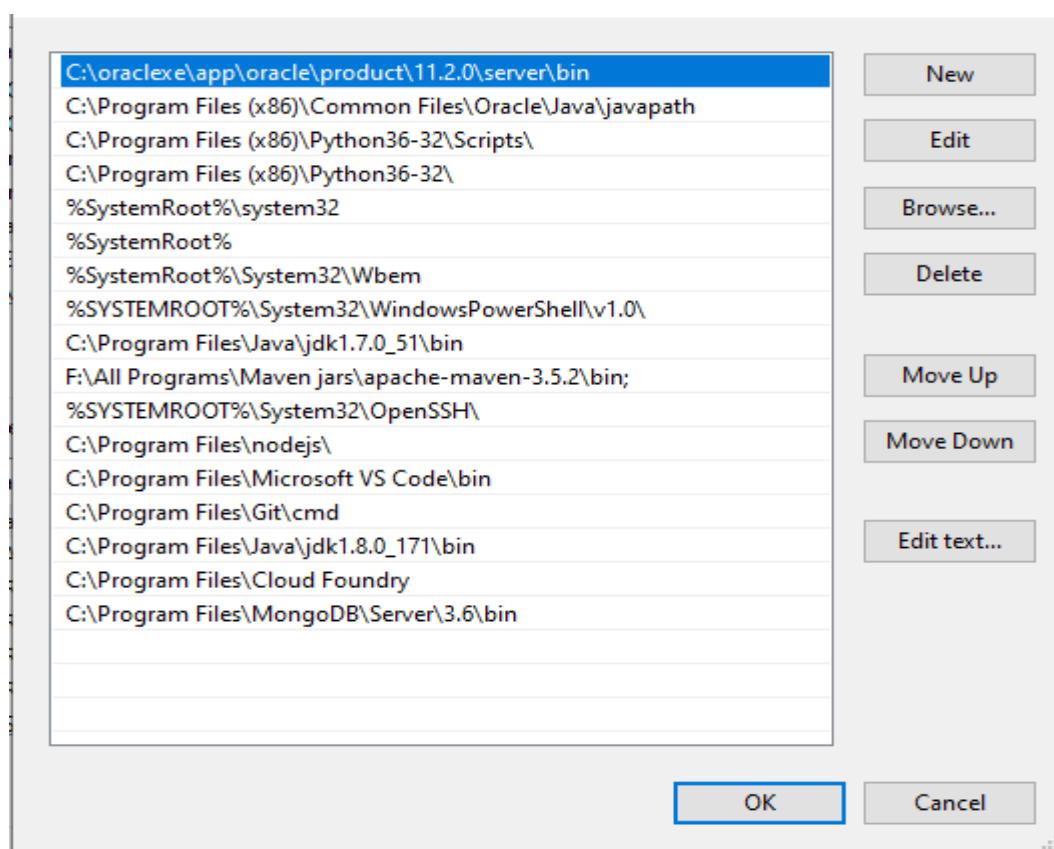
Ex:--"mongo-client.bat"

**Step#4:- Set Path to MongoDB Server**

=>Copy location of MongoDB installed address till bin folder

Ex:- C:\Program Files\MongoDB\Server\3.6\bin

=>Go to my Computer => Right click => Properties =>Advanced System Settings => Environment Variables => Chosse Path => Edit => Paste above location and symbol";" => save =>finish.

**Step#5:- Create service (server) folder in C:/ drive looks like "C:/data/db".**

NOTE:-- Here folder name can be any things.

**Step#6:- Start MongoDB server.**

=>Open cmd prompt-1 => type mongod => press enter

**Ex:-** C:\Program Files\MongoDB\Server\3.6\bin>mongod

\*\* To stop => press CTRL+C => Press y => enter

**Step#7:- Start MongoDB Client**

=>Open cmd prompt-2 =>type mongo => press Enter

**Ex:-** C:\Program Files\MongoDB\Server\3.6\bin>mongo

\*\* It will start client.

### MongoDB commands:--

- ❖ show dbs / show databases => View All Databases.
- ❖ use sathya => Get into one DB (sathya)
- ❖ show collections => View All collections in DB
- ❖ db.book.insert ({ "bcode" : "JAVA" , "bauth" : "ABC" , "bcost" : 3.3});

=>To insert one JSON row into Collection.

- ❖ db.<collectionName>.insert({“key”: val,...})
- ❖ db.book.find() [or] db.book.find.pretty();
- Or
- ❖ db.book.find({“bcode”：“JAVA”})

=>To get JSON rows from collection.

=>Delete one Collection object

- ❖ db.book.remove({“bookCode”：“JAVA”})

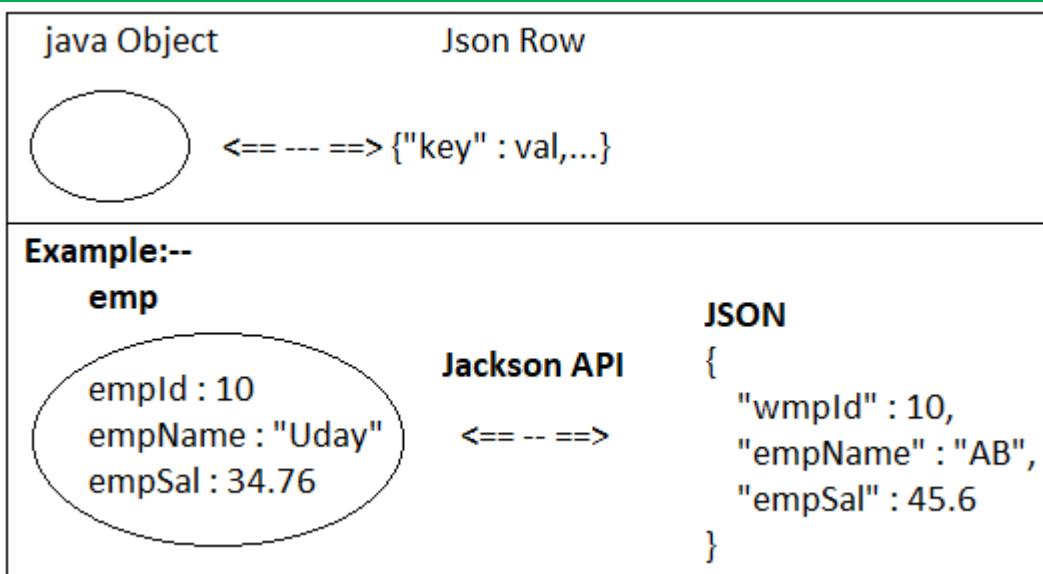
```
> show dbs
admin  0.000GB
config  0.000GB
local  0.000GB
sathya  0.000GB
> use sathya
switched to db sathya
> db.book.insert ({ "bcode" : "Spring Boot" , "bauth" : "ABCD" , "bcost" : 7.3});
WriteResult({ "nInserted" : 1 })
> db.book.find().pretty();
2019-06-27T14:11:09.065+0530 E QUERY    [thread1] TypeError: db.book.find is not a function :
@(shell):1:1
> db.book.find().pretty();
{
  "_id" : ObjectId("5d127845e6349e86fdf7f8ca"),
  "bcode" : "JAVA",
  "bauth" : "ABC",
  "bcost" : 3.3
}
{
  "_id" : ObjectId("5d1480fb09942fdc62f6eea1"),
  "bcode" : "Spring Boot",
  "bauth" : "ABCD",
  "bcost" : 7.3
}
> show collections
book
employee
> -
```

\*\*More Commands goto <https://docs.mongodb.com/manual/reference/command/>

=>Here Tables are called **Collections**. Rows are called as **JSON objects**.

=>MongoDB holds data in JSON format created from java object.

i.e one Java Object <=>One JSON object.



=>ORM concepts says

- One class = one table.
- One variable = one column.
- One Object = one row.

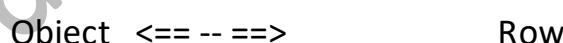
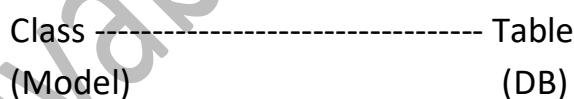
=>But, MongoDB follows (NoSQL concepts. It contains Collections (behaves like tables, but not).

=>Collection holds data in JSON format.

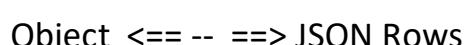
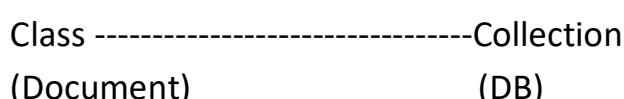
- One class = One collection.
- One Object = one JSON Row.

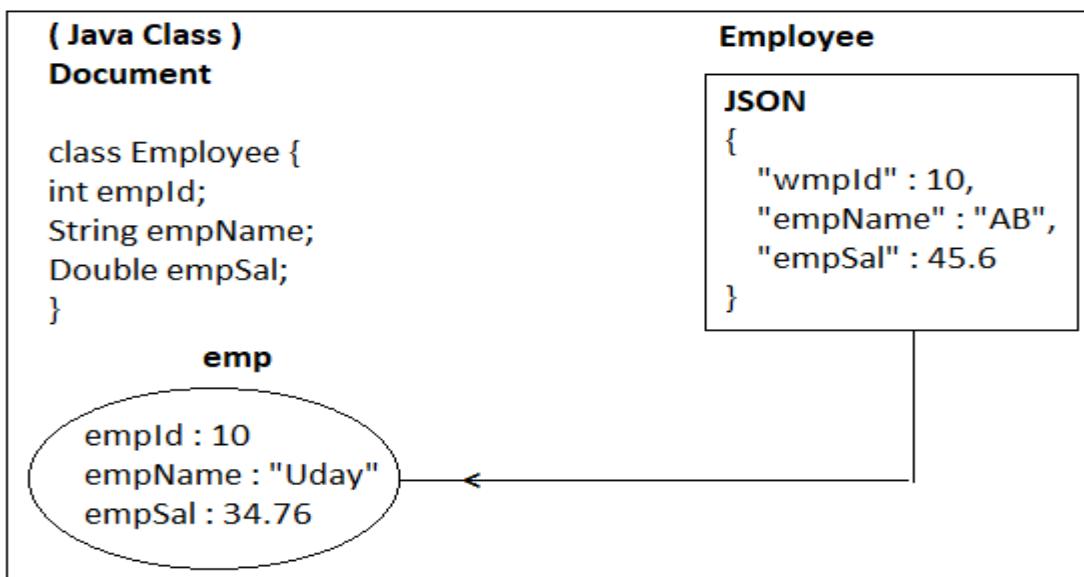
=>Every class must be called as **Document** which can have generated ID (UUID type).

**ORM (SQL):--**



**(NoSQL):--**





=>To work with MongoDB using spring boot we need to add its starter looks like.

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

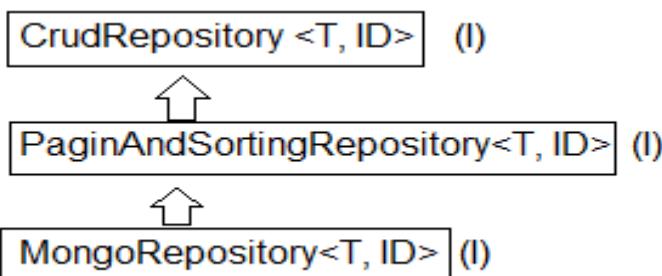
=>We can work with Installed MongoDB or embedded MongoDB. In case of embedded use dependency in pom.xml (remove `<scope>test </scope>`).

```
<dependency>
    <groupId>de.flapdoodle.embed</groupId>
    <artifactId>de.flapdoodle.embed.mongo</artifactId>
</dependency>
```

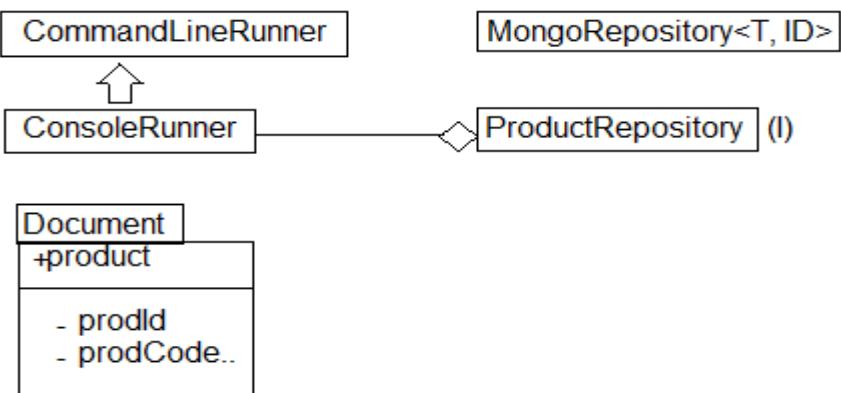
=>Boot also supports Embedded MongoDB for every coding and Testing Process.

=>Embedded MongoDB can be used in Dev, Test, Uat, but not in Production Environment.

### Design#1:- Spring Boot MongoDB Repository Levels:--



## Design#2:- Spring Boot MongoDB Coding files:-



### 2. Embedded MongoDB:-

#### Coding Steps:-- Embedded MongoDB

**Step#1:-** Create one starter project.

=>File => new => Spring Starter Project

=>Enter name : “SpringBootMongoDBEmbedded”

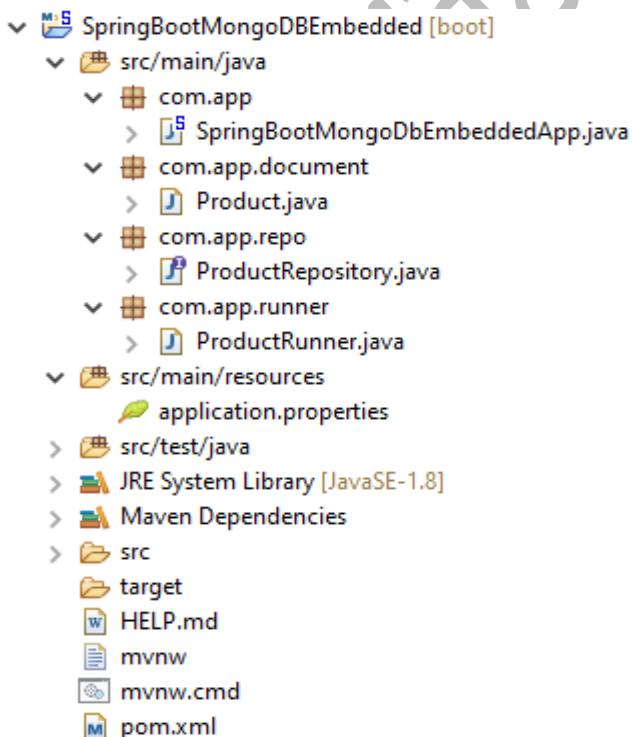
=>next => Search using “mongo”

=>Choose 2 dependencies

    ->Spring Data MongoDB

    ->Embedded MongoDB database

### 16. Folder Structure of SpringBootMongoDB Embedded Database :-



**Step#2:-** \*\*\* open pom.xml and remove  
<scope>test</scope>  
=>line for embedded mongo dependency.

**Step#3:-** Define one document class, primaryKey must be starting which is generated by MongoDB as Hexadecimal type using UUID concept.  
(UUID = Universal Unique Identifier).

### Document class---

```
package com.app.document;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

@Document
public class Product {
    @Id
    private String id;
    private Integer proId;
    private String prodName;
    private Double prodCost;

    public Product() {
        super();
    }

    public Product(Integer proId, String prodName, Double prodCost) {
        super();
        this.proId = proId;
        this.prodName = prodName;
        this.prodCost = prodCost;
    }

    public Product(String id, Integer proId, String prodName, Double prodCost) {
        super();
        this.id = id;
        this.proId = proId;
        this.prodName = prodName;
        this.prodCost = prodCost;
    }
}
```

```

public String getId() {
    return id;
}
public void setId(String id) {
    this.id = id;
}
public Integer getProdId() {
    return proId;
}
public void setProdId(Integer proId) {
    this.proId = proId;
}
public String getProdName() {
    return prodName;
}
public void setProdName(String prodName) {
    this.prodName = prodName;
}
public Double getProdCost() {
    return prodCost;
}
public void setProdCost(Double prodCost) {
    this.prodCost = prodCost;
}
@Override
public String toString() {
    return "Product [id=" + id + ", proId=" + proId + ", prodName=" +
prodName + ", prodCost=" + prodCost + "]";
}
}

```

**4. Repository:**-- Define one Repository interface that takes 2 generic params :

Document and ID types.

```

package com.app.repo;
import org.springframework.data.mongodb.repository.MongoRepository;
import com.app.document.Product;
public interface ProductRepository extends MongoRepository<Product, String> { }

```

**Step#5:-** Define runner class and make HAS-A with repository interface.

```
package com.app.runner;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;
import com.app.document.Product;
import com.app.repo.ProductRepository;
```

```
@Component
public class ProductRunner implements CommandLineRunner{

    @Autowired
    private ProductRepository repo;

    @Override
    public void run(String... args) throws Exception {
        repo.deleteAll();
        repo.save(new Product(10, "Mobile", 567.8));
        repo.save(new Product(11, "Laptop", 867.8));
        repo.save(new Product(12, "Computer", 467.8));
        System.out.println("-----");
        repo.findAll().forEach(System.out::println);
    }
}
```

\*\*\* Run Starter class.

#### Output:-

```
-----
Product [id=5d149cc97fa5112ea42513b4, prodId=10, prodName=Mobile, prodCost=567.8]
Product [id=5d149cc97fa5112ea42513b5, prodId=11, prodName=Laptop, prodCost=867.8]
Product [id=5d149cc97fa5112ea42513b6, prodId=12, prodName=Computer, prodCost=467.8]
Main Class Executed :
2019-06-27 16:09:06.181  INFO 11940 --- [           Thread-2] org.mongodb.driver.connection
```

#### Note:-

- @Document** is optional, but good practice to write must be provided in case of multiple concepts used to application.
- @Id** is also optional provide a variable named as “id” of type String only.
- @Id need to be provided in case of variable names is not ‘id’. Else value will be null.

Ex:-

```
@Id  
private String id;
```

- d. @Id type must be **String** only. Integer, Double... not accepted. It is UUID (Hexa Decimal value) which can be stored in string Datatype only.
- e. In case of wrong data type is used variable creation then boot throws Exception as : can't autogenerate id of type java.lang.Integer for entity of type com.app.document ! for example code :

```
@Id  
private Integer id;
```

### **3. Spring Boot External MongoDB Setup and code:--**

=>To work External MongoDB using Spring Boot we need to provide only details of DB in application properties of yml files like.

=>Default port no of MongoDB server is **27017**.

```
spring.data.mongodb.host=localhost  
spring.data.mongodb.port=27017  
spring.data.mongodb.database=sathya
```

=>By default MongoDB comes with NoSecure/NoAuth Setup, we can provide authentication details (username, password for login and logout). In that case we must provide extra keys like.

```
spring.data.mongodb.username=sa  
spring.data.mongodb.password=sa
```

#### **application.yml :--**

```
spring:  
  data:  
    mongodb:  
      host: localhost  
      port: 27017  
      database: sathya  
      username: sa  
      password: sa
```

**Step#1:-** Create one Spring Boot Starter project

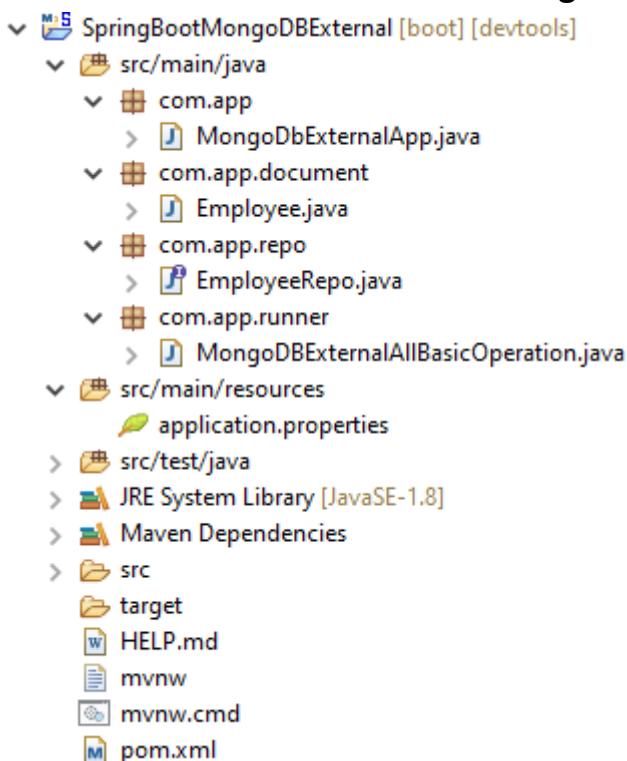
NAME : SpringBootMongoDBExternal

Dependency : Spring Data MongoDB

**MongoDB Dependency:**--

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

### 17. Folder Structure of External MongoDB:--



**pom.xml:--**

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.1.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
```

```
</parent>
<groupId>com.app</groupId>
<artifactId>MongoDBExternal</artifactId>
<version>1.0</version>
<name>MongoDBExternal</name>
<description>Spring Boot MongoDB Connectivity Application</description>

<properties>
    <java.version>1.8</java.version>
</properties>
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-mongodb</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
</project>
```

**Step#2:-** Open application.properties (.yml) and provide mongoDB host, port, database etc... details.

**application.properties:--**

```
spring.data.mongodb.host=localhost
spring.data.mongodb.port=27017
spring.data.mongodb.database=abc
```

**Step#3:-** Define, repo, Runner (same as before example).

```
package com.app.repo;
import org.springframework.data.mongodb.repository.MongoRepository;
import com.app.document.Employee;
```

```
public interface EmployeeRepository extends MongoRepository<Employee, String>{ }
```

**Step#4:- Define one Runner class.**

```
package com.app.runner;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;
import com.app.document.Employee;
import com.app.repo.EmployeeRepo;

@Component
public class MongoDBExternalAllBasicOperation implements CommandLineRunner {

    @Autowired
    private EmployeeRepo repo;

    @Override
    public void run(String... args) throws Exception {
        repo.save(new Employee(10, "UDAY", "Hyd"));
        repo.findAll().forEach(System.out::println);
    }
}
```

**Execution Process:**-- Goto MongoDB install directory till bin folder and open two cmd prompt.

=>Open cmd-1 and type > mongod (to start server)

=>Open cmd-2 and type > mongo (to open and work on DB)

=>Run starter class and go to mongo client to see output.

## 1. Console Output:--

```
2019-06-27 17:21:32.315 INFO 11112 --- [ restartedMain] com.app.MongoDbExternalApp
2019-06-27 17:21:32.329 INFO 11112 --- [ restartedMain] com.app.MongoDbExternalApp
2019-06-27 17:21:32.511 INFO 11112 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor
2019-06-27 17:21:34.310 INFO 11112 --- [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate
2019-06-27 17:21:34.613 INFO 11112 --- [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate
2019-06-27 17:21:39.755 INFO 11112 --- [ restartedMain] org.mongodb.driver.cluster
2019-06-27 17:21:39.756 INFO 11112 --- [ restartedMain] org.mongodb.driver.cluster
2019-06-27 17:21:40.003 INFO 11112 --- [localhost:27017] org.mongodb.driver.connection
2019-06-27 17:21:40.023 INFO 11112 --- [localhost:27017] org.mongodb.driver.cluster
2019-06-27 17:21:40.028 INFO 11112 --- [localhost:27017] org.mongodb.driver.cluster
2019-06-27 17:21:40.479 WARN 11112 --- [ restartedMain] o.s.d.m.c.m.BasicMongoPersistentProperty
2019-06-27 17:21:40.677 INFO 11112 --- [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer
2019-06-27 17:21:41.331 INFO 11112 --- [ restartedMain] com.app.MongoDbExternalApp
2019-06-27 17:21:41.523 INFO 11112 --- [ restartedMain] org.mongodb.driver.connection
Employee [Id=5d141fbf7fa5111f9c37ef33, empId=10, empName=Uday, empAdd=null]
Employee [Id=5d14ab787fa5110624af73fc, empId=10, empName=UDAY, empAdd=Hyd]
Employee [Id=5d14adcd7fa5112b68cd2e25, empId=10, empName=UDAY, empAdd=Hyd]
Executed :
2019-06-27 17:21:41.727 INFO 11112 --- [ Thread-8] org.mongodb.driver.connection
```

## 2. Mongo client output:--

```
> db.employee.find().pretty();
{
    "_id" : ObjectId("5d141fbf7fa5111f9c37ef33"),
    "empId" : 10,
    "empName" : "Uday",
    "empSal" : 34.8,
    "_class" : "com.app.collection.Employee"
}
{
    "_id" : ObjectId("5d14ab787fa5110624af73fc"),
    "empId" : 10,
    "empName" : "UDAY",
    "empAdd" : "Hyd",
    "_class" : "com.app.document.Employee"
}
{
    "_id" : ObjectId("5d14adcd7fa5112b68cd2e25"),
    "empId" : 10,
    "empName" : "UDAY",
    "empAdd" : "Hyd",
    "_class" : "com.app.document.Employee"
}
> ■
```

#### 4. Working with Multiple values of JSON in MongoDB:--

=>In general JSON key=value combination indicates primitive Data format.

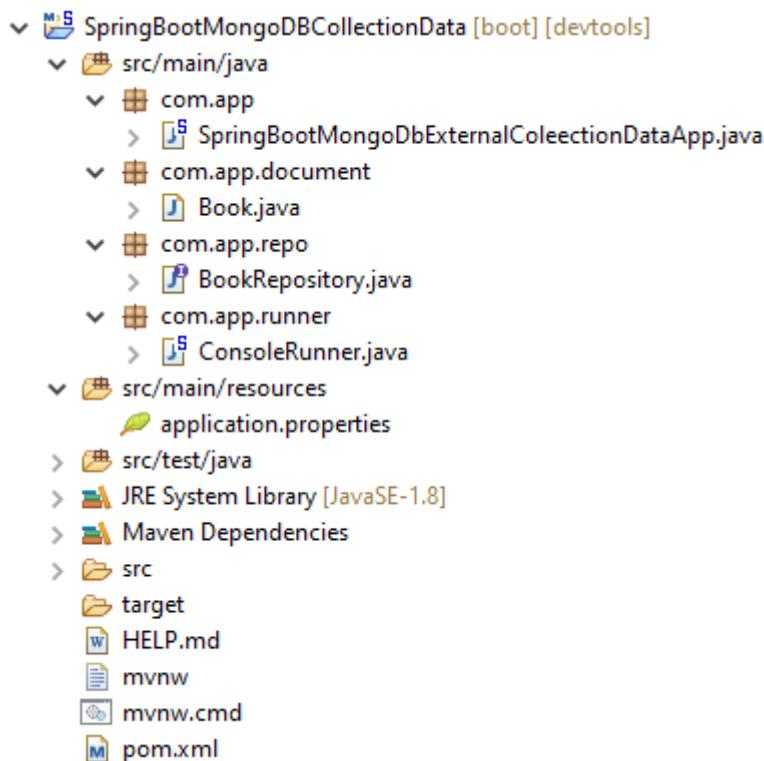
Ex:-- {"empId" : 25, "empName": "Uday"}

=>To store multiple values we can use collection type.

##### a. List/Set/Array :--

=>In case of java these are different but coming to JSON format holds as group of elements. Format is given as : ["value", "value"].

#### #18. Folder Structure of Collection (List/Set/Array) Data in MongoDB:--



##### application.properties:--

spring.data.mongodb.host=localhost

spring.data.mongodb.port=27017

spring.data.mongodb.database=sathya

##### 1. Document class:--

```

package com.app.document;
import java.util.Arrays;
import java.util.List;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;
  
```

@Document

```
public class Book {

    @Id //UUID
    private String id;
    private String bookCode;
    private String bookAuth;
    private Double bookCost;
    //Collection dataType
    private List<String> codes;
    private String[] grades;

    public Book() {
        super();
    }

    public Book(String bookCode, String bookAuth, Double bookCost, List<String>
               codes, String[] grades) {
        super();
        this.bookCode = bookCode;
        this.bookAuth = bookAuth;
        this.bookCost = bookCost;
        this.codes = codes;
        this.grades = grades;
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getBookCode() {
        return bookCode;
    }

    public void setBookCode(String bookCode) {
        this.bookCode = bookCode;
    }
}
```

```

public String getBookAuth() {
    return bookAuth;
}

public void setBookAuth(String bookAuth) {
    this.bookAuth = bookAuth;
}

public Double getBookCost() {
    return bookCost;
}

public void setBookCost(Double bookCost) {
    this.bookCost = bookCost;
}

public List<String> getCodes() {
    return codes;
}

public void setCodes(List<String> codes) {
    this.codes = codes;
}

public String[] getGrades() {
    return grades;
}

public void setGrades(String[] grades) {
    this.grades = grades;
}

@Override
public String toString() {
    return "Book [id=" + id + ", bookCode=" + bookCode + ", bookAuth=" +
bookAuth + ", bookCost=" + bookCost + ", codes=" + codes + ", grades=" +
Arrays.toString(grades) + "]";
}
}

```

## 2. Repository:-

```

package com.app.repo;
import org.springframework.data.mongodb.repository.MongoRepository;
import com.app.model.Book;
public interface BookRepository extends MongoRepository<Book, String> { }

```

### 3. Runner class:--

```

package com.app.runner;
import java.util.Arrays;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;
import com.app.document.Book;
import com.app.repo.BookRepository;

@Component
public class BookRunner implements CommandLineRunner {

    @Autowired
    private BookRepository repo;

    @Override
    public void run(String... args) throws Exception {
        repo.deleteAll();
        repo.save(new Book("REDR", "Uday Kumar", 45.76,
            Arrays.asList("A1", "A2"), new String[]{"A", "B", "C"}));
        repo.save(new Book("REDS", "Venkat Reddy", 85.26,
            Arrays.asList("B1", "B2"), new String[]{"X", "Y", "Z"}));
        repo.findAll().forEach(System.out::println);
        System.out.println("Completed");
    }
}

```

### Console Output:--

```

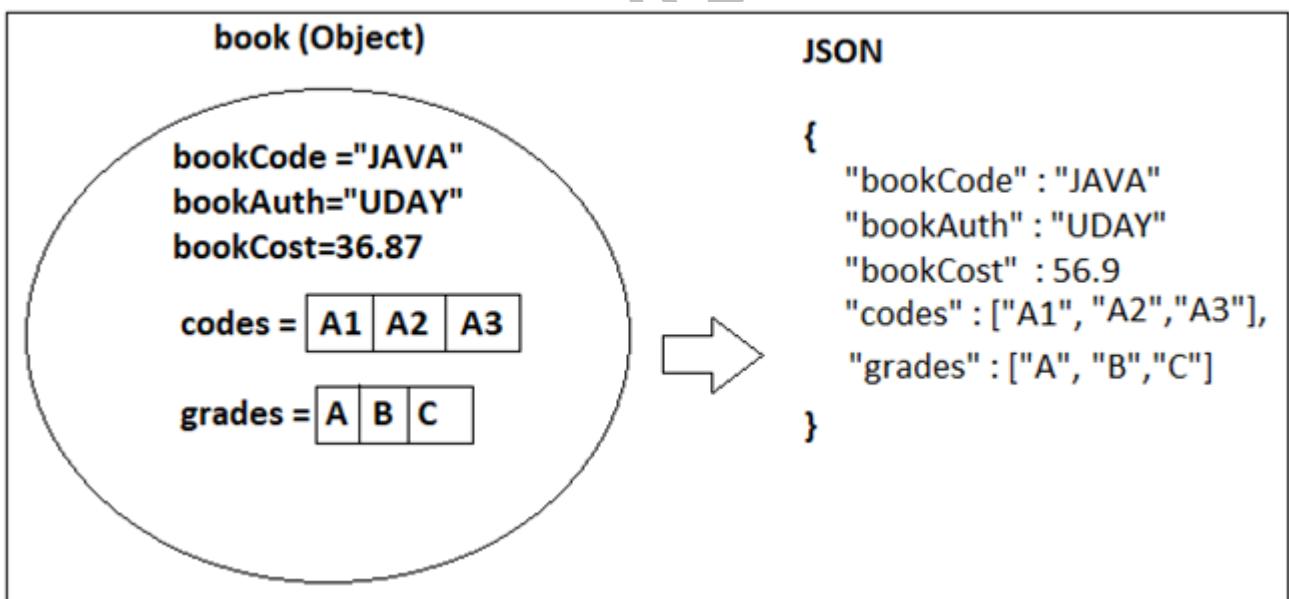
2019-06-27 18:51:42.798  INFO 7064 --- [ restartedMain] ringBootMongoDbCollectionDataApplication : Starting SpringBootMongoDbCol
2019-06-27 18:51:42.810  INFO 7064 --- [ restartedMain] ringBootMongoDbCollectionDataApplication : No active profile set, falling
2019-06-27 18:51:43.017  INFO 7064 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults act
2019-06-27 18:51:44.919  INFO 7064 --- [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data repo
2019-06-27 18:51:45.313  INFO 7064 --- [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repositor
2019-06-27 18:51:50.693  INFO 7064 --- [ restartedMain] org.mongodb.driver.cluster           : Cluster created with settings
2019-06-27 18:51:50.694  INFO 7064 --- [ restartedMain] org.mongodb.driver.cluster           : Adding discovered server local
2019-06-27 18:51:50.912  INFO 7064 --- [localhost:27017] org.mongodb.driver.connection : Opened connection [connection]
2019-06-27 18:51:50.927  INFO 7064 --- [localhost:27017] org.mongodb.driver.cluster           : Monitor thread successfully cc
2019-06-27 18:51:50.932  INFO 7064 --- [localhost:27017] org.mongodb.driver.cluster           : Discovered cluster type of STA
2019-06-27 18:51:51.507  INFO 7064 --- [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer      : LiveReload server is running c
2019-06-27 18:51:52.018  INFO 7064 --- [ restartedMain] ringBootMongoDbCollectionDataApplication : Started SpringBootMongoDbColle
2019-06-27 18:51:52.134  INFO 7064 --- [ restartedMain] org.mongodb.driver.connection           : Opened connection [connection]
Book [id=5d14c2f07fa5111b989631cd, bookCode=JAVA, bookAuth=UDAY, bookCost=36.87, codes=[A1, A2, A3], grades=[A, B, C]]
Book [id=5d14c2f07fa5111b989631ce, bookCode=Spring, bookAuth=Venkat Reddy, bookCost=85.26, codes=[B1, B2], grades=[X, Y, Z]]
Completed
2019-06-27 18:51:52.357  INFO 7064 --- [      Thread-8] org.mongodb.driver.connection           : Closed connection [connection]

```

### MongoDB Client console Output:--

```
C:\Windows\System32\cmd.exe - mongo
> db.book.find().pretty();
{
  "_id" : ObjectId("5d14c2f07fa5111b989631cd"),
  "bookCode" : "JAVA",
  "bookAuth" : "UDAY",
  "bookCost" : 36.87,
  "codes" : [
    "A1",
    "A2",
    "A3"
  ],
  "grades" : [
    "A",
    "B",
    "C"
  ],
  "_class" : "com.app.document.Book"
}
{
  "_id" : ObjectId("5d14c2f07fa5111b989631ce"),
  "bookCode" : "Spring",
  "bookAuth" : "Venkat Reddy",
  "bookCost" : 85.26,
  "codes" : [
    "B1",
    "B2"
  ],
  "grades" : [
    "X",
    "Y",
    "Z"
  ],
  "_class" : "com.app.document.Book"
}
> 
```

Activate Windows  
Go to Settings to activate Windows.



**b. Map/Properties:--** It holds data in 2D format i.e key =value format. JSON holds 2D as another JSON (child JSON) looks like {"key" : "val", "key": "val", "key" : "val".....} for both Map and Properties

## #19. Folder Structure of Map/Property Collection Data in MongoDB:--

```

    SpringBootMongoDBMapPropertyCollection [boot] [devtools]
      src/main/java
        com.app
          com.app.document
            Book.java
          com.app.repo
            BookRepository.java
          com.app.runner
            ConsoleRunner.java
      src/main/resources
        application.properties
      src/test/java
      JRE System Library [JavaSE-1.8]
      Maven Dependencies
      src
        target
        HELP.md
        mvnw
        mvnw.cmd
        pom.xml
  
```

**application.properties:--**

spring.data.mongodb.host=localhost

spring.data.mongodb.port=27017

spring.data.mongodb.database=sathya

**1. Collection class:--**

```

package com.app.document;
import java.util.Map;
import java.util.Properties;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;
  
```

@Document

```

public class Book {
  @Id
  private String id;
  private String bookCode;
  private String bookAuth;
  private Double bookCost;
  private Map<String, String> forms;
  private Properties models;
  
```

```
public Book() {
    super();
}

public Book(String bookCode, String bookAuth, Double bookCost,
Map<String, String> forms, Properties models) {
    super();
    this.bookCode = bookCode;
    this.bookAuth = bookAuth;
    this.bookCost = bookCost;
    this.forms = forms;
    this.models = models;
}

public String getId() {
    return id;
}

public void setId(String id) {
    this.id = id;
}

public String getBookCode() {
    return bookCode;
}

public void setBookCode(String bookCode) {
    this.bookCode = bookCode;
}

public String getBookAuth() {
    return bookAuth;
}

public void setBookAuth(String bookAuth) {
    this.bookAuth = bookAuth;
}

public Double getBookCost() {
    return bookCost;
}

public void setBookCost(Double bookCost) {
    this.bookCost = bookCost;
}
```

```

        public Map<String, String> getForms() {
            return forms;
        }
        public void setForms(Map<String, String> forms) {
            this.forms = forms;
        }
        public Properties getModels() {
            return models;
        }
        public void setModels(Properties models) {
            this.models = models;
        }
    }
    @Override
    public String toString() {
        return "Book [id=" + id + ", bookCode=" + bookCode + ", bookAuth=" + bookAuth + ",
bookCost=" + bookCost + ", forms=" + forms + ", models=" + models + "]";
    }
}

```

**2>Repository:--**

```

package com.app.repo;
import org.springframework.data.mongodb.repository.MongoRepository;
import com.app.document.Book;

public interface BookRepository extends MongoRepository<Book, String> { }

```

**3. Runner class:--**

```

package com.app.runner;
import java.util.HashMap;
import java.util.Map;
import java.util.Properties;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;
import com.app.document.Book;
import com.app.repo.BookRepository;

```

@Component

```
public class ConsoleRunner implements CommandLineRunner {

    @Autowired
    private BookRepository repo;

    @Override
    public void run(String... args) throws Exception {

        repo.deleteAll();
        Map<String, String> m1 = new HashMap<>();
        m1.put("A1", "B1");
        m1.put("A2", "B2");
        Properties p1 = new Properties();
        p1.put("M1", "N1");
        p1.put("M2", "N2");
        repo.save(new Book("JAVA", "Rama", 33.34, m1, p1));
        System.out.println(".....");
        repo.findAll().forEach(System.out::println);
        System.out.println("finished");
    }
}
```

### Console Output---

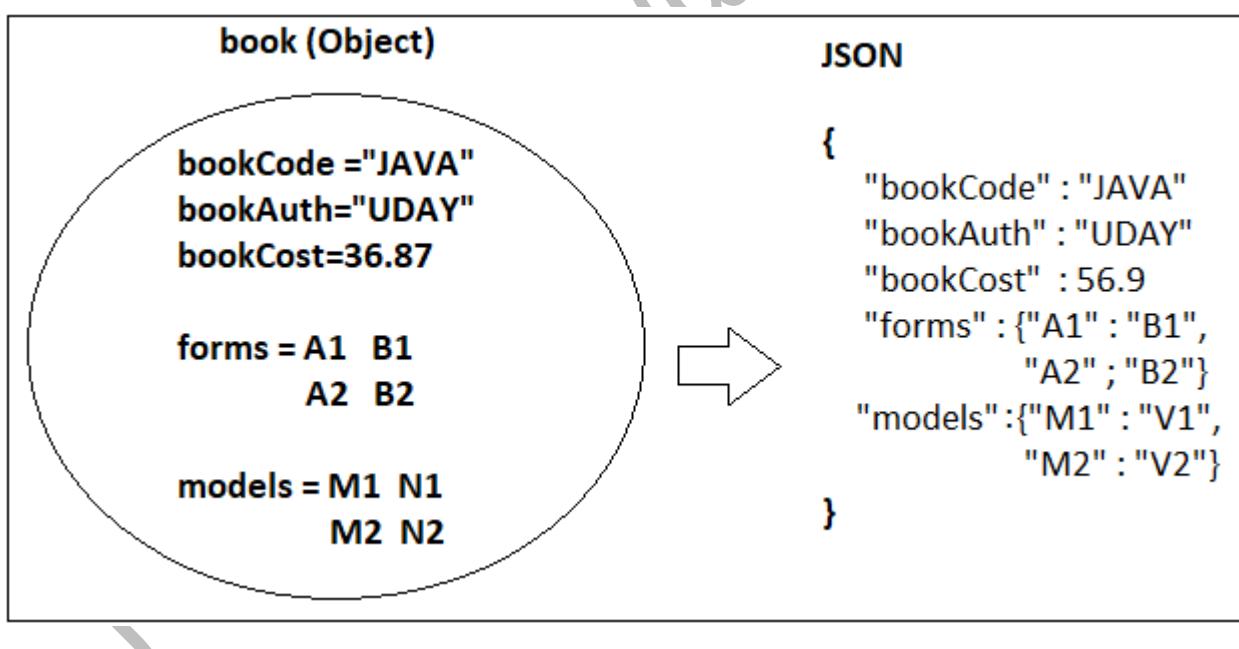
```
2019-06-27 19:22:32.999 INFO 13268 --- [ restartedMain] springBootMongoDbMapPropertyCollectionApp : Starting SpringBootMongoDbMapPropertyCollectio
2019-06-27 19:22:33.017 INFO 13268 --- [ restartedMain] springBootMongoDbMapPropertyCollectionApp : No active profile set, falling back to default
2019-06-27 19:22:33.163 INFO 13268 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'spring
2019-06-27 19:22:34.419 INFO 13268 --- [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data repositories in DEFA
2019-06-27 19:22:34.661 INFO 13268 --- [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 23
2019-06-27 19:22:39.333 INFO 13268 --- [ restartedMain] org.mongodb.driver.cluster : Cluster created with settings {hosts=[localhost
2019-06-27 19:22:39.333 INFO 13268 --- [ restartedMain] org.mongodb.driver.cluster : Adding discovered server localhost:27017 to cl
2019-06-27 19:22:39.548 INFO 13268 --- [localhost:27017] org.mongodb.driver.connection : Opened connection [connectionId{localValue:1,
2019-06-27 19:22:39.568 INFO 13268 --- [localhost:27017] org.mongodb.driver.cluster : Monitor thread successfully connected to serve
2019-06-27 19:22:39.573 INFO 13268 --- [localhost:27017] org.mongodb.driver.cluster : Discovered cluster type of STANDALONE
2019-06-27 19:22:40.207 INFO 13268 --- [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port 35729
2019-06-27 19:22:40.810 INFO 13268 --- [ restartedMain] springBootMongoDbMapPropertyCollectionApp : Started SpringBootMongoDbMapPropertyCollection
2019-06-27 19:22:40.929 INFO 13268 --- [ restartedMain] org.mongodb.driver.connection : Opened connection [connectionId{localValue:2,
.....
Book [id=5d14ca297fa51133d45923e4, bookCode=JAVA, bookAuth=Rama, bookCost=33.34, forms={A1=B1, A2=B2}, models={M2=N2, M1=N1}]
finished
2019-06-27 19:22:41.228 INFO 13268 --- [      Thread-8] org.mongodb.driver.connection : Closed connection [connectionId{localValue:2,
```

MongoDB client output:--

```
> db.book.find().pretty();
{
    "_id" : ObjectId("5d14ca297fa51133d45923e4"),
    "bookCode" : "JAVA",
    "bookAuth" : "Rama",
    "bookCost" : 33.34,
    "forms" : {
        "A1" : "B1",
        "A2" : "B2"
    },
    "models" : {
        "M2" : "N2",
        "M1" : "N1"
    },
    "_class" : "com.app.document.Book"
}
> .
```

\*\* In MongoDB : (Operation)

=>db.book.find().pretty();

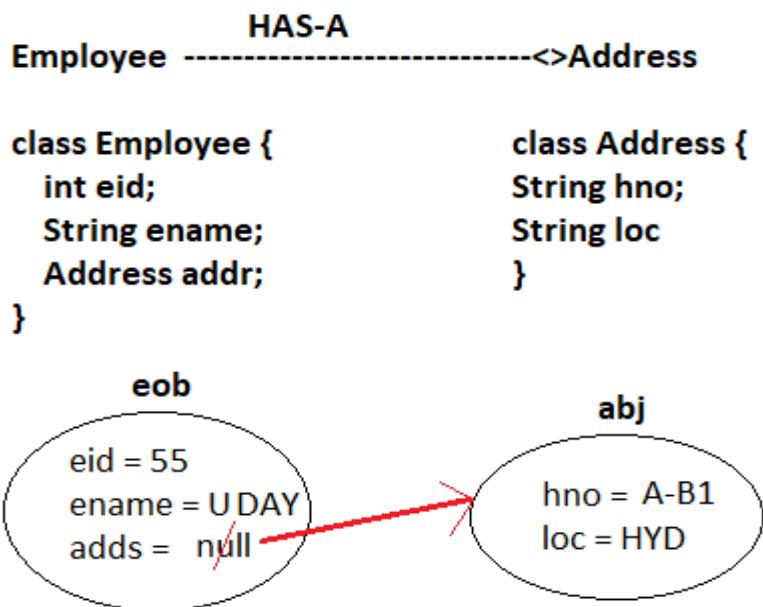


**C. Reference Type (HAS-A) :-** In this case JSON holds object as "Inner JSON format": i.e JSON Object having JSON object.

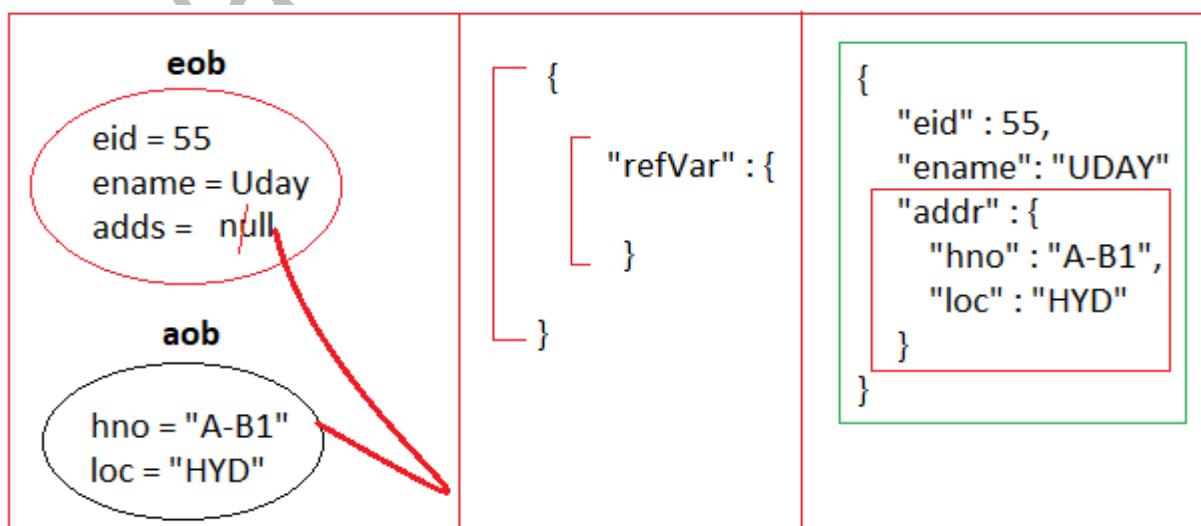
**Format Looks like:-**

```
{
    Key: val....
    ...
    "has-AVariable" : {
        Key: val...
    }
}
```

**Example:-**



=>for above Objects JSON format looks like



## #20. Folder Structure of HAS-A in MongoDB:--

```

    < SpringBootMongoDBHas-A [boot] [devtools]
      < src/main/java
        > com.app
        < com.app.document
          > Address.java
          > Employee.java
        < com.app.repo
          > EmployeeRepository.java
        < com.app.runner
          > ConsoleRunner.java
      < src/main/resources
        > application.properties
      < src/test/java
      > JRE System Library [JavaSE-1.8]
      > Maven Dependencies
      > src
      > target
      > HELP.md
      > mvnw
      > mvnw.cmd
      > pom.xml
  
```

**application.properties:--**

spring.data.mongodb.host=localhost  
 spring.data.mongodb.port=27017  
 spring.data.mongodb.database=sathya  
 spring.data.mongodb.username=SA  
 spring.data.mongodb.password=SA

**1. Collection classes:--****a. Address.java (Child class):--**

**package** com.app.document;

**public class** Address {

**private** String hno;  
**private** String loc;

**public** Address() {  
**super**();  
}

```

public Address(String hno, String loc) {
    super();
    this.hno = hno;
    this.loc = loc;
}

public String getHno() {
    return hno;
}
public void setHno(String hno) {
    this.hno = hno;
}
public String getLoc() {
    return loc;
}
public void setLoc(String loc) {
    this.loc = loc;
}
@Override
public String toString() {
    return "Address [hno=" + hno + ", loc=" + loc + "]";
}
}

```

**b. Employee.java (Parent class):--**

```

package com.app.document;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

@Document
public class Employee {

    @Id
    private String id;
    private Integer empld;
    private String empName;
    private Address addr; //Has-A
}

```

```
public Employee() {
    super();
}

public Employee(Integer empld, String empName, Address addr) {
    super();
    this.empld = empld;
    this.empName = empName;
    this.addr = addr;
}

public Employee(String id, Integer empld, String empName, Address addr) {
    super();
    this.id = id;
    this.empld = empld;
    this.empName = empName;
    this.addr = addr;
}

public String getId() {
    return id;
}

public void setId(String id) {
    this.id = id;
}

public Integer getEmpld() {
    return empld;
}

public void setEmpld(Integer empld) {
    this.empld = empld;
}

public String getEmpName() {
    return empName;
}

public void setEmpName(String empName) {
    this.empName = empName;
}
```

```

public Address getAddr() {
    return addr;
}

public void setAddr(Address addr) {
    this.addr = addr;
}

@Override
public String toString() {
    return "Employee [id=" + id + ", emplId=" + emplId + ", empName=" +
empName + ", addr=" + addr + "]";
}

```

## 2. Repository Interface:--

```

package com.app.repo;
import org.springframework.data.mongodb.repository.MongoRepository;
import com.app.document.Employee;

public interface EmployeeRepository extends
    MongoRepository <Employee, String> { }

```

## 3. Runner class code:--

```

package com.app.runner;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;
import com.app.document.Address;
import com.app.document.Employee;
import com.app.repo.EmployeeRepository;

@Component
public class ConsoleRunner implements CommandLineRunner {

    @Autowired
    private EmployeeRepository repo;
}

```

```

@Override
public void run(String... args) throws Exception {
    repo.deleteAll();
    repo.save(new Employee(10, "UDAY", new Address("2-3A", "HYD")));
    System.out.println(".....");
    repo.findAll().forEach(System.out::println);
    System.out.println("done");
}

```

### 1. Console Output Screen:--

```

2019-06-27 19:40:43.869  INFO 10352 --- [ restartedMain] c.app.SpringBootMongoDBHasAApplication
2019-06-27 19:40:43.889  INFO 10352 --- [ restartedMain] c.app.SpringBootMongoDBHasAApplication
2019-06-27 19:40:44.093  INFO 10352 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor
2019-06-27 19:40:45.729  INFO 10352 --- [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate
2019-06-27 19:40:45.981  INFO 10352 --- [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate
2019-06-27 19:40:50.727  INFO 10352 --- [ restartedMain] org.mongodb.driver.cluster
2019-06-27 19:40:50.728  INFO 10352 --- [ restartedMain] org.mongodb.driver.cluster
2019-06-27 19:40:50.957  INFO 10352 --- [localhost:27017] org.mongodb.driver.connection
2019-06-27 19:40:50.978  INFO 10352 --- [localhost:27017] org.mongodb.driver.cluster
2019-06-27 19:40:50.983  INFO 10352 --- [localhost:27017] org.mongodb.driver.cluster
2019-06-27 19:40:51.606  INFO 10352 --- [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer
2019-06-27 19:40:52.234  INFO 10352 --- [ restartedMain] c.app.SpringBootMongoDBHasAApplication
2019-06-27 19:40:52.627  INFO 10352 --- [ restartedMain] org.mongodb.driver.connection
.....
Employee [id=5d14ce6c7fa51128708bdbfa, empId=10, empName=UDAY, addr=Address [hno=A-B1, loc=HYD]]
done
2019-06-27 19:40:52.920  INFO 10352 --- [      Thread-8] org.mongodb.driver.connection

```

### 2. MongoDB client Output:--

```

> db.employee.find().pretty();
{
    "_id" : ObjectId("5d14ce6c7fa51128708bdbfa"),
    "empId" : 10,
    "empName" : "UDAY",
    "addr" : {
        "hno" : "A-B1",
        "loc" : "HYD"
    },
    "_class" : "com.app.document.Employee"
}
>

```

### 5. Securing DB in MongoDB:-- It is specific for database (MongoDB).

**Step#1:-** Start mongo server.

Ex:-- C:\Program Files\MongoDB\Server\3.6\bin>mongod

**Step#2:-** Start mongo client.

**Ex:-** C:\Program Files\MongoDB\Server\3.6\bin>mongo

#1. Switch to your db (which need security).

> use Sathya

#2. Create one user with user name, password and roles.

> db.createUser({ "user" : "SA", "pwd" : "SA", "roles" : [ "readWrite", "dbAdmin" ] })

=>Press Enter

Successfully added user: { "user" : "SA", "roles" : [ "readWrite", "dbAdmin" ] }

**Step#3:-** Close mongo client and server.

**Step#4:-** Start mongo server in authentication mode

**Ex:-** C:\Program Files\MongoDB\Server\3.6\bin>mongod –auth

**Step#5:-** Start mongo client

**Ex:-** Cmd > C:\Program Files\MongoDB\Server\3.6\bin>mongo

**Step#6:-** Switch to your db.

>use sathya

**Step#7:-** login as user.

> db.auth("SA", "SA")

**Screen Shot:-**

```
C:\Program Files\MongoDB\Server\3.6\bin>mongo
MongoDB shell version v3.6.11
connecting to: mongodb://127.0.0.1:27017/?gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("7f698817-9606-4df3-bddb-b327b8add6b4") }
MongoDB server version: 3.6.11
> use sathya
switched to db sathya
> db.auth("SA", "SA")
1
>
```

**Step#8:-** \*\* Now, goto Spring Boot application and add below key=val pairs.

**application.properties:-**

spring.data.mongodb.host=localhost

spring.data.mongodb.port=27017

spring.data.mongodb.database=sathya

spring.data.mongodb.username=SA

spring.data.mongodb.password=SA

**application.yml:--**

```

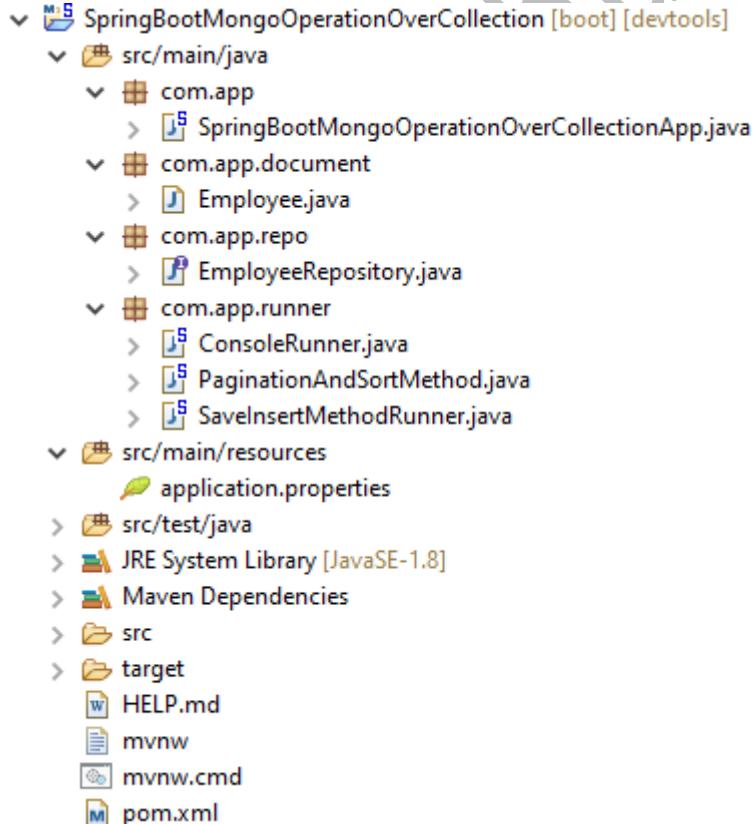
spring:
  data:
    mongodb:
      host: localhost
      port: 27017
      database: sathya
      username: SA
      password: SA
  
```

**6. Working with MongoDB Operations over Collection:--**

=>In general Collection is called as Data holder in JSON Format (we can compare with table).

=>To insert data into MongoDB, Repository interfaces has provided methods like save() and saveAll() which returns same object updated with primary Key.

=>PrimaryKey is String type, it is UUID format. (Universal Unique Identifier) which is hexa decimal number (0-9, A-F).

**#21. Folder Structure of Operation over collection in MongoDB:--**

**1. Collection class:--**

```
package com.app.collection;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

@Document
public class Employee {

    @Id
    private String id;
    private Integer empld;
    private String empName;
    private Double empSal;

    public Employee() {
        super();
    }

    public Employee(Integer empld, String empName, Double empSal) {
        super();
        this.empld = empld;
        this.empName = empName;
        this.empSal = empSal;
    }

    public Employee(String id, Integer empld, String empName, Double empSal) {
        super();
        this.id = id;
        this.empld = empld;
        this.empName = empName;
        this.empSal = empSal;
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }
}
```

```

public Integer getEmpld() {
    return empld;
}
public void setEmpld(Integer empld) {
    this.empld = empld;
}
public String getEmpName() {
    return empName;
}
public void setEmpName(String empName) {
    this.empName = empName;
}
public Double getEmpSal() {
    return empSal;
}
public void setEmpSal(Double empSal) {
    this.empSal = empSal;
}
@Override
public String toString() {
    return "Employee [id=" + id + ", empld=" + empld + ", empName=" +
empName + ", empSal=" + empSal + "]";
}
}

```

**2. repository Interface:--**

```

package com.app.repo;
import org.springframework.data.mongodb.repository.MongoRepository;
import com.app.collection.Employee;

public interface EmployeeRepository extends
    MongoRepository <Employee, String> { }

```

**3. Runner class#1:--**

```

package com.app.runner;
import java.util.Arrays;
import java.util.List;

```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;
import com.app.document.Employee;
import com.app.repo.EmployeeRepository;
```

```
@Component
```

```
public class ConsoleRunner implements CommandLineRunner {
    @Autowired
    private EmployeeRepository repo;
    @Override
    public void run(String... args) throws Exception {
```

#### //1. Delete Previous records

```
        repo.deleteAll();
        System.out.println("-----");
```

#### //2. Insert one row

```
        Employee e = repo.save(new Employee(10, "Uday", 34.8));
        repo.findAll().forEach(System.out::println);
        System.out.println("-----");
        System.out.println(e.getId());
```

#### //3. Insert Bulk records

```
        List<Employee> emps = repo.saveAll(Arrays.asList(
            new Employee(11, "Rama", 45.7),
            new Employee(12, "Neha", 49.7),
            new Employee(13, "Raja", 42.7)
        ));
        for (Employee es:emps) {
            System.out.println(es.getId());
        }
        System.out.println("-----");
        repo.findAll().forEach(System.out::println);
        System.out.println("=====");
    }
}
```

**Output:-**

```
-----
not found
-----
Employee [id=5d1516be7fa5110bd81c8943, empId=10, empName=Uday, empSal=67.76]
Employee [id=5d1516be7fa5110bd81c8944, empId=11, empName=Venkat, empSal=98.36]
Employee [id=5d1516be7fa5110bd81c8945, empId=12, empName=Neha, empSal=37.46]
-----
Employee [id=5d1516be7fa5110bd81c8944, empId=11, empName=Venkat, empSal=98.36]
Employee [id=5d1516be7fa5110bd81c8943, empId=10, empName=Uday, empSal=67.76]
Employee [id=5d1516be7fa5110bd81c8945, empId=12, empName=Neha, empSal=37.46]
Employee [id=5d1516be7fa5110bd81c8943, empId=10, empName=Uday, empSal=67.76]
Employee [id=5d1516be7fa5110bd81c8944, empId=11, empName=Venkat, empSal=98.36]
-----
```

**Runner class#2:- (with save () )--****Case#1:-** id =null, then generate ID and insert`repo.save (new Employee(10, "Uday", 85.7));`**Case#2:-** id 1234ad is not exist then insert`repo.save(new Employee("1234ad", 11, "Raja", 65.8));`**Case#3:-** id 1234ad is exist then update/modify`repo.save(new Employee("1234ad", 12, "Neha", 45.6));`**Runner code:-- for save() /Insert() method**

```
package com.app.runner;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;
import com.app.document.Employee;
import com.app.repo.EmployeeRepository;

@Component
public class saveOrInsertRunner implements CommandLineRunner {

    @Autowired
    private EmployeeRepository repo;

    @Override
    public void run(String... args) throws Exception {
        repo.deleteAll();
```

```
//Save() code
```

```

    //Case#1:- id =null, then generate ID and insert
    repo.save (new Employee (10, "Uday", 85.7));
    //Case#2:- id 1234ad is not exist then insert
    repo.save(new Employee ("1234ad", 11, "Raja", 65.8));
    repo.findAll().forEach(System.out::println);
    //Case#3:- id 1234ad is exist then update/modify
    repo.save(new Employee ("1234ad", 12, "Neha", 45.6));
}
}
```

**Output:-**

```
> db.employee.find().pretty();
{
  "_id" : ObjectId("5d14de927fa5113224dfd3f6"),
  "empId" : 10,
  "empName" : "Uday",
  "empSal" : 85.7,
  "_class" : "com.app.document.Employee"
}
{
  "_id" : "1234ad",
  "empId" : 12,
  "empName" : "Neha",
  "empSal" : 45.6,
  "_class" : "com.app.document.Employee"
}
>
```

**Runner class#3:- run method code (with insert()):-**

**Case#1:-** if id = **null**, then generate ID and insert

```
repo.insert (new Employee(10, "Uday", 85.7));
```

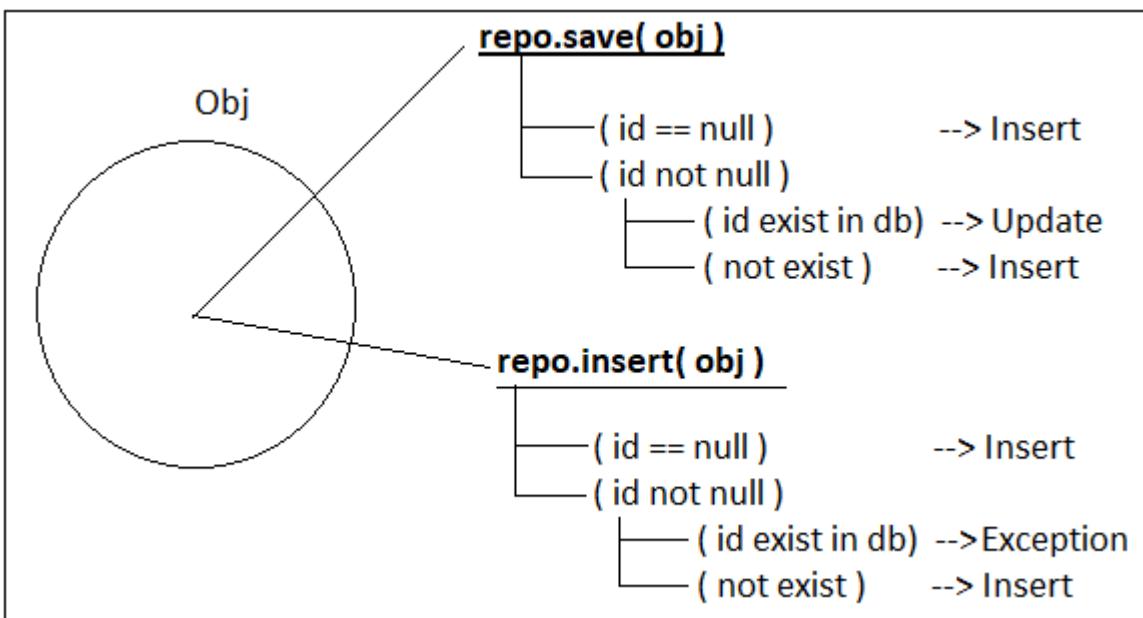
**Case#2:-** if id = **1234ad** is not exist then insert

```
repo.insert(new Employee("1234ad", 11, "Raja", 65.8));
```

**Case#3:-** if id = **1234ad** is exist then throw Exception

```
repo.insert(new Employee("1234ad", 12, "Neha", 45.6));
```

=>Code is same as above runner code just replace save with insert.



**Q> What is the difference b/w save() and insert() method in MongoDB Operations?**

Save()	Insert()
=>save() define in CrudRepository.	=>insert() method define in MongoRepository in Boot 1.7 version (not exist in before version).
=>Save() method updates if ID exist in Collection	=>where insert() throws Exception As : MongoWriteException E11000 duplicate key error collection: hello.employee index: -id-dup key: { : "1234ad"}

**NOTE:-**In other cases (if id is null or id not exist) then both behaves as insert new JSON Row only

\*\* Spring Boot MongoRepository (I) supports all pagination and sort methods for findAll() method (same as JPA).

### 3. Runner class code:--

```

package com.app.runner;
import java.util.Arrays;
import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;

```

```
import org.springframework.boot.CommandLineRunner;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Sort;
import org.springframework.data.domain.Sort.Direction;
import org.springframework.stereotype.Component;
import com.app.document.Employee;
import com.app.repo.EmployeeRepository;

@Component
public class PaginationAndSortMethod implements CommandLineRunner {

    @Autowired
    private EmployeeRepository repo;
    @Override
    public void run(String... args) throws Exception {
        repo.deleteAll();
        System.out.println("-----");
        repo.saveAll(Arrays.asList(new Employee(10, "Uday", 67.76),
            new Employee(11, "Venkat", 98.36), new Employee(12, "Neha", 37.46)
        ));

        Optional<Employee> e = repo.findById("5d1514b07fa51133787a2001");
        if(e.isPresent()) {
            System.out.println(e.get());
        } else {
            System.out.println("not found");
        }
        System.out.println("-----");

        repo.findAll().forEach(System.out::println);
        System.out.println("-----");
        repo.findAll(Sort.by(Direction.DESC, "empSal"))
            .forEach(System.out::println);
        repo.findAll(PageRequest.of(0, 2)).forEach(System.out::println);
    }
}
```

**Output:--**

```
-----  
not found  
-----|  
Employee [id=5d1516be7fa5110bd81c8943, empId=10, empName=Uday, empSal=67.76]  
Employee [id=5d1516be7fa5110bd81c8944, empId=11, empName=Venkat, empSal=98.36]  
Employee [id=5d1516be7fa5110bd81c8945, empId=12, empName=Neha, empSal=37.46]  
-----  
Employee [id=5d1516be7fa5110bd81c8944, empId=11, empName=Venkat, empSal=98.36]  
Employee [id=5d1516be7fa5110bd81c8943, empId=10, empName=Uday, empSal=67.76]  
Employee [id=5d1516be7fa5110bd81c8945, empId=12, empName=Neha, empSal=37.46]  
Employee [id=5d1516be7fa5110bd81c8943, empId=10, empName=Uday, empSal=67.76]  
Employee [id=5d1516be7fa5110bd81c8944, empId=11, empName=Venkat, empSal=98.36]  
-----
```

# CHAPTER#4: SPRING BOOT WEB-MVC

## 1. Introduction:--

=>Spring Boot has provided one starter for web application.

=>It is similar to spring WEB MVC execution process but reduces work done by programmer for,

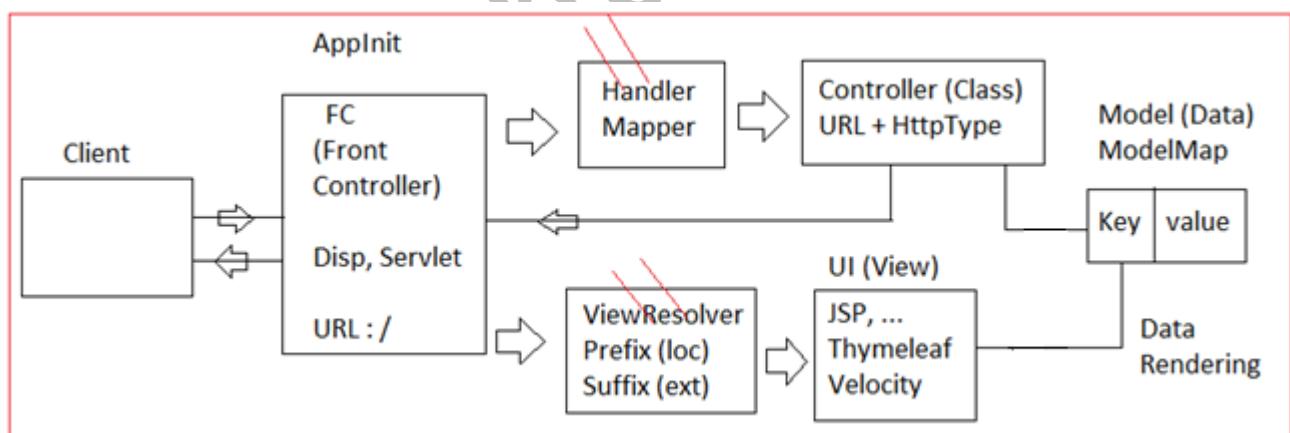
- a. Maven project creation and setup.
- b. Pom.xml dependencies and plugins.
- c. Writing common code (AppInit, AppConfig).
- d. Handle Runtime Environment and creating JAR/WARs.

=>Such process is taken care by spring boot and called as “AutoConfiguration”.

=>Even coming to Handler Mapping is configured by FC.

=>ViewResolver needs not to be configured. But Programmer has to provide (Prefix and Suffix) using properties/yml file.

=>FC (DispatcherServlet) is configured by spring boot and mapped to URL = “/”.



=>FC, ViewResolver, HandlerMapper taken care by Boot, Controller and UI files should be defined by Programmer.

**Data Rendering:**-- Reading data from Model (I) or ModelMap(C) at runtime and send to UI is known as Data Rendering, It is implemented using EL Programming.

=>Programmer should provide inputs like port number view resolver details using Properties or yml file. Example files like:

**application.properties:--**

```
server.port=9898
spring.mvc.view.prefix=/WEB-INF/views/
spring.mvc.view.suffix=.jsp
```

=>Default port no mapped to '**8080**' by using key 'server.port'. We can change even.

=>Spring boot has provided 3 embedded servers. [No download and No install]

Those are : Tomcat (default server), Jetty and Undertow.

=>In General Tomcat container 2 engines **Servlet Engine (Catalina) and JSP Engine (JASPER)**. In Spring boot, tomcat comes with only **Servlet engine**. That's why it is also called as **light weight** engine that works for "DispatcherServlet", nothing else.

=> Default Static Resource Handler is added to folder static and template which are provided under src/main/resources folder.

=>To work with JSP files in Spring Boot WEB apps, we need to add dependencies in pom.xml.

```
<dependency>
    <groupId>org.apache.tomcat.embed</groupId>
    <artifactId>tomcat-embed-jasper</artifactId>
    <scope>provided</scope> //Version taken care by Spring boot (and Tomcat)
</dependency>
```

=>To avoid/remove tomcat server (default server) from Boot application, we need to add **<Exclusion>** for Tomcat under web dependency. Given as,

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
    <exclusions>
        <exclusion>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-tomcat</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

=>To use jetty in spring boot, first we need to exclude ‘tomcat’, then add below dependency in pom.xml.

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

=>In case of ‘JBoss Undertow Server’ add below dependency.

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

=>Tomcat provided by Apache.

=>Jetty Provided by Eclipse.

=>Undertow provided by JBoss.

=>Final packing of Spring Boot Application: after coding and unit testing, our project will be converted to JAR/WAR based on <packing> type selected while creating project.

#### **Execute commands:-**

**Step#1:-** Do JDK setup.

**Step#2:-** Clear target folder.

**Step#3:-** Generate jar/war file.

=>Refresh target folder after message “build success”.

=>If pom.xml contains <packing>war</packing> then it will be converted to [ProjectName]-[version].war

=>Else it is considered as jar

[ProjectName]-[version].jar

=>To indicate Spring container, spring f/w has provided two container interfaces.

Those are.

- a. BeanFactory (I) [Legacy]
- b. ApplicationContext (I) [new]

=>In Spring boot programming, for stand-alone type application impl class used “**AnnotationConfigApplicationContext**”.

=>For Spring Boot web and related application impl class used: “**AnnotationConfigServletWebServerApplicationContext**”.

=>In case of Spring Application for web Programming Programmer has to define Configuration code as.

Ex:--

```
@Configuration
@ComponentScan (basePackage="---")
public class AppConfig {---}
```

=>Here, basePackage must be provided by programmer. If we are using Spring boot then ‘basePackage’ is set to Spring Boot Starter class package.

=>All our classes which are annotated with StereoType annotation must be under starter class package or its sub package.

=>All our classes which are annotated with stereotype annotations must be under starter class manually. It is not a good approach.

=>We can run Starter class only one time if type is web application with one port number.

=>If we want to run again, then must stop last process which is already running.

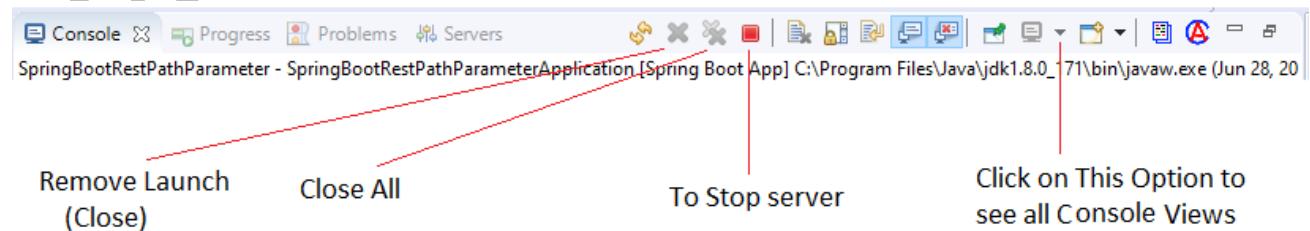
->Goto Console option.

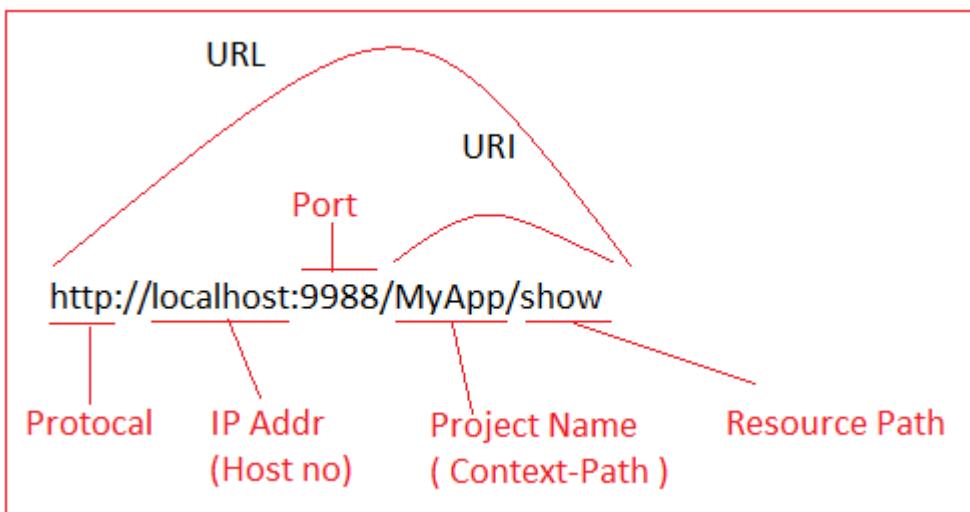
->Look at Console Symbol.

->Click on DropDown (In case of multiple).

->Click on Red Color Box Symbol (Stop Symbol).

### Diagram:--





=>URL gets changed from server to server where as URI remains same.  
=>If URL doesn't contain any PORT number then default mapped to '80' [HTTP Default port number].  
Ex:-- <http://localhost/MyApp/show>  
->In above example PORT number is : 80  
\*\*\*It means if server is running on PORT number: 80, then PORT number not required to write in URL.

#### Ex#1:--

##### **application.properties:--**

```
server.port=9898
server.servlet.context-path=/myapp
spring.application.name=SAMPLEAPP
```

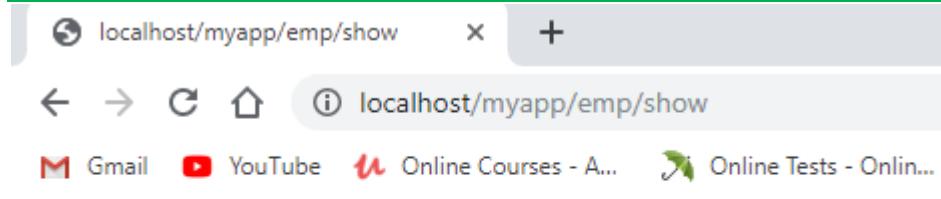
=>then sample URL : <http://localhost:9898/myapp>

#### Ex#2:--

##### **application.properties:--**

```
server.port=80
server.servlet.context-path=/myapp
```

=>then sample URL : <http://localhost:80/myapp> or  
<http://localhost:/myapp>



## Welcome to Boot!!

Welcome App:Fri Jun 28 18:18:26 IST 2019

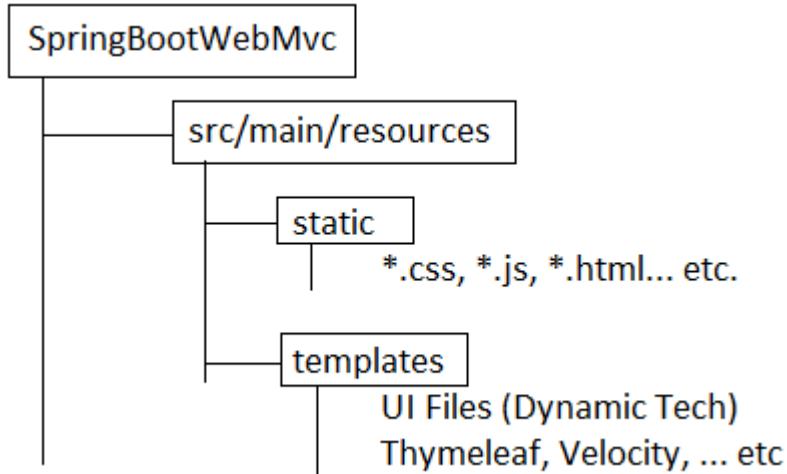
**Ex#3:--**

**application.properties:--**

server.servlet.context-path=/myapp

=>Then Sample URL : <http://localhost:8080/myapp>

**Q>Where to place CSS/JS/HTML files in Spring Boot web applications?**



**Coding Steps:--**

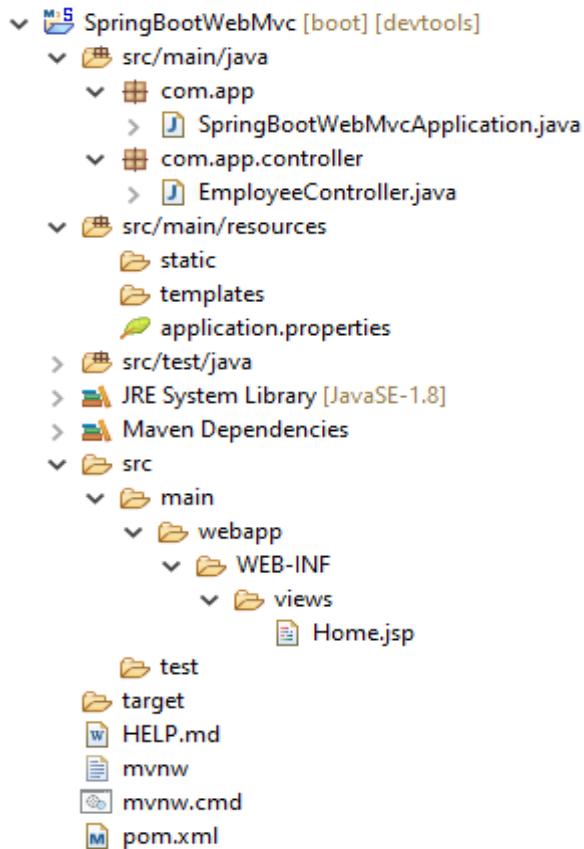
**Step#1:-** Create one Spring Boot Starter application and choose dependencies 'Spring Web Starter'.

**Spring Boot Web-MVC Dependency:--**

```

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

## #22. Folder Structure of WEB-MVC Project:-



**Step#2:-** Open pom.xml file and add below dependency to work with JSP pages only.

```

<dependency>
  <groupId>org.apache.tomcat.embed</groupId>
  <artifactId>tomcat-embed-jasper</artifactId>
  <scope>provided</scope>
</dependency>
  
```

**Step#3:-** Define application.properties given below

```

## Server keys ##
server.port=9898
server.servlet.context-path=/myapp
## View Resolver Keys ##
spring.mvc.view.prefix=/WEB-INF/views/
spring.mvc.view.suffix=.jsp
  
```

**Step#4:-** Create folders

- webapp folder under main
- WEB-INF under webapp
- views under WEB-INF

**Step#5:-** Define Controller class under 'src/main/java'

```
package com.app.controller;
import java.util.Date;
import org.springframework.stereotype.Component;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

@Component
@RequestMapping("emp")
public class EmployeeController {
    @RequestMapping("show")
    public String showPages(Model m) {
        m.addAttribute("msg", "Welcome App:" + new Date());
        return "Home";
    }
}
```

**Step#6:-** Define JSP Page under 'views' folder

=>Right click on 'views' folder => new => other

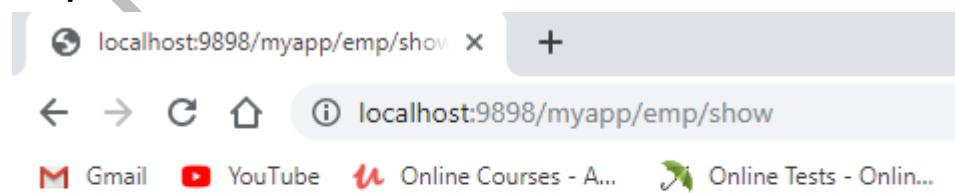
=>Search and choose 'JSP file' => Next

=>Enter name ex : Home.jsp

**Home.jsp:--**

```
<html><body><h2>Welcome to Boot!!</h2>
    ${msg}
</body></html>
** Run Starter class and Enter URL in Browser
http://localhost:9898/myapp/emp/show
```

**Output:--**



## Welcome to Boot!!

Welcome App:Fri Jun 28 18:12:43 IST 2019

**Q>What is the difference between Model (I) and ModelMap (C) ?.**

**A> ModelMap:**-- It is a shared Memory holds data in key=val which allocates memory based on data added to it In simple onDemand Memory Creation.

**B> Model:**-- Model also similar to ModelMap, but un-used key=val pairs (at UI) are removed from memory.

**NOTE:**-- In above example “Applnt” and AppConfig” are provided by spring Boot. Annotation like @EnableWebMvc, @ComponentScan, @PropertySource not required to provide.

### Spring Boot Development to Deployment Process (End-to-end Application process) :-

Part#1:- WEB MVC +Data JPA + ORACLE DB (or any Embedded Data).

Part#2:- UI Design (Bootstrap).

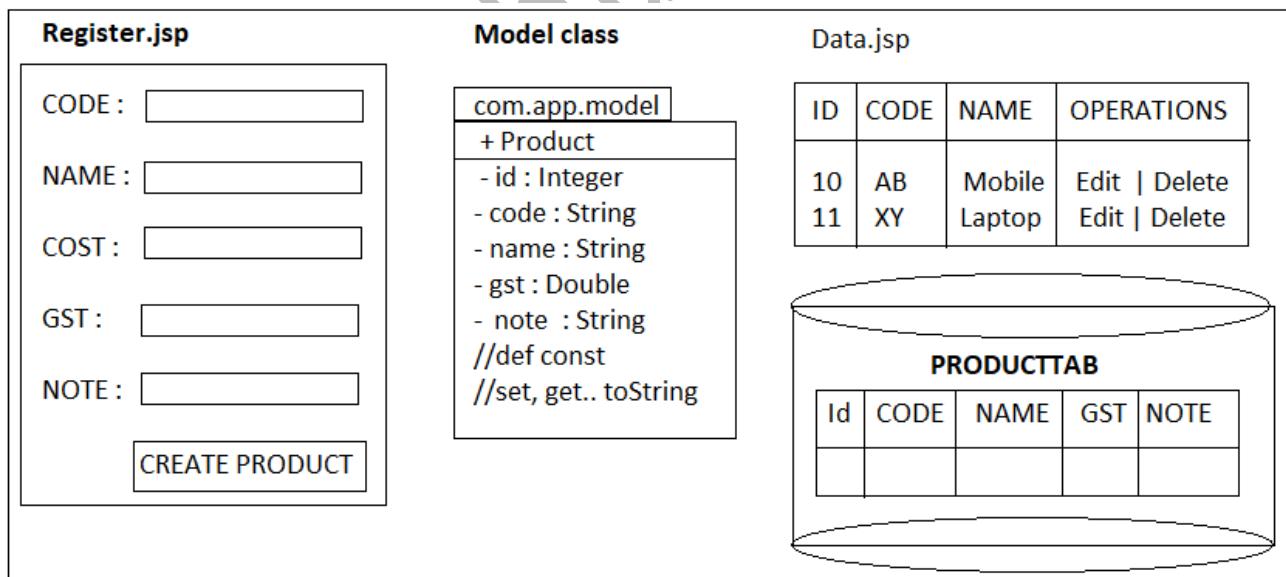
Part#3:- Cache Management.

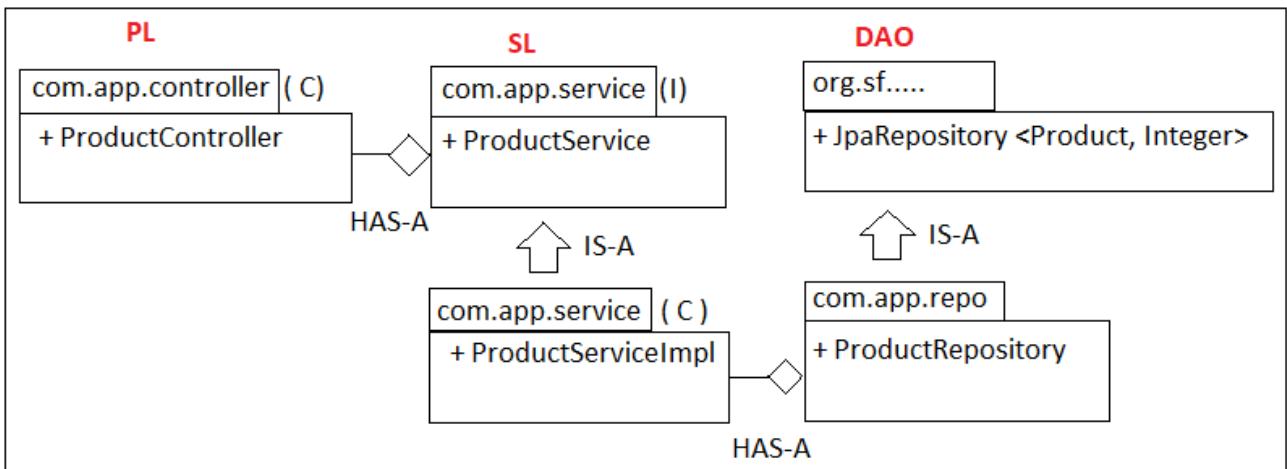
Part#4:- Email configuration programming.

Part#5:- Logging, UnitTesting and Integration with GIT.

Part#6:- Cloud deployment (PCF =pivotal Cloud foundry).

### Design:-



**UML Diagram:--****Part#1:- Setup and code files:--**

**Step#1:-** Create a web project with dependencies web, spring data jpa, lombok, mySQL, h2 or any database dependency (Oracle), jstl, jasper, hikari.

**Step#2:-** application.properties

DataSource, JPA, Server, Mvc (Suffix, Prefix)

**Step#3:-** Model class (Product).

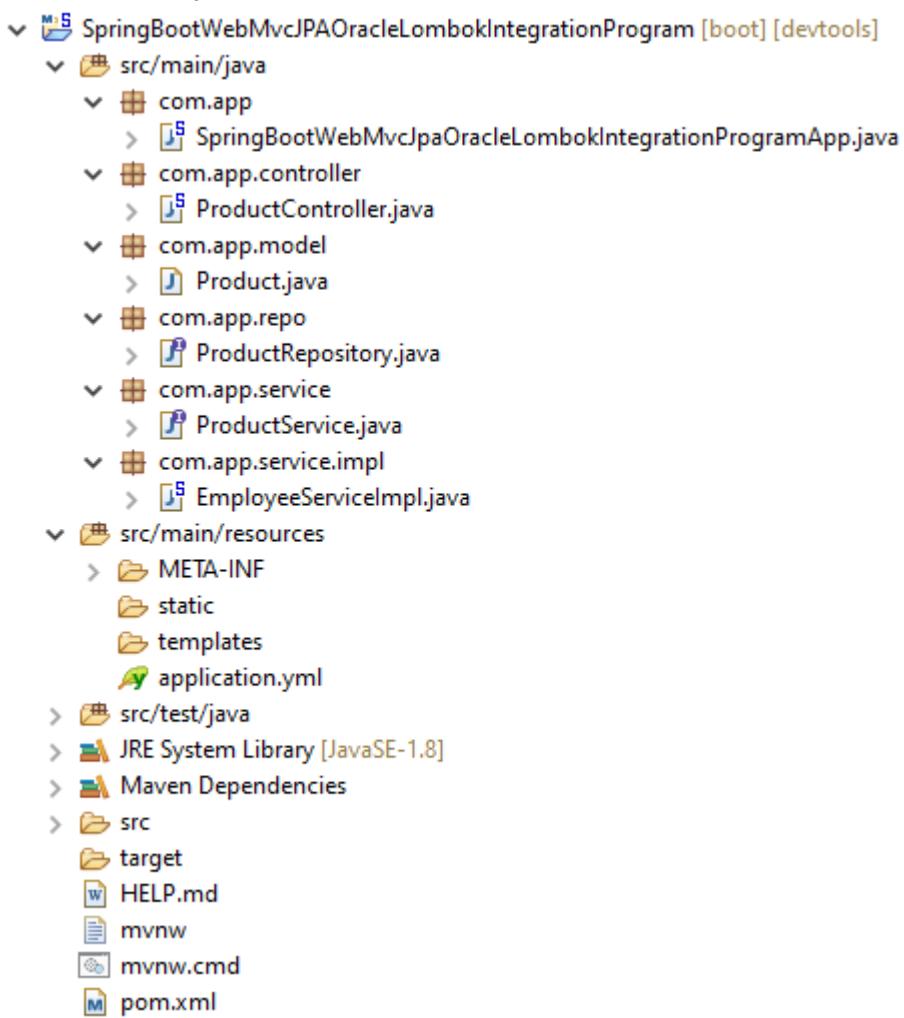
**Step#4:-** ProductRepository Interface.

**Step#5:-** ProductService and ServiceImpl.

**Step#6:-** ProductController.

**Step#7:-** UI Files : Register.jsp, Data.jsp.

## #23. Folder Structure of MVC Mini Project (Using Oracle DB, Tomcat Server, Lombok API, Data JPA) :-



### 1. application.properties:--

```
##Server##
server.port=2019
server.servlet.context-path=/myapp

##WEB MVC##
spring.mvc.view.prefix=/WEB-INF/views/
spring.mvc.view.suffix=.jsp

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

```
##JPA##  
spring.jpa.show-sql=true  
spring.jpa.hibernate.ddl-auto=create  
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.Oracle10gDialect  
spring.jpa.properties.hibernate.format_sql=true
```

**application.yml:--**

```
##Server##  
server:  
    port: 2019  
    servlet:  
        context-path: /myapp
```

**##WEB MVC##**

```
spring:  
    mvc:  
        view:  
            prefix: /WEB-INF/views/  
            suffix: .jsp
```

**##DataSource##**

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

**##JPA##**

```
jpa:  
    show-sql: true  
    hibernate:  
        ddl-auto: create  
    properties:  
        hibernate:  
            dialect: org.hibernate.dialect.Oracle10gDialect  
            format_sql: true
```

F3 :- Goto code (Go from usage to definition).

F4 :- See all Implementation Interface / class.

Ctrl + shift + R :- Enter full name

## 2. Model class (Product.java):--

```
package com.app.model;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;
import lombok.Data;

@Entity
//@Table(name="PRODUCTTAB")
@Data
public class Product {

    // @Id
    @Column(name="id")
    @GeneratedValue
    private Integer id;
    @Column(name="code")
    private String code;

    @Column(name="name")
    private String name;
    @Column(name="cost")
    private Double cost;
    @Column(name="gst")
    private String gst;
    @Column(name="note")
    private String note;
}
```

**3. Repository Interface (ProductRepository.java):--**

```
package com.app.repo;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import com.app.model.Product;
```

```
@Repository
```

```
public interface ProductRepository extends JpaRepository <Product, Integer> {
    Product getProductById(Integer id);
}
```

**4. Service Interface (ProductService.java):--**

```
package com.app.service;
import java.util.List;
import com.app.model.Product;
```

```
public interface ProductService {
```

```
    public Integer saveProduct(Product p);
    public List<Product> getAllProducts();
    public void deleteProduct(Integer id);
    public Product getProductById(Integer id);
```

```
}
```

**5. Service Impl class (ProductServiceImpl.java):--**

```
package com.app.service.impl;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import com.app.model.Product;
import com.app.repo.ProductRepository;
import com.app.service.ProductService;
```

```
@Service
```

```
public class EmployeeServiceImpl implements ProductService {
```

```
    @Autowired
```

```
    private ProductRepository repo;
```

```

//1. Save method
@Transactional
public Integer saveProduct(Product p) {
    //calculations here..
    //gstAmt= cost*gst/100
    //totalAmt=cost+ gstAmt - disc
    p=repo.save(p);
    return p.getId();
}

//2. Get all Product details from Database
@Transactional(readOnly= true)
public List<Product> getAllProducts() {
    return repo.findAll();
}

//3. Delete Record based on ID
@Transactional
public void deleteProduct(Integer id) {
    repo.deleteById(id);
}

//4. Get Record based on ID
@Transactional
public Product getProductById(Integer id) {
    return repo.getProductById(id);
}
}

```

#### **6. Controller class (ProductController.java):--**

```

package com.app.controller;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import com.app.model.Product;

```

```
import com.app.service.ProductService;

@Controller
@RequestMapping("/emp")
public class ProductController {

    @Autowired
    private ProductService service;

    //1. Show Product Form with Backing Object
    @RequestMapping("/reg")
    public String showReg(Model map) {
        //Form Backing Object
        map.addAttribute("product", new Product());
        return "Register";
    }

    //2. Read Form Data on click submit
    @RequestMapping(value="/save",method=RequestMethod.POST)
    public String saveData(@ModelAttribute Product product, Model map)
    {
        //call service layer
        Integer id=service.saveProduct(product);
        map.addAttribute("message", "Product "+id+" created!!");
        //clean Form Backing Object
        map.addAttribute("product", new Product());
        return "Register";
    }

    //3. Fetch all Rows from DB to UI
    @RequestMapping("/all")
    public String showAll(Model map) {
        //fetch all rows from DB
        List<Product> obs=service.getAllProducts();
        //send to UI
        map.addAttribute("list", obs);
        return "Data";
    }
}
```

```

//4. Delete row based on ID
@RequestMapping("/delete")
public String remove(@RequestParam Integer id) {
    //delete row based on ID
    service.deleteProduct(id);
    //response.sendRedirect
    return "redirect:all";
}

//5.Show edit Page
@RequestMapping(value="edit")
public String showEdit(@RequestParam Integer id, Model map)
{
    //Load Objet from DB
    Product p =service.getProductById(id);
    //From BACKING OBJECT
    map.addAttribute("product", p);
    map.addAttribute("Mode", "EDIT");
    return "Register";
}
}

```

## 7. Home.jsp:--

```

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1" isELIgnored="false"%>
<%@taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<!DOCTYPE html>
<html><head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h3>WELCOME TO PRODUCT REGISTER</h3>
<form:form action="save" method="POST" modelAttribute="product">
<pre>
CODE : <form:input path="code"/><br>
NAME : <form:input path="name"/><br>
COST : <form:input path="cost"/><br>

```

```

GST : <form:select path="gst"><br>
      <form:option value="5">5%-SLAB</form:option>
      <form:option value="12">12%-SLAB</form:option>
      <form:option value="18">18%-SLAB</form:option>
      <form:option value="22">22%-SLAB</form:option>
      <form:option value="30">30%-SLAB</form:option>
</form:select><br>
NOTE : <form:textarea path="note"/><br>
<input type="submit" value="CREATE PRODUCT"/>
</pre>
</form:form>
${message}
</body>
</html>

```

#### 8. Data.jsp (To Display all records in browser):--

```

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h3>WELCOME TO PRODUCT DATA</h3>
<table>
  <tr>
    <th>ID</th>
    <th>CODE</th>
    <th>NAME</th>
    <th>COST</th>
    <th>GST</th>
    <th>NOTE</th>
    <th colspan=2>OPERATIONS</th>
  </tr>

```

```

<c:forEach items="${list}" var="ob">
    <tr>
        <td><c:out value="${ob.id}" /></td>
        <td><c:out value="${ob.code}" /></td>
        <td><c:out value="${ob.name}" /></td>
        <td><c:out value="${ob.cost}" /></td>
        <td><c:out value="${ob.gst}" /></td>
        <td><c:out value="${ob.note}" /></td>
        <td>
            <a href="delete?id=${ob.id}">DELETE</a>
            <a href="edit?id=${ob.id}">EDIT</a>
        </td>
    </tr>
</c:forEach>
</table>
</body></html>

```

=>Run the application and type below URL one by one

1><http://localhost:2019/myapp/emp/reg>

2><http://localhost:2019/myapp/emp/all>

### Output:-

localhost:2019/myapp/emp/save

Gmail YouTube Online Courses - A... Online Tests - Onlin...

### WELCOME TO PRODUCT REGISTER

CODE :	<input type="text" value="APPX"/>
NAME :	<input type="text" value="APPLE X"/>
COST :	<input type="text" value="98000"/>
GST :	<input type="text" value="5%-SLAB ▾"/>
It is a Good Product.	
NOTE :	<input type="text" value=""/>
<b>CREATE PRODUCT</b>	

Product '2' created!!

**Code: For Edit Operation :-**

**Step#1:- Add below Hyperlink in Data.jsp**

```
<td><a href="edit?id=${ob.id}">EDIT</a></td>
```

**Step#2:- Add below method in ProductController**

```
//5.Show Edit Page
@RequestMapping("edit")
public String showEdit(@RequestParam Integer id, Model map)
{
    //Load Objet from DB
    Product p =service.getProductById(id);
    //From BACKING OBJECT
    map.addAttribute("product", p);
    map.addAttribute("Mode", "EDIT");
    return "Register";
}
```

**#3:-- Do below modifications in “Register.jsp” Add code:-**

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
Under <form>after<pre> tag)
<pre>
ID : <form:input path="id" readOnly="true"/>
<c:if>
=>In button, at submit button place
<c:choose>
    <c:when test="${'EDIT' eq Mode}">
        <input type="submit" value="UPDATE"/>
    </c:when>
    <c:otherwise>
        <input type="submit" value="CREATE"/>
    </c:otherwise>
</c:choose>
```

=>After Adding above code in Home.jsp:--

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
```

```

pageEncoding="ISO-8859-1" isELIgnored="false" %>
<%@taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<!DOCTYPE html>
<html><head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head><body>
<h3>WELCOME TO PRODUCT REGISTER</h3>
<form:form action="save" method="POST" modelAttribute="product">
<pre>
ID : <form:input path="id" readOnly="true"/><br>
CODE : <form:input path="code"/><br>
NAME : <form:input path="name"/><br>
COST : <form:input path="cost"/><br>
GST : <form:select path="gst"><br>
        <form:option value="5">5%-SLAB</form:option>
        <form:option value="12">12%-SLAB</form:option>
        <form:option value="18">18%-SLAB</form:option>
        <form:option value="22">22%-SLAB</form:option>
    </form:select><br>
NOTE : <form:textarea path="note"/><br>
<c:choose>
    <c:when test="${'EDIT' eq Mode}">
        <input type="submit" value="UPDATE"/>
    </c:when>
    <c:otherwise>
        <input type="submit" value="CREATE"/>
    </c:otherwise>
</c:choose>
</pre>
</form:form>
${message}
</body></html>
*)choose-when-otherwise (JSTL Core Tags) behaves as switch-case in sample. Here
'test=' indicates condition check (Boolean).

```

## 2. Hot Deployment using DevTools:--

=>If we do any modification in application files after starting server, Boot will not update them into server (war/jar) unless we started i.e. to see changes “STOP and START” server again.

=>To avoid this process “**Spring Boot DevTools**” are introduced which runs as parallel execution and links to war file.

=>It is also called as **HotDeployment/ HotSwapping**. This process is used only to save development time not recommend using in Production.

=>DevTools executions LiveReload in server on port **35729**

### Dependency:--

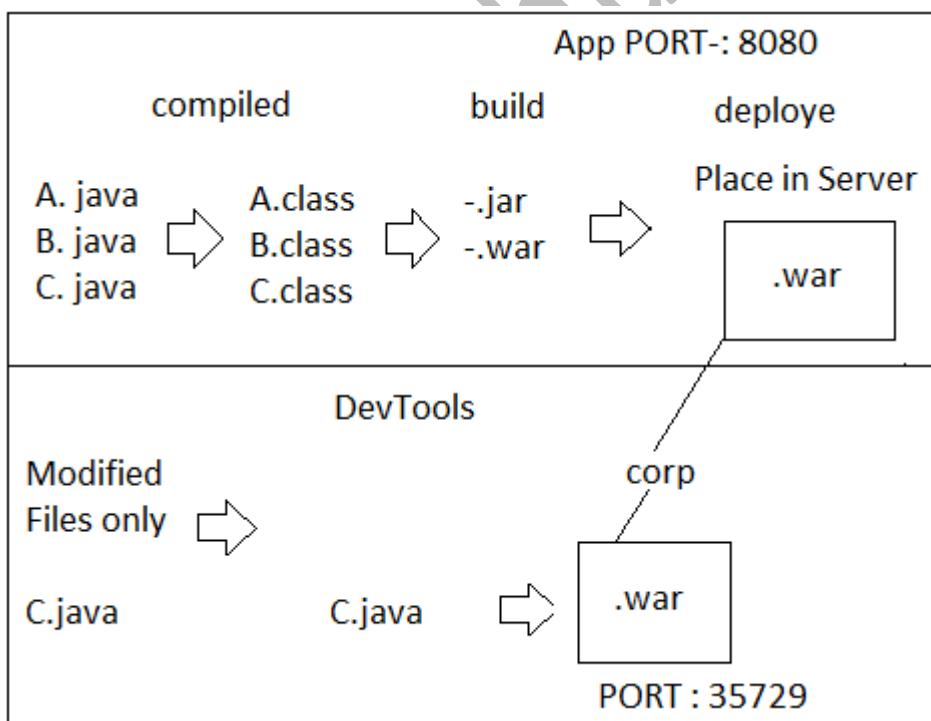
<dependency>

```
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-devtools</artifactId>
<scope>runtime</scope>
```

<dependency>

=>Choose Spring DevTools while creating your project.

### Diagram:--



**Deployment:--** Place jar/war in server and start server.

**HotDeployment**-- Without stopping server, update only new files into jar/war which is in server.

**NOTE**--

1. To change PORT Number use key  
**spring.devtools.liveload.port=---**
2. By default few files are not re-loaded by devTools  
Given using key "**spring.devTools.restart.exclude**"

**Locations are:-**

META-INF/maven/**	META-INF/resources/**,
resources/**	static/**,
public/**	template/**,
**/*Test.class	**/*Tests.class,
git.properties,	META-INF/build-info.properties

1. To avoid extra files which need to be re-loaded can be given key (with example values) **spring.devtools.restart.additional-exclude=/sample/app.properties**

=>To specify location which need to be included for re-loading (key-val example) use key: **spring.devtools.restart.additional-paths=static/\*\*, templates/\*\***

### 3. Thymeleaf (UI):--

=>It is a UI technology used to implement Dynamic web pages with **Lightweight UI engine**.

=>Compared to JSP its coding and memory is less and execution is faster.

=>JSP (JASPER) is a heavy weight engine; Thymeleaf is a light engine (less memory).

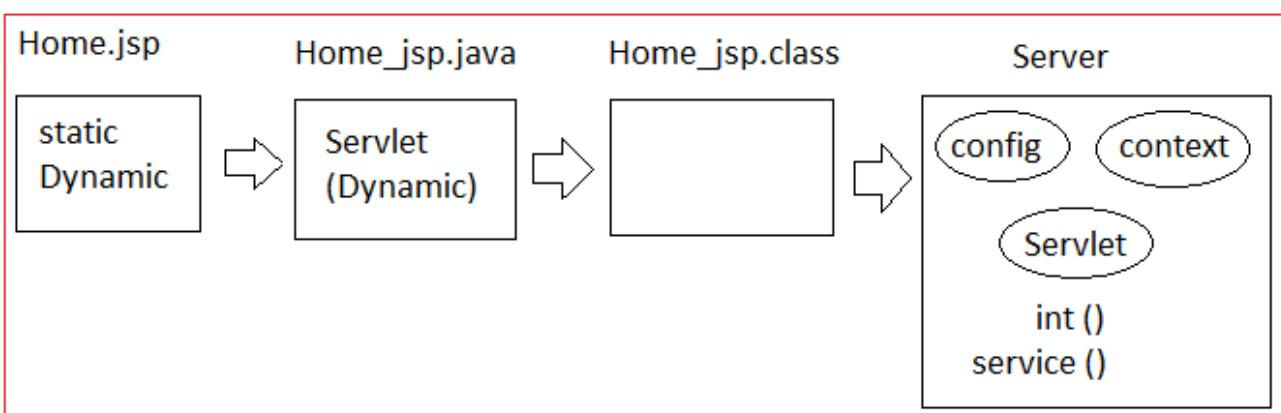
**JSP Work flow**--

- a>Translation Phase
- b>Compilation Phase
- c>Execution Phase

**a>Translation**-- In this phase both static content and dynamic content are converted to JAVA format.

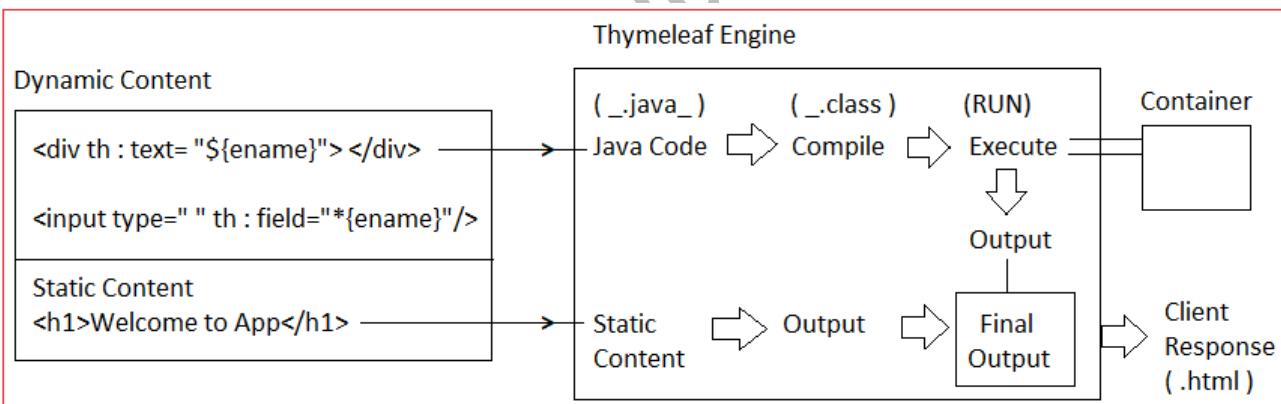
**b>Compilation**-- JASPER converts \_\_.java file into \_\_.class format (heavy weight file = static code in dynamic format).

**Execution:**-- \_\_.class is loaded into server and Servlet execution process is started.  
Here multiple objects are created like config, context, Servlet, Request, Response... etc.



### Thymeleaf work flow:-

- =>Boot supports working with Thymeleaf UI engine which converts only Dynamic content into its equal java code, static content is placed as it is.
- =>Dynamic content will be converted to java format and then compiled, finally executed and mixed with static output.
- =>It may read data from spring container using Thymeleaf EL.



=>To enable Thymeleaf UI in Spring boot apps add below starter in pom.xml.

```

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

```

=>Thymeleaf tags(code) starts with prefix “**th**”.

Ex:- th:action, th: href, th:object ... etc.

=>Thymeleaf supports Symbols like ‘@’, ‘\$’, and ‘\*’ for dynamic coding.

    @{url},    \${EL},    \*{Link}

=>Here @{} Symbol indicates base URL/Upto contextpath.

Consider example URL : <http://localhost:8080/myapp/emp/req>

Can be written using Thymeleaf as : @{/emp/req}.

=>Using css file in Thymeleaf

```
<link rel="stylesheet" th:href="@{/css/myui.css}" />
```

=>Using Java script files in Thymeleaf

```
<script type="text/javascript" th:src="@{/js/jquery.js}"></script>
```

=>Using forms in Thymeleaf:

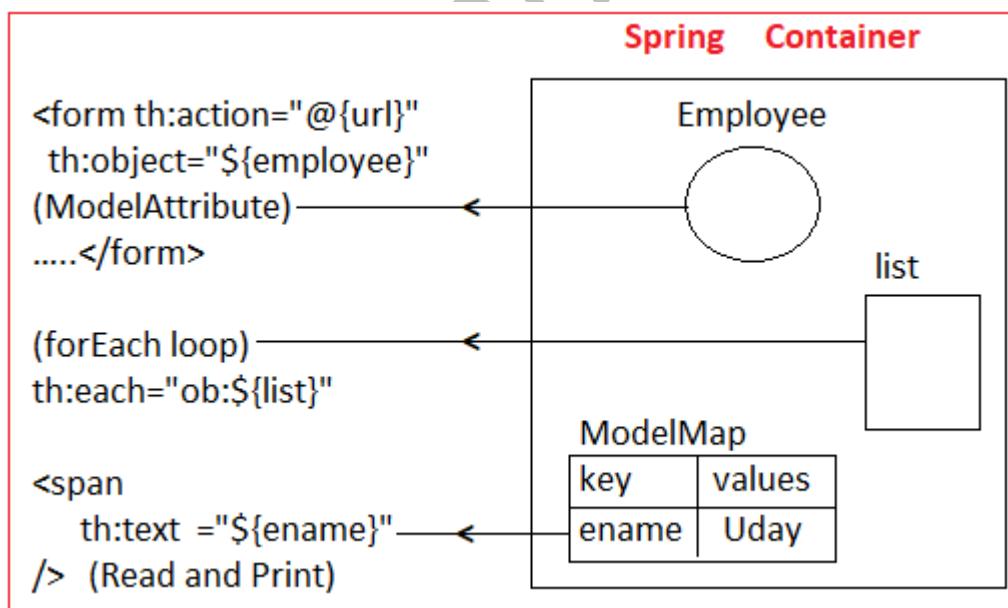
```
<form action="#" th:action="@{/student/register}" ....
```

=>using Hyperlinks in Thymeleaf:

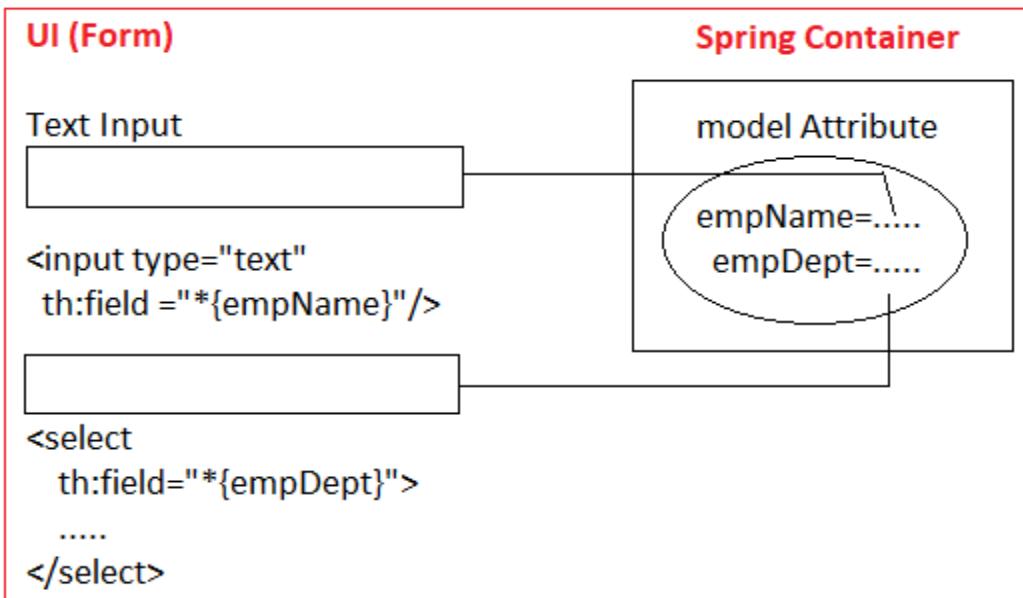
```
<a th:href="@{/student/export}" EXPORT</a>
```

=>Here \${ } symbol indicates EL in Thymeleaf. It supports data reading from spring container(may read data from memory).

=>UI form can read data using ModelAttribute, Text Data, can be taken from modelMap or a Collection data can be read using for each loop... by using EL.



=>Here \*{} Symbol indicates link to variable/Property in ModelAttribute. This is used incase of UI forms.

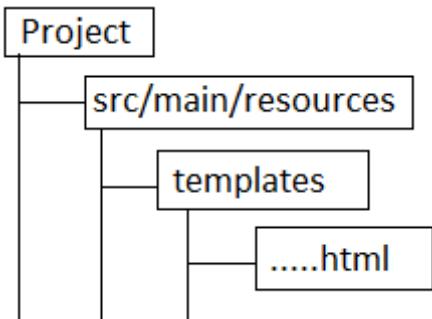


### Data Exchange between UI and Controller using Thymeleaf:--

=>We can send data from controller to UI using **ModelMap**. Here, Thymeleaf UI reads data using EL and TH NAMESPACE.

=>To use Thymeleaf tags we should provide Thymeleaf Engine Location to current UI page, as:

```
<html xmlns:th="http://www.thymeleaf.org"/>
```



=>To read data ModelMap and print at UI used Thymeleaf code.

th:text="\\${key}"

Using any Html tag example

```
<div th:text="\${message}"></div>
<span th:text="\${emp}"></span>
<td th:text="\${ob.empId}"></td>
```

=>To read collection data and Iterate (loop) use Thymeleaf code.

th:each="tempvar:\\${CollectionKey}"

**Example:--**

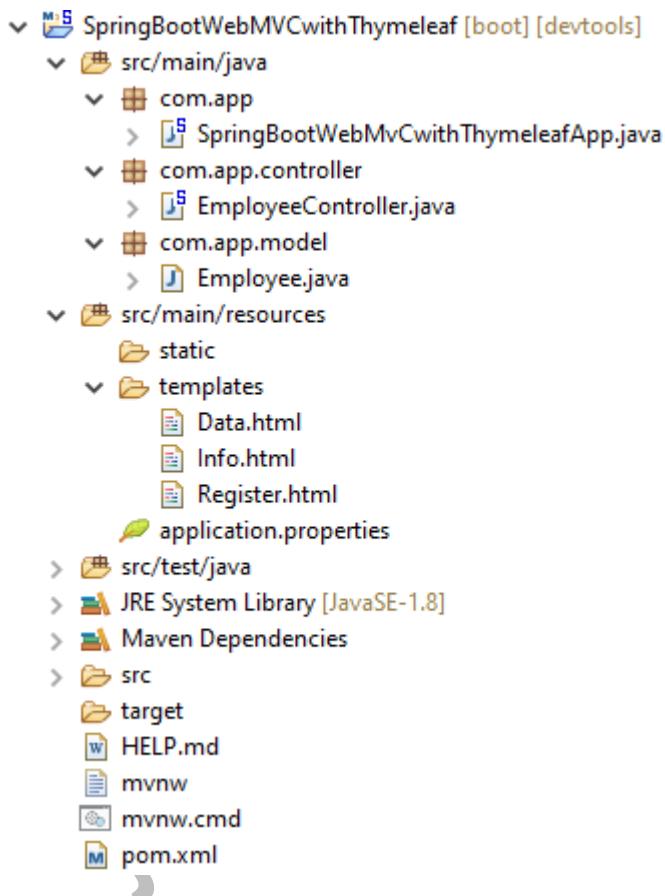
```
<tr th:each="ob:${products}">
<td th:text="${ob.prodCode}"></td>
<td th:text="${ob.prodCost}"></td>
</tr>
```

**Code:--**

**Step#1:-** Create one Spring boot Starter Project

Name : SpringBootWebMVCWithThymeleaf

Dependencies : Web, Thymeleaf

**#24. Folder Structure of Thymeleaf Application:--****1. Model class:--**

```
package com.app.model;
import lombok.Data;
import javax.persistence.Id;
import javax.persistence.Table;
```

```

@Data
@Entity
@Table(name="employeeTab")
public class Employee {

    @Id
    private Integer empld;
    private String empName;
    private Double empSal;
}

```

## 2. Controller class:-

```

package com.app.controller;
import java.util.Arrays;
import java.util.List;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotationModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import com.app.model.Employee;

@Controller
public class EmployeeController {

    @RequestMapping("/reg")
    public String regPage(ModelMap map) {
        //Form Backing Object
        Employee e= new Employee();
        map.addAttribute("employee", e);
        return "Register";
    }

    @RequestMapping(value="/save", method=RequestMethod.POST)
    public String saveData(@ModelAttribute Employee employee, ModelMap map) {
        map.addAttribute("emp", employee);
    }
}

```

```

        return "Info";
    }

    @RequestMapping("/all")
    public String showDates(ModelMap map) {
        map.addAttribute("message", "Hello UI");
        List<Employee> emps=Arrays.asList(
            new Employee(10, "Uday", 2.2),
            new Employee(11, "Venkat", 6.3),
            new Employee(12, "Neha", 9.8)
        );
        map.addAttribute("list", emps);
        return "Data";
    }
}

```

**Step#3:-** UI Pages (Under templates folder)

**a. Register.html:--**

```

<html xmlns:th="http://www.thymeleaf.org">
<body><h3>Welcome to Register Page</h3>
<form action="#" method="post" th:action="@{/save}" th:object="${employee}">

    ID : <input type="text" th:field="*{empId}"><br><br>
    NAME : <input type="text" th:field="*{empName}"><br><br>
    SALARY : <input type="text" th:field="*{empSal}"><br><br>

    <input type="submit" value="Create"/>
</form>
</body></html>

```

**b. Info.html:--**

```

<html xmlns:th="http://www.thymeleaf.org">
<body>
    Your form data is : <div th:text="${emp}"></div>
</body>
</html>

```

**c. Data.html:--**

```

<html xmlns:th="http://www.thymeleaf.org/">
<body><h3>Welcome to data Page</h3>
<span th:text="${message}"></span>
<table>
<tr>
<th>ID</th>
<th>NAME</th>
<th>SALARY</th>
</tr>
<tr th:each="ob:${list}">
<td th:text="${ob.empId}"></td>
<td th:text="${ob.empName}"></td>
<td th:text="${ob.empSal}"></td>
</tr>
</table>
</body></html>

```

**Execution:**-- Run the application and type below urls.

1><http://localhost:2017/myapp/employee/reg>

2><http://localhost:2017/myapp/employee/all>

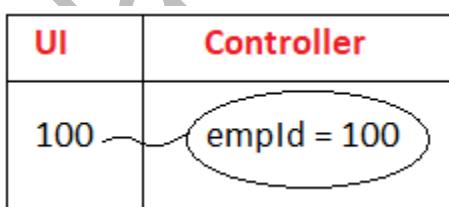
#### NOTE:--

**#1:**-- Here th:field="\*{variable}" links one form Input /select/text area with one ModelAttribute( Form Backing Object) variable.

Example, consider below code

<input type="text" th:field="\*{empId}">

Then it looks like.



**#2:**-- At <form> tag level we need to provide URL (action) using format.

th:action="@{/path}"

Ex:-- <form.... th:action="@{“save”}...>

At runtime @ will be replaced with full path (base Url)

ex:-- <http://localhost:9898/save>

**#3:**--To indicate ModelAttribute (Form Backing Object) use syntax:

th:object="\${modelAttribute(name)}"

Ex:-- <form... th:object="\${employee}"></form>

=>It creates two way binding. It means form data will be converted to object and object data can be shown at UI form.

**#4:**--All these Thymeleaf tags/attributes are defined in Thymeleaf engine. Its location (xml Namespace) must be provided at UI file level using code:

<html xmlns:th=<http://www.thymeleaf.org/>>

#### Thymeleaf Sample code:--

#a :-- Link CSS file with UI file

<link rel="stylesheet" th:href="@{/---.css}">

#b :-- Link Javascript files with UI file

<script type="text/javascript" th:src="@{/ / .js}"></script>

#c :-- Define one form.

<form action="#" th:action="@{/url}" th:object="\${modelClassObject}">

#d :-- Input field (variable) linking

<input type="—" th:field="\*{variable}">

#e :-- Display text taken from container

<span th:text="\${message}"></span>

#f :-- Read List (Collection) and print using for-each

<tr th:each="ob : \${list}">

  <td th:text="\${ob.variable}"></td>

.....

</tr>

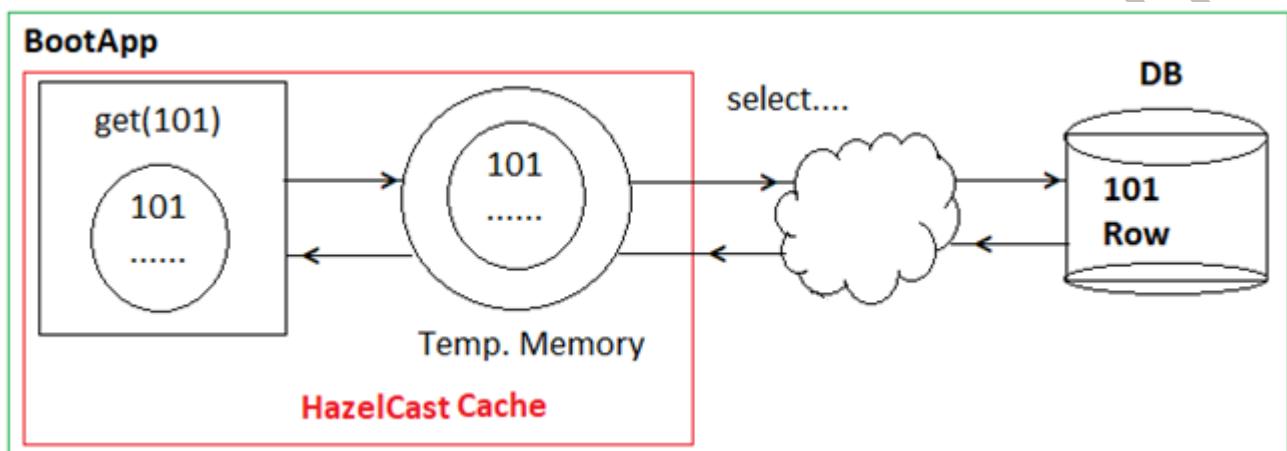
#### 4. Cache Management in Spring Boot:-

=>Cache is **Temp memory** used at application side to reduce network calls between Application and Database for commonly accessed data.

=>Caches are used to save time for network calls that improves the app performance.

=>Cache should never hold **large data**, apply only for few modules which are accessed more times by client.

\*\*\*Do't apply cache for **findAll()** type method



=>To implement Cache Management in Spring Boot Application. We should use any one vendor. ex: HazelCast Cache

=>First, we need to indicate "Create Cache Memory at application side, using class **Config (com.hazelcast.config)**".

\*\*\*This Cache Memory can also be called as "**HazelCast Instance**".

=>To enable cache for one module, use **MapConfig(C)** object and provide details like "name, lifeTime, cacheSize, Eviction..." one time.

=>This MapConfig (C) can be repeated for multiple module [i.e one module = one MapConfig object].

=>MapConfig stores data in key=val format Here key=PrimaryKey (ID) and val=Model class object.

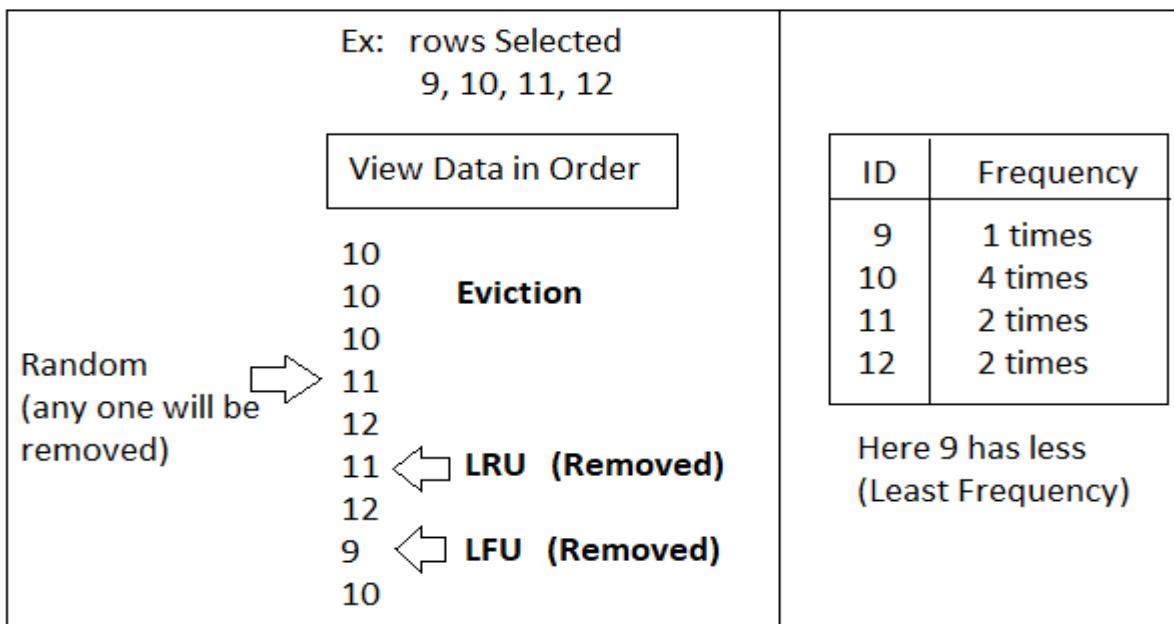
=>EvictionPolicy is an enum defined with Eviction possible values. It means removing one object from cache based on Condition.

=>EvictionPolicy Possible values and meaning are.

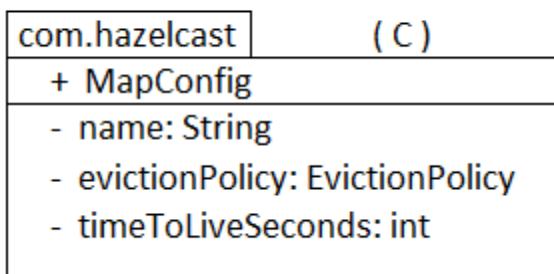
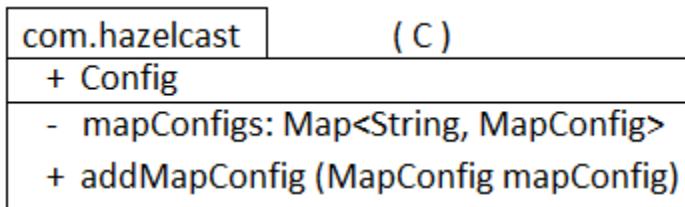
- ❖ LRU:- Least Recently used.
- ❖ LFU:- Least Frequently Used.
- ❖ NONE:- Don't remove any Object.
- ❖ RANDOM:- Any one Object.

**Frequency**-- No. of times object is accessed.

**Recent**-- Which is seen last in order (or) latest object.

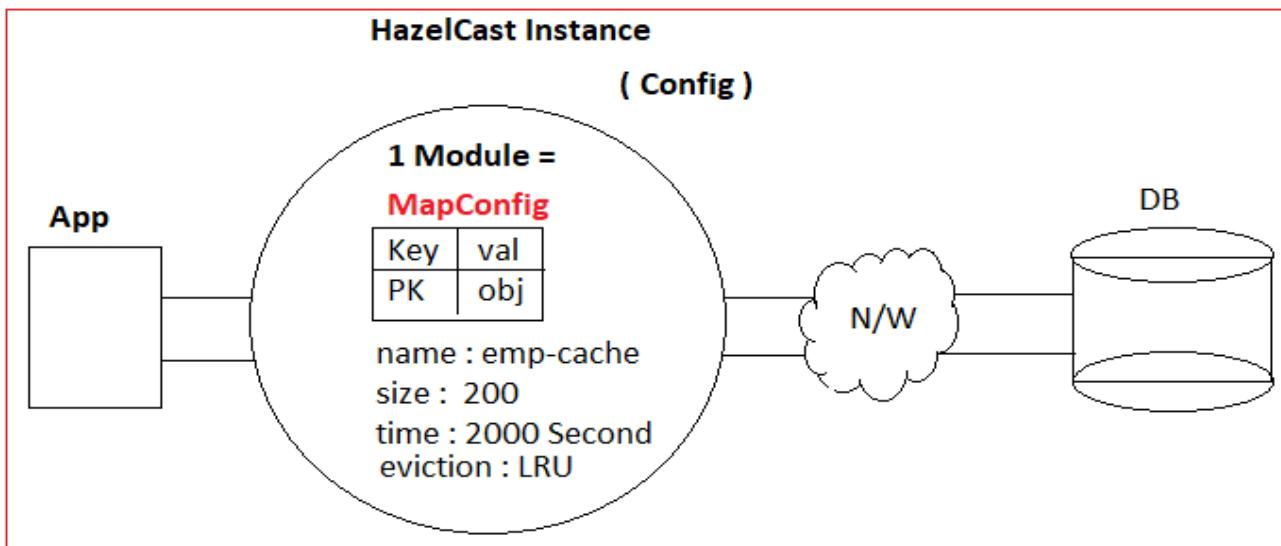


### UML DESIGN:--



=>Here, Config is used to create Cache Instance where as MapConfig is used to module memory. Example, looks like below.

**Hazelcast-cache Config** :-- It is a 3<sup>rd</sup> party Cache configuration process, supported for any java application caching.



**Step#1:-** Provide Dependencies for Starter-cache, HazelCast and Hazelcast-spring integration.

=>Choose **Spring cache abstraction** dependency

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

**Step#2:-** On starter class level we need to specify enable annotation for Cache Management.

**@EnableCaching**

**Step#3:-** Model class which needs cache support must implement one interface java.io.serializable

Ex:-- class Employee implements Serializable { }

**Step#4:-** Define Java config class for HazelCast Cache Configuration and MapConfig Objects code looks like:  
=> In application write Java configuration code for Config (C) (com.hazelcast).

```
package com.app.config;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import com.hazelcast.config.Config;
import com.hazelcast.config.EvictionPolicy;
import com.hazelcast.config.MapConfig;
import com.hazelcast.config.MaxSizeConfig;
import com.hazelcast.config.MaxSizeConfig.MaxSizePolicy;

@Configuration
public class MyCacheConfig {

    @Bean
    public Config cacheConfig() {
        return new Config()
            .setInstanceName("hazelCast-instance")
            .addMapConfig(
                new MapConfig()
                    .setName("emp-cache")
                    .setTimeToLiveSeconds(2000)
                    .setMaxSizeConfig(
                        new MaxSizeConfig(200, MaxSizePolicy.FREE_HEAP_SIZE))
                    .setEvictionPolicy(EvictionPolicy.LRU)
                );
    }
}
```

**Step#5:-** At service Layer methods need to apply cache annotation given by spring.

- a. **@Cacheable**:- This one must be applied over select method (getOneById).
- b. **@CacheEvict**:- This one must be applied over delete method (deleteById).
- c. **@CachePut**:- CachePut annotation is used to update object in cache before updating in Database.

A >Over getOne (findOne) method **@Cacheable(value="cache-name", key="#PKId")**  
b>Over delete method (delete byId) **@CacheEvict(vale="cache-name", key="#PKId")**

-----Example service layer methods with cache annotation:-----

```
@Service
public class Employee {

    @Cacheable (value="emp-cache", key="#emplId")
    public Employee getOneEmployee(Integer emplId) {
        //select * from empTab where eid=?
    }

    @CacheEvict(value="emp-cache", key="#emplId")
    public void deleteById(Integer emplId) {
        //delete * from emptab where eid=?
    }
}
```

### Spring Boot Application#2:--

Dependencies: Oracle, Spring Data JPA, Spring WEB, Thymeleaf and Lombok, DevTools.

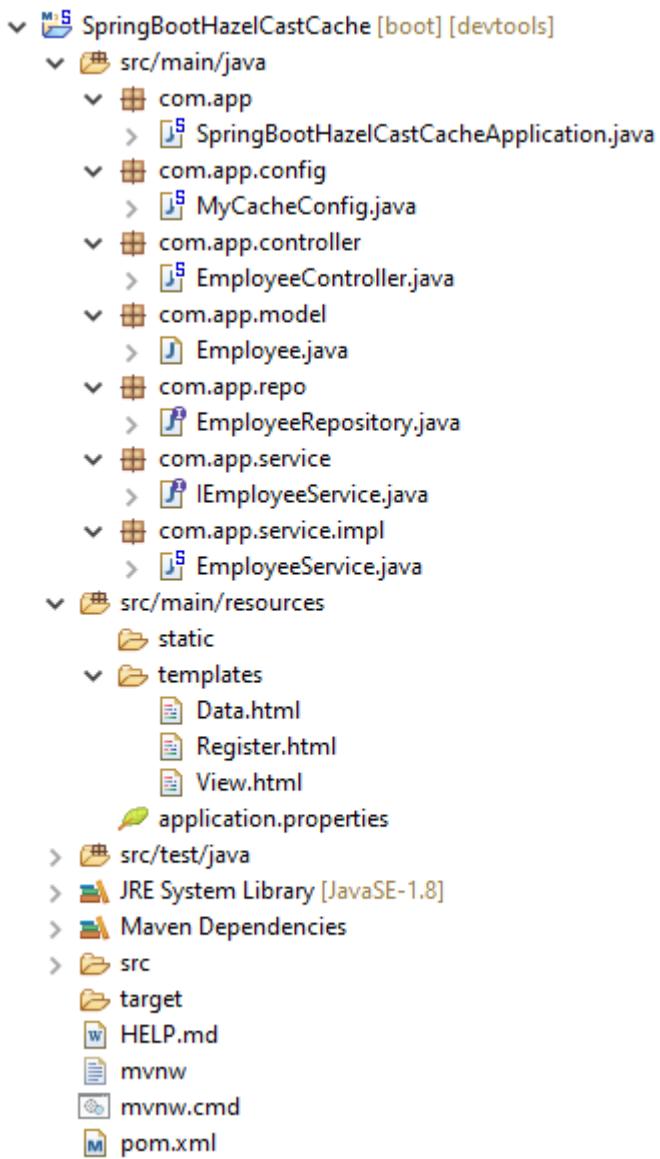
#### Coding order:--

1. Model class : Employee.java
2. Repository : EmployeeRepo.java
3. Service (I) : IEmployeeService  
And Impl (C) : CustomerServiceImpl.java
4. Controller : EmployeeController
5. UI : a. Register.html  
b. Data.html  
c. View.html

#### Note:--

1. Thymeleaf UI default prefix is “**template**” folder and suffix is “.html”.
2. **URL-Rewriting**:- Creating URL with static path and dynamic path.  
Ex:-- @{/employee/delete/{id}=\${ob.custId}} will be converted to (ex)  
<http://... Employee/delete/101>
3. To read this ID value in Controller, use Path Variable Annotation Syntax:  
    @PathVariable Integer id
4. At ServiceImpl (C) need to apply Transaction Management using annotation:  
    @Transactional (readOnly=true) for select  
    @Transactional for non-select methods

## #25. Spring Boot Application for Cache Implementation:--

**1>Starter class:--**

```
package com.app;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cache.annotation.EnableCaching;

@SpringBootApplication
@EnableCaching
public class SpringBootHazelCastCacheApplication {

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

**2>CacheConfig class (MyCacheConfig.java):--**

```
package com.app.config;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import com.hazelcast.config.Config;
import com.hazelcast.config.EvictionPolicy;
import com.hazelcast.config.MapConfig;
import com.hazelcast.config.MaxSizeConfig;
import com.hazelcast.config.MaxSizeConfig.MaxSizePolicy;

@Configuration
public class MyCacheConfig {

    @Bean
    public Config cacheConfig() {
        return new Config()
            .setInstanceName("hazelCast-instance")
            .addMapConfig(
                //add per module one MapConfig
                new MapConfig()
                    .setName("emp-cache")
                    .setTimeToLiveSeconds(2000)
                    .setMaxSizeConfig(
                        new MaxSizeConfig(200, MaxSizePolicy.FREE_HEAP_SIZE))
                    .setEvictionPolicy(EvictionPolicy.LRU)
                );
    }
}
```

**3>Model class (Employee.java):--**

```
package com.app.model;
import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.Id;
import lombok.Data;
```

```
@Entity  
@Data  
public class Employee implements Serializable {  
  
    private static final long serialVersionUID = 1L;  
    @Id  
    @GeneratedValue(strategy=GenerationType.IDENTITY)  
    private Integer empld;  
    private String empCode;  
    private Integer empName;  
    private String empType;  
    private Double empSal;  
    private String Addr;  
}
```

#### 4>Repository (EmployeeRepository.java):--

```
package com.app.repo;  
import org.springframework.data.jpa.repository.JpaRepository;  
import com.app.model.Employee;  
  
public interface EmployeeRepository extends JpaRepository<Employee, Integer>  
{ }
```

#### 5>Service interface (IEmployeeService.java):--

```
package com.app.service;  
import java.util.List;  
import com.app.model.Employee;  
  
public interface IEmployeeService {  
  
    public Integer saveEmployee(Employee e);  
    public List<Employee> getAllEmployees();  
    public Employee getOneEmployee(Integer empld);  
    public void deleteEmployee(Integer empld);  
    public void update(Employee employee); //Update Data  
}
```

**6>ServiceImpl class (EmployeeServiceImpl.java):--**

```
package com.app.service.impl;
import java.util.List;
import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.cache.annotation.CacheEvict;
import org.springframework.cache.annotation.CachePut;
import org.springframework.cache.annotation.Cacheable;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import com.app.model.Employee;
import com.app.repo.EmployeeRepository;
import com.app.service.IEmployeeService;

@Service
public class EmployeeService implements IEmployeeService {

    @Autowired
    private EmployeeRepository repo;

    @Transactional
    public Integer saveEmployee(Employee e) {
        /*Employee e1=repo.save(e);
        Integer id=e1.getEmpId();
        return id;*/
        return repo.save(e).getEmpId();
    }

    @Transactional(readOnly = true)
    public List<Employee> getAllEmployees() {
        return repo.findAll();
    }

    @Transactional(readOnly = true)
    @Cacheable(value="emp-cache", key="#empId")
    public Employee getOneEmployee(Integer empId) {
        Optional<Employee> e= repo.findById(empId);
        if(e.isPresent()) {
            return e.get();
        }
        return null;
    }
}
```

```

@Transactional
@CacheEvict(value="emp-cache", key="#empId")
public void deleteEmployee(Integer empId) {
    repo.deleteById(empId);
}

@Override
@CachePut(value="emp-cache", key="#empId")
public void update(Employee e) {
    repo.save(e);
}

```

**7>EmployeeController:--**

```

package com.app.controller;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import com.app.model.Employee;
import com.app.service.IEmployeeService;

@Controller
@RequestMapping("/employee")
public class EmployeeController {

    @Autowired
    private IEmployeeService service;

    //1. Display Reg Page with Form Backing Object
    @RequestMapping("/reg")
    public String showReg(Model map) {
        //Form backing object
        map.addAttribute("employee", new Employee());
        return "Register";
    }
}

```

```
//2. On click submit read form data and save into DB
@RequestMapping(value="/save",method=RequestMethod.POST)
public String saveData(@ModelAttribute Employee employee, Model map)
{
    //insert into DB
    Integer id=service.saveEmployee(employee);
    map.addAttribute("message", "employee "+id+" created");
    //clear form backing object
    map.addAttribute("employee", new Employee());
    return "Register";
}

//3. Display all records in DB at UI
@RequestMapping("/all")
public String showAll(Model map) {
    //fetch data from DB
    List<Employee> emps=service.getAllEmployees();
    //send data to UI
    map.addAttribute("list", emps);
    return "Data";
}

//4. fetch data based on id and display
@RequestMapping("/view/{id}")
public String viewOne(@PathVariable Integer id, Model map) throws Exception
{
    Employee e=service.getOneEmployee(id);
    map.addAttribute("ob", e);
    return "View";
}

//5. delete row based on Id
@RequestMapping("/delete/{id}")
public String deleteOne(@PathVariable Integer id, Model map) {
    service.deleteEmployee(id);
    //fetch all new data
    List<Employee> emps=service.getAllEmployees();
    //send data to UI
    map.addAttribute("list", emps);
    //success message
    map.addAttribute("message", "Employee "+id+" deleted");
    return "Data";
}
```

```

//6. Edit Specific record by emplId
@RequestMapping("/edit")
public String edit(@RequestParam Integer id, Model map)
{
    Employee emp = service.getOneEmployee(id);
    map.addAttribute("employee", emp);
    return "EmployeeDataEdit";
}

//7. Update Specific record by emplId
@RequestMapping("/update")
public String update(@ModelAttribute Employee e)
{
    service.update(e);
    return "redirect:all";
}

```

**8>UI Pages:--**

**a>Register.html**

```

<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org/">
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h3>Welcome to Employee Register Page</h3>
<form action="#" method="POST" th:action="@{/employee/save}"
th:object="${employee}">
<pre>
CODE : <input type="text" th:field="*{empCode}" /><br>
NAME : <input type="text" th:field="*{empName}" /><br>
TYPE : <select th:field="*{EmpType}">
        <option value="PER">PER</option>
        <option value="CONT">CONT</option>
    </select>

SALARY : <input type="text" th:field="*{empSal}" /><br>
ADDR : <textarea th:field="*{addr}"></textarea><br>
        <input type="submit" value="Create" />
</pre>

```

```
</form>
<span th:text="${message}"></span>
</body>
</html>
```

**b>Data.html:--**

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org/">
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head><body>
<h3>Welcome to All Records Page</h3>
<table border="1">
<tr>
<th>ID</th>
<th>CODE</th>
<th>NAME</th>
<th>TYPE</th>
<th>SALARY</th>
<th>ADDR</th>
<th colspan="2">OPERATIONS</th>
</tr>
<tr th:each="ob:${list}">
<td th:text="${ob.empId}"></td>
<td th:text="${ob.empCode}"></td>
<td th:text="${ob.empName}"></td>
<td th:text="${ob.empType}"></td>
<td th:text="${ob.empSal}"></td>
<td th:text="${ob.addr}"></td>
<td>
<a th:href="@{/employee/view/{id}(id=${ob.empId})}">VIEW</a>
</td>
<td>
<a th:href="@{/employee/delete/{id}(id=${ob.empId})}">DELETE</a>
</td>
</tr>
</table>
<span th:text="${message}"></span>
</body></html>
```

```
c>View.html:--  
<!DOCTYPE html>  
<html xmlns:th="http://www.thymeleaf.org/">  
<head>  
<meta charset="ISO-8859-1">  
<title>Insert title here</title>  
</head>  
<body>  
<h3>Welcome to View One Page</h3>  
<table border="1">  
    <tr>  
        <th>ID</th>  
        <td th:text="${ob.empId}"></td>  
    </tr>  
  
    <tr>  
        <th>CODE</th>  
        <td th:text="${ob.empCode}"></td>  
    </tr>  
  
    <tr>  
        <th>NAME</th>  
        <td th:text="${ob.empName}"></td>  
    </tr>  
    <tr>  
        <th>TYPE</th>  
        <td th:text="${ob.empType}"></td>  
    </tr>  
  
    <tr>  
        <th>TYPE</th>  
        <td th:text="${ob.empSal}"></td>  
    </tr>  
  
    <tr>  
        <th>NOTE</th>  
        <td th:text="${ob.addr}"></td>  
    </tr>  
</table></body>  
</html>  
//http://localhost:2019/myapp/employee/reg
```

**Output:-**

```

2019-07-19 01:24:29.979 INFO 2504 --- [nio-2019-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/myapp] : Initializing Spring DispatcherServlet 'dis...
2019-07-19 01:24:29.983 INFO 2504 --- [nio-2019-exec-1] o.s.web.servlet.DispatcherServlet      : Initializing Servlet 'dispatcherServlet'
2019-07-19 01:24:30.017 INFO 2504 --- [nio-2019-exec-1] o.s.web.servlet.DispatcherServlet      : Completed initialization in 33 ms
2019-07-19 01:24:30.106 INFO 2504 --- [nio-2019-exec-1] c.h.i.p.impl.PartitionStateManager   : [10.0.75.1]:5701 [dev] [3.11] Initializing
Hibernate: select employee0_.emp_id as emp_id1_0_0_, employee0_.addr as addr2_0_0_, employee0_.emp_code as emp_code3_0_0_, employee0_.emp_name
Hibernate: select employee0_.emp_id as emp_id1_0_0_, employee0_.addr as addr2_0_0_, employee0_.emp_code as emp_code3_0_0_, employee0_.emp_name
Hibernate: delete from emptab where emp_id=?
2019-07-19 01:25:08.542 INFO 2504 --- [nio-2019-exec-7] o.h.h.i.QueryTranslatorFactoryInitiator : HHH000397: Using ASTQueryTranslatorFactory
Hibernate: select employee0_.emp_id as emp_id1_0_0_, employee0_.addr as addr2_0_0_, employee0_.emp_code as emp_code3_0_0_, employee0_.emp_name as er

```

=>In above output we can see the only single select query even if we are click on multiple times on “View/Select” option.

**Task:-**

#1:- Implement Edit Operation for Module

2>Methods in controller

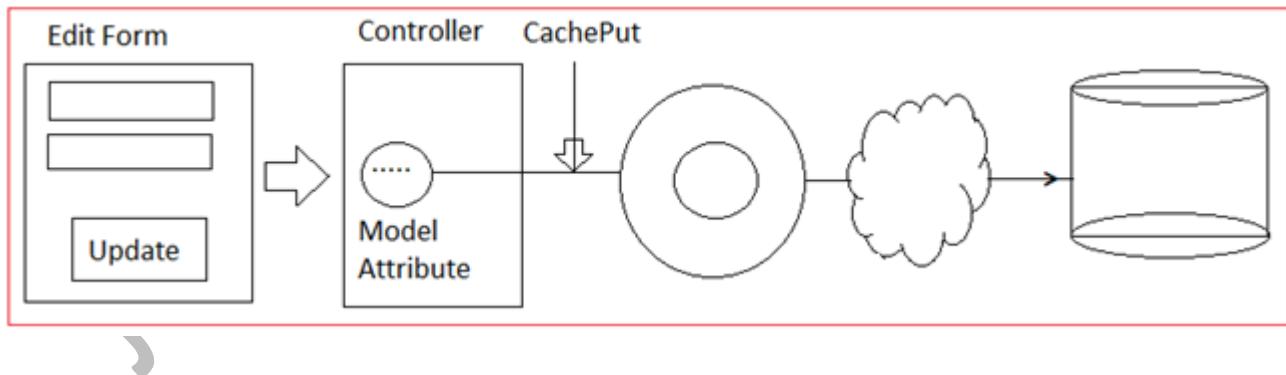
a>showEditPage with Object

b>updateData with Object

#2:-- Define Service Layer method i.e Update() with Transaction Management and Cache Annotation “CachePut”.

#3:-- Define Edit.html using Thymeleaf

=>CachePut annotation is used to update object in cache before updating in Database.



**1>@CacheEvict** --- This annotation is used to remove object from cache (not from DB).

=>On calling service.delete..() method, SQL query delete row only in DB but not in cache.

=>At same time object to be removed from cache, for that add this Annotation over delete(..) method in service layer.

**2>@Cacheable** --- On calling select... SQL (load one row as object) same object will be placed in cache before give to Service (App).

**3> CachePut:**-- This annotation is used to update object in cache before updating in Database.

**NOTE:--**

- ❖ Define one MapConfig memory for one module cache.  
`new MapConfig().setName("prod-cache");`
- ❖ Provide MAX size of cache (maximum no. of objects to be hold by cache)  
`setMaxSizeConfig(new MaxSizeConfig(100, MaxSizePolicy.FREE_HEAP_SIZE))`
- ❖ Provide EvictionPolicy.
  - \*\*If cache is full and another object is in waiting state to get into cache then EvictionPolicy will not allow another obj. (Default is : NONE).
  - \*\*EvictionPolicy .LRU : Will remove last accesses Object (Least Recently Used) from cache.
- ❖ Provide Life Time of object to be in cache (in second)  
`setTimeToLiveSeconds (3000);`

**5. Lombok API:--**

=>This is open source **JAVA API** used to avoid writing (or generating) common code for Bean/Model/Entity classes.

That is like:

- 1.>Setters and Getters
- 2.>`toString()` method
- 3.>Default and Parameterized Constructor
- 4.>`hashCode()` and `equals()` methods.

=>Programmer can write these methods manually or generate using IDE. But if any modification (s) are done in those classes then again generate set/get methods also delete and write code for new : `toString`, `hashCode`, `Equals` and `Param const` (it is like represented task).

=>By using Lombok API which reduces writing code or generating task for Beans.  
Just apply annotations, it is done.  
=>To use Lombok, while creating Spring Boot Project choose Dependency : **Lombok** (or)  
Add below dependency in pom.xml.

(for Spring Boot Project: Do not provide version provided by spring boot Parent only.)

```
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
</dependency>
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <scope>provided</scope>
</dependency>
```

**NOTE:-- (For Non Spring Boot Projects)**

```
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.18.6</version>
</dependency>
```

#### **Install Lombok in IDE:--**

**Step#1:-** Open STS/Eclipse (any workspace).

**Step#2:-** Create Spring Boot Project with Lombok Dependency (or) maven project with above Lombok Dependency.

**Step#3:-** Update Maven Project (Alt+F5).

**Step#4:-** Close STS.

**Step#5:-** Goto Lombok JAR location

Ex:-- C:\Users\<username>\.m2\repository\org\projectlombok\lombok\1.18.6

**Step#6:-** Open Command Prompt Here

=>Shift + Mouse Right Click

=>Choose “Open Command Window Here”

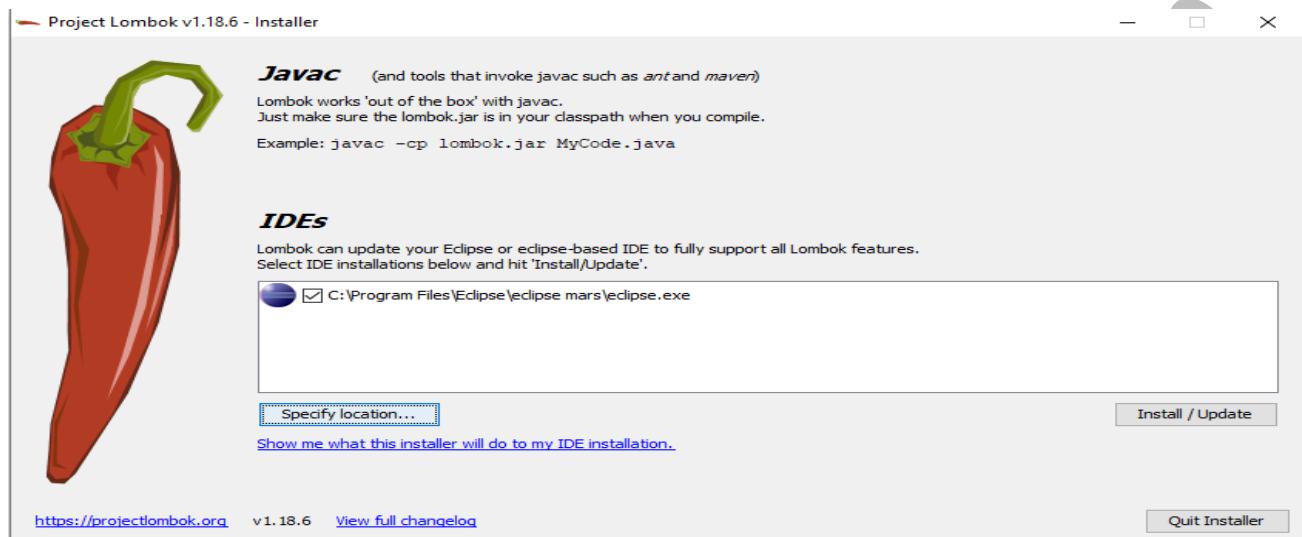
=>Type cmd given there (java –jar Lombok-1.18.6.jar)

```
C:\Windows\System32\cmd.exe - java -jar Lombok-1.18.6.jar
Microsoft Windows [Version 10.0.18342.8]
(c) 2018 Microsoft Corporation. All rights reserved.

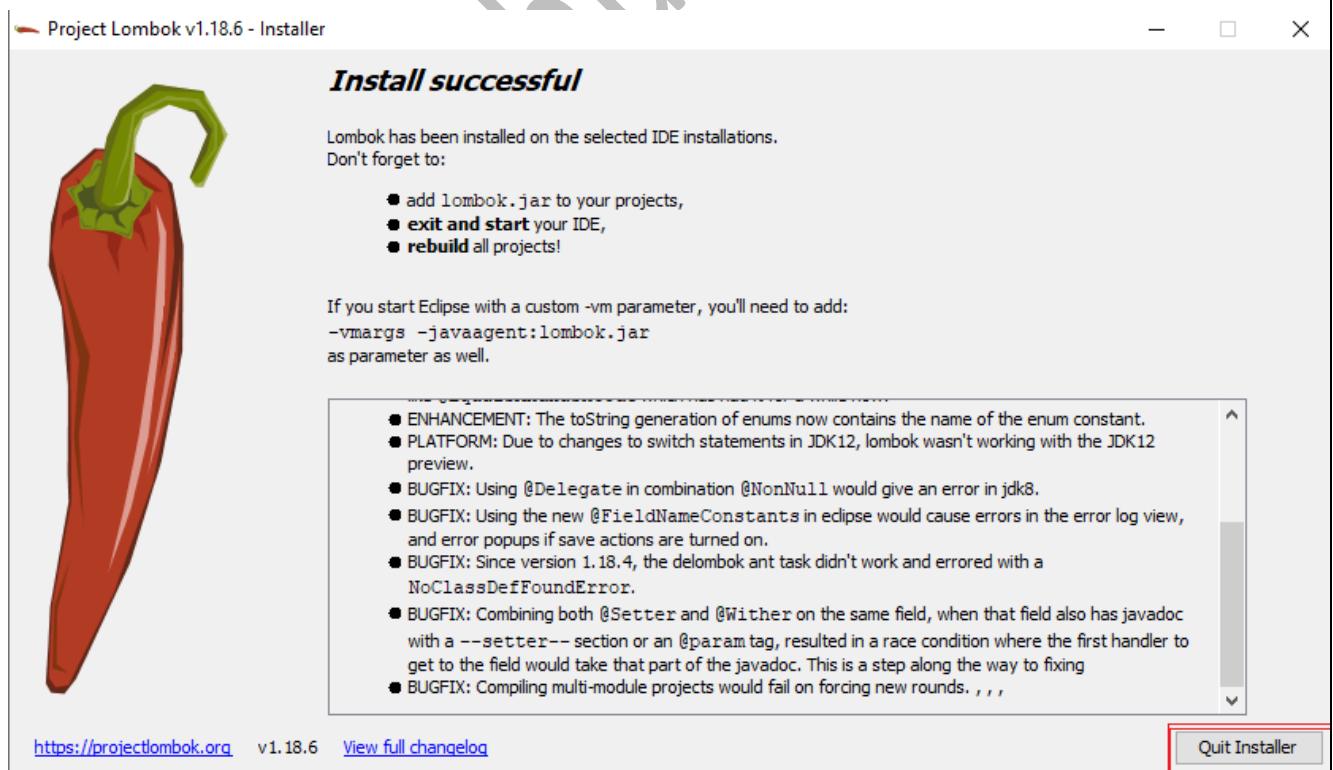
C:\Users\Uday\.m2\repository\org\projectlombok\lombok\1.18.6>java -jar Lombok-1.18.6.jar
```

=>wait for few minutes (IDEs are detected).

## Screen#1:- Click on Install/Update



## Screen#2:--



=>Finish.

**Step#7:-** Open STS/Eclipse and Start coding.

### Example Application:--

**Step#1:-** Create Spring Boot Starter Project

=> File => new => Spring Starter Project

=> Enter Details

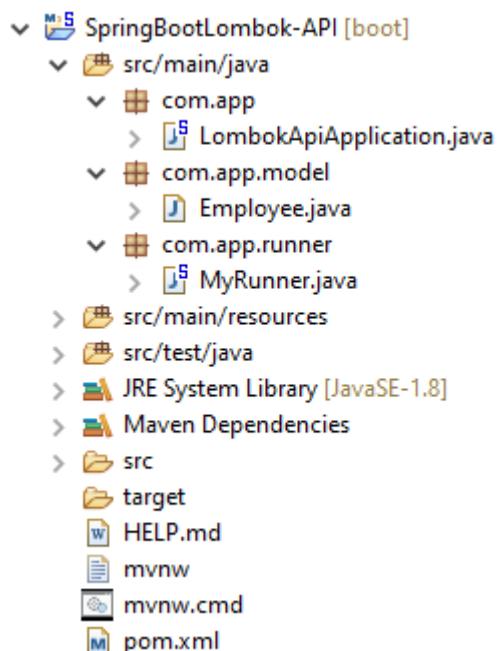
GroupId : com.app

ArtifactId : SpringBootLombok-API

Version :1.0

=> Choose dependency : Lombok (only)

### #26. Folder Structure of Lombok API Application:--



### Step#2:- Create Model class with Lombok annotations

```

package com.app.model;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.NonNull;
import lombok.RequiredArgsConstructor;
import lombok.Setter;
import lombok.ToString;
  
```

```

@Getter //generates get methods
@Setter //generates set method
@ToString //override toString method
@NoArgsConstructorConstructor ///generate default constructor
@RequiredArgsConstructor //Generate param const
@EqualsAndHashCode //Override hashCode, equals Methods
public class Employee {
    @NotNull
    private Integer emplId;
    @NotNull
    private String empName;
    private Double empSal;
}

```

\*\*\*To use **@RequiredArgsConstructor** which generates constructor using variables annotated with **@NotNull**. If no variable found having **@NotNull**, then it is equal to generating “Default constructor” only.

### Step#3:- Runner code

```

package com.app.runner;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;
import com.app.model.Employee;

@Component
public class MyRunner implements CommandLineRunner {

    public void run (String... args) throws Exception
    {
        Employee e1 = new Employee();
        e1.setEmplId(10);
        e1.setEmpName("Uday");
        e1.setEmpSal(34.87);

        Employee e2 = new Employee();
        e2.setEmplId(10);
        e2.setEmpName("Uday");
    }
}

```

```
e2.setEmpSal(76.65);

System.out.println(e2.equals(e1));
}

}
```

**NOTE:**-- Apply **@Data** (package : Lombok.Data) over Bean/Model which generates Getter, Setter toString, equals, hashCode and RequiredArgsConstructor ( ).

Example:--

```
@Data
public class Employee {...}
```

Ex:--

```
package com.app.model;
import javax.persistence.Id;
import javax.persistence.Table;
import lombok.Data;
```

```
@Data
@Table
public class Employee {

    @Id
    private Integer empld;
    //@NonNull
    private String empName;
    private Double empSal;
}
```

**NOTE:**-- For more details about Lombok follow bellow links

<https://projectlombok.org/changelog>

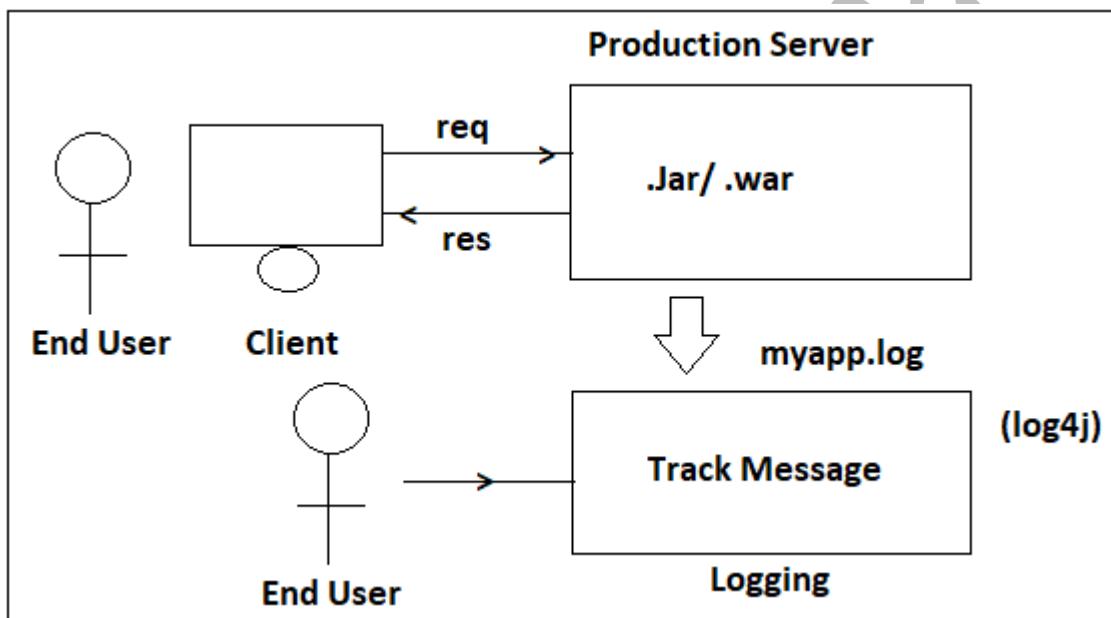
\*\*\* Click on show whitespace character symbol (looks like PI symbol :  $\pi$ )

**6. Logging:**-- This process is used to track all types of messages when any request is processed by server.

=>Here tracking means, finding messages like SUCCESS, WARNING, EXCEPTIONS, (N/W, DB...), INFORMATIONS... etc.

#### Types of server:--

- DEV / QA Server
- UAT Server (User Acceptance Test):- Client does testing before purchase the project/product.
- Production Server :-- Application placed in real-time server and this can be accessed by end users.
- MS service (Management Support).



=>To implement Logging concept, we need to provide 3 components. Those are:

- a>Logger (C)
- b>Appender (I)
- c>Layout (AC)

**a>Logger:**-- This object need to be created in class which needs Logging concept.

=>If Logger object is created then Logging is enabled for that class, else not.

=>All classes may not need Logger object, Example Model class.

=>Logger object can be created using Different Vendors few are: Apache Log4J, SELF4J, Java UTIL Logging, JBoss Logging, Apache JCL (Commons Logging)...etc.

**b>Appender** --- It indicates where to store Log messages Possible Appender are.

- 1>ConsoleAppender (C)
- 2>FileAppender \*\*\* (C)
- 3>SmtpAppender (C)
- 4>TelnetAppender (C)
- 5>JdbcAppender (C)

=>FileAppender is mostly used one for projects which stores all log messages in **---.log** file.

**c>Layout**-- This one used to specify “Message Format” to be printed on Appender.

=>PossibleLayouts are / implemented classes are.

- 1>SimpleLayout:- Prints only message as it is provided in log methods.
- 2>XMLLayout:- Converts message into XML Format.
- 3>HTMLLayout:- Prints in HTML format.
- 4>PatternLayout:- Programmer can define Message Format (like Date-Time-class-method-line-message ... etc).

=>For more details about pattern visit bellow link :

<https://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/PatternLayout.html>

**NOTE**-- In one Application we can use more than one Appender also.

=>In development mostly used Appender is FileAppender and Layout is Pattern Layout.

=>If No appender is provided to Logger object then Log4J throws WARNING as log4j : WARN No appenders could be found for logger (com.app.Product).

=>At last Appender object must be added to Logger Object then only it is considered by using **log.addAppender()** method.

#### Priority Methods with Order---

=>This are used to print log messages with type indications.

=>List of Methods with order is given below.

Order	Method	Meaning
1	TRACE	=>State of execution.
2	DEBUG	=>Message with final Result.
3	INFO	=>Message of current step.

4	WARN	=>Warning message.
5	ERROR	=>Exception messages.
6	FATAL	=>High Level Exceptions/ Errors
0	OFF	=>Do TURN OFF Logging

### 1. Sample Logging code:--

```

package com.app;
import java.io.IOException;
import org.apache.log4j.Appender;
import org.apache.log4j.ConsoleAppender;
import org.apache.log4j.FileAppender;
import org.apache.log4j.HTMLLayout;
import org.apache.log4j.Layout;
import org.apache.log4j.Logger;
import org.apache.log4j.PatternLayout;
import org.apache.log4j.SimpleLayout;
import org.apache.log4j.jdbc.JDBCAppender;
import org.apache.log4j.net.JMSAppender;
import org.apache.log4j.net.SMTPAppender;
import org.apache.log4j.xml.XMLLayout;

public class MyLogging {

    private static Logger log = Logger.getLogger(MyLogging.class);

    public static void main(String[] args)throws IOException {

        //Layout layout = new SimpleLayout();
        //Layout layout1 = new HTMLLayout();
        //Layout layout2 = new XMLLayout();
        Layout layout3 = new PatternLayout("%d{dd-mm-yy hh:ss} %c-%m:[%p]%m %n");

        Appender app = new ConsoleAppender(layout3);
        //Appender app1 = new FileAppender(layout, "D:\\Source\\aa.log");
        //Appender app2 = new JDBCAppender();
        //Appender app3 = new JMSAppender();
    }
}

```

```

//Appender app4 = new SMTPAppender();
log.addAppender(app);
log.debug("WELCOME");
log.warn("SAMPLE");
log.info("Hello");
log.error("OK ! FAIL");
}
}

```

**Output:--**

```

26-48-19 02:13 com.app.runner.MyLogging-WELCOME:[DEBUG]WELCOME
26-48-19 02:13 com.app.runner.MyLogging-SAMPLE:[WARN]SAMPLE
26-48-19 02:13 com.app.runner.MyLogging-Hello:[INFO]Hello
26-48-19 02:13 com.app.runner.MyLogging-OK ! FAIL:[ERROR]OK ! FAIL

```

**Log4j.properties:--** This file is used to provide all the configuration details of Log4J. Especially like **appender** and **layout** details with Pattern and also root Logger details.  
=>We should not create Appender in every class only logger object must be created in a class.

=>Data will be shown in below order like

1. rootLogger
2. appenders
3. layouts

- ❖ Appender name must be defined before use at rootLogger level.
- ❖ We can specify multiple appenders in log4j.properties file like 2 File Appenders, one JdbAppender, one smtpAppender etc...
- ❖ Appender name can be any think ex:-- abc, hello, sysout, file, db, email etc...
- ❖ log4j.properties file must be created under src/main/resource folder (Maven).
- ❖ Make rootLogger =OFF to disable log4j completely without deleting code in any class or properties file.
- ❖ log4j.properties file will be auto detected by log4j tool. No special coding is required.

**Format : Log4j.properties**

```

log4j.rootLogger=PRIORITY, APPENDER NAMES, ....
log4j.appender.<name>.layout=.....

```

**VM Arguments**-- A variable created with value at JVM level which can be stored to one or services (Projects/Threads/Apps... etc)

Format for creating System / VM Argument

-Dkey = value

=>To read data in Application code is :

```
String s = System.getProperty("key");
```

## 2. Working with logging using spring boot:--

**Step#1**:- Creating logger object using multiple APIs.

a>SLF4J API:--

```
private static Logger log = LoggerFactory.getLogger(----.class);
```

b>Apache Log4J:--

```
private static Logger log = LogManager.getLogger(----.class);
```

c>Java util Logging

```
private static logger log = LogManager.getLogger(---.class, getName());
```

**Step#2**:- Provide Appender and Layout details using Properties files/yml file in Boot.

**NOTE**:- Application No need to create another **log4j.properties** file.

=>Default logging level in INFO.

Example:-- key = value pairs are given below.

**application.properties**:--

logging.level.root=INFO

logging.file=d://logs/my.log

logging.level.org.springframework=error

logging.pattern.file=%p %d{dd-mm-yy}%L %c[%m]-%d %n

logging.pattern.console=%d{yyyy-mm-dd HH:mm:ss}-%m %n

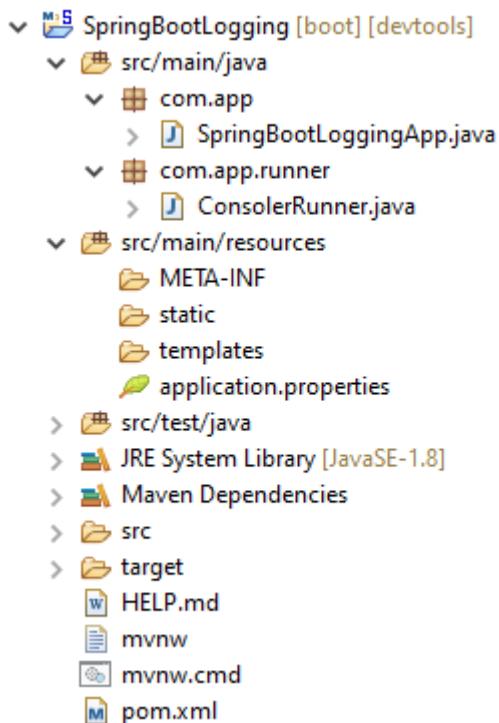
**Step#1:-** Create one spring boot starter application with web dependency.

Name :- SpringBootLogging

### Dependencies:-

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-log4j</artifactId>
    <version>1.3.8.RELEASE</version>
</dependency>
```

### #27. Folder Structure of Logging using Apache log4J:-



**Step#2:-** Open application.properties file and add below keys.

### application.properties:-

```
logging.level.root=INFO
logging.file=d://logs/myapp.log
logging.level.org.springframework=ERROR
logging.level.org.apache=DEBUG
logging.file.max-size=1MB
logging.file.max-history=5
logging.pattern.file=%p %d{dd-mm-yy}%L %c[%m]-%d %n
logging.pattern.console=%d{yyyy-mm-dd HH:mm:ss}-%m %n
```

Step#3:- Create logger object in any class and call priority methods.

```
package com.app.runner;
import org.apache.log4j.LogManager;
import org.apache.log4j.Logger;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

@Component
public class ConsolerRunner implements CommandLineRunner {

    @Override
    public void run(String... args) throws Exception {

        //Apache log4j
        private static Logger log = LogManager.getLogger(ConsolerRunner.class);

        log.info("Console Runner Starter");
        int a=10, b=20, c=-1;
        log.info("Data init.. done");

        if(a>0 && b>0)
        {
            c=a+b;
            log.info("if block executed");
        } else {
            c=a-b;
            log.info("else block executed");
        }
        log.debug("Final Result is: "+c);
        if(c > 0)
            try {
                throw new RuntimeException("Hello Sample");
            }catch (Exception e) {
                e.printStackTrace();
                //oa.error("Problem found..." +e);
            }
        //stop server
        System.exit(0);
    }
}
```

**StopWatch(c)**--This class is used to find execution time of JAVA code (method /block /loop... etc).

=>Support method start(), stop(), getTotalMillis(), getTotalTimeSecond()... etc

**prettyPrint()**-- It will print stopwatch details part by part.

Example ConsoleRunner (run method):--

```
StopWatch s = new StopWatch("MY SERVICE TEST");
```

```
    log.info("Block Started");
    s.start("Block#1");
    //s.start();
    for (int i=0; i<=9999.99; i++) {
        double d = Math.random();
        d=Math.sin(d);
        d=Math.abs(d);
    }
    s.stop();

    s.start("Block#2");
    for (int i=0; i<=9999; i++) {
        double d = Math.random();
        d=Math.sin(d);
        d=Math.abs(d);
    }
    s.stop();

    long ms=s.getTotalTimeMillis();
    double sec = s.getTotalTimeSeconds();
    log.info("Block end:"+ms);
    log.info(s.prettyPrint());
```

#### Color-coded log output:--

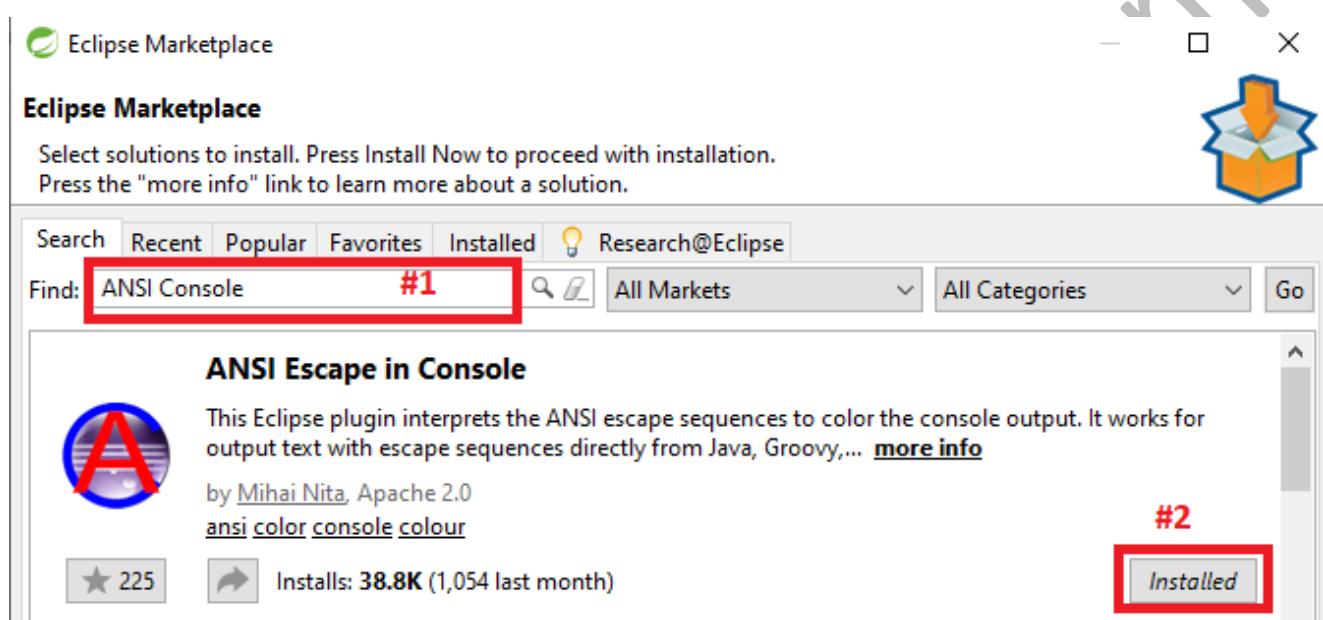
=>If your terminal supports ANSI, color output will be used to aid readability. You can set `spring.output.ansi.enabled` value to either **ALWAYS**, **NEVER** or **DETETC(Default)**.

=>Color coding is configured using the `%clr` conversion word. In its simplest form the converter will color the output according to the log level.

- ❖ FATAL and ERROR – Red
- ❖ WARN – Yellow
- ❖ INFO, DEBUG and TRACE – Green

### Step to implement Colour code in Logging:--

**Step#1:-** Go Eclipse market place and search with “ANSI Console” and install the software for supporting of ANSI scape sequences.



**Step#2:-** Add below properties in application.properties file.

```
spring.output.ansi.enabled=ALWAYS
```

# CHAPTER#5: SPRING BOOT EMAIL

## **1. Introduction:--**

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

=>Spring Boot mail API is defined on top of JAVA Mail API which reduces common lines of code for Email Application.

=>To implement Email Service, we need to provide below keys in application.properties file.

=>Example given with Gmail Server Details.

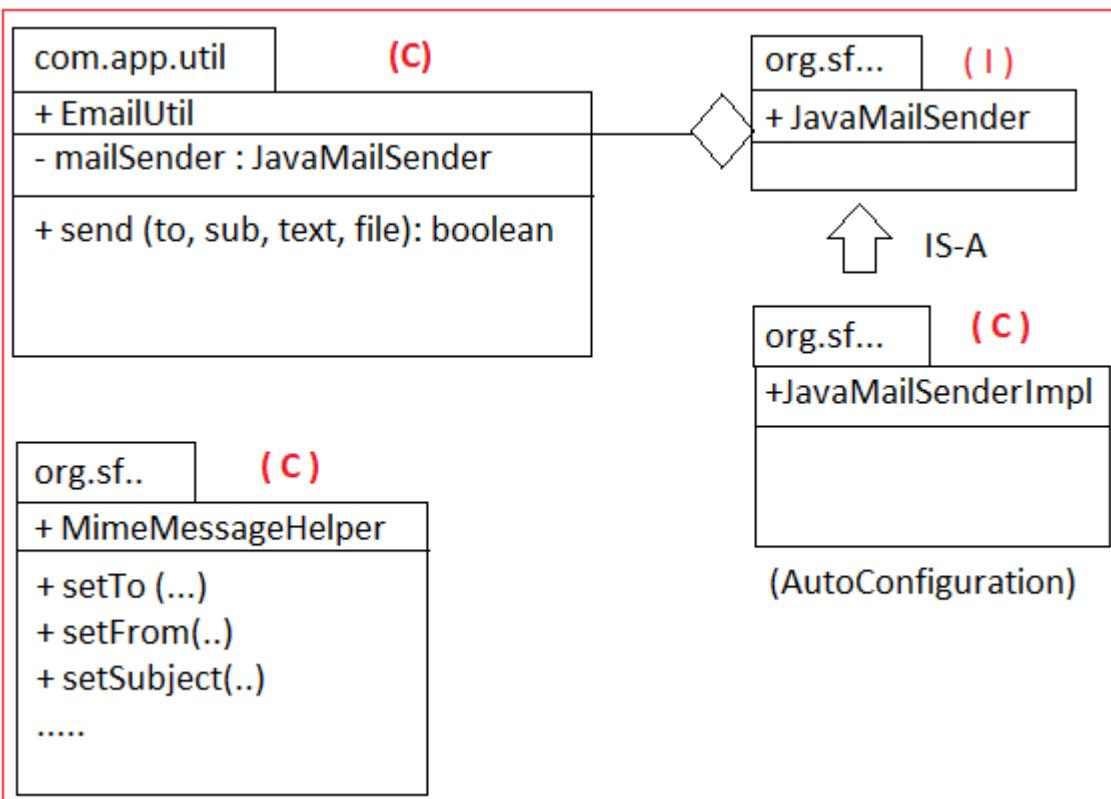
## **Application.properties:--**

```
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=udaykumar461992@gmail.com
spring.mail.password=Uday1234
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
```

## **application.yml:--**

```
spring:
  mail:
    host: smtp.gmail.com
    port: 587
    username: udaykumar461992@gmail.com
    password: Uday1234
    properties:
      mail:
        smtp:
          auth: true
          starttls:
```

enable: true

**UML Diagram:--**

Spring Boot AutoConfiguration does,

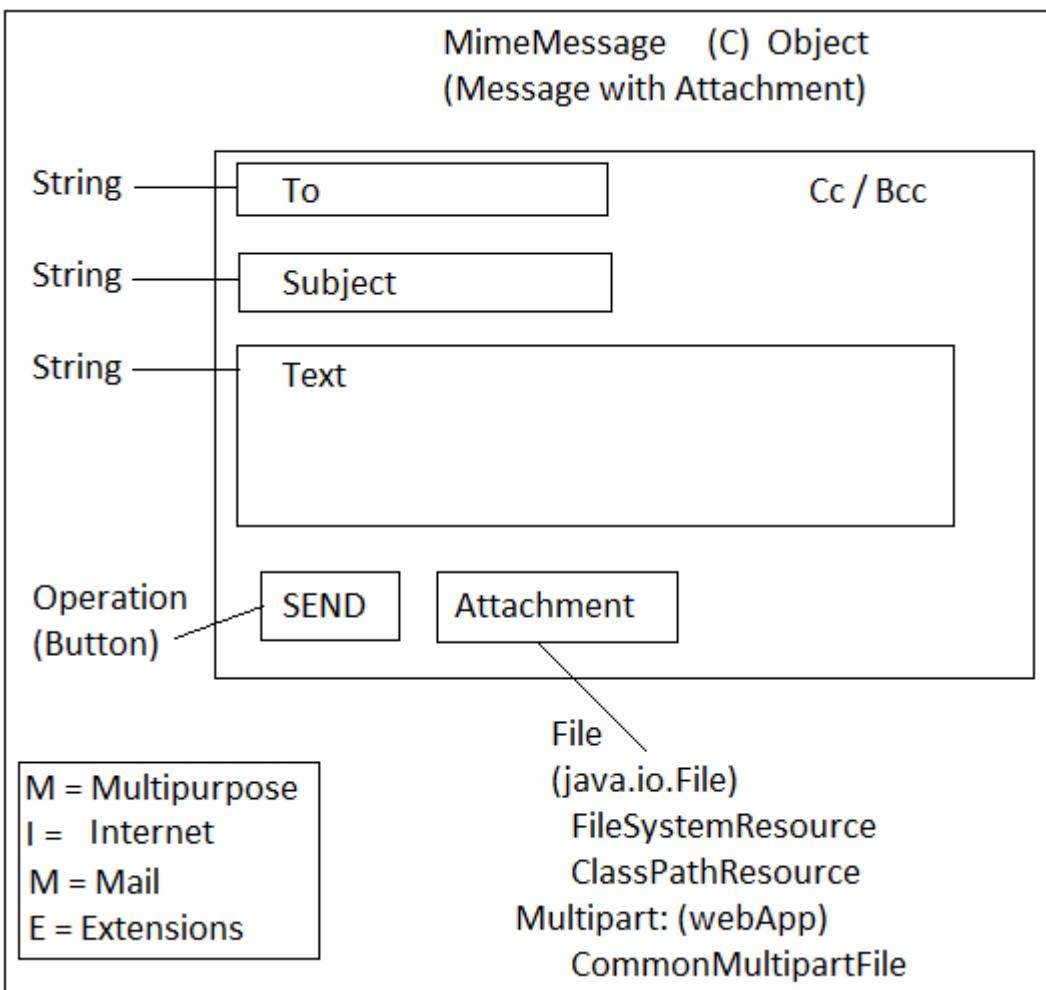
- 1> Get all Required Jars to Project.
- 2> Find for keys properties/yml  
(auto-load based on prefix =spring.mail)
- 3> \*\*\*Writes Configuration for JavaMailSenderImpl Bean (i.e @Bean)
- 4> Creates Email session with Mail sender.

=Programmer not required defining bean for JavaMailSenderImpl in AppConfig (like Spring f/w).

=>But need to provide all inputs like host, port.. using Properties file.

=>We need to construct one object i.e "**MimeMessage**" which is equals to "New Message in Mail Application".

=>Spring f/w has provided one helper class "**MimeMessageHelper**", to construct object for "**MimeMessage**".



=>To Load Files (Attachments) in case of stand-alone mode application in spring or Boot use “**Resource**” impl classes. For Example

- a. **FileSystemResource** --- This is used to load one file from System drives.  
(ex:- d:/images/mydata).
- b. **ClassPathResource**--- If file is available in src/main/resources folder then use this concept.

**Step#1:**-- Create Spring Boot starter Application using dependency : Java Mail Sender

#### Java Mail Sender Dependency:

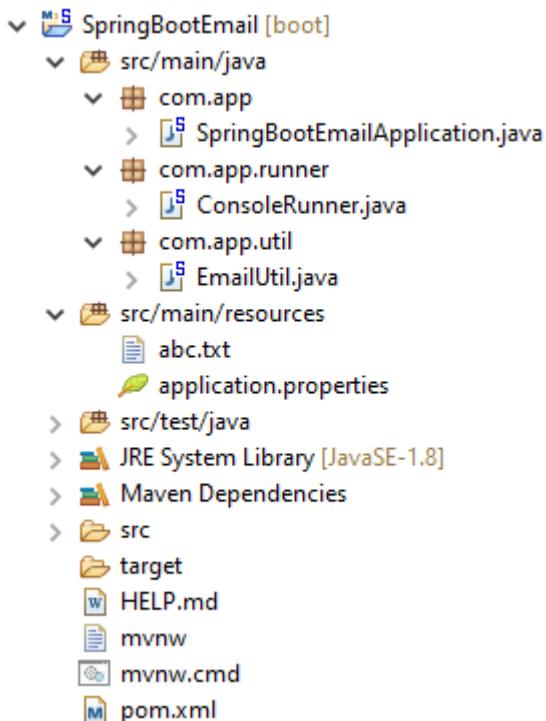
```

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

**Step#2:**-- Provide details in application.properties/yml  
 host, port = To identify Mail service Provider.  
 user, pwd = To do login from boot Application.  
 mail properties = Extra information to server.

**Step#3:**-- Define Service Utility class

## #28. Folder Structure of Spring Boot Mail Service:--



### 1. application.properties:--

```

spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=udaykumar461992@gmail.com
spring.mail.password=Uday1234
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
  
```

### 2. EmailUtil.java:--

```

package com.app.util;
import javax.mail.internet.MimeMessage;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.core.io.FileSystemResource;
  
```

```
import org.springframework.mail.javamail.JavaMailSender;
import org.springframework.mail.javamail.MimeMessageHelper;
import org.springframework.stereotype.Component;
```

```
@Component
public class EmailUtil {

    @Autowired
    private JavaMailSender mailSender;

    public boolean send(String to, String subject, String text,
            //String[] cc,
            //String[] bcc,
            FileSystemResource file)
            //ClassPathResource file)
    {
        boolean flag=false;

        try {
            //1. Create MimeMessage object
            MimeMessage message=mailSender.createMimeMessage();

            //2. Helper class object
            MimeMessageHelper helper=new MimeMessageHelper(message,
                    file!=null?true:false);

            //3. set details
            helper.setTo(to);
            helper.setFrom("udaykumar461992@gmail.com");
            helper.setSubject(subject);
            helper.setText(text);
            //helper.setCc(cc); //array Inputs
            //helper.setBcc(bcc);

            if(file!=null)
                helper.addAttachment(file.getFilename(),file);
        }
    }
}
```

```

        //4. send button
        mailSender.send(message);

        flag=true;
    } catch (Exception e) {
        flag=false;
        e.printStackTrace();
    }
    return flag;
}

```

**3. ConsoleRunner:--**

```

package com.app.runner;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.core.io.FileSystemResource;
import org.springframework.stereotype.Component;
import com.app.util.EmailUtil;

@Component
public class ConsoleRunner implements CommandLineRunner {

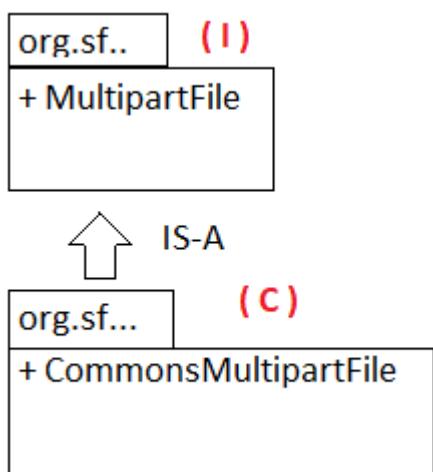
    @Autowired
    private EmailUtil util;

    @Override
    public void run(String... args) throws Exception
    {
        //ClassPathResource file=new ClassPathResource("abc.txt");
        FileSystemResource file=new FileSystemResource("F:\\Uday Kumar\\Photo.jpg");
        boolean flag=util.send("udaykumar0023@gmail.com", "AA", "Hello", file);
        if(flag) System.out.println("SENT");
        else System.out.println("CHECK PROBLEMS");
    }
}

```

## **Modification in project for Email setup:-**

- =>To indicate any attachment (file) use concept '**Spring Boot multipart Programming**'.
- =>Multipart is a concept used to indicate 'java.io.File' with Input/Output Stream operations given.
- =>Multipart concept works only for Web Applications.
- =>Multipart is **enabled** by default in spring boot.  
(spring.servlet.multipart.enabled=true)  
“spring.servlet.multipart.max-file-size=10MB”.



## **Code changes in project:-**

**Step#1:-** In model class define Transient variable for email (which never gets stored in DB, only at UI).

## Product.java

```
@Transient  
private String email;
```

**Step#2:-** Indicate Form has attachment using encoding type

```
<form: form.... enctype="multipart/form-data">
```

Ex:-- <form:form action="save" method="POST" modelAttribute="product" enctype="multipart/form-data">

**Step#3:-** add two inputs in Register.JSP page

EMAIL: <form:input path="email"/>

DOCUM: <input type="file" name="fileOb">

**Step#4:-** Copy EmailUtil.java, email properties, dependency (Starter-mail) into Project.

**Step#5:-** Use EmailUtil in Controller class

ProductController-----< > EmailUtil

**Step#6:-** Read file (attachment) in controller method (on save) **@RequestParam**

**MultipartFile fileOb**

**Step#7:-** call send(..) method from EmailUtil

```
mailUtil.send(product.getEmail(), "Product added", "Hello User", fileOb);
```

**Step#8:-** Change Parameter Type from FileSystemResource to multipartFile in EmailUtil.

```
package com.app.util;
import javax.mail.internet.MimeMessage;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.mail.javamail.JavaMailSender;
import org.springframework.mail.javamail.MimeMessageHelper;
import org.springframework.stereotype.Component;
import org.springframework.web.multipart.MultipartFile;

@Component
public class EmailUtil {

    @Autowired
    private JavaMailSender mailSender;

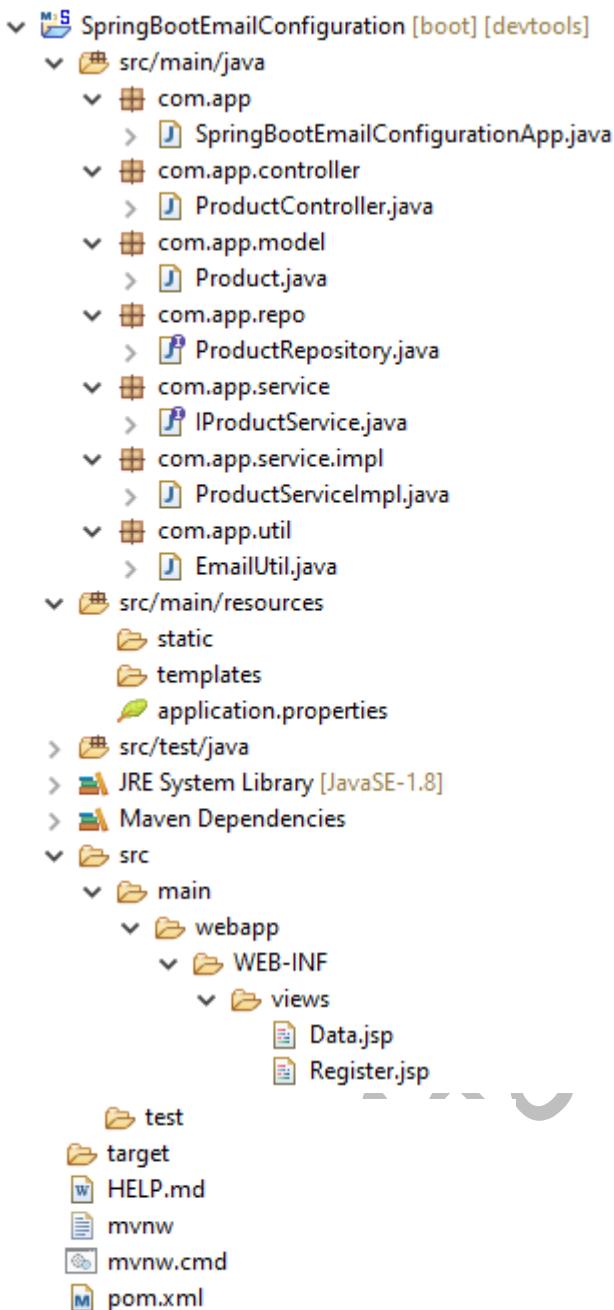
    public boolean send(String to, String subject, String text,
            //String[] cc,
            //String[] bcc,
            MultipartFile file) //change from FileSystemResource to MultipartFile
            //ClassPathResource file)
    {
        boolean flag=false;

        try {

```

```
//1. Create MimeMessage object  
MimeMessage message=mailSender.createMimeMessage();  
  
//2. Helper class object  
MimeMessageHelper helper=new MimeMessageHelper(message,  
file!=null?true:false);  
//3. set details  
helper.setTo(to);  
helper.setFrom("udaykumar461992@gmail.com");  
helper.setSubject(subject);  
helper.setText(text);  
//helper.setCc(cc); //array Inputs  
//helper.setBcc(bcc);  
if(file!=null)  
helper.addAttachment(file.getOriginalFilename(),file); //changes  
  
//4. send button  
mailSender.send(message);  
  
flag=true;  
} catch (Exception e) {  
flag=false;  
e.printStackTrace();  
}  
return flag;  
}  
}
```

## #29. Folder Structure of EmailConfiguration with Web MVC application:-



## application.properties:--

```

## SERVER ##
server.port=9090
server.servlet.context-path=/myapp

## WEB MVC ##
spring.mvc.view.prefix=/WEB-INF/views/
spring.mvc.view.suffix=.jsp
  
```

```
## DataSource ##
spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:XE
spring.datasource.username=system
spring.datasource.password=system

## JPA ##
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.Oracle10gDialect
spring.jpa.properties.hibernate.format-sql=true

## Email ##
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=udaykumar461992@gmail.com
spring.mail.password=Uday1234
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true

1>Model class:--
package com.app.model;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;
import org.springframework.data.annotation.Transient;
import lombok.Data;

@Entity
@Data
@Table(name="producttab")
public class Product {

    @Id
```

```
@Column(name="pid")
@GeneratedValue
private Integer id;

@Column(name="PCODE")
private String code;
@Column(name="PNAME")
private String name;

@Column(name="PCOST")
private Double cost;
@Column(name="PGST")
private Integer gst;

@Column(name="PNOTE")
private String note;

@Transient
private String email;
}
```

**2>ProductRepository:--**

```
package com.app.repo;
import org.springframework.data.jpa.repository.JpaRepository;
import com.app.model.Product;

public interface ProductRepository extends JpaRepository<Product, Integer> { }
```

**3>IProductService:--**

```
package com.app.service;
import java.util.List;
import com.app.model.Product;

public interface IProductService {
```

```
    public Integer saveProduct(Product p);
    public List<Product> getAllProducts();
    public void deleteProduct(Integer id);
    public Product getProductById(Integer id);
}
```

**4>ProductServiceImpl:--**

```
package com.app.service.impl;
import java.util.List;
import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import com.app.model.Product;
import com.app.repo.ProductRepository;
import com.app.service.IProductService;

@Service
public class ProductServiceImpl implements IProductService {

    @Autowired
    private ProductRepository repo;

    @Transactional
    public Integer saveProduct(Product p) {
        //calculations here..
        //gstAmt= cost*gst/100
        //totalAmt=cost+ gstAmt - disc
        p=repo.save(p);
        return p.getId();
    }

    @Transactional(readOnly= true)
    public List<Product> getAllProducts() {
        return repo.findAll();
    }
}
```

```
@Transactional  
public void deleteProduct(Integer id) {  
    repo.deleteById(id);  
}  
  
@Transactional(readOnly=true)  
public Product getProductById(Integer id) {  
    Optional<Product> p=repo.findById(id);  
    return p.get();  
}  
}
```

**5>EmailUtil.java:--**

```
package com.app.util;  
import javax.mail.internet.MimeMessage;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.mail.javamail.JavaMailSender;  
import org.springframework.mail.javamail.MimeMessageHelper;  
import org.springframework.stereotype.Component;  
import org.springframework.web.multipart.MultipartFile;
```

```
@Component
```

```
public class EmailUtil {
```

```
    @Autowired  
    private JavaMailSender mailSender;
```

```
    public boolean send(  
        String to,  
        String subject,  
        String text,  
        //String[] cc,  
        //String[] bcc,  
        MultipartFile file  
    )
```

```
    {  
        boolean flag=false
```

```
try {  
    //1. Create MimeMessage object  
    MimeMessage message=mailSender.createMimeMessage();  
  
    //2. Helper class object  
    MimeMessageHelper helper=new MimeMessageHelper(message,  
file!=null?true:false);  
  
    //3. set details  
    helper.setTo(to);  
    helper.setFrom("udaykumar461992@gmail.com");  
    helper.setSubject(subject);  
    helper.setText(text);  
    //helper.setCc(cc); //array Inputs  
    //helper.setBcc(bcc);  
    if(file!=null)  
        helper.addAttachment(file.getOriginalFilename(),file);  
  
    //4. send button  
    mailSender.send(message);  
  
    flag=true;  
} catch (Exception e) {  
    flag=false;  
    e.printStackTrace();  
}  
return flag;  
}  
}
```

**6>ProductController:--**

```
package com.app.controller;  
import java.util.List;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.web.bind.annotation.ModelAttribute;
```

```
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.multipart.MultipartFile;
import com.app.model.Product;
import com.app.service.IProductService;
import com.app.util.EmailUtil;

@Controller
@RequestMapping("/product")
public class ProductController {

    @Autowired
    private IProductService service;
    @Autowired
    private EmailUtil mailUtil;

    //1. Show Product Form with Backing Object
    @RequestMapping("/reg")
    public String showReg(Model map) {
        //Form Backing Object
        Product p=new Product();
        map.addAttribute("product",p);
        return "Register";
    }

    //2. Read Form Data on click submit
    @RequestMapping(value="/save",method=RequestMethod.POST)
    public String saveData(@ModelAttribute Product product,
    @RequestParam MultipartFile fileOb, Model map)
    {
        //call service layer
        Integer id=service.saveProduct(product);
        //send email
        mailUtil.send(product.getEmail(), "Product added", "Hello User", fileOb);
        map.addAttribute("message", "Product "+id+" created!!");
    }
}
```

```
//clean Form Backing Object  
map.addAttribute("product", new Product());  
return "Register";  
}  
  
//3. Fetch all Rows from DB to UI  
@RequestMapping("/all")  
public String showAll(Model map) {  
    //fetch all rows from DB  
    List<Product> obs=service.getAllProducts();  
    //send to UI  
    map.addAttribute("list", obs);  
    return "Data";  
}  
  
//4. Delete row based on ID  
@RequestMapping("/delete")  
public String remove(@RequestParam Integer id) {  
    //delete row based on ID  
    service.deleteProduct(id);  
    //response.sendRedirect  
    return "redirect:/all";  
}  
//5. Show Edit Page  
@RequestMapping("/edit")  
public String showEdit(  
    @RequestParam Integer id  
    ,Model map) {  
    //load object from DB  
    Product p=service.getProductById(id);  
    //FORM BACKING OBJECT  
    map.addAttribute("product",p);  
    map.addAttribute("Mode", "EDIT");  
    return "Register";  
}  
}
```

7>UI Pages:--

a. Register.jsp

```

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1" isELIgnored="false" %>
<%@taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<!DOCTYPE html>
<html><head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h3>WELCOME TO PRODUCT REGISTER</h3>
<form:form action="save" method="POST" modelAttribute="product"
enctype="multipart/form-data">
<pre>
<c:if test="${'EDIT' eq Mode }">
    ID : <form:input path="id" readonly="true"/><br>
</c:if>
    CODE : <form:input path="code"/><br>
    NAME : <form:input path="name"/><br>
    COST : <form:input path="cost"/><br>
    GST : <form:select path="gst">
        <form:option value="5">5%-SLAB</form:option>
        <form:option value="12">12%-SLAB</form:option>
        <form:option value="18">18%-SLAB</form:option>
        <form:option value="22">22%-SLAB</form:option>
        <form:option value="30">30%-SLAB</form:option>
    </form:select>
</pre>
EMAIL: <form:input path="email"/><br>
DOCUM: <input type="file" name="fileOb"/><br>
NOTE : <form:textarea path="note"/><br>
<c:choose>

```

```

<c:when test="#{'EDIT' eq Mode}">
    <input type="submit" value="UPDATE PRODUCT"/>
</c:when>
<c:otherwise>
    <input type="submit" value="CREATE PRODUCT"/>
</c:otherwise>
</c:choose>
</pre>
</form:>
${message}
</body></html>

```

**Execution:**-- Run the application and type the below URL in browser.

<http://localhost:9090/myapp/product/reg>

**Output:**--

localhost:9090/myapp/product/save

Gmail YouTube Online Courses - A... Online Tests - Onlin...

## WELCOME TO PRODUCT REGISTER

CODE :

NAME :

COST :

GST :

EMAIL:

DOCUM:  No file chosen

NOTE :

Q>Define on Spring Boot web application using Email service.

#1:- Design one Spring Boot web application using email service.

#2:- On click submit, form data should be converted to model class Object (com.app.model.message).

#3:- Use EmailUtil and send message to do Addr.

#4:- Clean Form and send source message back to UI.

\*\*\* Use Spring Validation API to avoid null or empty message (even Mail formats).

Ex:-- To address cannot be null Email is invalid format subject is required

Text min 10 Characters. (Validate messages)

The form consists of the following elements:

- To:** Input field for recipient email address.
- Cc / Bcc:** Input field for carbon copy or blind carbon copy email addresses.
- Subject:** Input field for the email subject.
- Text:** Large input area for the email body text, with a placeholder "Text".
- File:** A section labeled "File" with a "Choose file" button for attaching files.
- SEND:** A blue rectangular button for sending the email.
- Attachment:** A blue rectangular button for managing attachments.

## CHAPTER#6 SPRING BOOT BATCH PROCESSING

### 1. Introduction:--

**Batch:**-- Multiple Operations are executed as one Task or one large/Big Task executed step by Step.

=>Every task is called as “**JOB**” and sub task is called as “**STEP**”.

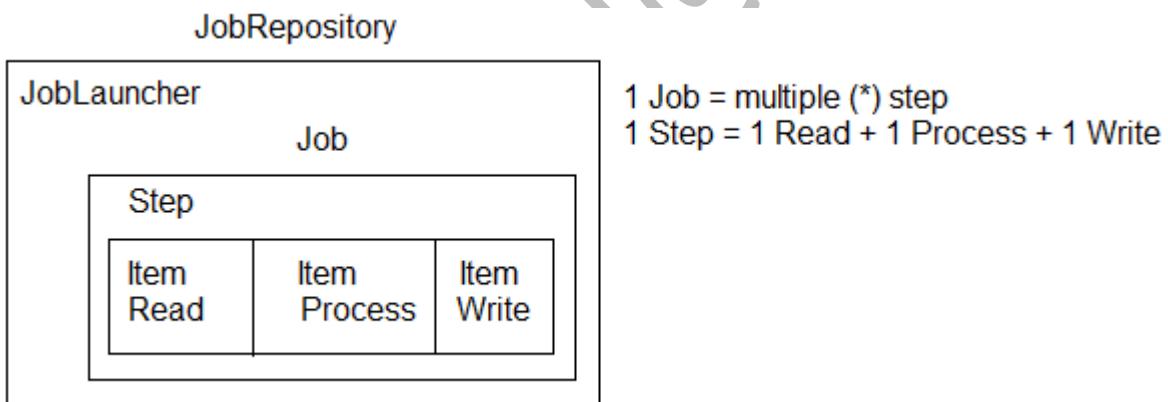
=>One JOB can contain one or more Steps (Step can also called as Sub Task).

=>One Step contains.

- a. Item Reader (Read data from Source).
- b. Item Process (Do calculations and logics/operations etc....).
- c. Item writer (Provide output to next Step or final output).

=>JOBS are invoked by JobLauncher also known as JobStarter.

=>JOB, Steps and Lanuncher details must be stored in JobRepository (Config file).



### Step Implementation:--

=>In a Job (work) we can define one or multiple steps which are executed in order (step by step).

=>Job may contain 1 step, job may contain 2 step..... job may contains many Steps so, finally 1-job = \* (many) Step.

=>Every step having 3 execution stages

- a. ItemReader<T>
- b. ItemProcessor<I, O>
- c. ItemWriter<T>.

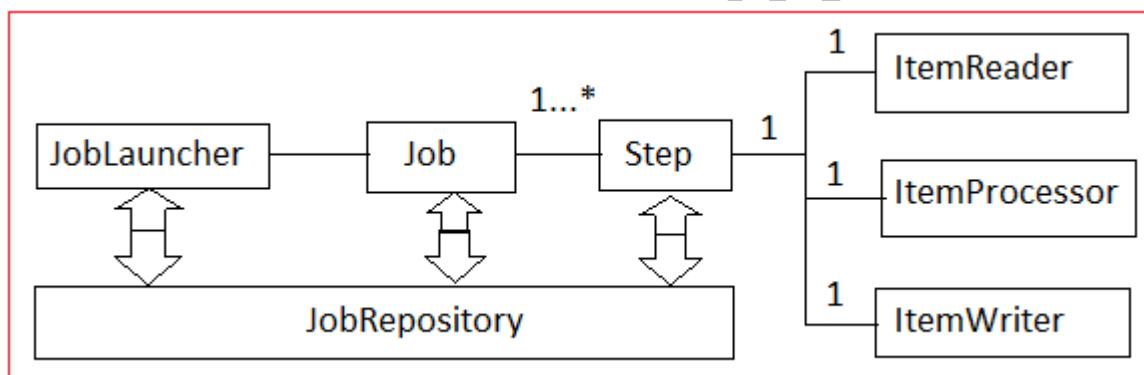
**a>ItemReader<T>** :-- It is used to read data (**Item**) from source (**Input location**) as input to current Step. A source can be File, DB, MQ, (Message Queues), Networks, XML, JSON... etc.

**b>ItemProcessor<I, O>** :-- It is used to process input data given by Reader and returns in Modifier (or same) format of data.

**c>ItemWriter<T>** :-- It will read bulk of Items from Processor at a time and writes to one destination. Even Destination can be a File (ex: DB, Text File, CSV, EXCEL, XML...etc).

=>To start Job, we need one **JobLauncher**... which works in general way or Scheduling based.

=>Here details like: Job, Steps, Launcher... are store in one memory which is called as **JobRepository** [Ex: H2DB or MySQL, DB ... any one DB].



#### NOTE:--

1. An Item can be String, Array, Object of any class, Collection (List/Set....).
2. ItemReader will read one by one Item from Source. For example Source has 10 items then 10 times ItemReader will be executed.
3. ItemReader GenericType must match with ItemProcessor Input GenericType.
4. ItemProcess will read item by item from Reader and does some process (calculate, convert, check conditions, and convert to any class Object etc...).
5. ItemProcessor will provide output (may be same as Input type) which is called as Transformed Type.
6. ItemWriter will collect all output Items into one List type from Processor at a time (Only one time).
7. ItemWriter writes data to any destination.

8. Source/Destination can be Text, DB, Network, MessageQueue, Excel, CSV, JSON data, XML etc...

### Step Execution Flow :-

=>Step contains 3 interfaces (impls classes) given with generic types.

- a. ItemReader<T> : (T= Input Data Type)
- b. ItemProcessor<I, O> : ( I must match with Reader T type and O must match with writer T type).
- c. ItemWriter<T> : (T = Type after processing)

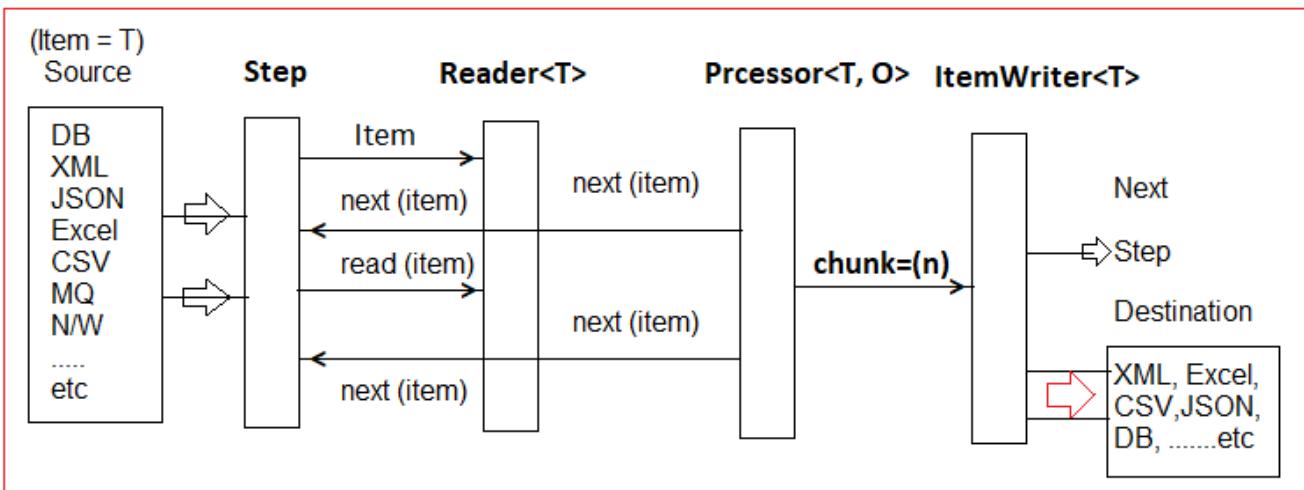
=>Here T/I/O can be String Object of any class or even collection (List, Set...).

=>Here ItemReader reads data from source with the helps of steps.

=>Reader and Processor are executed for every item, but writer is called based on chunk size.

Ex-- No. of Items = 200, chunk=50 then Reader and Processor are executed 200 times but writer is called one time after 50 items are processed (Here 4 times executed).

=>ItemWriter writes data to destination with the help of step.



### Spring Boot Batch coding Files and Steps:-

Batch programming is implemented in 3 stages.

1. Step Creation
2. Job Creation
3. Job Executions

**1. Step Creation:-** This process is used to construct one (or more) Step(s).

Ex:-- Step1, Step2, Step3..

=>here, Step is interface, It is constructed using "**StepBuilderFactory (C)**" with 3 inputs taken as (interfaces Impl class object) :

a>ItemReader<T>  
b>ItemProcessor<I, O>  
c>ItemWriter<T>

=>Programmer can define above interfaces Impl classes or can be also use exist (pre-defined) classes.

=>Reader, Writer, Processor is functional interfaces. We can define them using Lambda Expression and methods references even.

**2>Job Creation:**-- One job is collection of Steps executed in order (one by one).

=>Even job may contain one Step also. Here, job is interface which is constructed using "**JobBuilderFactory <C>**" and Step (I) instances.

=>To execute any logic before or after job, define Impl class for "**JobExecutionListener (I)**" having method

Like: beforeJob(...) : void and  
afterJob(...) : void

=>Here Listener is optional, It may be used to find current status of Batch (Ex: COMPLETED, STOPPED, FAILED...) start date and time, end date and time etc.

**3>Job Execution:**-- Once Steps and job are configured, then we need to start them using "**JobLauncher (I)**" run(...) method.

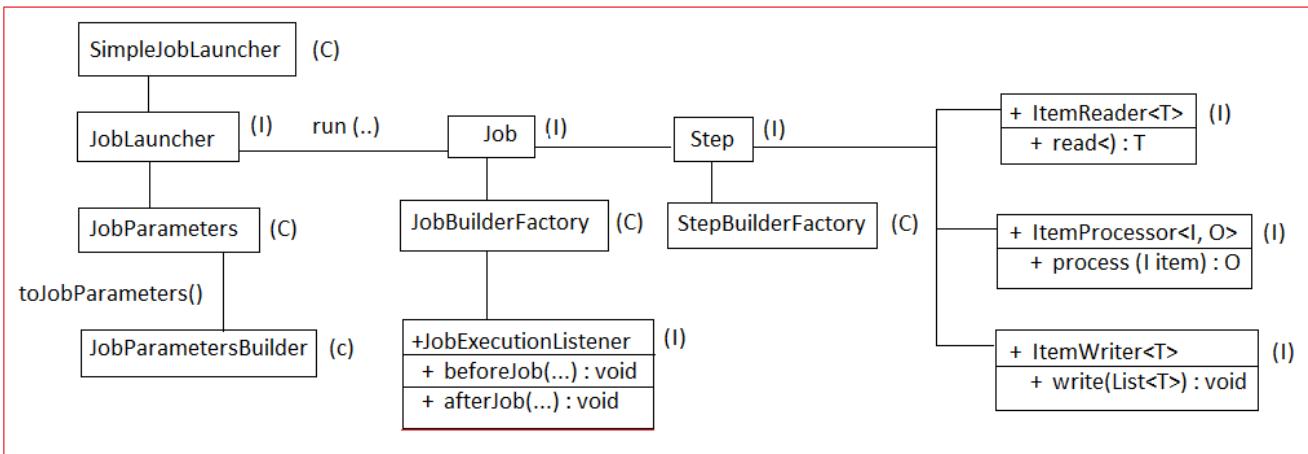
=>This run(...) method takes 2 parameters

- a. Job (I) and
- b. JobParameter (C)

=>Here, JobParameters (C) are inputs given to Job While starting this one.

Ex:- Parameters are : Server Data and Time, Customer Name flags (true/false), Task Name... etc.

=>JobParameters (C) object is created using "**JobParametersBuilder**" and its method toJobParameters().

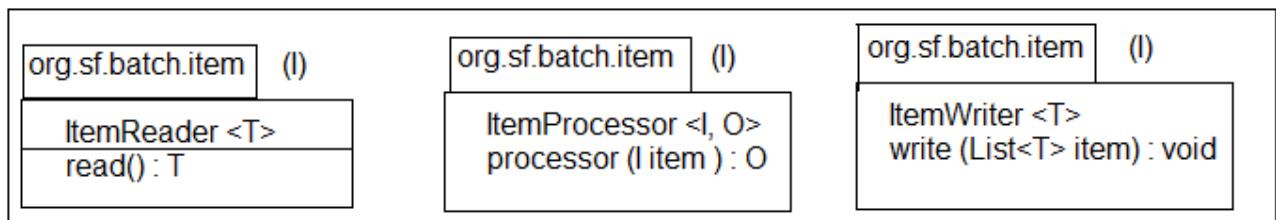


**Step** ::= One **Step** can be constructed using **StepBuilderFactory** (sf) class by providing name, chunk size, reader, processor and writer.

### StepBuilderFactory (sf):--

sf.get("step1")	=>Provide name of Step.
.<String, String> chunk(1)	=> No. of Items to be read at a time.
.reader (readerObj)	=>Any Impl class of IteamReader<T> (I)
.processor(processorObj)	=>Any Impl class of itemProcessor <I, O>
.writer(writerObje	=>Any Impl class of ItemWriter <T> (I)
.build();	=>Convert to step (impl class) Object.

### UML Notation:--



### JobBuilderFactory (C) :--

=>This class is used to create one or more Jobs using **Steps, listener, Incrementer....**  
 =>Job (I) Construction flow

### JobBuilderFactory jf :--

jf.get("jobA")	=>Job Name
.incremental(runIdIncrementar)	=>Incrementer
.listener (jobExListener)	=>Job Execution Listener
.start (stepA	=>First Step

.next (stepB)	=>Steps in Order
.next (stepC)	
.next (stepD)	
.build();	=>Create job Type

**JobExecutionListener (I):--**

=>This Interface is provided Spring Batch f/w, which gets called automatically for our Job.

=>For one job – one Listener can be configured.

=>It is an Interface which has provided two abstract methods.

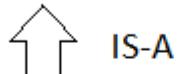
- a. beforeJob (JobExecution) : void
- b. afterJob (JobExecution) : void

=>If we write any impl class for above Listener (I) we need to implement two abstract method in our class.

=>Some times we need only one method in our class file afterJob() method is required. Then go for JobListenerAdapter(C) which has provided default impl logic for both **beforeJob; and afterJob();** methods.

=>JobExecution is a class which is used to find current job details like jobParameters, BatchStatus, stepExecutions etc...

JobExecutionListener (I)



IS-A

JobListenerAdapter (C)



IS-A

MyJobListener (C)

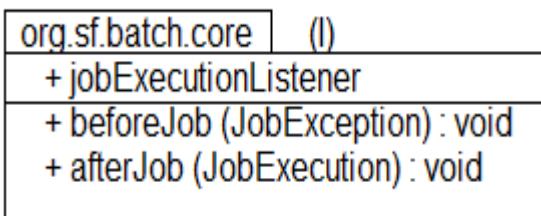
=>\*\*\*BatchStatus is an enum having possible values : COMPLETED, STARTING, STARTED, STOPPING, STOPPED, FAILED, ABANDONED, UNKNOWN.

**Q>What are possible Job Status/Batch Status in Batch Programming?**

A> All possible status in Batch Execution are given as enum “**BatchStatus**”. Those are

1	COMPLETED	=>Successfully done.
2	STARTING	=>About to call run(...).
3	STARTED	=>Entered into run(...).

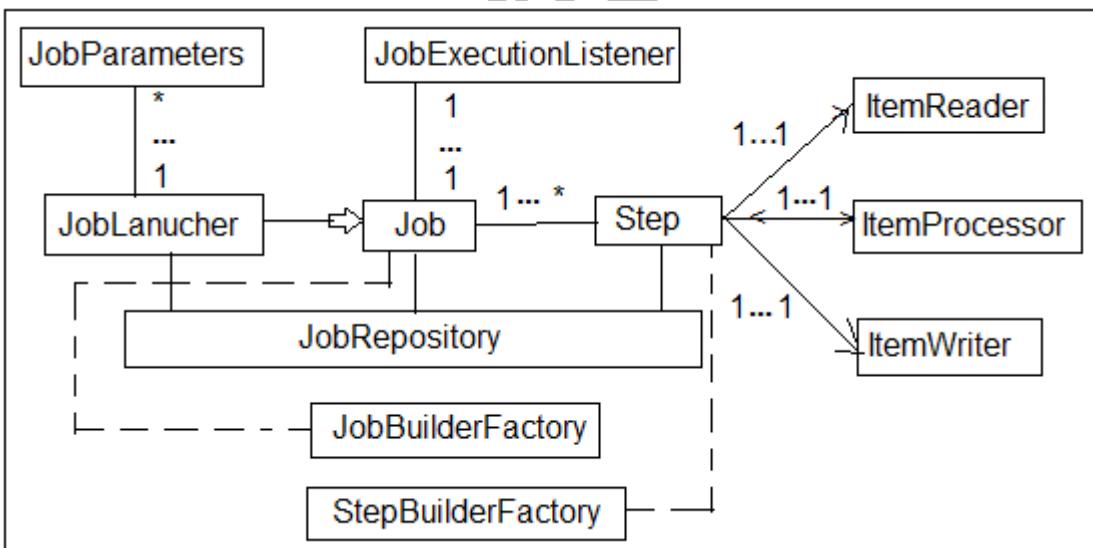
4	STOPPED	=>Abort & come out of run(...).
5	STOPING	=>Run(...) method finished.
6	FAILED	=>Exception in run(...).
7	ABANDONED	=>Stopped by External service.
8	UNKNOWN	=>Unable to provide.



### \*\*JobLauncher (I) :-

- =>This interface is used to invoke our Job with Parameters (input) like creationTime, uniqueId (name), programmerData etc.
- =>This Interface is having method run (Job, JobParameters).
- =>This Interface object is created by JobParametersBuilder which holds data in Map <key, Value> style.

### Final Spring Boot Batch Implementation Diagram:-



**Step#1:-** Define one Spring Starter Project and select **Spring Batch** or else add below dependencies.

```

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

```

**Step#2:-** Define Impl classes for IteamReader, ItemProcessor and IteamWriter  
Ex: Reader (C), Processor (C), Writer (C).

**Step#3:-** Define one class for Batch Configuration.

a>Create Beans for Reader, Processor, Writer.

b>Create Bean for Step using StepBuilderFactory  
(BatchConfig-----<>StepBuilderFactory)

c>Create Bean for job using JobBuilderFactory  
(BatchConfig-----<>JobBuilderFactory)

d>Define JobExecutionListener (I) Impl class (It is optional)

**Step#4:-** Create ConsoldeRunner to make job Execution using JobLauncher [run(...) method].

ConsoleRunner-----<>JobLauncher

ConsoleRunner-----<>Job

=>Provider JobParameters using its builder obj.

**Step#5:-** At Starter class level add annotation: @EnableBatchProcessing

**Step#6:-** add below key in application.properties

spring.batch.job.enabled=false

=>By default Starter class executes Job one time on startup application and even JobLauncher is executing another time. To avoid starter execution by starter class add above key as false value.

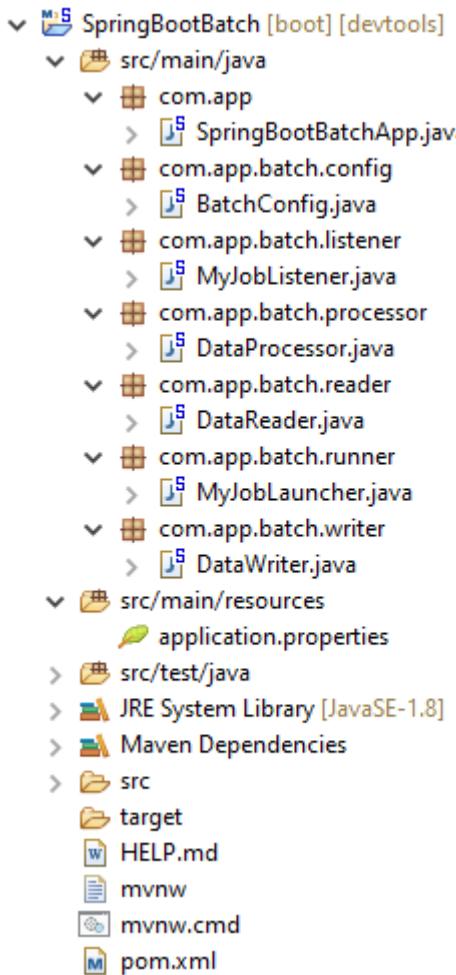
**Coding order:--**

1. Reader
2. Processor
3. Writer
4. Step Configuration using StepBuilderFactory
5. JobExecutionListener
6. Job Config –job BuilderFactory
7. JobParameters using JobParametersBuilder
8. JobLauncher using CommandLineRunner
9. \*\* Add key in application.properties

**Spring.batch.job.enabled=false**

=>To avoid execution of job multiple times (by Starter class)

## #30. Folder Structure of SpringBoot batch Programming:--

**application.properties:--**

#Disable this otherwise job executed one time by SpringBoot on startup  
 And also one more time by our launcher  
 spring.batch.job.enabled=false  
 spring.batch.initialize-schema=always  
 spring.datasource.driverClassName=oracle.jdbc.driver.OracleDriver  
 spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe  
 spring.datasource.username=system  
 spring.datasource.password=system

**1>DataReader:--**

```

package com.app.batch.reader;
import org.springframework.batch.item.ItemReader;
import org.springframework.batch.item.NonTransientResourceException;
import org.springframework.batch.item.ParseException;
import org.springframework.batch.item.UnexpectedInputException;
import org.springframework.stereotype.Component;
  
```

```
@Component
public class DataReader implements ItemReader<String> {

    String message[] = {"hi","hello","hoe"};
    int index;

    @Override
    public String read() throws Exception, UnexpectedInputException,
ParseException, NonTransientResourceException {
        if(index < message.length)
        {
            return message[index++];
        } else {
            index=0;
        }
        return null;
    }
}
```

## 2. DataProcessor:--

```
package com.app.batch.processor;
import org.springframework.batch.item.ItemProcessor;
import org.springframework.stereotype.Component;
```

```
@Component
```

```
public class DataProcessor implements ItemProcessor<String, String> {
```

```
    @Override
    public String process(String item) throws Exception {
        return item.toUpperCase();
    }
}
```

## 3. DataWriter:--

```
package com.app.batch.writer;
import java.util.List;
import org.springframework.batch.item.ItemWriter;
import org.springframework.stereotype.Component;
```

```
@Component
```

```
public class DataWriter implements ItemWriter<String> {
```

```
    @Override
    public void write(List<? extends String> items) throws Exception {
        for(String item:items) {
            System.out.println(item);
        }
    }
}
```

#### 4. BatchConfig.java:--

```
package com.app.batch.config;
import org.springframework.batch.core.Job;
import org.springframework.batch.core.JobExecutionListener;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.configuration.annotation.EnableBatchProcessing;
import org.springframework.batch.core.configuration.annotation.JobBuilderFactory;
import org.springframework.batch.core.configuration.annotation.StepBuilderFactory;
import org.springframework.batch.core.launch.support.RunIdIncrementer;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import com.app.batch.listener.MyJobListener;
import com.app.batch.processor.DataProcessor;
import com.app.batch.reader.DataReader;
import com.app.batch.writer.DataWriter;
```

```
@Configuration
@EnableBatchProcessing
public class BatchConfig {

    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    @Autowired
    private StepBuilderFactory stepBuilderFactory;

    @Bean
    public Job jobA() {
        return jobBuilderFactory.get("jobA")
                .incrementer(new RunIdIncrementer())
                .listener(listener())
                .start(stepA())
                // .next(step)
    }
}
```

```

        .build();
    }
    @Bean
    public Step stepA() {
        return stepBuilderFactory.get("stepA")
            .<String, String>chunk(50)
            .reader(new DataReader())
            .processor(new DataProcessor())
            .writer(new DataWriter())
            .build();
    }
    @Bean
    public JobExecutionListener listener() {
        return new MyJobListener();
    }
}

```

**5. MyJobListener.java:--**

```

package com.app.batch.listener;
import org.springframework.batch.core.JobExecution;
import org.springframework.batch.core.JobExecutionListener;
import org.springframework.stereotype.Component;

@Component
public class MyJobListener implements JobExecutionListener {

    @Override
    public void beforeJob(JobExecution jobExecution) {
        System.out.println(jobExecution.getStartTime());
        System.out.println(jobExecution.getStatus());
    }
    @Override
    public void afterJob(JobExecution jobExecution) {
        System.out.println(jobExecution.getEndTime());
        System.out.println(jobExecution.getStatus());
    }
}

```

**6. MyJobLauncher.java:--**

```

package com.app.batch.runner;
import org.springframework.batch.core.Job;
import org.springframework.batch.core.JobParameters;
import org.springframework.batch.core.JobParametersBuilder;

```

```
import org.springframework.batch.core.launch.JobLauncher;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;
```

```
@Component
public class MyJobLauncher implements CommandLineRunner {

    @Autowired
    private JobLauncher jobLauncher;
    @Autowired
    private Job job;

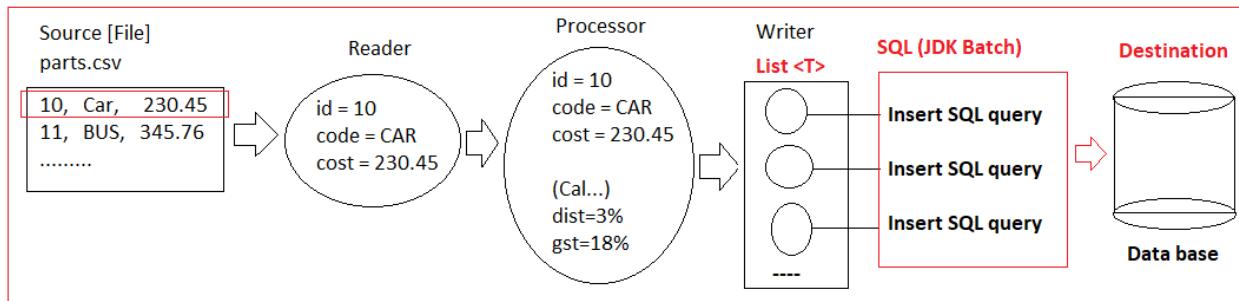
    @Override
    public void run(String... args) throws Exception {
        JobParameters jobParameters = new JobParametersBuilder()
            .addLong("time", System.currentTimeMillis())
            .toJobParameters();
        jobLauncher.run(job, jobParameters);
    }
}
```

### **Output:-**

```
2019-07-03 00:55:17.461 INFO 8488 --- [ restartedMain] o.s.b.c.l.support.SimpleJobLauncher : Job: [SimpleJob: [name=jobA]] launched v
Wed Jul 03 00:55:17 IST 2019
STARTED
2019-07-03 00:55:17.590 INFO 8488 --- [ restartedMain] o.s.batch.core.job.SimpleStepHandler : Executing step: [stepA]
HI
HELLO
HOE
Wed Jul 03 00:55:17 IST 2019
COMPLETED
2019-07-03 00:55:17.666 INFO 8488 --- [ restartedMain] o.s.b.c.l.support.SimpleJobLauncher : Job: [SimpleJob: [name=jobA]] completed
Main Class Executed
2019-07-03 00:55:17.705 INFO 8488 --- [      Thread-9] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown initiated...
2019-07-03 00:55:17.753 INFO 8488 --- [      Thread-9] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown completed.
```

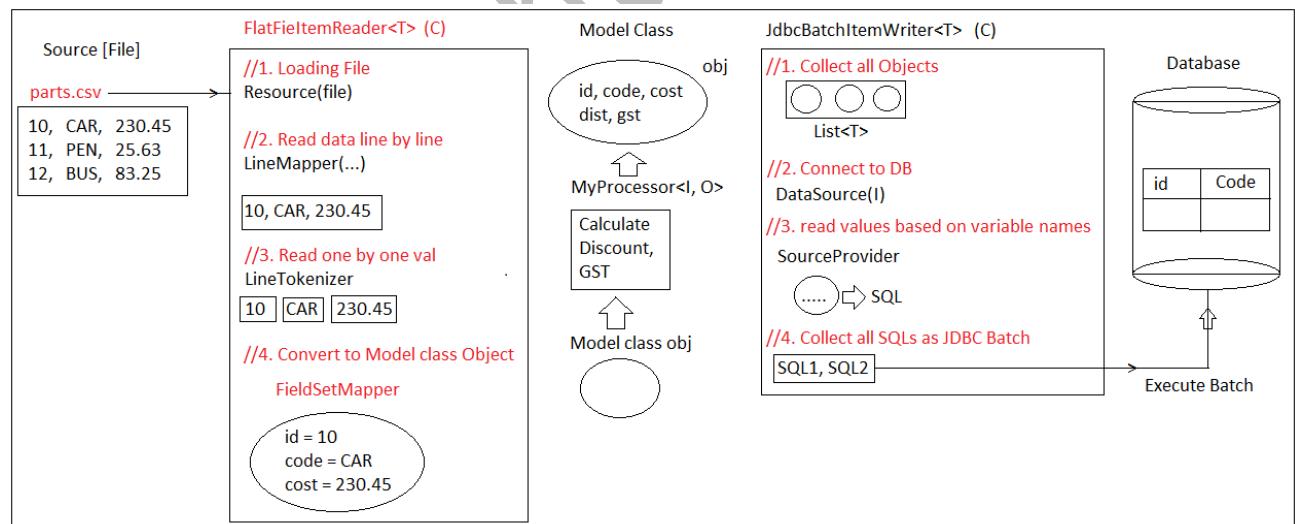
## **2. Spring Boot Batch Processing : Converting .csv file data to DataBase table:-**

- =>Consider input data given by csv file is related to products having product id, name, cost, by using one item reader convert .csv file data to Product class object.
- =>Define one Processor class to calculate gst (Goods and Service Tax) and discount of product.
- =>Finally Product should have id, name, cost, gst, discount.
- =>Use one ItemWriter class to convert one object to one Row in DB table.



- =>In realtime CSV (Comma Separated Values) Files are used to hold large amount of data using symbol / Tokenizer ',' or (It will be , . -, \, / ).
- =>This data will be converted to one Model class(T) object format.
- =>It means one line data(id, code, cost...) converted to one java class object by Reader.
- =>Calculate Discount and GST... etc using Processor class. Processor may return same class (or) different Object (i.e l==0)
- =>Based on chunk(int) size all objects returned by Processor will be collected into one List by Writer.
- =>Every Object in List will be converted into its equal "INSERT SQL..." .
- =>Multiple SQLs are converted to one JDBC Batch and sent to DB at a time.
- =>No of calls between Writer and Database depends on chunk size (and no of Items).

### Technical Flow for csv to DB using Spring Boot Batch:--



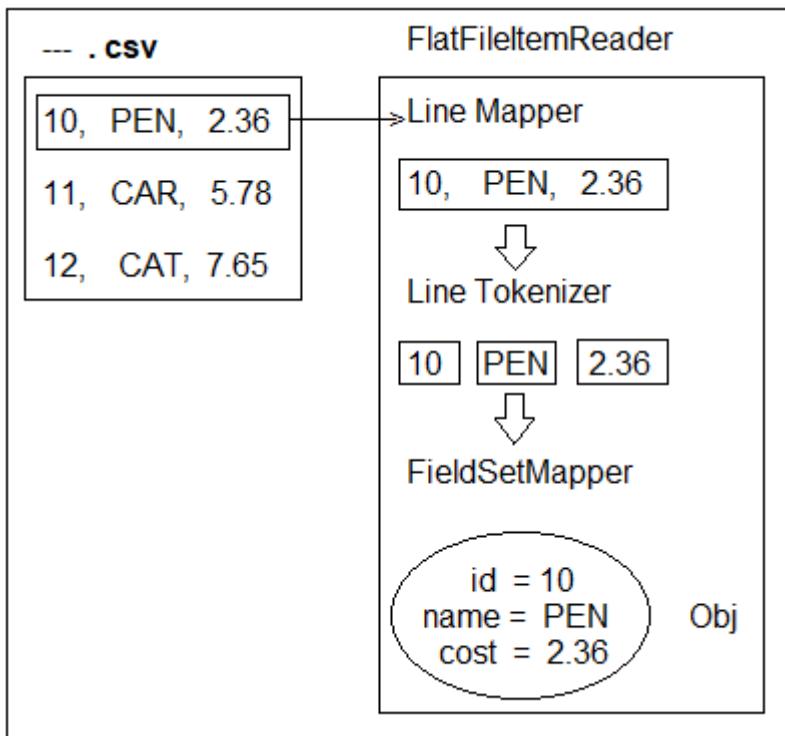
### FlatFileItemReader:---

- =>This class is provided by Spring Batch to read data from any Text Related file (.txt, .csv,...).

### DefaultLineMapper:-- It will load one row(/n) at a time as one Line Object.

**DelimitedLineTokenizer** :-- It will devide one line values into multiple values using any Delimiter (like comma, space, tab, new, line, dash.. etc).  
=>Default Delimiter is "," [Comma].

**BeanWrapperFieldSetMapper** :-- It maps data to variables, finally converted to one class type (TargetType).



#### Execution Flow:--

- =>Spring Boot Batch f/w has provided pre-defined ItemReaders and ItemWriters.
- =>FlatFileItemReader <T> is used to load any file (source) as input to read data example .txt, .csv,... etc.
- =>It will read one line data based on LineMapper (\n).
- =>One Line data is divided into multiple values based on Tokenizer (Delimiter = ,).
- =>These values are mapped to one class (T) type object also known as Target.
  
- =>This Target object is input to ItemProcessor. After processing object (calculations, Logical checks, data modifications... etc) Processor returns output type Object.
- =>JdbcBatchItemWriter collects all Items from ItemProcessor into List<T> based on **chunk (int)** size.
- =>Provide one DataSource (DB Connection) to communicate with DB Tables.

=>Define one SQL which inserts data into table based on Parameter (Variable Name) SourceProvider.

=>Multiple SQLs are converted to one batch and send to DB table.

\*\*\* Here Batch Size = Chunk Size.

Example chunk size = 150 (int value)

=>Chunk indicates maximum Items to be sent to Writer in one network call from Step.

=>For last network call chunk may contain few Items (no. of Items <chunk size).

---

=>In application.properties add below key-value pairs :

spring.batch.job.enabled=false

spring.batch.initialize-schema=always

=>Here **job.enabled=false** will disable execution of job by one time on app starts by Starter class.

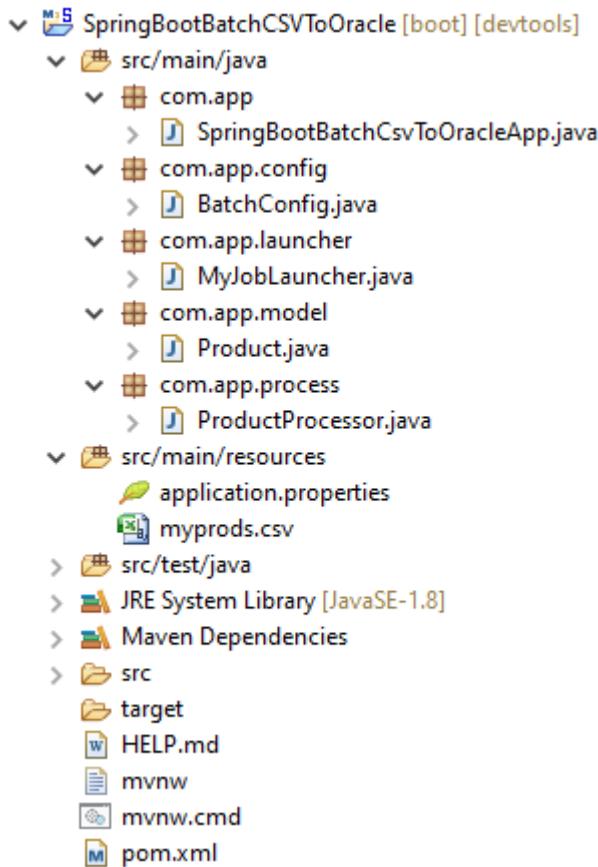
=>**Initialize-schema=always** will allow Spring Batch to communicate DB to hold its Repository details (like Job, Step, current status details...).

=>\*\*\*In case of Embedded Database initialize-schema not required.

### Coding Order for Batch :- csv to ORACLE

- 1> Model class
- 2> Processor class\*\*\*
- 3> BatchConfig
- 4> ConsoleRunner
- 5> Application.properties
- 6> Starter class
- 7> Pom.xml

## #31. Folder Structure of Spring Boot Batch transferring data from csv file to DB:--

**application.properties:--**

```
spring.batch.job.enabled=false
spring.batch.initialize-schema=always
spring.datasource.driverClassName=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe
spring.datasource.username=system
spring.datasource.password=system
```

**1. Model class (Product.java):--**

```
package com.app.model;
import lombok.Data;

@Data
public class Product {
    private Integer prodId;
    private String prodName;
    private Double prodCost;
    private Double prodGst;
    private Double prodDisc;
}
```

**2. ProductProcess.java:--**

```
package com.app.process;
import org.springframework.batch.item.ItemProcessor;
import com.app.model.Product;

public class ProductProcessor implements ItemProcessor<Product, Product> {

    @Override
    public Product process(Product item) throws Exception {
        item.setProdGst(item.getProdCost()*12/100.0);
        item.setProdDisc(item.getProdCost()*25/100.0);
        return item;
    }
}
```

**3. BatchConfig.java:--**

```
package com.app.config;
import javax.sql.DataSource;
import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.configuration.annotation.EnableBatchProcessing;
import org.springframework.batch.core.configuration.annotation.JobBuilderFactory;
import org.springframework.batch.core.configuration.annotation.StepBuilderFactory;
import org.springframework.batch.core.launch.support.RunIdIncrementer;
import org.springframework.batch.item.ItemProcessor;
import org.springframework.batch.item.ItemReader;
import org.springframework.batch.item.ItemWriter;
import org.springframework.batch.item.database.BeanPropertyItemSqlParameterSourceProvider;
import org.springframework.batch.item.database.JdbcBatchItemWriter;
import org.springframework.batch.item.file.FlatFileItemReader;
import org.springframework.batch.item.file.mapping.BeanWrapperFieldSetMapper;
import org.springframework.batch.item.file.mapping.DefaultLineMapper;
import org.springframework.batch.item.file.transform.DelimitedLineTokenizer;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.io.ClassPathResource;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
import com.app.model.Product;
import com.app.process.ProductProcessor;
```

```
@EnableBatchProcessing
@Configuration
public class BatchConfig
{
    //STEP
    @Autowired
    private StepBuilderFactory sf;

    @Bean
    public Step stepA() {
        return sf.get("stepA")
            .<Product, Product> chunk(3)
            .reader(reader())
            .processor(processor())
            .writer(writer())
            .build();
    }

    //JOB
    @Autowired
    private JobBuilderFactory jf;

    @Bean
    public Job jobA() {
        return jf.get("jobA")
            .incrementer(new RunIdIncrementer())
            .start(stepA())
            .build();
    }
}

//dataSource -- Creates DB connection
@Bean
public DataSource dataSource() {
    DriverManagerDataSource ds = new DriverManagerDataSource();
    ds.setDriverClassName("oracle.jdbc.driver.OracleDriver");
    ds.setUrl("jdbc:oracle:thin:@localhost:1521:xe");
    ds.setUsername("system");
    ds.setPassword("system");
    return ds;
}
```

```

//1. Item Reader from CSV file
/** Code snippet for CSV to ORACLE DB**/

@Bean
public ItemReader<Product> reader() {
    FlatFileItemReader <Product> reader = new FlatFileItemReader<Product>();
    //--reader.setResource(new FileSystemResource("d:/abc/products.csv"));
    //--loading file/Reading file
    reader.setResource(new ClassPathResource("myprods.csv"));
    //--read data line by line
    reader.setLineMapper(new DefaultLineMapper<Product>(){{
        //--make one into multiple part
        setLineTokenizer (new DelimitedLineTokenizer() { {
            //-- Store as variables with names
            setNames ("prodId", "prodName", "prodCost");
        }});
        setFieldSetMapper(new BeanWrapperFieldSetMapper<Product>() {{{
            //--Convert to model class object
            setTargetType (Product.class);
        }});
    }});
    return reader;
}

//2. Item Processor
@Bean
public ItemProcessor<Product, Product> processor() {
    //return new ProductProcessor();
    return (p)-> {
        p.setDisc(p.getCost()*3/100.0);
        p.setGst(p.getCost()*12/100.0);
        return p;
    }
}

//3. Item Writer
@Bean
public ItemWriter<Product> writer() {
    JdbcBatchItemWriter<Product> writer = new JdbcBatchItemWriter<>();
    writer.setDataSource(dataSource ());
    writer.setItemSqlParameterSourceProvider(new
        BeanPropertyItemSqlParameterSourceProvider<Product>());
}

```

```

writer.setSql("INSERT INTO PRODSTAB(PID, PNAME, PCOST, PGST, PDISC)
VALUES(:prodId, :prodName, :prodCost, :prodGst, :prodDisc)");

return writer;
}

```

#### 4. MyJobLauncher.java---

```

package com.app.launcher;
import org.springframework.batch.core.Job;
import org.springframework.batch.core.JobParametersBuilder;
import org.springframework.batch.core.launch.JobLauncher;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

@Component
public class MyJobLauncher implements CommandLineRunner{

    @Autowired
    private JobLauncher jobLauncher;
    @Autowired
    private Job job;

    @Override
    public void run(String... args) throws Exception {
        jobLauncher.run(job,new JobParametersBuilder()
            .addLong("time",System.currentTimeMillis())
            .toJobParameters());
    }
}

```

**DB Table:**-- Execute below SQL query before execution of program to create table.  
SQL>CREATE TABLE prodstab (PID **number** (10), PNAME varchar2 (50), PCOST number,  
PGST number, PDISC number);

#### Create a file under src/main/resources folder:--

myprods.csv : under src/main/Resources.  
10, PEN, 2.36  
11, CAR, 8.8  
12, BUS, 99.56

- =>To read file from outside Application from File System (D:/driver or C:/driver) then use **FileSystemResource**.
- =>It same way to read file from network (internet URL based) then use **UrlResource** in place of **ClassPathResource**.

**Task:--**

#1. Write Spring boot Batch Application To read data from database (Oracle DB) using “**JdbcCursorItemReader**” and write data to csv file using “**FlatFileItemWriter**”.

Task#1: (JDBC ->CSV)

**JdbcCursorItemReader<T>**

**FlatFileItemWriter<T>**

#2. Write data from MongoDB using “**MongoItemReader**” and write data to JSON file using “**JsonFileItemWriter**”.

Task#2: MongoDB to JSON file

**MongoItemReader<T>**

**JsonFileItemWriter<T>**

Task#2: ORACLE to CSV

(ORM -> CSV)

**HibernateCursorItemReader<T>**

**FlatFileItemWriter<T>**

**Note:--** Every step contains

a> ItemReader

b> ItemProcessor

c> ItemWriter

=>All these are functional Interface (contains only one (1) abstract method).

So Logic can be provided to these interfaces using Lambda Expression.

=>Lambda coding syntax:--

(parameters) -> {

    method body;

}

```

interface Sample { #1
    void m1();
}

class A implements Sample { #2

    public void m1() {
        System.out.println("OK");
    }
}

Sample s = new A(); #3

s.m1(); #4

```

↗

```

interface Sample { #1
    void m1();
}

#2 & #3 = Lambda
Sample s=
    () -> {
        System.out.println("OK");
    }

s.m1(); #4

```

=>Writing Lambda expression

**Simple code:--**

```

public class MyProcessor implements ItemProcessor<Product, Product> {
    public Product process(Product p) throws Exception {
        double cost=p.getCost();
        p.setDisc(cost*3/100.0);
        p.setGst(cost*12/100.0);
        return p;
    }
}
@Bean
public ItemProcessor<Product, Product>
process() {
    return new MyProcessor();
}

```

**Lambda Exp:--**

```

@Bean
ItemProcessor<Product, Product> process() {
    return (p) -> {
        double cost=p.getCost();
        p.setDisc(cost*3/100.0);
        p.setGst(cost*12/100.0);
        return p;
    };
}

```

\*\*\*Different ways of creating object and calling method:--

Consider below class:--

```
class Sample {  
  
    Sample() {  
        System.out.println("Constructor");  
    }  
    void show () {  
        System.out.println("Method");  
    }  
}
```

**Text class:--**

```
public class WayOfCreatingObject  
{  
    public static void main(String[] args) {
```

//1. Creating object and calling method

```
Sample s = new Sample();  
s.show();
```

//2. Creating Object and calling method

```
new Sample().show();
```

//3. Creating object (add extra code, override methods) and calling method

```
new Sample () {  
    public void show() {  
        System.out.println("NEW LOGIC");  
    }  
}.show();
```

//4. While creating object invoke method

```
new Sample() {  
    //Instance initialize block  
    {  
        show();  
    }  
};  
}
```

Ex:-- **Code Extension and instance block:--**

```
package com.app;

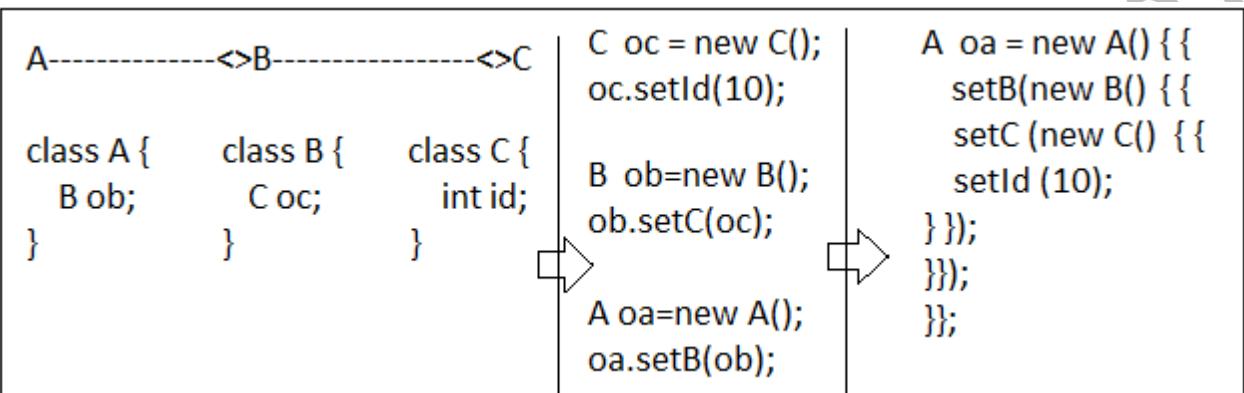
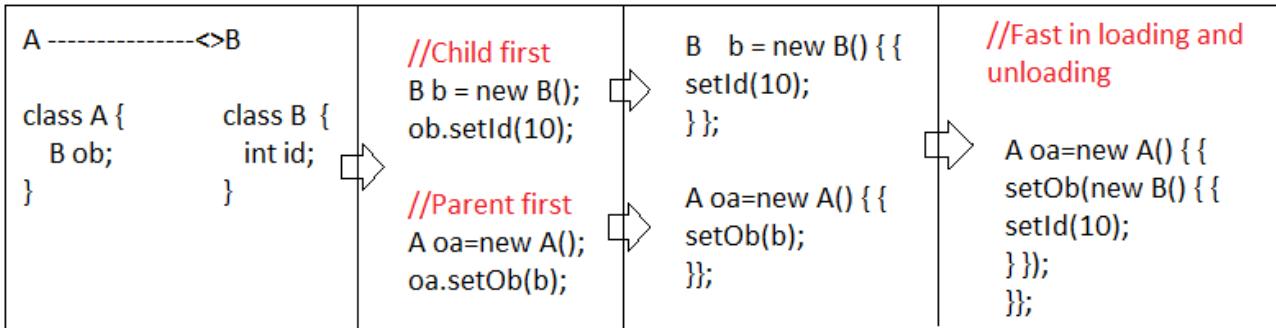
public class Sample {
    System.out.println("From instance-inside");
}
public sample() {
    System.out.println("From Constructor");
}
//getters setters.. toString
}

package Com.app;
public class Test
Public static void main {

    //1. Create object then call method
    Sample s1 = new Sample();
    s1.setSid(10);

    //2. calling method while creating obj
    Sample s2 = new Sample() {
        setSid(10);
    };

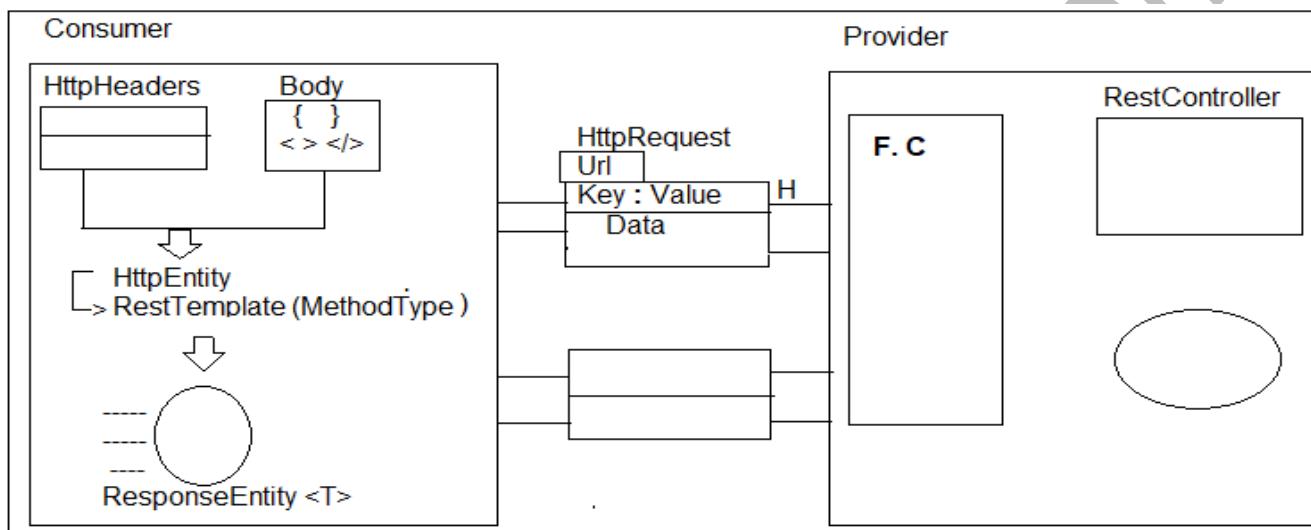
    //3. Code Extension for current object
    Sample s3 = new Sample() {
        void m1() {
            System.out.println("OK");
        }
        {
            setSid(10);
            m1();
        }
    };
}
```



## CHAPTER#7: SPRING BOOT REST (PROVIDER & CONSUMER)

### 1. Introduction:--

=>To implement ReSTful webservices API using simple annotations and Templates Spring boot ReST F/w has been introduced.  
=>Even to implement Microservices design Spring boot ReST API is used.



### NOTE:--

- a.>Consumer and Provider interchanges the data Primitive (String format only), Class Type (Object) and Collections.
- b.>Data will be converted to either **JSON or XML** format also known as Global Formats
- c.> String data will be converted to any other type directly.
- d.>ReST Controller supports 5 types of Request Methods Handling. Those are HTTP Method : GET, POST, PUT, DELETE and PATCH.
- e.>Controller class level use annotations `@RestController` (Stereotype), `@RequestMapping` (for URL).
- f.>Controller methods level use annotations.

TYPE	Annotations
GET	<code>@GetMapping</code>
POST	<code>@PostMapping</code>
PUT	<code>@PutMapping</code>
DELETE	<code>@DeleteMapping</code>
PATCH	<code>@PatchMapping</code>

g.>To provide Inputs to Request Methods in RestController, use annotations.

TYPE	Annotation
url?key=val	@RequestParam
/url/val/	@PathVariable
/url/key=val	@MatrixVariable
Http Header	@RequestHeader
Http Body	@RequestBody

\*\*\*All above annotations works on HttpRequest only, supports reading input data.

h.>By default Matrix Parameter is disabled (may not work properly). To enable this write below code in MVC config file.

**Example:**--

```
package com.app.controller;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.PathMatchConfigurer;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
import org.springframework.web.util.UrlPathHelper;

@Configuration
public class AppConfig implements WebMvcConfigurer
{
    @Override
    public void configurePathMatch(PathMatchConfigurer configure)
    {
        UrlPathHelper helper= new UrlPathHelper();
        helper.setRemoveSemicolonContent(false);
        configure.setUrlPathHelper(helper);
    }
}
```

i.>DispatcherServlet is autoConfigured in Spring Boot with URL Pattern mapped to /.

j.>AutoConfiguration Provided even for @EnableWebMvc,

@ComponentScan(Startsclass), @PropertySource("ClassPath:application.property")

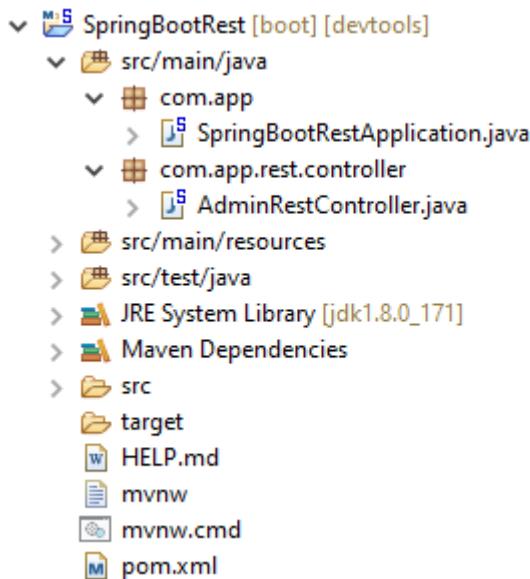
k.>Embedded server.

## #1. Spring Boot ReST Provider Example #1

**Step#1** -- Create Starter Project using web and devtools Dependencies. Details are

GroupId : com.app  
ArtifactId : SpringBootRestProvider  
Version : 1.0

## #32. Folder Structure of Spring Boot RestController with possible HttpRequest:--



**Step #2** -- Write one Controller class with class lever URL different method with HTTP MethodTypes.

```
package com.app.controller;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PatchMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/admin") //Optional
public class AdminRestController {
    @GetMapping("/show")
    public String helloMsgGet () {
        return "Hello From GET";
    }
}
```

```

    }

    @PostMapping("/show")
    public String helloMsgPost () {
        return "Hello From POST";
    }

    @PutMapping("/show")
    public String helloMsgPut () {
        return "Hello From PUT";
    }

    @DeleteMapping("/show")
    public String helloMsgDelete () {
        return "Hello From DELETE";
    }

    @PatchMapping("/show")
    public String helloMsgPatch () {
        return "Hello From PATCH";
    }
}

```

**application.properties:--**

server.port=2019

server.servlet.context-path=/myApp

\*\*\* Run Starter class and Test Application using POSTMAN

**-----POSTMAN SCREEN-----**

**GET http://localhost: 2019/ myapp /admin/show** **SEND**

**Postman Screen:--**

The screenshot shows the Postman application window. At the top, there's a toolbar with 'NEW', 'Runner', 'Import', and other icons. Below the toolbar, the URL 'http://localhost:2019/' is entered in the address bar, along with a collection named 'myApp'. The 'Builder' tab is selected. In the main area, a GET request is configured with the URL 'http://localhost:2019/myApp/admin/show'. The 'Authorization' tab is selected, showing 'Type' as 'No Auth'. Under the 'Body' tab, the response is displayed as a JSON object with one key-value pair: '1 Hello From GET'. The status bar at the bottom right indicates 'Status: 200 OK' and 'Time: 181 ms'.

**NOTE:--**

a. URL is case-sensitive, same URL can be used at multiple (GET, POST, PUT...) must be different.

b. Difference between PUT and Patch :

**PUT** :-- It indicates modify complete (full) data, based on Identity (ID-PK).

**PATCH** :-- It indicates modify partial (few data, based on Identity (ID-PK)).

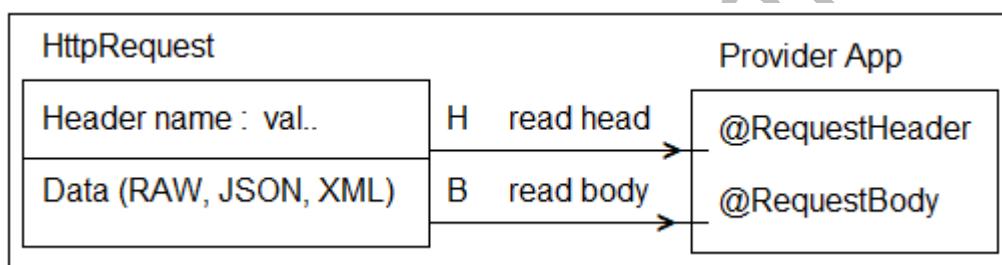
**Example#2 Read Data from Http Request (Head and Body).**

=>Http Request sends data using Header and body Section (both).

=>To Read any Header Parameter use code

- ❖ @RequestHeader DataType localVariable
- ❖ @RequestHeader (required=false) DataType localVariable
- ❖ @RequestHeader ("key") DataType localVariable

=>To read data from Body (RAW Data) use code @RequestBody

**Code:--**

```

package com.app.rest.controller;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestHeader;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/admin")
public class AdminRestController {
    @PostMapping("/head")
    public String readHead(@RequestHeader(required=false) String dept,
                          @RequestHeader("Content-Type") String type, @RequestBody String mydata) {
        return "Hello Head :" + dept + "," + type + ",Body:" + mydata;
    }
}
  
```

```

POST Http://localhost:2019/admin/head      SEND

  head
  dept          SAMPLE
  Content-Type   application/json

POST Http://localhost:2019/admin/head      SEND

  Body
    (*) raw   application/json

  {"message" : This is from Request Body"}

```

**NOTE:--**

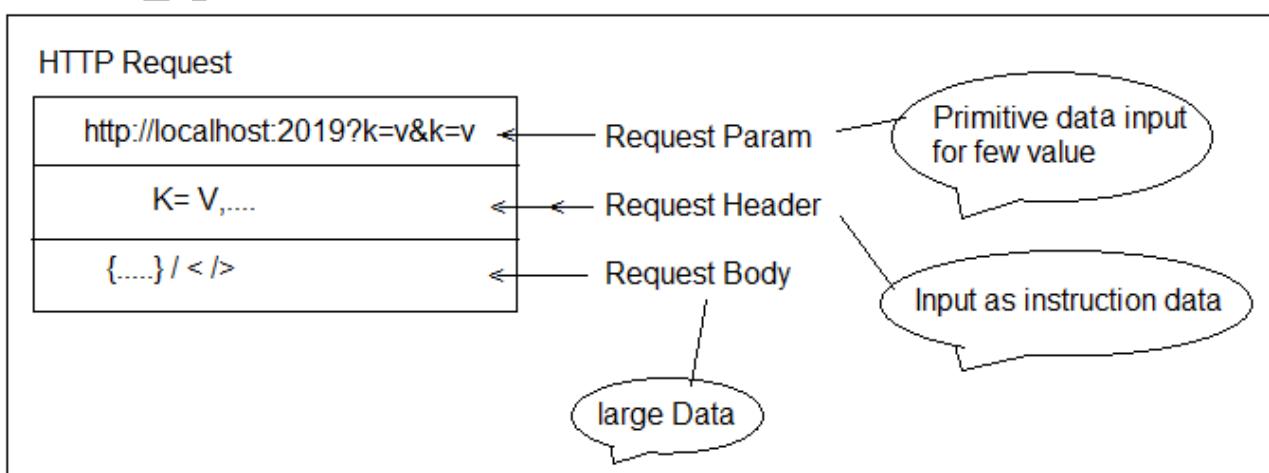
- ❖ Request Header is required (true) by default to make it optional, add code required=false.
- ❖ If key name localVariable Name same then providing key name is optional.
- ❖ Request Body Raw data (Characters or any...).
- ❖ Can be Stored in String with any variableName.

**2. Passing Input to RestController:--**

=>RestController will read input data either in primitive or in Type (Object/Collection).  
 =>To pass data/input to rest controller Spring has provided different concepts.

**Those are :--**

- 1>Request Header
- 2>Request Body \*\*\*
- 3>Request Parameter
- 4>Path Variable \*\*\*
- 5>Matrix Variable (disable mode by default)



**1>Request Header:**-- It provides data in key=value format (both are String type).

=>Request header also called as Header Parameters.

=>These are used to provide instructions to server (application/Controller).

Ex:-- Content-Type, Accept, host, Date, Cookies...

**2>Request Body:**-- To send large/bulk data like objects and collections data in the form of JSON/XML.

=>Even supported large text (Raw data).

=>Spring boot enables JSON conversion by default, but not XML (no JAXB API).

**3>Request Parameter:**-- To pass primitive data (like id, name, codes ... etc) as input, request parameters are used.

=>format looks like url?key=val&key=val...

=>Both key and value are String type by default.

#### Request Parameter Format:--

```
@RequestParam(
    value="key",
    required=true/false,
    defaultValue="-value-")
    DataType localVariable
```

#### Syntax #1:-

```
@RequestParam("key") DataType localVariable
```

Ex#1: @RequestParam("sname") String sn

#### Syntax #2:-

```
@RequestParam DataType localVar
```

Ex#2: @RequestParam String sname

=>If key name and local variable name are same then key is not

#### Syntax #3:-

```
@RequestParam(required=false) DT localVariable
```

=>To make key-value is optional @RequestParam (required=false) String sname

#### Syntax #4:-

```
@RequestParam (required=false, defaultValue="-DATA-")DT localVariable
```

=>To change **default** value from **null** to any other value.

```
@RequestParam(required=false, defaultValue="No DATA") String sname
```

**Controller code:--**

```
package com.app.rest.controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class AdminRestController {
    @GetMapping("/show")
    public String showMsg(@RequestParam (value="sname", required=false,
        defaultValue="NO DATA") String sid) {
        return "Hello:" +sid;
    }
}
```

**Path Variable [Path Parameters]:--**

=>We can send data using URL (path).

=>It supports only Primitive Data.

**Paths are two types:--**

a.>Static Path [format:/url]

b.>Dynamic Path [format:/{key}]

=>Static Path indicates **URL**, where as Dynamic Path indicates **Data** at runtime.

=>While sending data using Dynamic Path key should not be used. Only data

=>Order must be followed in case of sending multiple Parameters.

=>To read data at controller method, use annotation : **@PathVariable**.

**Syntax:--**

@PathVariable datatype keyName

**Controller code:--**

```
package com.app.controller.rest;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;
```

@RestController

**public class** StudentController

{

    @GetMapping("/show/{sid}/{sname}/{sfee}")

**public** String showMsg(

        @PathVariable **int** sid,

        @PathVariable **String** sname,

        @PathVariable **double** sfee)

}

**return** "Heloo :" +sid+"," +sname+ "," +sfee;

}

}

Example URL :-- <http://localhost:2019/show/10/Uday/56.8>

### 3. RestController : Method ReturnType:--

=>We can use String (for primitive Data), A classType or Any CollectionType(List, Set, Map...) as Method ReturnType.

=>If return types String then same data will be send to Controller.

=>If return type is non-String (Class or Collection Type) then Data converted to Global Format (ex: JSON/XML).

=>Default conversion Type supported by Boot is JSON (Java Script Object Notation).

=>Even “**ResponseEntity<T>**” can be used as Return Type which holds Body (T) and status (HttpStatus enum).

### Possible Http Status are (5):--

Code	Types
1xx	Informational
2xx	Success
3xx	Redirect
4xx	Client Side Error
5xx	Server Side Error

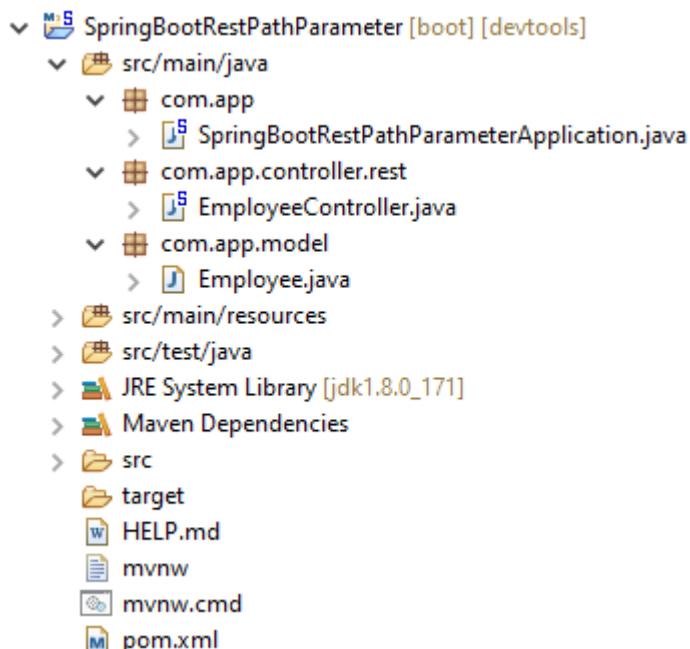
=>To Convert Object to JSON (and JSON to Object) SpringBoot uses JACKSON API.

**Step #1:-** Create one SpringBoot starter App with web, devtools dependencies.

**Step #2:-** Add below dependency in pom.xml for XML (JAXB) supports.

```
<dependency>
<groupId>com.fasterxml.jackson.dataformat</groupId>
<artifactId>jackson-dataformat-xml</artifactId>
</dependency>
```

### #33. Folder Structure of RestController with HttpStatus methods:--



#### Step#3 Model class:--

```
package com.app.model;
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement
public class Employee {
    private Integer empld;
    private String empName;
    private double empSal;

    public Employee() {
        super();
    }
```

```

public Employee(Integer empld, String empName, double empSal) {
    super();
    this.empld = empld;
    this.empName = empName;
    this.empSal = empSal;
}
public Integer getEmpld() {
    return empld;
}
public void setEmpld(Integer empld) {
    this.empld = empld;
}
public String getEmpName() {
    return empName;
}
public void setEmpName(String empName) {
    this.empName = empName;
}
public double getEmpSal() {
    return empSal;
}
public void setEmpSal(double empSal) {
    this.empSal = empSal;
}
@Override
public String toString() {
return "Employee [empld=" + empld + ", empName=" + empName + ", empSal=" +
empSal + "]";
}
}

```

#### Step #4 Controller class:-

```

package com.app.controller.rest;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

```

```
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
import com.app.model.Employee;

@RestController
public class EmployeeController {
    @GetMapping("/showA")
    public String showA() {
        return "Hello-String";
    }
    @GetMapping("/showB")
    public Employee showB() {
        return new Employee(22, "UDAY", 3.8);
    }
    @GetMapping("/showC")
    public List<Employee> showC() {
        return Arrays.asList(new Employee(22, "Uday", 6.8),
                new Employee(23, "Neha", 6.8),
                new Employee(24, "Ramu", 6.8));
    }
    @GetMapping("/showD")
    public Map<String, Employee> showD() {
        Map<String, Employee> map= new HashMap<>();
        map.put("e1", new Employee(22, "UDAY", 4.6));
        map.put("e1", new Employee(23, "NEHA", 8.2));
        map.put("e1", new Employee(24, "RAJA", 9.5));
        return map;
    }
    @GetMapping("/showE")
    public ResponseEntity<String> showE() {
        ResponseEntity<String> resp = new ResponseEntity<String> ("Hello RE ",
        HttpStatus.OK);
    }
}
```

```

        return resp;
    }
}

```

=>Run the application and enter below urls one by one and show output.

- 1> <http://localhost:2019/showA>
- 2> <http://localhost:2019/showB>
- 3> <http://localhost:2019/showC>
- 4> <http://localhost:2019/showD>

#### **POSTMAN SCREEN :--**

GET <http://localhost:2019/showD> SEND

<b>Headers</b>	
<b>Key</b>	<b>Value</b>
Accept	application/xml

#### **Output Screen:--**

The screenshot shows the Postman interface with a GET request to <http://localhost:2019/showD>. The 'Headers' tab is active, containing the key-value pair 'Accept: application/json'. The 'Body' tab shows the JSON response:

```

1 [{}]
2   "e1": {
3     "empId": 24,
4     "empName": "RAJA",
5     "empSal": 9.5
6   }
7 ]

```

#### **Spring Boot ReST + Data JPA +MySQL CRUD Operations (Rest API with Swagger):--**

=>Here, define ReST Collector which works for JSON and XML Data input/output.

=>For Primitive inputs use PathVariable.

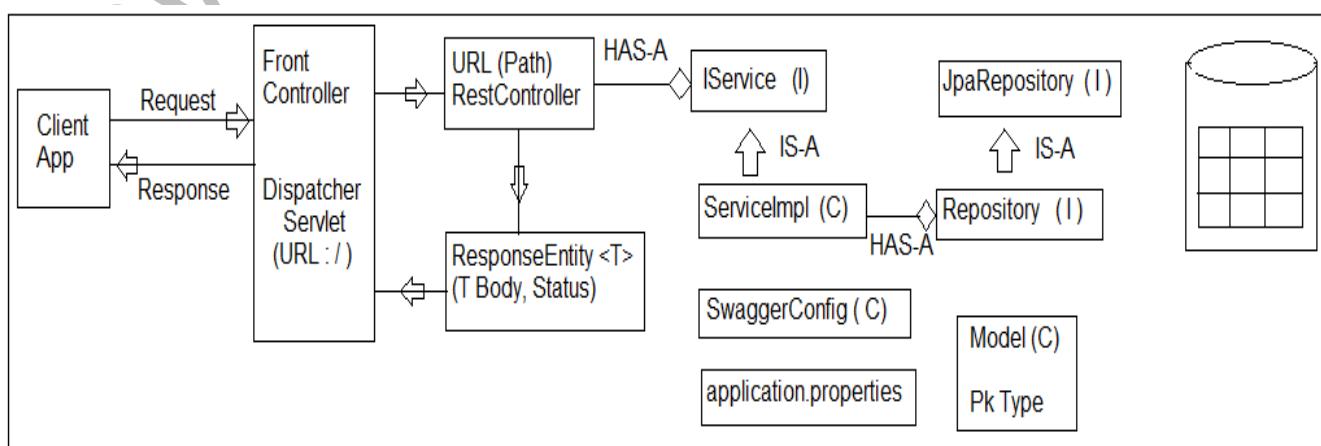
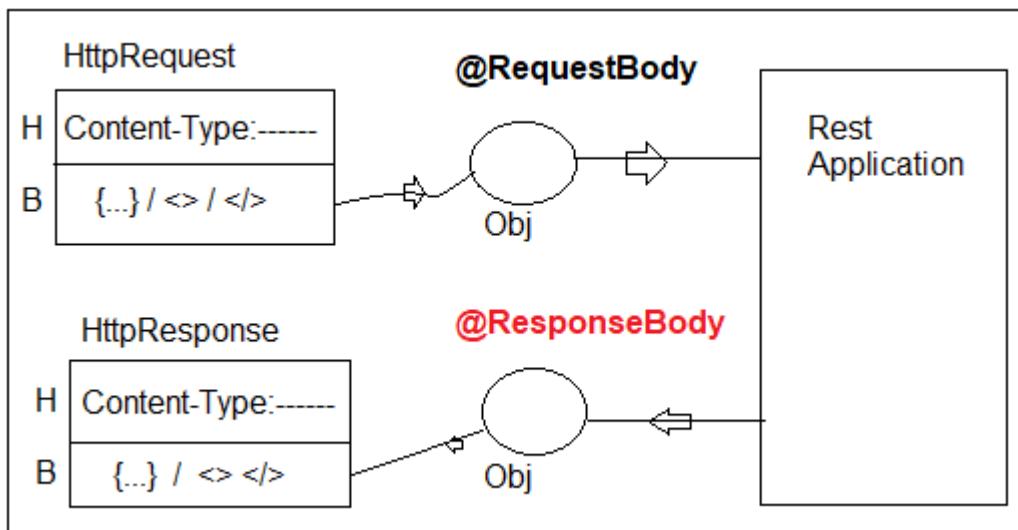
=>ReST Controller must return output type as ResponseEntity<T> which holds Body(T) and HttpStatus Ex:-- 500, 404, 200, 400 etc..

Operation Type	Http Method Annotation
Save	@PostMapping
Update	@PutMapping
Delete	@DeleteMapping
getOne	@GetMapping
get all	@GetMapping

### MediaType Conversion annotations are:--

- =>@RequestBody and @ResponseBody are the MediaType annotations.
- =>Here, @ResponseBody is applied when we write @RestController annotation over class (\*\*Not handled by programmer). It converts **Class/Collection** type to JSON/XML.
- =>@RequestBody should be handled by programmer. It converts JSON/XML Data to Object format.

### Spring Boot MediaType:--



**Coding Order:--**

**Step#1:-** Create Project with web, Jpa, devtools, mysql dependencies.

**Step#2:-** add jackson-dataformat-xml in pom.xml.

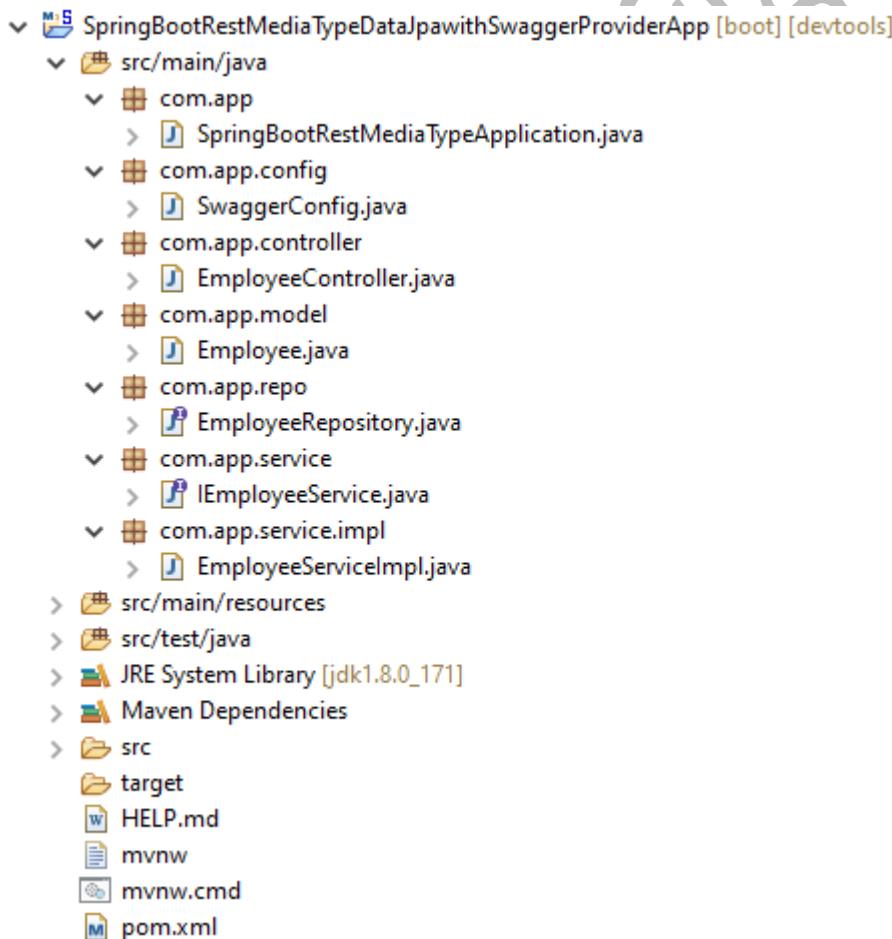
**Step#3:-** Define Model class, Repository, IService and ServiceImpl in order.

=>Student.java = Model  
 =>StudentRepository.java = Repository  
 =>IService.java = IService  
 =>StudentServiceImpl.java = Impl

**Step#4:-** In application.properties provide keys details server port, datasource and jpa (dialect, show-sql...).

**Step#5:-** Define RestController

### #34. Folder Structure of Spring Boot Rest with DataJpa and Swagger

**Implementation:--**

**1. application.properties:--**

```
server.port=2019
## DataSource Config ##
spring.datasource.driverClassName=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe
spring.datasource.username=system
spring.datasource.password=system

## Hibernate Config ##
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.database-platform=org.hibernate.dialect.Oracle10gDialect
```

**#2 Model class (Employee.java):--**

```
package com.app.model;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.xml.bind.annotation.XmlRootElement;

@Entity
@XmlRootElement
@Table(name="emp_tab")
public class Employee {

    @Id
    @GeneratedValue
    @Column(name="emp_id")
    private Integer empld;

    @Column(name="emp_name")
    private String empName;
    @Column(name="emp_sal")
    private double empSal;
```

```
public Employee() {
    super();
}

public Employee(Integer empId, String empName, double empSal) {
    super();
    this.empId = empId;
    this.empName = empName;
    this.empSal = empSal;
}

public Integer getEmpId() {
    return empId;
}

public void setEmpId(Integer empId) {
    this.empId = empId;
}

public String getEmpName() {
    return empName;
}

public void setEmpName(String empName) {
    this.empName = empName;
}

public double getEmpSal() {
    return empSal;
}

public void setEmpSal(double empSal) {
    this.empSal = empSal;
}

@Override
public String toString() {
    return "Employee [empId=" + empId + ", empName=" + empName + ",
empSal=" + empSal + "]";
}
```

**#3 Repository Interface (EmployeeRepository.java):--**

```
package com.app.repo;
import org.springframework.data.jpa.repository.JpaRepository;
import com.app.model.Employee;

public interface EmployeeRepository extends JpaRepository<Employee, Integer> { }
```

**#4 Service Interface (IEmployeeService.java):--**

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

public interface IEmployeeService {

    public Integer saveEmployee(Employee e);
    public void updateEmployee(Employee e);
    public void deleteEmployee(Integer id);
    public Optional<Employee> getOneEmployee(Integer id);
    public List<Employee> getAllEmployees();
    public boolean isPresent(Integer id);
}
```

**#5 ServiceImp class(EmployeeserviceImpl.java):--**

```
package com.app.service.impl;
import java.util.List;
import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import com.app.model.Employee;
import com.app.repo.EmployeeRepository;
import com.app.service.IEmployeeService;
```

```
@Service
```

```
public class EmployeeserviceImpl implements IEmployeeService {
```

```
@Autowired  
private EmployeeRepository repo;  
  
@Transactional  
public Integer saveEmployee(Employee e) {  
    return repo.save(e).getEmpId();  
}  
  
@Transactional  
public void updateEmployee(Employee e) {  
    repo.save(e);  
}  
  
@Transactional  
public void deleteEmployee(Integer id) {  
    repo.deleteById(id);  
}  
  
@Transactional(readOnly=true)  
public Optional<Employee> getOneEmployee(Integer id) {  
    return repo.findById(id);  
}  
  
@Transactional(readOnly = true)  
public List<Employee> getAllEmployees() {  
    return repo.findAll();  
}  
  
@Transactional(readOnly = true)  
public boolean isPresent(Integer id) {  
    return repo.existsById(id);  
}  
}
```

#### #6 Controller class (EmployeeController.java):--

```
package com.app.controller;  
import java.util.List;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.GetMapping;
```

```
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import com.app.model.Employee;
import com.app.service.IEmployeeService;

@Controller
@RequestMapping("/rest/employee")
public class EmployeeController {

    @Autowired
    private IEmployeeService service;

    //1. save student data
    @PostMapping("/save")
    public ResponseEntity<String> save(@RequestBody Employee employee){
        ResponseEntity<String> resp=null;
        try {
            Integer id=service.saveEmployee(employee);
            resp=new ResponseEntity<String>("Employee "+id+" created", HttpStatus.OK);
        } catch (Exception e) {
            resp=new ResponseEntity<String>(e.getMessage(),
                HttpStatus.INTERNAL_SERVER_ERROR);
            e.printStackTrace();
        }
        return resp;
    }

    //2. get All Records
    @GetMapping("/all")
    public ResponseEntity<?> getAll(){
        ResponseEntity<?> resp=null;
        List<Employee> list=service.getAllEmployees();
        if(list==null || list.isEmpty()) {
```

```

        String message="No Data Found";
        resp=new ResponseEntity<String>(message,HttpStatus.OK);
    } else {
        resp=new ResponseEntity<List<Employee>>(list,HttpStatus.OK);
    }
    return resp;
}

//3. delete based on id , if exist
@DeleteMapping("/delete/{id}")
public ResponseEntity<String> deleteById(@PathVariable Integer id)
{
    ResponseEntity<String> resp=null;
    //check for exist
    boolean present=service.isPresent(id);
    if(present) {
        //if exist
        service.deleteEmployee(id);
        resp=new ResponseEntity<String>("Deleted "+id+" successfully",HttpStatus.OK);
    } else { //not exist
        resp=new ResponseEntity<String>(""+id+" Not
Exist",HttpStatus.BAD_REQUEST);
    }
    return resp;
}

//4. update data
@PutMapping("/update")
public ResponseEntity<String> update(@RequestBody Employee employee){
    ResponseEntity<String> resp=null;

    //check for id exist
    boolean present=service.isPresent(employee.getEmplId());
    if(present) { //if exist
        service.updateEmployee(employee);
        resp=new ResponseEntity<String>("Updated Successfully",HttpStatus.OK);
    } else {
}

```

```

        //not exist
        resp=new ResponseEntity<String>("Record "+employee.getEmpId()+" not
found",HttpStatus.BAD_REQUEST);
    }
    return resp;
}

```

### #1. POSTMAN SCREEN for POST Method:--

**POST** <http://localhost:2019/rest/employee/save> **SEND**

#### Body

raw application/Json

```
{
  "empName" : "Uday Kumar",
  "empSal"   : 5563.3
}
```

### POSTMAN OUTPUT SCREEN for POST Method:--

The screenshot shows the Postman interface with the following details:

- Request URL:** http://localhost:2019/rest/employee/save
- Method:** POST
- Headers:** (2)
- Body:** (selected)
  - Content Type: raw
  - JSON (application/json)

```
{"empName": "Uday Kumar", "empSal": 45.87}
```
- Response:**
  - Status: 200 OK
  - Time: 200 ms
  - Body: Employee '4' created

## #2. POSTMAN SCREEN for GET Method:--

**GET** [Http://localhost:2019/ rest/employee/all](http://localhost:2019/rest/employee/all) **SEND**

**Header**

Key	Value
-----	-------

Accept application/json

**POSTMAN OUTPUT SCREEN for GET Method:--**

The screenshot shows the Postman interface with a GET request to <http://localhost:2019/rest/employee/all>. The Headers tab is selected, showing a key 'Accept' with value 'application/json'. The response body is displayed in Pretty JSON format:

```

1 [ ]
2 {
3   "empId": 1,
4   "empName": "Uday Kumar",
5   "empSal": 45.87
6 }
7 ]
  
```

**#3. POSTMAN SCREEN for DELETE Method:--**

**DELETE** <http://localhost:2019/employee/delete/4> **SEND**

**POSTMAN SCREEN for DELETE Method:--**

The screenshot shows the Postman interface with a DELETE request to <http://localhost:2019/rest/employee/delete/3>. The Body tab is selected. The response body is displayed in Text format:

```

1 Deleted '3' successfully
  
```

**NOTE:--** If request Id is not present then Return Http Status – **400 BAD-REQUEST**

DELETE <http://localhost:2019/rest/employee/delete/3>

Params Send Save

Authorization Headers Body Pre-request Script Tests

Key	Value	Description
New key	Value	Description

Body Cookies Headers (4) Test Results

Status: 400 Bad Request Time: 165 ms

Pretty Raw Preview Text `1 '3' Not Exist|`

#### #4. POSTMAN SCREEN for PUT Method:--

**PUT** <http://localhost:2019/employee/update> SEND

Body

raw application/json

```
{
  "empId": 1,
  "empName" :"Venkat Redy",
  "empsal" : 67.8
}
```

#### POSTMAN SCREEN for PUT Method:--

PUT <http://localhost:2019/rest/employee/update>

Params Send Save

Authorization Headers (2) Body Pre-request Script Tests

form-data x-www-form-urlencoded raw binary JSON (application/json)

1 {"empId" : 1, "empName" : "Venkat Redy", "empsal" : 98.87|

Body Cookies Headers (3) Test Results

Status: 200 OK Time: 281 ms

Pretty Raw Preview Text `1 Updated Successfully|`

#### 4. Enable Swagger UI in Spring Boot ReST Application:--

=>Compared to all other tools Swagger is a RichSet API provides dynamic UI based on code written for Rest Controller with common Paths.

**Step#1:-** Add below dependencies in pom.xml

```
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.7.0</version>
</dependency>
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.7.0</version>
</dependency>
```

Flow	Meaning
->Docket ()	=>Create Docket
->select ()	=>Choose Rest classes
->apis(basepackage())	=>Classes are in package
->paths (regex())	=>Having common path
->build()	=>Create final output

**Step#2:- Define Swagger Configuration class in Application (SwaggerConfig.java):--**

```
package com.app.config;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spring.web.plugins.Docket;
import springfox.documentation.swagger2.annotations.EnableSwagger2;
import springfox.documentation.builders.RequestHandlerSelectors;
import springfox.documentation.builders.PathSelectors;

@Configuration
@EnableSwagger2
public class SwaggerConfig {
```

```

@Bean
public Docket myApi() {
    return new Docket(DocumentationType.SWAGGER_2)
        .select()
        .apis(RequestHandlerSelectors.basePackage("com.app.controller.rest"))
        .paths(PathSelectors.regex("/rest.*"))
        .build();
}

```

\*\* basePackage () is a static method defined in RequestHandlerSelectors (C) and in same way regex() is a static method defined in PathSelectors (C).

**Step#3:-** Run starter class and enter URL :

<http://localhost:2019/swagger-ui.html>

<http://localhost:2019/rest/employee/save>

{ "empId" : 10, "empName" : "Uday Kumar", "empSal" : 45.87}

**Output:-**

The screenshot shows a web browser window with the following details:

- Address Bar:** localhost:2023/swagger-ui.html#/
- Toolbar:** Back, Forward, Home, Refresh, Search, Favorites, etc.
- Header:** **swagger** (highlighted in green), default (/v2/api-docs), Explore
- Content Area:**
  - Section Title:** Api Documentation
  - Sub-section:** Api Documentation
  - Version:** Apache 2.0
  - API Endpoint:** employee-controller : Employee Controller
  - Operations:**
    - GET /rest/employee/all** (blue button)
    - DELETE /rest/employee/delete/{id}** (red button)
    - POST /rest/employee/save** (green button)
    - PUT /rest/employee/update** (orange button)
  - Buttons:** Show/Hide, List Operations, Expand Operations
  - Notes:** [ BASE URL: / , API VERSION: 1.0 ]

## SCREEN#1:-- Swagger Screen for Save Operation

**POST** /rest/employee/save save

**Response Class (Status 200)**  
string

**Response Content Type** `*/*` ▾

**Parameters**

Parameter	Value	Description	Parameter Type	Data Type
employee	<pre>{   "empId": 34,   "empName": "Uday Kumar",   "empSal": 55,000 }</pre>	employee	body	<a href="#">Model</a>   <a href="#">Example Value</a> <pre>{   "empId": 0,   "empName": "string",   "empSal": 0 }</pre>

Parameter content type: `application/json` ▾

**Response Messages**

HTTP Status Code	Reason	Response Model	Headers
201	Created		
401	Unauthorized		
403	Forbidden		
404	Not Found		

[Try it out!](#)

## SCREEN#2:-- Swagger Screen for Delete Operation.

**DELETE** /rest/employee/delete/{id} deleteById

**Response Class (Status 200)**  
string

**Response Content Type** `*/*` ▾

**Parameters**

Parameter	Value	Description	Parameter Type	Data Type
id	(required)	id	path	integer

**Response Messages**

HTTP Status Code	Reason	Response Model	Headers
204	No Content		
401	Unauthorized		
403	Forbidden		

[Try it out!](#)

**email : javabyraghu@gmail.com**

FB: <https://www.facebook.com/groups/thejavatemple/>

# **SPRING BOOT END**

# MICROSERVICES

## 1. Monolithic Application:--

A Project which holds all modules together and converted as one Service (one .war file).

=>All most every project in realtime is implemented using this format only.

=>In this case, if no. of users are getting increased, then to handle multiple request (load) use LBS (Load Balancing Server).

=>But few modules needs Extra load, not all. In this case other modules get memory which is waste (no use). Hence reduces performance of server (application).

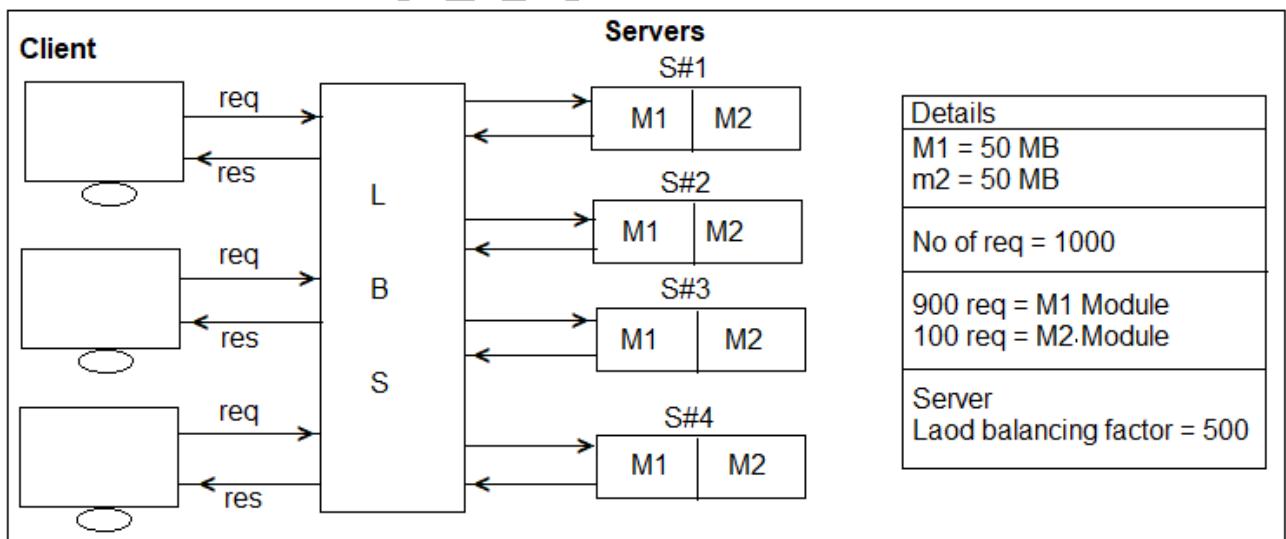
\*\*Consider Project P1 is having 2 modules M1(Register), M2(Login) and their runtime memories are M1=50 MB, M2=50 MB.

=>Here, End user Register only onetime (first time) and login is mostly used module.

->Consider Application needs 100MB size in server as register 50MB and Login 50MB.

->To handle multiple client request multiple instances of sample App must be provided using Load balancing server with Load Register.

**Diagram:--** Load Balancing is done using Servers looks like.



=>In above example, M2 Module is getting fewer (100) requests from Client. So, max 2 instances are may be enough. Other 2 instances (memories) are no use. It means near 100MB memory is not used, which impacts server performance.

**2. Microservices:**-- It is an independent deployment component.

=>It is a combination of one (or more) modules of a Project runs as one Service.

=>To avoid monolithic Limitations like memory and time (performance) problems use this design.

#### Nature of Microservices:

1>Every Service must be implemented using **Webservices** concept.

2>One or multiple module as one Project.

3>Every service must be **independent** (One service should not affect another one, like for code modifications, version upgrades, downtime for few servers... etc).

4>It should be able to communicate with any type of client (Mobile, Web based, 3<sup>rd</sup> party).

5>Every Services should be able to communicate with each other Microservice, It is also called as "**Intra Communication**".

6>Required Services must be supported for **Dynamic Load Balancing** (i.e. one service runs in multiple instances) based on click request.

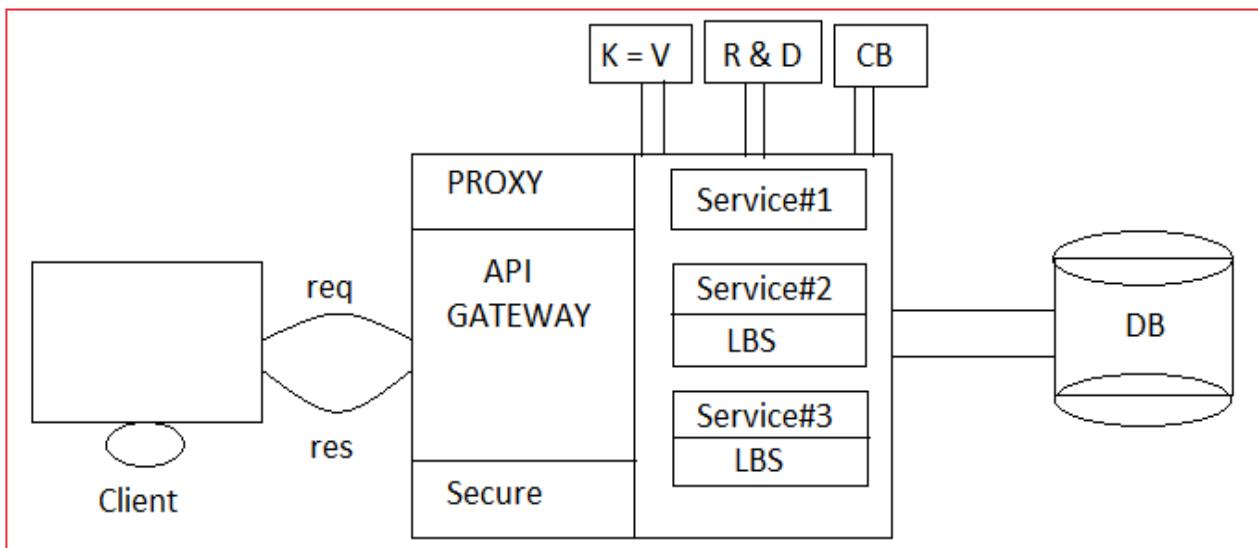
7>Every microservice should support being able to read input data from External

**Configuration Server** [Config Server] (Dynamic inputs using `_.properties/_.yml`), with GIT/Config Server).

8>Service communication (Chain of execution) problems should be able to solve using **Circuit Breaker** [Find other possible...].

9>All Servers must be accessed to Single Entry known as Gateway Service [ Proxy Gateway or API Gateway], It supports securing, **metering** and **Routing**.

#### Microservice Generic Architecture:



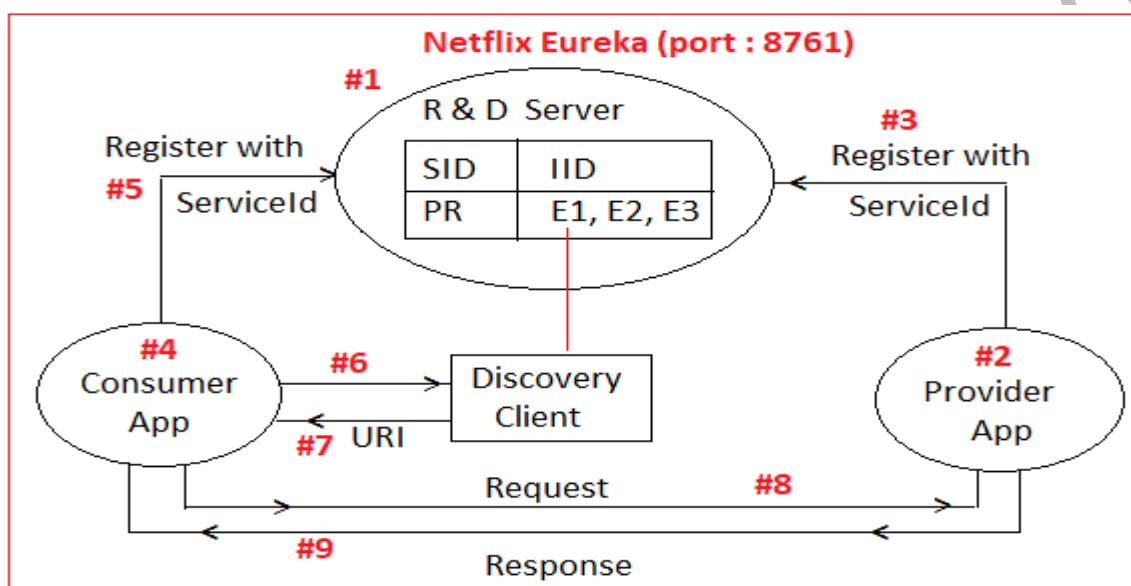
=>This design provides ability to communicate any kind of client application.

Ex:-- Android, Angular, RFID, Web, 3<sup>rd</sup> party, Swings,... etc.

=>Eureka is a R & D server which supports microservices register, discover with each other for intra communication.

=>Every Microservice should get registered with Eureka using one ServiceId (one or more instanceId).

=>For consumer application using Discovery client, get URI and make http Request to producer Application.



#### Component Names:--

1>Service Registry & Discovery	= Eureka
2>Load Balancing Server	= Ribbon
3>Circuite Breaker	= Hystrix
4>API Gateway	= Zuul
5>Config Server	= GitHub
6>Secure Server	= OAuth2
7>Log and Trace	= Zipkin + Sleuth
8>Message Queues	= Kafka
9>Declarative Rest Client	= Feign
10>Integration Service	= Camel
11>Production Ready Endpoint	= Actuator
12>Metrics UI	= Admin (Server/Client)
13>Cloud Platform	= PCF, Docker

With Deploy services

**SOA (Service Oriented Architecture):--**

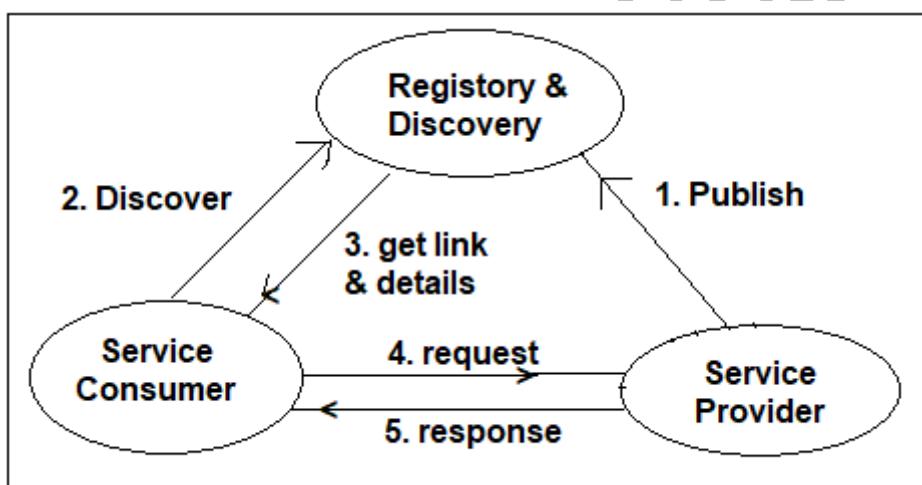
=>It is a Design Pattern used to create communication links between multiple services providers and Consumers (users).

**Components of SOA:--**

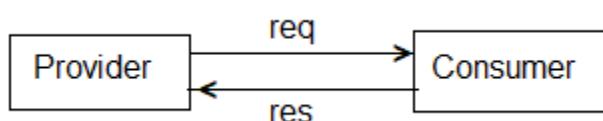
- a>Service Registry and Discovery [Eureka]
- b>Service Provider [Webservice Provider]
- c>Service Consumer [Webservice Client]

**Operations:--**

- 1>Publish
- 2>Discover
- 3>Link Details of Provider
- 4>Query Description (Make Http Request).
- 5>Access Service (Http Response).

**Implementing MicroService Application Using Spring Cloud:--**

Design #1:-- A Simple Rest WebService using Spring Boot.

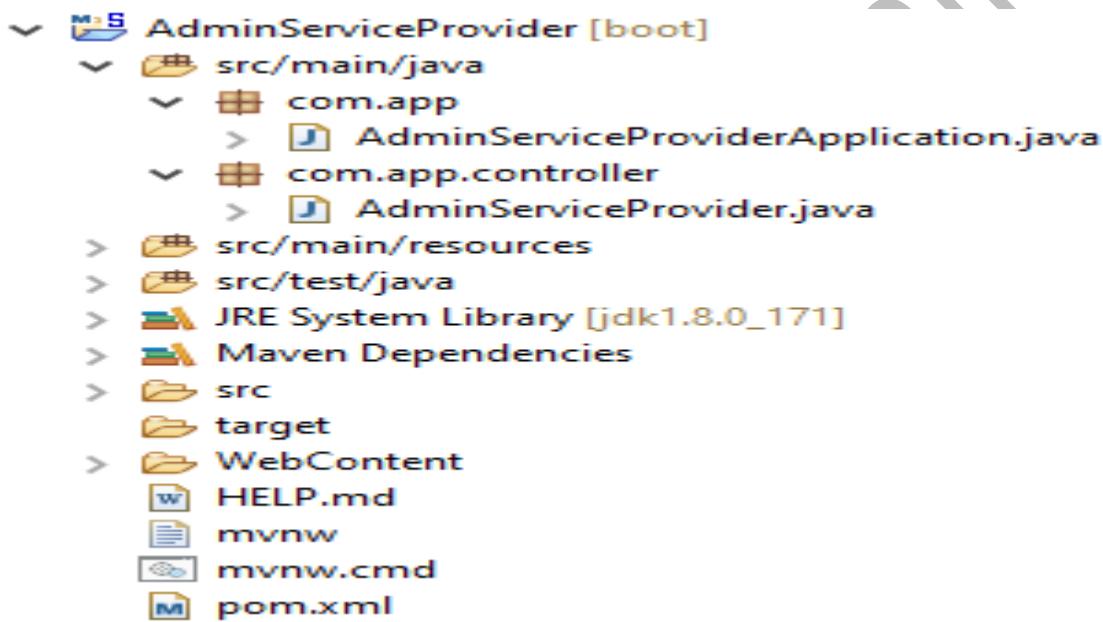


=>This application is implemented using Spring Boot Restfull webservices which provides Tightly coupled design, It means any changes in provider application effects consumer application, specially server changes port no changes, context changes etc.  
=>This design will not support load balancing.  
=>It is implemented using RestController and RestTemplate.

### Step#1:- Create Provider Application (Dependencies : web only)

Group Id : org.verizon  
ArtifactId : AdminServiceProvider  
Version : 1.0

#### #1. Folder Structure of AdminServiceProvider:--



#### Step #2:- Define one RestController (AdminServiceProvider.java):--

```

package com.app.controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/provider")
public class AdminServiceProvider {
    @GetMapping("/show")
    public String showMsg() {
        return "Hello from AdminServiceProvider";
    }
}
  
```

```

        return "Hello";
    }
}

```

### Step #3:- Create Consumer Application (Dependencies : web only)

GroupId : org.verizon  
 ArtifactId : AdminServiceConsumer  
 Version : 1.0

### #2. Folder Structure of AdminServiceConsumer:--



### Step #4:- Define Consumer (call) code (AdminConsumer.java):--

```

package com.app.consumer;
import org.springframework.boot.CommandLineRunner;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Component;
import org.springframework.web.client.RestTemplate;

@Component
public class AdminConsumer implements CommandLineRunner {

    public void run (String... args) throws Exception {
        RestTemplate rt = new RestTemplate();
        ResponseEntity<String> resp =
rt.getEntity("http://localhost:2019/provider/show", String.class);
    }
}

```

```
        System.out.println(resp.getBody());
        System.exit(0);
    }
}
```

## **Execution flow SCREEN :-**

=>First Run provider (Starter), then consumer and enter below url

1><http://localhost:2019/provider/show>

## **#1 PROVIDER SCREEN:--**

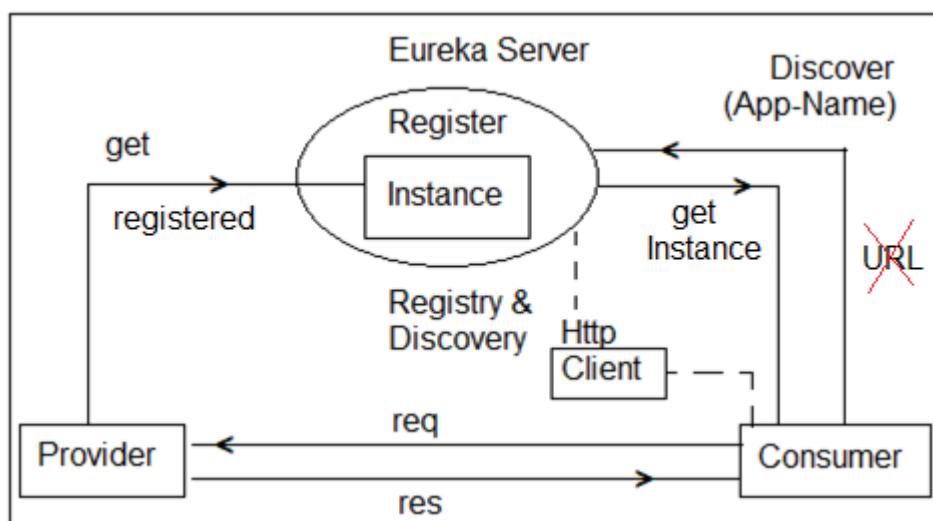
## **CONSUMER SCREEN:--**

### 3. MicroService Design and Implementation using Spring Cloud

#### (Netflix Eureka Registry & Discovery):--

- => Registry and Discovery server hold details of all Client (Consumer/Producer) with its serviced and Instance Id.
- => Netflix Eureka is one R & D Server.
- => Use default port no is 8761.

#### Design#1:- [Basic – No Load Balancing]



**Step #1: Create Eureka Server:**-- Create one Spring Boot Starter Project with Dependencies : Eureka Server

#### Eureka Server Dependencies:--

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-server </artifactId>
</dependency>
```

GroupId : org.verizon

ArtifactId : EurekaServerApp

Version : 1.0

### #3. Folder Structure of Eureka Server:--



#### pom.xml of Eureka Server:--

```

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.1.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.app</groupId>
  <artifactId>EurekaServerApp</artifactId>
  <version>1.0</version>
  <name>EurekaServerApp</name>
  <description>Demo project for Eureka Server</description>
  <properties>
    <java.version>1.8</java.version>
    <spring-cloud.version>Greenwich.SR1</spring-cloud.version>
  </properties>
  <dependencies>
    <dependency>
  
```

```
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
</dependencies>

<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>${spring-cloud.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
</project>
```

**Step#2:-** At Starter class level add Annotation `@EnableEurekaServer` Annotation:--

```
package com.app;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;
```

```
@SpringBootApplication
@EnableEurekaServer
public class EurekaServerApp {
    public static void main(String[] args) {
        SpringApplication.run(EurekaServerApp.class, args);
        System.out.println("Eureka Server Executed:");
    }
}
```

**Step #3:-** In application.properties add keys

```
server.port=8761
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
```

**Step #4:-** Run starter class and Enter URL <http://localhost:8761> in browser

**NOTE:--** Default port no of Eureka Server is 8761.

### Screen Short of Eureka Server Dashboard:--

The screenshot shows the Spring Eureka Server dashboard at <http://localhost:8761>. The top navigation bar includes links for Gmail, YouTube, Online Courses, Online Tests, Tutorials, Youth4work, Testpot.com, Facebook, and Hello Python. The dashboard has a dark header with the Spring Eureka logo and the text "HOME LAST 1000 SINCE STARTUP".

**System Status:**

Environment	test	Current time	2019-04-14T00:29:22+0530
Data center	default	Uptime	00:02
		Lease expiration enabled	false
		Renew threshold	1
		Renews (last min)	0

**DS Replicas:**

localhost
Instances currently registered with Eureka

**General Info:**

Name	Value
total-avail-memory	299mb
environment	test
num-of-cpus	4

#### 4. Spring Boot Provider (Producer) Application:-

=>Every client service should get registered with Eureka and implement load balancing in case of multiple request processing.

**Step #1:-** Create one Spring starter App with web and **Eureka Discovery Client** dependencies

**Eureka Discovery Client Dependencies:-**

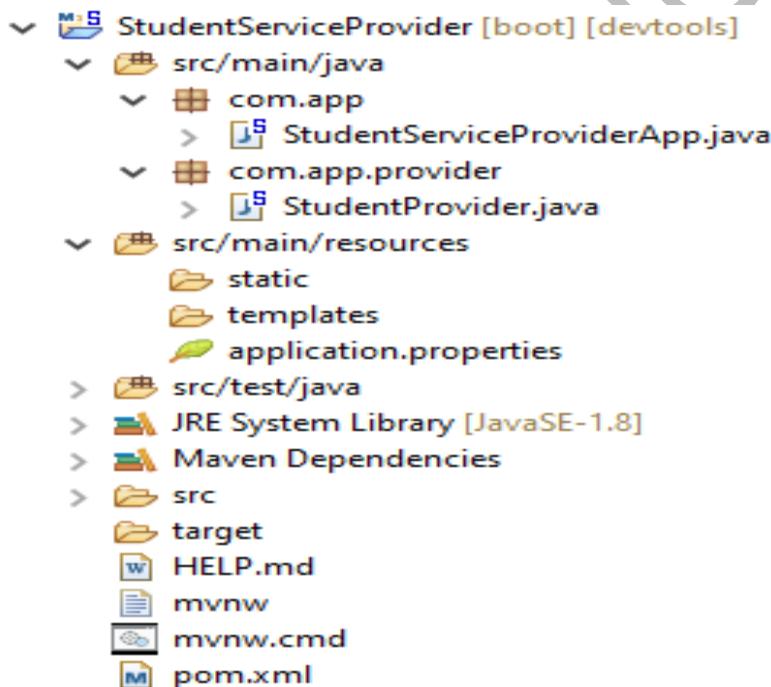
```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client </artifactId>
</dependency>
```

GroupId : org.app

ArtifactId : StudentServiceProvider

Version : 1.0

#### #4. Folder Structure of Provider Application:-



**pom.xml:-**

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0  
http://maven.apache.org/xsd/maven-4.0.0.xsd">  
    <modelVersion>4.0.0</modelVersion>  
    <parent>  
        <groupId>org.springframework.boot</groupId>  
        <artifactId>spring-boot-starter-parent</artifactId>  
        <version>2.1.1.RELEASE</version>  
        <relativePath/> <!-- lookup parent from repository -->  
    </parent>  
    <groupId>com.app</groupId>  
    <artifactId>StudentServiceProvider</artifactId>  
    <version>1.0</version>  
    <name>StudentServiceProvider</name>  
    <description>Demo project for Microservice Studenet Provider</description>  
    <properties>  
        <java.version>1.8</java.version>  
        <spring-cloud.version>Greenwich.SR1</spring-cloud.version>  
    </properties>  
    <dependencies>  
        <dependency>  
            <groupId>org.springframework.boot</groupId>  
            <artifactId>spring-boot-starter-web</artifactId>  
        </dependency>  
        <dependency>  
            <groupId>org.springframework.cloud</groupId>  
            <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>  
        </dependency>  
        <dependency>  
            <groupId>org.springframework.boot</groupId>  
            <artifactId>spring-boot-devtools</artifactId>  
            <scope>runtime</scope>  
        </dependency>  
        <dependency>  
            <groupId>org.springframework.boot</groupId>  
            <artifactId>spring-boot-starter-test</artifactId>  
            <scope>test</scope>  
        </dependency>
```

```

        </dependency>
    </dependencies>

    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-dependencies</artifactId>
                <version>${spring-cloud.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>

```

**Step #2:-** Add below annotation at Starter class level **@EnableEurekaClient** (given by Netflix) or **@EnableDiscoveryClient** (given by Spring Cloud) both are optional.

#### Spring Starter class (**StudentServiceProviderApp.java**)--

```

package com.app;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;

@SpringBootApplication
@EnableEurekaClient
public class StudentServiceProviderApp {

```

```
public static void main(String[] args) {  
    SpringApplication.run(StudentServiceProviderApp.class, args);  
    System.out.println("Student Service provider");  
}  
}
```

**Step #3:- In application.properties file**

```
server.port=9800  
spring.application.name=STUDENT-PROVIDER  
eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka
```

**Step #4:- Define one Provider Controller (StudentProvider) :-**

```
package com.app.provider;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
@RequestMapping("/provider")  
public class StudentProvider {  
  
    @GetMapping("show")  
    public String showMsg() {  
        return "Hello from Provider";  
    }  
}
```

**Execution Order:- (RUN Starter Classes)**

1. EUREKA SEVER
  2. PROVIDER APPLICATION
- =>Goto eureka dashboard and check for application  
=>Click on url and add /provider/show PATH

## #1 Screen Short of Eureka Server Dashboard:--

The screenshot shows the Spring Eureka Server Dashboard. At the top, it displays the Eureka logo and navigation links for HOME and LAST 1000 SINCE STARTUP. Below this, the 'System Status' section provides configuration details:

Environment	test	Current time	2019-04-14T00:43:08 +0530
Data center	default	Uptime	00:16
		Lease expiration enabled	true
		Renews threshold	3
		Renews (last min)	4

The 'DS Replicas' section shows a single entry for 'localhost'. The 'Instances currently registered with Eureka' section lists one instance:

Application	AMIs	Availability Zones	Status
STUDENT-PROVIDER	n/a (1)	(1)	UP (1) - 192.168.100.39:STUDENT-PROVIDER:9800

**NOTE:--** Click on URI (Instance) (192.168.100.39:STUDENT-PROVIDER:9800) then add provider path after URI (<http://192.168.100.39:9800/provider/show>)

## #2 Screen Short of Provider Application:--

The screenshot shows a web browser window with the URL [192.168.100.39:9800/provider/show](http://192.168.100.39:9800/provider/show). The page displays the message "Hello from Provider".

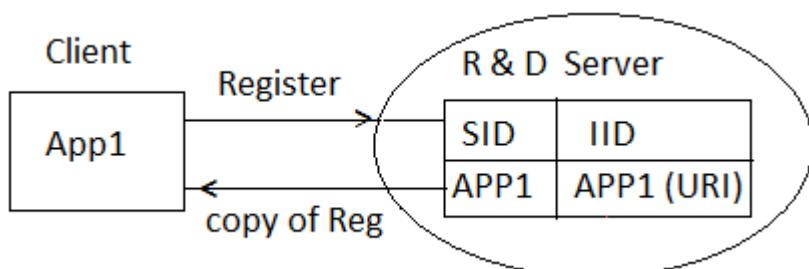
## Work flow of Eureka and Producer:--

Step#1:- On startup Eureka server, one Registry is created (it is like a Map).

Step#2:- Every client is identified using serviced and their servers are identified using Instance Id (URI).

Step#3:- Client identifies R & D using Service-url and it is enabled by using @EnableEurekaClient.

Step#4:- Client gets registered first and then fetches the registry (copy).



**5. Consumer Application:**-- In general Spring Boot application, by using any HTTP Client code, Consumer makes request based on URL (static/hard coded) given in code.  
 =>This application need to be registered first with R & D server then fetch service registry using any one client component.given as  
 a>Discovery Client  
 b>Load Balancing Client (Feign client)

=>Every client component need “Service-Instance” object to communicate with producer application.

=>In general, one instance means One-service where application is running.

=>Every instance must have:

1. Service Id : Project Name
2. Instance Id : Server Instance name)
3. URI : Host, Port, ...etc)
4. MetaData : (Headers, Cookies, Sessions...)
5. Current Load Factor... etc.

**\*\*\* Hard coding:**-- Providing a direct value to a variable in .java file or fixed value.

Ex:- int a = 5;

=>It provides always same output for multiple runs.

=>By using RestTemplate with URL (hard coded) we can make request. But it will not support.

a>If provider IP/PORT number gets changed.

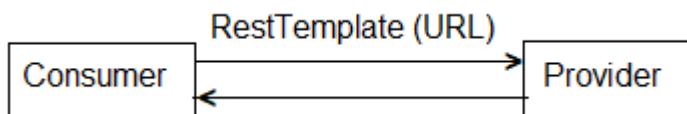
b>Load Balancing Implementation.

=>So, we should use Dynamic Client that gets URL at runtime based on Application name registered in “Registry and Discovery Server (Eureka)”.

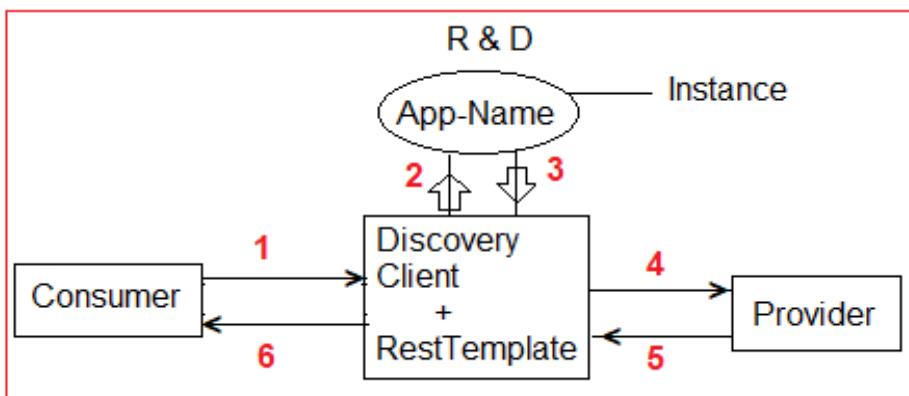
=>DiscoveryClient is used to fetch Instance based on Application Name and we can read URI of provider at runtime.

=>RestTemplate uses URI (+path)= URL and makes Request to Provider and gets ResponseEntity which is given back to the Consumer.

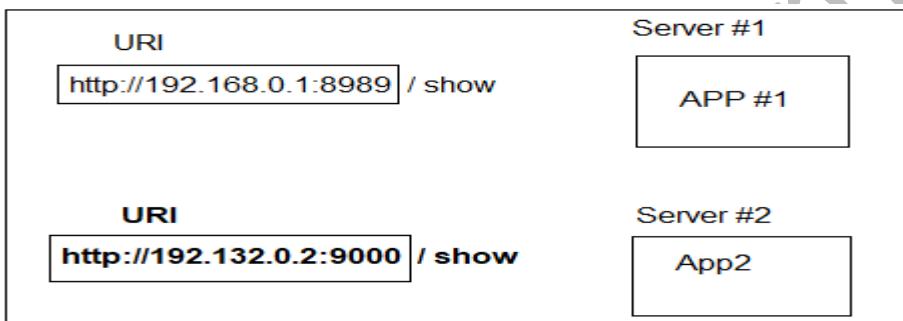
### Simple Web Service Example:-



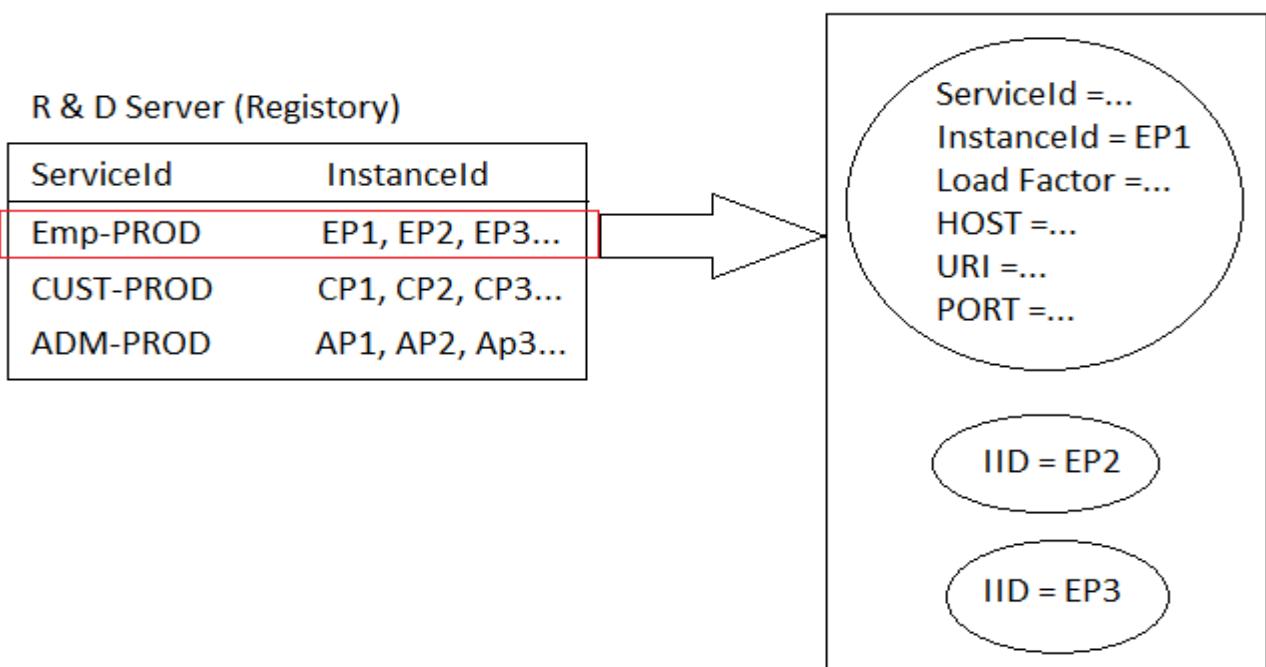
### Microservices Example:--



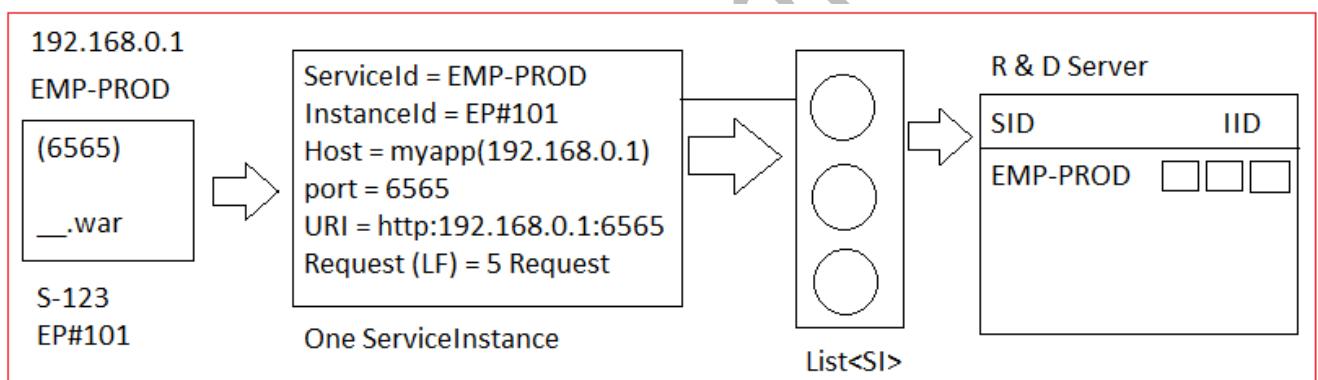
=>If one Application is moved from one Server to another server then URI gets changed (Paths remain same).



=>If one Project is running in 3 Server then current Project is having 3 instances. All these are store in R & D Server and we can fetch details as : List<ServiceInstance>



=>Consider one example producer application running in server 192.168.0.1 (IP).



### Consumer code:-

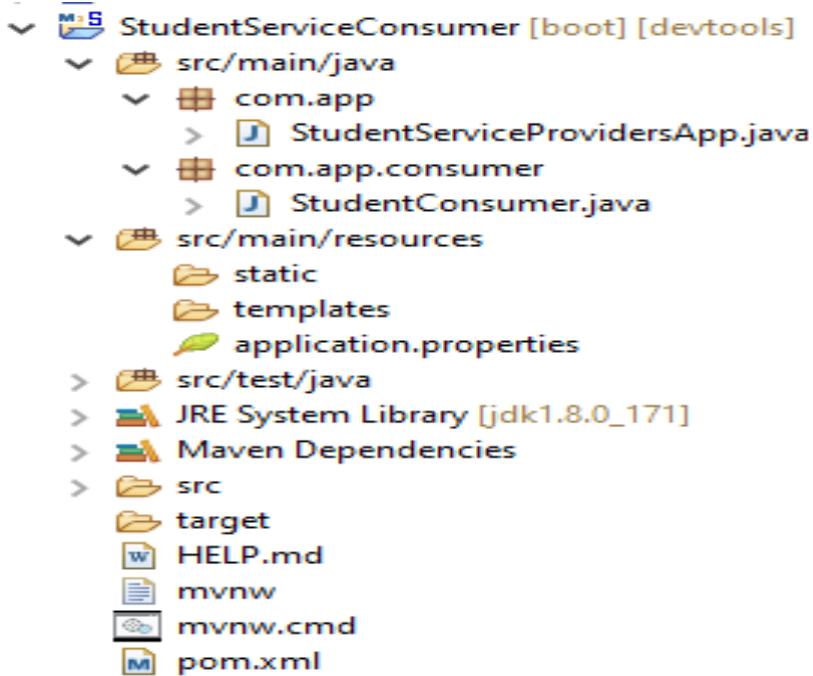
**Step #1:-** Create one Spring Starter Project using Dependencies web, Eureka Discovery.

GroupId : org.app

ArtifactId : StudentServiceConsumer

Version : 1.0

### #5. Folder Structure of Consumer Application:-



**Step #2:-** At Starter class level add Annotation either **@EnableEurekaClient** or **@EnableDiscoveryClient** (both are optional)

**Step #3:-** In application.properties file

server.port=9852

spring.application.name=STUDENT-CONSUMER

eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka

**Step #4:- Define Consumer code with RestTemplate and DiscoveryClient:--**

**StudentConsumer.java:--**

```
package com.app.consumer;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.cloud.client.ServiceInstance;
import org.springframework.cloud.client.discovery.DiscoveryClient;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.client.RestTemplate;
```

@RestController

**public class** StudentConsumer

{

@Autowired

**private** DiscoveryClient client; //Where DiscoveryClient is a n Instance

@GetMapping("/consume")

**public** String consumeData ()

{

    RestTemplate rt = **new** RestTemplate();

List&lt;ServiceInstance&gt; list=client.getInstances("STUDENT-PROVIDER");

ResponseEntity&lt;String&gt; resp =rt.getEntity(list.get(0).getUri()+"show", String.class);

**return** "FROM CONSUMER=>" +resp.getBody();

}

}

**Execution order:--**

#1 Run Starter classes in order:--

Eureka server, provider App, Consumer App

#2 Goto Eureka Server Dashboard and click on Client Consumer URI and enter "/consume" path after PORT number (<http://192.168.100.39:9852/consume>).**Screen#1 Eureka Server:--**

The screenshot shows the Spring Eureka Server dashboard at [localhost:8761](http://localhost:8761). The top navigation bar includes links for Gmail, YouTube, Online Courses, Online Tests, Tutorials, Youth4work Assessments, Testpot.com, Facebook, and Hello Python! Python.

**System Status:**

Environment	test	Current time	2019-05-12T14:31:47 +0530
Data center	default	Uptime	00:14
		Lease expiration enabled	true
		Renew threshold	5
		Renews (last min)	8

**DS Replicas:**

localhost
-----------

**Instances currently registered with Eureka:**

Application	AMIs	Availability Zones	Status
STUDENT-CONSUMER	n/a (1)	(1)	UP (1) - localhost:STUDENT-CONSUMER:9859
STUDENT-PROVIDER	n/a (1)	(1)	UP (1) - 192.168.100.11:STUDENT-PROVIDER:9800

**6. Load Balancing in Spring Cloud:--**

=>To handle multiple requests made by any HTTP client (or consumer) in less time, one provider should run as multiple instances and handle request in parallel. Such concept is called as Load Balancing.

=>Spring cloud has provided one Interface “Load Balance Client” which is used to define LBS (Load balancing server) Register.

=>This register holds Load factor over Instances and stores in a map format based on ServiceId. Load factor also called as current load over level over any instance.

=>Load balancing is to make request handling faster (reduce waiting time in queue).

### **Step to implement Load Balancing:--**

a>Create one provider application.

b>Register and provider as multiple instances in Registry & Discovery [Eureka] server every instance with unique ID.

Ex:-- P1-58266, P2-2353424, P3-740986 etc.

c>Define consumer with any one Load Balancing Component (Ex:-- Ribbon, Feign).

d>Ribbon chooses one Provider URL, based on instance Id with the help of LBS register which maintains request count.

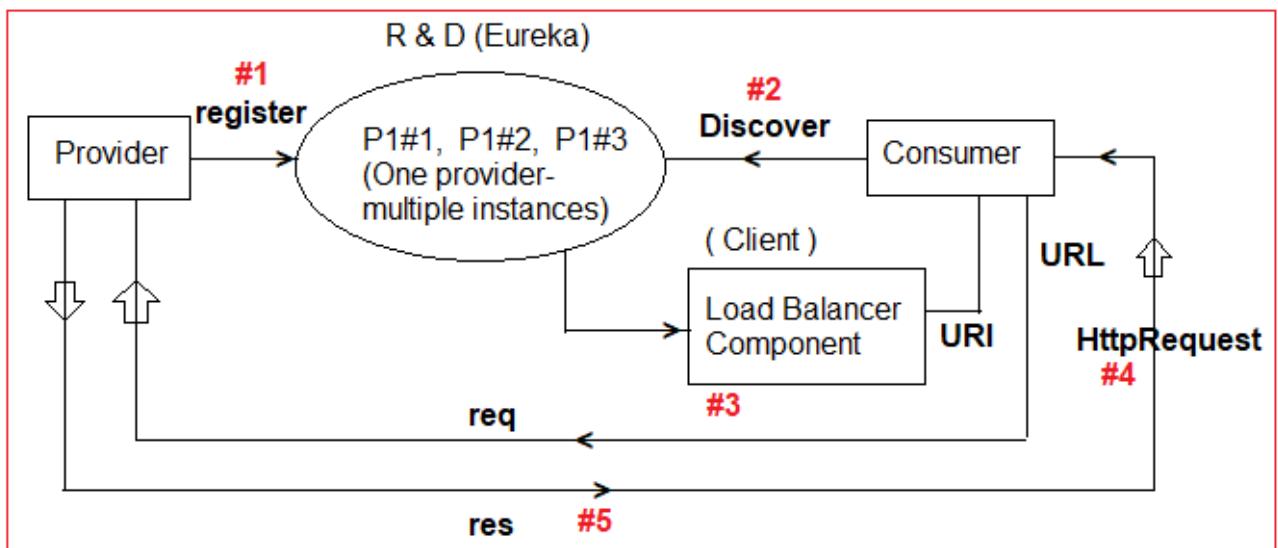
e>Consumer will uses paths to URL and makes Request using “RequestClient”.

[ Ex:- LoadBalancerClient (I) or @FeignClient]

f>ResponseEntity is returned by Provider to Consumer.

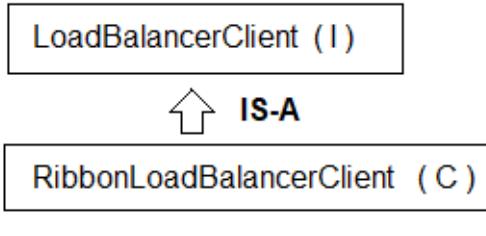
**5.1 Ribbon:--** It is a Netflix component provided for spring boot cloud Load Balancing.

=>It is also called as Client side load Balancing. It means Consumer App, reads URI (which one is free) using LBS register.



Load Balancer Component : Ribbon, Feign

Request Component :



Temp memory by Ribbon

Load Balancer Server

Registry

Instance Id	Req Count
P1#1	11
P1#2	10
P1#3	10

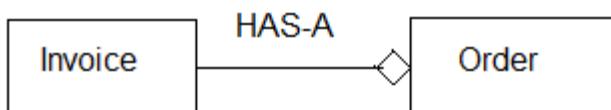
=>Ribbon should be enabled by every Consumer.

=>Spring cloud uses **LoadBalancerClient (I)** for choose an invoice process.

=>Its implementation is provided by Ribbon

=>Netflix has provided one Impl class for above Interface (**LoadBalancerClient**). It supports creating, updating, choosing, ... etc of a LBS Register and Instances details.

**Consider below Example for Ribbon:-**

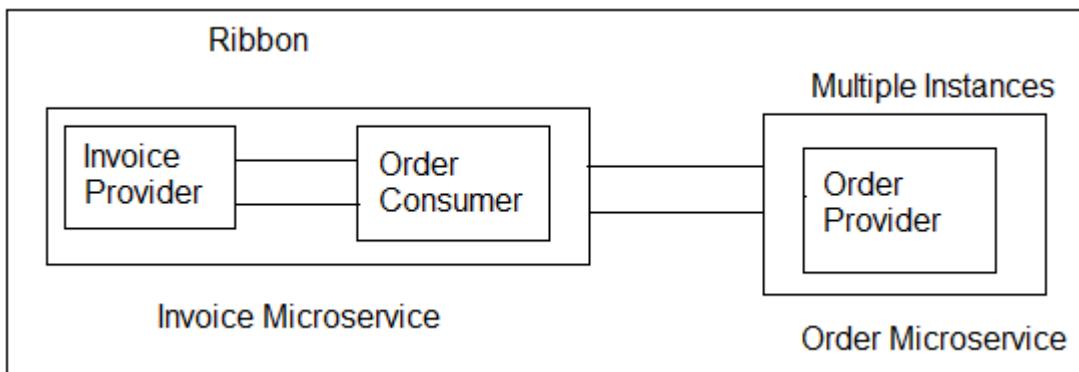


=>Then two projects are created here.Order Application and Invoice Application.

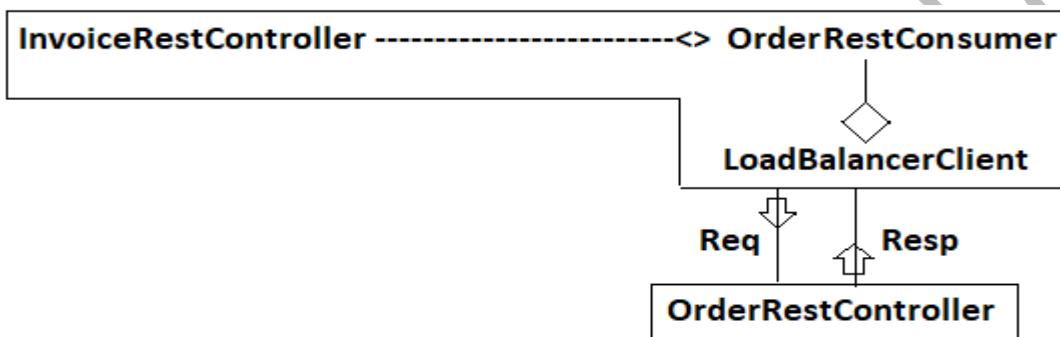
=>Both apps contain their Service Producer codes (RestController).

=>This time child Producer will become ChildConsumer in ParentProducer application.

=>ChildConsumer and childProducer codes communicate using HTTP protocols.



**NOTE:**-- Ribbon at Consumer side



=>Consumer Application (Invoice Producer) needs LoadBalancerClient which is provided by Netflix-Ribbon. Below dependency must be provided in pom.xml

```

<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-ribbon</artifactId>
</dependency>
    
```

=>By default application behaves as ServiceId and InstanceId, but this time to provide multiple InstanceIds use below key in properties file.

```
eureka.instance.instance-id=
```

#### Example: application.properties:--

```
spring.application.name=ORDER-PRO
```

```
eureka.instance.instance-id=${spring.application.name}: ${random.value}
```

#### Coding Steps:--

**Step#1:-** In provider, define Instance Id for Provider Application using key.

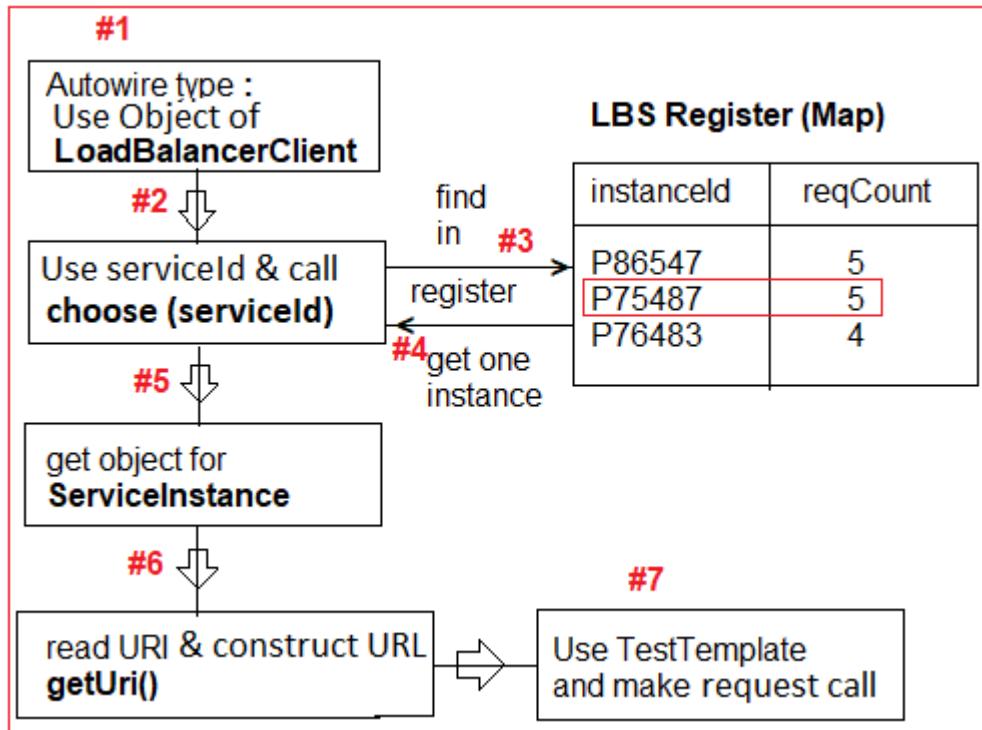
**“eureka.instance.intance-id”.** If not provided default is taken as Spring App name.

```
eureka.instance.instance-id=${spring.application.name}: ${random.value}
```

[Add in application.properties]

**Step#2:-** In consumer, add dependency for Ribbon and use LoadBalancerClient (Autowired) and call choose ("App-Name") method to get serviceInstance (for URI)

**Consumer Execution flow for Request:-**



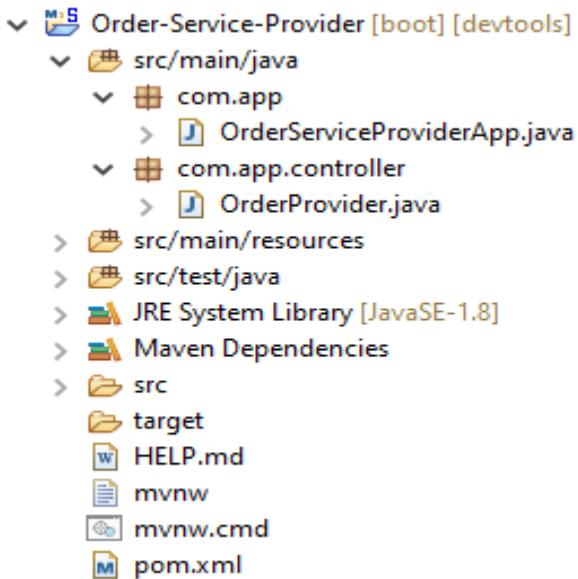
### APPLICATION #1:- Configure Eureka Server Dependencies : Eureka Server only

**application.properties:-**

```
server.port=8761
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
```

### APPLICATION#2:- Create Order Service Provider Application(Child) Dependencies : Eureka Discovery, Web.

### #6. Order Service Provider:-

**application.properties:--**

```
server.port=9800
spring.application.name=ORDER-PROVIDER
eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka
eureka.instance.instance-id=${spring.application.name}:${random.value}
```

=>If no instance-id is provided then application name (service Id) behaves as instance Id.

**#1. Starter class:--**

```
package com.app;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

@SpringBootApplication
@EnableDiscoveryClient
public class OrderServiceProviderApp
{
    public static void main(String[] args)
    {
        SpringApplication.run(OrderServiceProviderApp.class, args);
    }
}
```

**#2. Controller:--**

```

package com.app.controller;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("order")
public class OrderProvider {

    @Value("${server.port}")
    private String port;

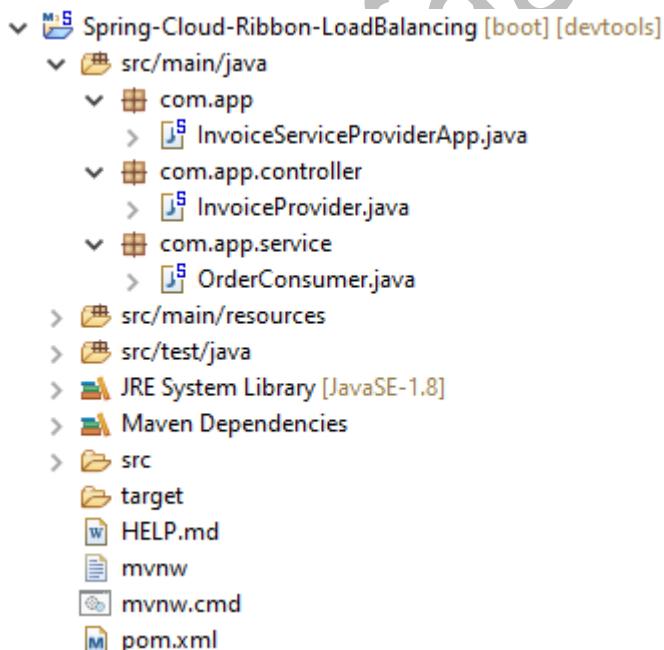
    @GetMapping("/status")
    public String getOrderStatus() {
        return "FINISHED:\\"Hello from Order Provider\":"+port;
    }
}

```

### **APPLICATION#3:- Define Invoice Service Consumer(Parent):--**

Dependencies : Eureka Discovery, Web, Ribbon

### **#7. Invoice Service Consumer:--**



### **Ribbon Dependency:--**

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-ribbon</artifactId>
</dependency>
```

**application.properties:--**

```
server.port=8802
spring.application.name=INVOICE-CONSUMER
eureka.client.serviceUrl.defaultZone =http://localhost:8761/eureka
```

**#1. Consumer starter class:--****#2. Consumer code(OrderConsumer.java):--**

```
package com.app.service;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.cloud.client.ServiceInstance;
import org.springframework.cloud.client.loadbalancer.LoadBalancerClient;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestTemplate;
```

```
@Service
```

```
public class OrderConsumer {
    @Autowired
    private LoadBalancerClient client;
    public String getStatus() {
        String path="/order/status";
        //Choose Service instance based on SID
        ServiceInstance instance=client.choose("ORDER-PROVIDER");
        //Read URI from instance
        String uri=instance.getUri().toString();
        //Make http Request
        RestTemplate rt = new RestTemplate();
        ResponseEntity <String> resp=rt.getForEntity(uri+path, String.class);
        return "CONSUMER=>"+resp.getBody();
    }
}
```

**#2 Controller code (InvoiceProvider):--**

```
package com.app.controller;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
import com.app.service.OrderConsumer;  
  
@RestController  
@RequestMapping("/invoice")  
public class InvoiceProvider {  
  
    @Autowired  
    private OrderConsumer consumer;  
  
    @GetMapping("/info")  
    public String getOrderStatus() {  
        return consumer.getStatus();  
    }  
}
```

**Execution:--**

- a>Start Eureka Server
- b>Run OrderProvider starter class 3 times
- \*\*\* Change every time Port number like : 9800,9801, 9802
- c>Run InvoiceProvider Starter class
- d>Goto Eureka server Dashboard and Execute Invoice Consumer Instance and type full URL <http://localhost:8800/invoice/info>

**OutPut:--**

## 5.2 Declarative RestClient : [Feign Client]:--

Spring cloud supports any HTTP Client to make communication between (Microservices) Provider and Consumer.

=>RestTemplate is a legacy style which is used to make HTTP calls with URL and Extra inputs. Feign Client reduces burden on programmer by avoiding legacy style coding for RestTemplate.

=>RestTemplate with DiscoveryClient makes mask to Provider URL. It means works based on Application Name (Service ID). Even URI gets changed it works without any modification at consumer side.

=>RestTemplate combination always makes Programmer to write manual coding for HTTP calls.

=>Spring Cloud has provided one Action Client [which behaves as Client, but not]. It means, Provide Abstraction at code level by programmer and Implementation is done at runtime by Spring cloud.

=>It is Http client used to create communication links between Consumer and Producer application.

- >It is provided by Netflix.

- >It works based on Proxy design Pattern.

- >At runtime Http calls logic is implemented.

- >Supports Load balancing Also.

- >Internally uses Ribbon only.

- >It is called as Declarative Client.

=>By taking above Inputs Http calls are implemented using one Instance (URI).

=>Feign is **Declarative Client**, which supports generating code at runtime and **proxy Objects** by using **Proxy HTTP Request** calls can be made.

=>It supports even Parameters (Path/Query...) and Global Data Conversion (XML/JSON).

### Example Feign client looks like:--

```
@FeignClient("TRACT-PROD")
public interface TrackConsumer {
    @GetMapping("/tract/info")
    public String getModel();
    @PostMapping("track/insert")
    public String create(@RequestBody Track tr);
}
```

=>It is similar to rest controller but it is **interface** with **abstract** methods.

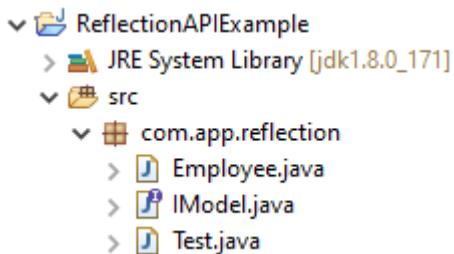
**Proxy**-- It is a code/Object generated at runtime which acts as actual one, but not exist as physical file.

- >Proxy Type (Proxy Class)
- >Proxy Data (Proxy Objects)

=>Proxies are designed using **CGLibs and Reflection**.

=>Javapoet is a child type of CGLibs(Code Generator Library).

## 1. Reflection API Example(Normal Java Application):--



### 1. IModel interface:--

```

package com.app.reflation;

public interface IModel {
    public String getModelName();
    public IModel getModelObject();
}
  
```

### 2. Employee Class:--

```

package com.app.reflation;

public class Employee implements IModel {

    public Employee() {
        System.out.println("Employee Constructor...");
    }
    @Override
    public String getModelName() {
        return "Employee";
    }
    @Override
    public IModel getModelObject() {
        return this;
    }
}
  
```

```

    public String toString(){
        return "FROM EMPLOYEE OBJECT";
    }
}

```

**3. Test class:--**

```
package com.app.reflation;
```

```
public class Test {
```

```

    public static void main(String[] args) throws Exception {
        //load runtime class
        Class c = Class.forName(args[10]);
        //Create instance
        Object ob = c.newInstance();

        //Downcast to IModel Type
        if(ob instanceof IModel){
            IModel m =(IModel)ob;
            System.out.println(m.getModelName());
            System.out.println(m.getModelObject());
        }
    }
}
```

=>Right click > Run As > Run Configuration

->Click on Arguments > Progarmming args

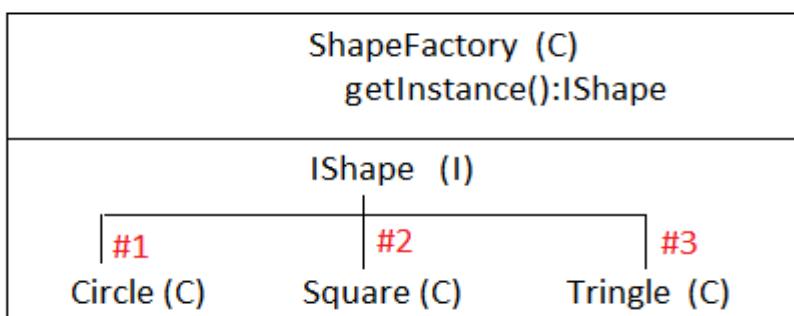
->Enter Ex: com.app.Employee >Apply & Run

**Static Factory Design Pattern using Reflection API and JDK 1.8 Interface features:--**

->Input (int)

1. Circle
2. Square
3. Triangle

( int )  
choice →

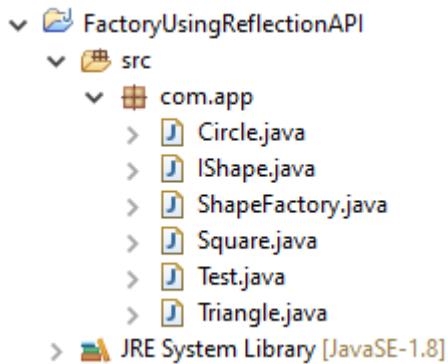


**Output:**-- Object of class and calling methods.

=>Factory Components

1. IShape (I)
2. ShapeFactory (C)

## 2. Folder Structure of Factory Design Pattern Using Reflection API:--



**Code:**--

### 1. IShape Interface:--

```
package com.app;
```

```
public interface IShape {
```

```
  //must be overridden
```

```
  public void showInfo();
```

```
  //can be overridden
```

```
  public default void shapeMsg() {
```

```
    System.out.println("Welcome to ShapeFactory");
```

```
}
```

```
  //can not be overridden
```

```
  public static void commonMsg() {
```

```
    System.out.println("Hello Shape.. ");
```

```
}
```

### 2. Circle Class:--

```
package com.app;
```

```
public class Circle implements IShape {
```

```
  public Circle() {
```

```
    System.out.println("circle is created.. ");
```

```
}
```

```

public void showInfo() {
    System.out.println("This is a circle Object");
}
@Override
public void shapeMsg() {
    System.out.println("Message from circle");
}
}

```

**3. Square Class:--**

```
package com.app;
```

```

public class Square implements IShape {

    public Square() {
        System.out.println("Square object is created..");
    }
    public void showInfo() {
        System.out.println("This is Square Shape ");
    }
    @Override
    public void shapeMsg() {
        IShape.super.shapeMsg();
        System.out.println("Also Square message Printed..");
    }
}

```

**4. Triangle Class:--**

```
package com.app;
```

```

public class Triangle implements IShape {

    public Triangle() {
        System.out.println("triangle object is created");
    }
    public void showInfo() {
        System.out.println("THis is triangle object");
    }
}

```

**5. ShapeFactory Class:--**

```
package com.app;
```

```
public class ShapeFactory {
```

```

public static IShape getShape(int ch){
    String cls=choose(ch);
    IShape shpae=null;
    try {
        Object ob=Class.forName(cls).newInstance();
        if(ob instanceof IShape) {
            shpae=(IShape)ob;
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return shpae;
}

```

```
private static String choose(int ch) {
```

```

    String cls=null;
    switch (ch) {
        case 1:
            cls="com.app.Circle";
            break;
        case 2:
            cls="com.app.Square";
            break;
        case 3:
            cls="com.app.Triangle";
            break;
        default:
            break;
    }
    return cls;
}

```

#### 6. Test Class:-

```

package com.app;

import java.util.Scanner;

public class Test {

    public static void main(String[] args) {

```

```

Scanner sc=new Scanner(System.in);
System.out.println("Please Choose One option");
System.out.println("1.Circle");
System.out.println("2.Square");
System.out.println("3.Triangle");
int ch=sc.nextInt();
if(ch>0 && ch<4) {
    IShape shape=ShapeFactory.getShape(ch);
    shape.showInfo();
    shape.shapeMsg();
    IShape.commonMsg();
} else {
    System.out.println("invalid selection");
}
sc.close();
}
}

```

**Execution process:--**

1>Run Test class and Enter number 1-3 and see the Output on Console.

**2. Export Project as Executable:--**

- =>Right Click on Project > Export
- =>Search Using “JAR” word
- =>Choose “Runnable JAR File”
- =>Select Main class to launch
- =>Browse for Location and Enter Jar name ex:- d:/myapps/factory.jar
- =>Finish

**Execute Jar:--**

- =>Goto Location where JAR is created
- =>Shift + Right Click
- =>Open cmd Window Here
- =>type below command  
‘java -jar factory.jar’

**Create a windows batch file:--**

- =>Open notepad =>Type below commands  
java -jar factory.jar
- =>and save with .bat extension

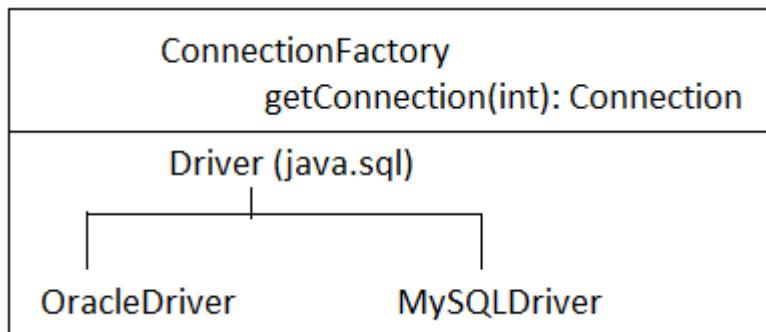
**Myapp.bat:--**

java -jar factory.jar

pause;

=>Double click on batch and enter inputs like (1, 2, 3)

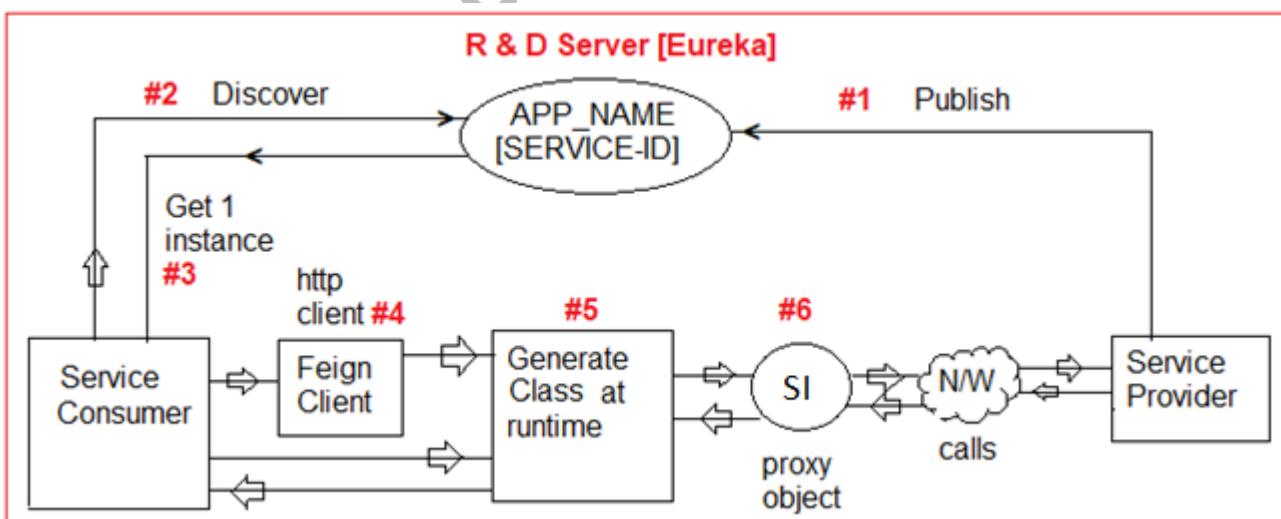
**Task:--**



**Work Flow Design:--**

1. Interface defined with **@FeignClient** validates all details (ServiceId, Paths, MethodTypes...).
2. Uses ServiceId and communicates with R&D it gets LBSR (Load Balancing).
3. A Proxy class is generated which makes all this process. Based on load Factor one Service Instance is chosen.  
\*\*\*Ribbon is used internally.
4. Http Logic is generated using Service Instance (URI).
5. Proxy class supports making Http Request and Response (Using dynamic SI).

**Work Flow Design:--**



=>Feign Client is an Interface and contains abstraction details, it takes I/P details from programmer like:

a>path (Provider Paths at class and method).

b>Http Method Type (GET, POST...).

c>ServiceId (Application Name).  
 d>Input Details and Output Type (String, Object, Collection).  
 e>Parameters (Request, Path,...).  
 =>We need to apply Annotation at starter class level **@EnableFeignClients**.  
 =>At interface level apply **@FeignClient(name="servieId")**.

### Syntax:-- Feign Client

```
@FeignClient(name="serviceld")
public interface <ClientName> {

  @GetMapping("/path")
  //or @RequestMapping("/path")
  public <Return> <method>(<Params>);

  ...
}
```

### Example:--

#### Provider Code (SID : EMP-PROV):--

```
com.app.rest;
```

```
@RestController
@RequestMapping("/emp")
public class EmpProvider {

  @GetMapping("/show")
  public String findMsg() {
    .....
  }
}
```

#### Consumer Code : Feign Client

```
@FeignClient(name="EMP_PROV")
public interface EmpConsumer {

  @GetMapping ("/emp/show")
  public String getMsg(); //return type and path must be same as Provider
}
```

## 1. Consider last Ex Eureka Server and Provider Application (ORDER-PROVIDER):--

**Step#1:-** Create one Spring Boot Starter Project for Consumer (using Feign, web, Eureka Discovery)

### Feign Client Dependency:--

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

### Eureka Discovery Client Dependency:--

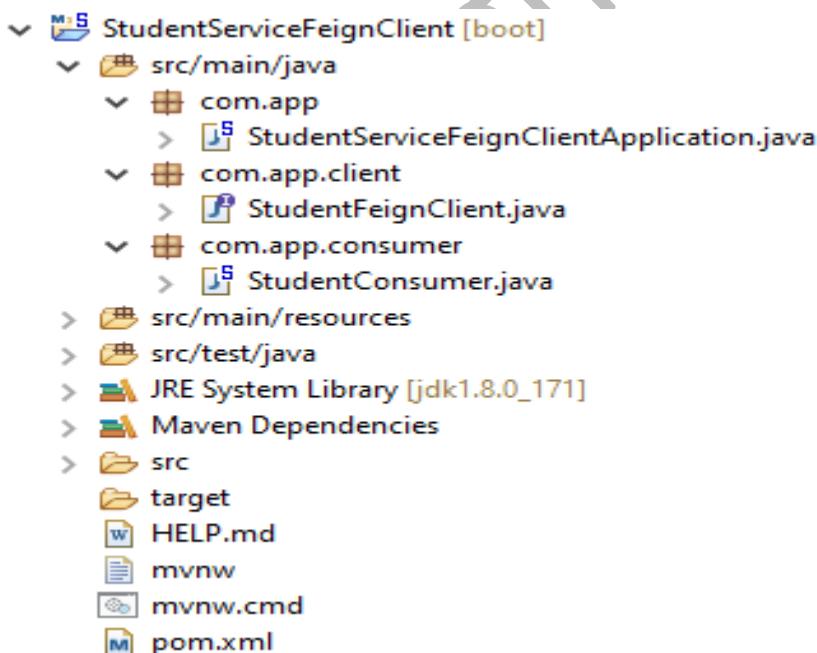
```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
```

GroupId : org.app

ArtifactId : StudentServiceConsumerFeign

Version : 1.0

## #8. Folder Structure of Consumer Application with Declarative RestClient:--



### Step#1:- Starter class for Feign client:--

```
package com.app;
import org.springframework.boot.SpringApplication;
```

```

import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;

@SpringBootApplication
@EnableEurekaClient
@EnableFeignClients
public class StudentServiceFeignClientApp {

    public static void main(String[] args) {
        SpringApplication.run(StudentServiceFeignClientApp.class, args);
        System.out.println("Student Consumer Service executed");
    }

    @Bean
    public RestTemplate restTemplate(RestTemplateBuilder builder) {
        return builder.build();
    }
}

```

**Step#2:- Define one public interface as**

```

package com.app.client;
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.GetMapping;

//Name must be same as Provide Service-Id
@FeignClient(name="STUDENT-PROVIDER")
public interface StudentFeignClient {

    @GetMapping("order/status")
    public String getMsg(); //Path and Return type same as Provider method
}

```

**Step#3:- Use in any consumer class (HAS-A) and make method call (HTTP CALL)**

```

package com.app.consumer;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
import com.app.client.StudentFeignClient;

```

```

@RestController
public class StudentConsumer {
    @Autowired
    private StudentFeignClient client;

    @GetMapping("consume")
    public String showData() {
        System.out.println(client.getClass().getName());
        return "CONSUMER=>" + client.getMsg();
    }
}

```

**NOTE:**-- Here client.getMsg() method is nothing but HTTP Request call.

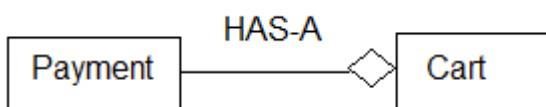
#### Execution Order:--

- 1>Start Eureka Server
- 2>Run Provider Application
- 3>Run Consumer Application and click on consumer App service id on Eureka Server Dashboard and provider consumer method level path:

<http://192.168.100.27:9841/consume>

#### 5.2 Load Balancing using Feign Client:--

- Incase of Manual Coding for load Balancing Ribbon Component is used with Type “LoadBalancingClient” (I).
- =>Here, using this programmer has to define logic of consumer method.
- =>Feign Client reduces coding lines by Programmer, by generating logic/code at runtime.
- =>Feign Client uses abstraction Process, means Programmer has to provide path with Http Method Type and also input, output details.
- =>At Runtime RibbonLoadBalancerClient instance is used to choose serviceInstance and make HTTP call.



**Feign Client:-**

**Step#1:-** Create Spring Starter Project for Eureka Server with port 8761 and dependency “Eureka Server”.

**Step#2:-** Create Cart provider Application

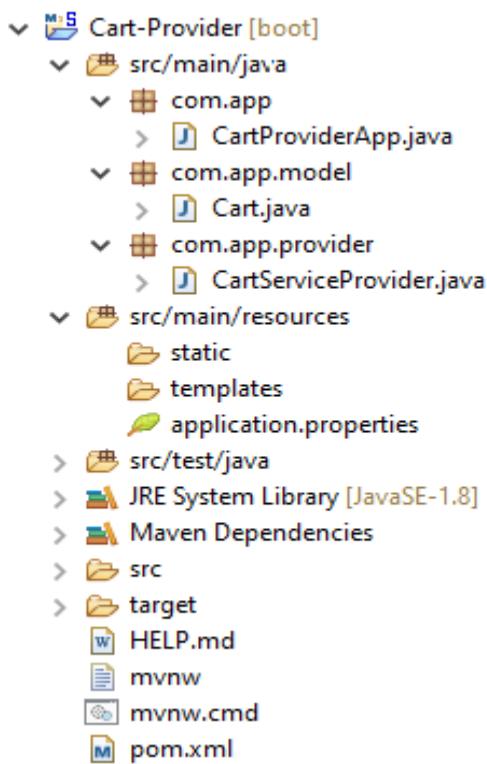
Dependencies : web, Eureka Discovery.

GroupId : com.app

ArtifactId : Cart-Provider

Version : 1.0

### #9. Folder Structure of LoadBalancing Using Feign Client (Cart-Provider):-



**Step #3:-** application.properties

server.port=8600

spring.application.name=CART-PROVIDER

eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka

eureka.instance.instance-id=\${spring.application.name}:\${random.value}

**Step #4:-** At starter class level : @EnableDiscoveryClient

**1>CartProviderApplication.java:--**

```
package com.app;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

@SpringBootApplication
@EnableDiscoveryClient
public class CartProviderApplication {
    public static void main(String[] args) {
        SpringApplication.run(CartProviderApplication.class, args);
    }
}
```

**2>Model class (Cart.java):--**

```
package com.app.model;
//import lombok.Data;

//@Data
public class Cart {

    private Integer cartId;
    private String cartCode;
    private Double cartFinalCost;

    //Default Constructor
    public Cart() {
        super();
    }

    //Parameterized constructor
    public Cart(Integer cartId, String cartCode, Double cartFinalCost) {
        super();
        this.cartId = cartId;
        this.cartCode = cartCode;
        this.cartFinalCost = cartFinalCost;
    }
}
```

```

//Set/get Method
public Integer getCartId() {
    return cartId;
}
public void setCartId(Integer cartId) {
    this.cartId = cartId;
}
public String getCartCode() {
    return cartCode;
}
public void setCartCode(String cartCode) {
    this.cartCode = cartCode;
}
public Double getCartFinalCost() {
    return cartFinalCost;
}
public void setCartFinalCost(Double cartFinalCost) {
    this.cartFinalCost = cartFinalCost;
}

@Override
public String toString() {
    return "Cart [cartId=" + cartId + ", cartCode=" + cartCode + ",
        cartFinalCost=" + cartFinalCost + "]";
}
}

```

### 3>Cart Service Provider code:-

```

package com.app.provider;
import java.util.Arrays;
import java.util.List;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.app.model.Cart;

```

```

@RestController
@RequestMapping("/cart")
public class CartServiceProvider
{
    @Value("${server.port}")
    private String port;

    @GetMapping("/info")
    public String getMsg() {
        return "CONSUMER:"+port;
    }

    @GetMapping("/data")
    public Cart getObj() {
        return new Cart(109, "ABC: "+port, 7868.98);
    }

    @GetMapping("/list")
    public List<Cart> getObjs() {

        return Arrays.asList(
            new Cart(101, "A :" + port, 876.98),
            new Cart(102, "B :" + port, 856.98),
            new Cart(103, "C :" + port, 883.98));
    }
}

```

**Step #3:- Payment Provider App with Cart Consumer code**

Dependencies : web, Eureka Discovery, Feign

#### **Feign Dependency:--**

```

<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>

```

GroupId : org.app  
 ArtifactId : Payment-Provider  
 Version : 1.0

## #10. Folder Structure of Payment-Provider Service:--

```

  ✓ Spring-cloud-Feign-LoadBalancing [boot]
    ✓ src/main/java
      ✓ com.app
        > PaymentProviderApplication.java
      ✓ com.app.consumer
        > CartServiceConsumer.java
      ✓ com.app.model
        > Cart.java
      ✓ com.app.provider
        > PaymentServiceProvider.java
    > src/main/resources
    > src/test/java
    > JRE System Library [jdk1.8.0_171]
    > Maven Dependencies
    > src
      ✓ target
      ✓ HELP.md
      ✓ mvnw
      ✓ mvnw.cmd
    ✓ pom.xml
  
```

### 1>application.properties:--

```

server.port=9890
spring.application.name=PAYMENT-PROVIDER
eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka
  
```

**2>\*\*At Starter class level add annotation : @EnableFeignClients**

### (PaymentProviderApplication.java):--

```

package com.app;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.openfeign.EnableFeignClients;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.openfeign.EnableFeignClients;

@SpringBootApplication
@EnableFeignClients
public class PaymentProviderApplication {
```

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

### 3>Cart Service Consumer (CartServiceConsumer):--

```
package com.app.consumer;  
import java.util.List;  
import org.springframework.cloud.openfeign.FeignClient;  
import org.springframework.web.bind.annotation.GetMapping;  
import com.app.model.Cart;
```

```
@FeignClient(name="CART-PROVIDER")  
public interface CartServiceConsumer  
{  
    @GetMapping("/cart/info")  
    public String getMsg();
```

```
    @GetMapping("/cart/data")  
    public Cart getObj();
```

```
    @GetMapping("/cart/list")  
    public List<Cart> getBulk();
```

```
}
```

### 4>Model class (Cart.java):-- Cart Model class same as above Project

```
package com.app.model;
```

```
public class Cart {  
    private Integer cartId;  
    private String cartCode;  
    private Double cartFinalCost;
```

```
//Default Constructor
```

```
    public Cart() {  
        super();  
    }
```

```

    //Parameterized constructor
    public Cart(Integer cartId, String cartCode, Double cartFinalCost) {
        super();
        this.cartId = cartId;
        this.cartCode = cartCode;
        this.cartFinalCost = cartFinalCost;
    }
    //Set/get Method
    public Integer getCartId() {
        return cartId;
    }
    public void setCartId(Integer cartId) {
        this.cartId = cartId;
    }
    public String getCartCode() {
        return cartCode;
    }
    public void setCartCode(String cartCode) {
        this.cartCode = cartCode;
    }
    public Double getCartFinalCost() {
        return cartFinalCost;
    }
    public void setCartFinalCost(Double cartFinalCost) {
        this.cartFinalCost = cartFinalCost;
    }
    @Override //toString method
    public String toString() {
        return "Cart [cartId=" + cartId + ", cartCode=" + cartCode + ",
cartFinalCost=" + cartFinalCost + "]";
    }
}

```

### 5>PaymentServiceProvider.java:--

```

package com.app.provider;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;

```

```
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.app.consumer.CartServiceConsumer;
import com.app.model.Cart;
@RestController
@RequestMapping("/payment")
public class PaymentServiceProvider {

    @Autowired
    private CartServiceConsumer consumer;

    @GetMapping("/message")
    public String getMsg() {
        return consumer.getMsg();
    }
    @GetMapping("/one")
    public Cart getOneRow() {
        return consumer.getObj();
    }
    @GetMapping("/all")
    public List<Cart> getAllRows() {
        return consumer.getBulk();
    }
}
```

**Step #4 Execution Order:--**

=>Run Eureka Server

=>Run Cart Provider 3 times (with different port)

The screenshot shows the Spring Eureka dashboard. In the top right, it says "HOME" and "LAST 1000 SINCE STARTUP". Under "System Status", there's a table with two rows: Environment (test) and Data center (default). On the right, another table shows current time (2019-04-21T22:49:44 +0530), uptime (00:09), lease expiration enabled (true), renew threshold (6), and renew count (12). Below that is a section for "DS Replicas" with a table showing one instance registered under "localhost".

=>run Payment Provider 1 time

=>Goto Eureka server and Run Payment service and provide bellow URLs one by one.

1><http://192.168.100.17:9890/payment/message>

2><http://192.168.100.17:9890/payment/one>

A screenshot of a browser window. The address bar shows "192.168.100.17:9890/payment/one". The page content displays a JSON object: {"cartId":109,"cartCode":"ABC: {server.port}","cartFinalCost":7868.98}

3><http://192.168.100.17:9890/payment/all>

A screenshot of a browser window. The address bar shows "192.168.100.17:9890/payment/all". The page content displays a JSON array: [{"cartId":101,"cartCode":"A :{server.port}","cartFinalCost":876.98}, {"cartId":102,"cartCode":"B :{server.port}","cartFinalCost":856.98}, {"cartId":103,"cartCode":"C :{server.port}","cartFinalCost":883.98}]

## 7. Spring Cloud Config Server:--

It is also called as Configuration Server. In Spring Boot/Cloud Projects, it contains files like : Properties files [application.properties]

=>In every application (Microservice) properties/yml file contain setup related keys like. (Database, ORM, Batch, Email, AOP, Security, Eureka Server, Gateway... etc) in the form of key=value.

=>In some cases Key=Value need to be changed or new key=value need to be added.

At this time, we should

- >Stop the Server (Application)
- >Open/find application.properties file
- >add External key=value pairs or do modifications.
- >save changes [save file]
- >Re-build file [re-create jar/war]
- >Re-Deploy [re-start server]

=>In case of multiple microservices these common keys related for every Project.

And this is done in All related Project [Multiple Microservices] which is repeated task for multiple applications.

=>\*\*In this case, Config Server concepts is used

\*\*To avoid this repeated (or lengthy) process use application.properties this is placed outside of your Project i.e. known as “config Server” .

=>Writing common properties only one time outside of all project.

=>Easy to modify common properties (one place to modify properties files that effects all projects).

=>Every service instances not required to **stop** restart for effect for properties file modifications (Refersh Scope).

=>Config server Process maintains three (3) properties file. Those are:

- a>One in = Under Project (Microservice)
- b>One in = Config Server (link file)
- c>One in = Config server (External) also called as Source file.

#### **Types of Config Server:--**

Spring Cloud has provided Setup and support for configuration properties. It can be handled in two ways. Those are

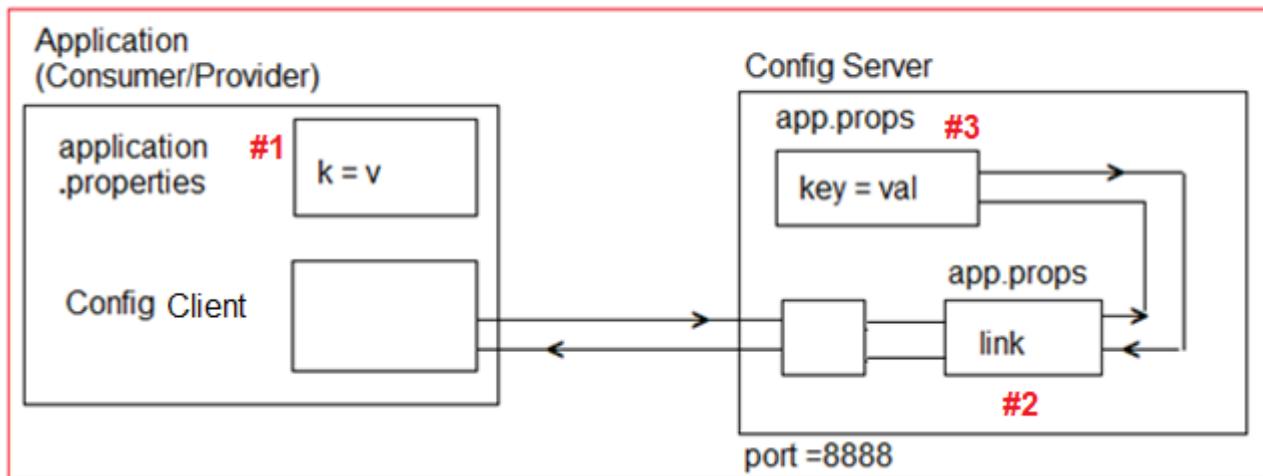
- a>Native Config Server
- b>External Config Server

**a>Native Config Server:--** It is also called as local file placed in Config server only. In this case one properties file is created inside folder System (or drive).

Ex:-- D:/abc/myapp(Or Under config server location file).

### #1 Native Config Server

Local File Config Server



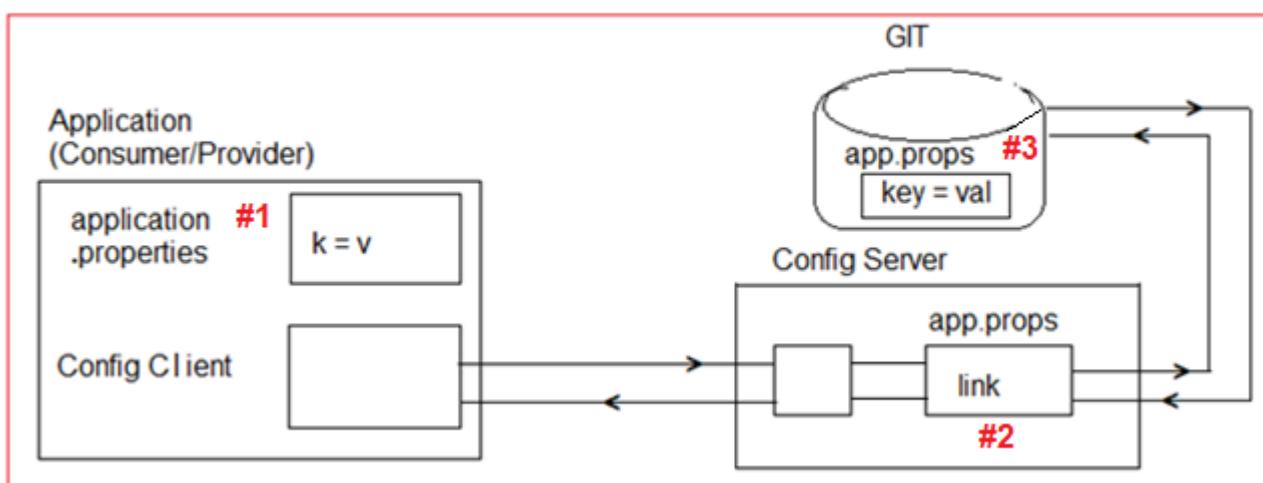
**b>External Config Server:**-- In this case properties file is placed outside the Config server. Ex: GIT (github).

=> Here, Properties file is placed in External and accesses using its URL. It is also called as Global Location.

\*\*"GIT HUB" is used in this process.

### #2 GIT Config Server

External Config Server



=>In Consumer/Producer Project we should add Config Client dependency which gets config server details at runtime.

=>Config server runs at default port=8888.

**Step to implements Spring Cloud Config Server-Native & External:--**

**Step#1:-** Create Spring Boot Starter Project for “Config-Server” with dependency : Config Server.

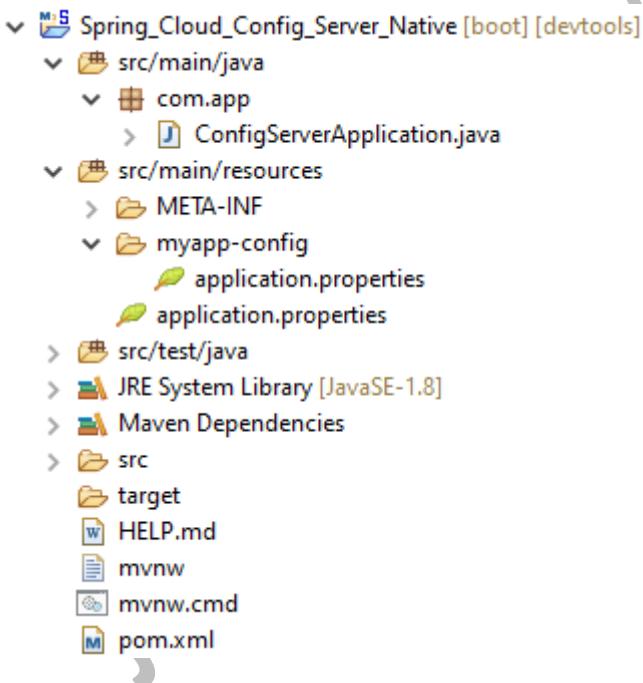
### Config Server Dependency:--

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-config-server</artifactId>
</dependency>
```

GroupId : org.app  
 ArtifactId : Config-Server  
 Version : 1.0

### 7.1 Native Config Server:--

#### #11. Spring cloud config server component using Native:--



**Step#2.a:-** Native application.properties:-- Create folder “myapp-config” under src/main/resources.

server.port=8888

spring.profiles.active=native

spring.cloud.config.server.native.searchLocations=classpath:/myapp-config

**Case#2.b:-** External application.properties

server.port=8888

spring.cloud.config.server.git.uri=https://github.com/javabyraghu/configserverex

**Step#3:-** Create sub folder “myapp-config” in src/main/resources folder.

=>Create file under “myapp-config” **application.properties** inside this folder. It behaves like source

Having ex key: eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka

Or

Create Git Project “configserverex” and create application.properties inside this. Place above key and save file.

**Step#4:-** Provider include resource code in pom.xml to consider properties files to be loaded into memory.

```
<resources>
  <resource>
    <filtering>true</filtering>
    <directory>src/main/resources</directory>
    <includes>
      <include>*.properties</include>
      <include>myapp-config</include>
    </includes>
  </resource>
</resources>
```

**pom.xml:-**

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
```

```
<artifactId>spring-boot-starter-parent</artifactId>
<version>2.1.1.RELEASE</version>
<relativePath/> <!-- lookup parent from repository -->
</parent>
<groupId>com.app</groupId>
<artifactId>Config-Server</artifactId>
<version>1.0</version>
<name>Config-Server</name>
<description>Demo project for Config-Server</description>
<properties>
    <java.version>1.8</java.version>
    <spring-cloud.version>Greenwich.SR1</spring-cloud.version>
</properties>
<dependencies>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-config-server</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>${spring-cloud.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>

```

```

        </dependency>
    </dependencies>
</dependencyManagement>
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
    <resources>
        <resource>
            <filtering>true</filtering>
            <directory>src/main/resources</directory>
            <includes>
                <include>*.properties</include>
                <include>myapp-config</include>
            </includes>
        </resource>
    </resources>
</build>
</project>

```

**Step#5:**-- Starter class of ConfigServer App, add annotation: **@EnableConfigServer**

```

package com.app;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.config.server.EnableConfigServer;
```

```

@SpringBootApplication
@EnableConfigServer
public class ConfigServerApplication {
    public static void main(String[] args) {
        SpringApplication.run(ConfigServerApplication.class, args);
        System.out.println("Config Server Executed");
    }
}
```

}

**Step#6:**-- In Consumer/Provider Projects pom.xml file add dependency : Config Client  
(or copy below dependency)

**Config Client Dependency:**--

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-config-client</artifactId>
</dependency>
```

**Execution Order:**--

=>Eureka Server

=>Config Server

=>Producer (Provider)

=>Consumer

**Step#1:-** Eureka Server Project Dependency : Eureka Server

=>Write application.properties.

=>Add @EnableEurekaServer annotation

**Step#2:-** Define Config Server Project Dependency : Config Server

=>Write application.properties

=>Add @EnableConfigServer annotation

**Step#3:-** Create Provider Project (Order)

Dependency : Web, Eureka Discovery, Config Client...

=>Write application.properties

=>Add @EnableDiscoveryClient annotation

=>Write Provider (RestController)

**Step #4:-** Create Consumer Project (Invoice)

Dependency : Web, Eureka Discovery, Config Client, Feign....

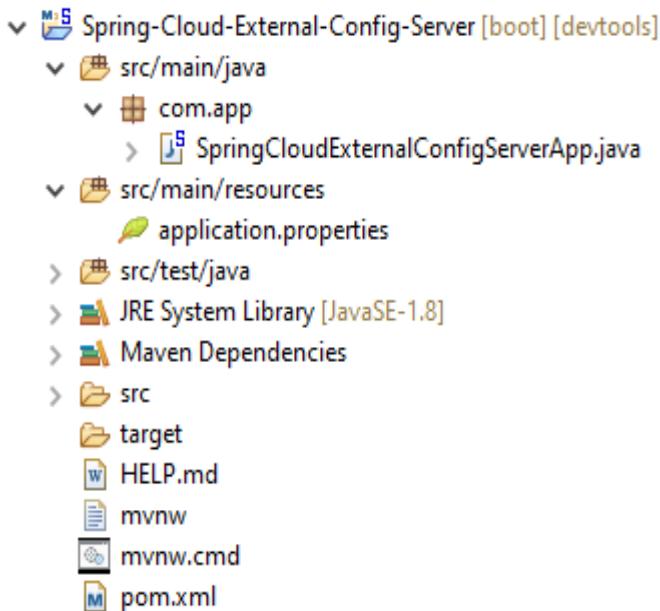
=>Write application.properties

=>Add @EnableFeignClient annotation

=>Write Consumer [Feign Client] and Invoice Provider (RestController)

## 7.2 External Config-Server:--

## #12. Folder Structure of External Config Server:--



### Step to configure External Config Server (only modifications):--

**Step#1:-** In Config Server Project, delete folder myapp-config

**Step#2:-** In Config Server Project modify application.properties with Git URI

server.port=8888

spring.cloud.config.server.git.uri=https://github.com/javabyraghu/configserverex  
or

spring.cloud.config.server.git.uri=https://gitlab.com/udaykumar0023/configserverex

**Step#3:-** In config Server Project, in pom.xml delete line

<include>myapp-config </include>

**pom.xml:--**

```

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  
```

<modelVersion>4.0.0</modelVersion>

```

<parent>
  <groupId>org.springframework.boot</groupId>
  
```

```
<artifactId>spring-boot-starter-parent</artifactId>
<version>2.1.1.RELEASE</version>
<relativePath /> <!-- lookup parent from repository -->
</parent>
<groupId>com.app</groupId>
<artifactId>Spring-Cloud-External-Config-Server</artifactId>
<version>1.0</version>
<name>Spring-Cloud-External-Config-Server</name>
<description>Project for Spring config server</description>

<properties>
    <java.version>1.8</java.version>
    <spring-cloud.version>Greenwich.SR1</spring-cloud.version>
</properties>
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-config-server</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
```

```
</dependency>
</dependencies>

<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>${spring-cloud.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
    <resources>
        <resource>
            <filtering>true</filtering>
            <directory>src/main/resources</directory>
        </resource>
    </resources>
</build>
</project>
```

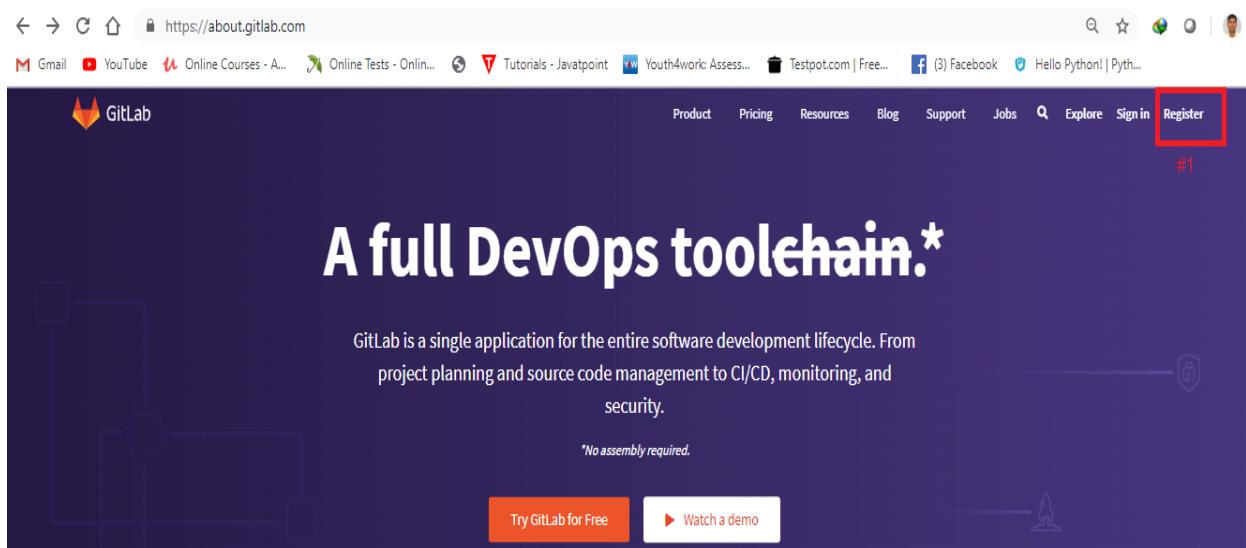
**Step#4:-** Create git account and create configserverex Repository. Under this create file application.properties.

**How to create GitLab Account:--** Follow bellow You tube Links

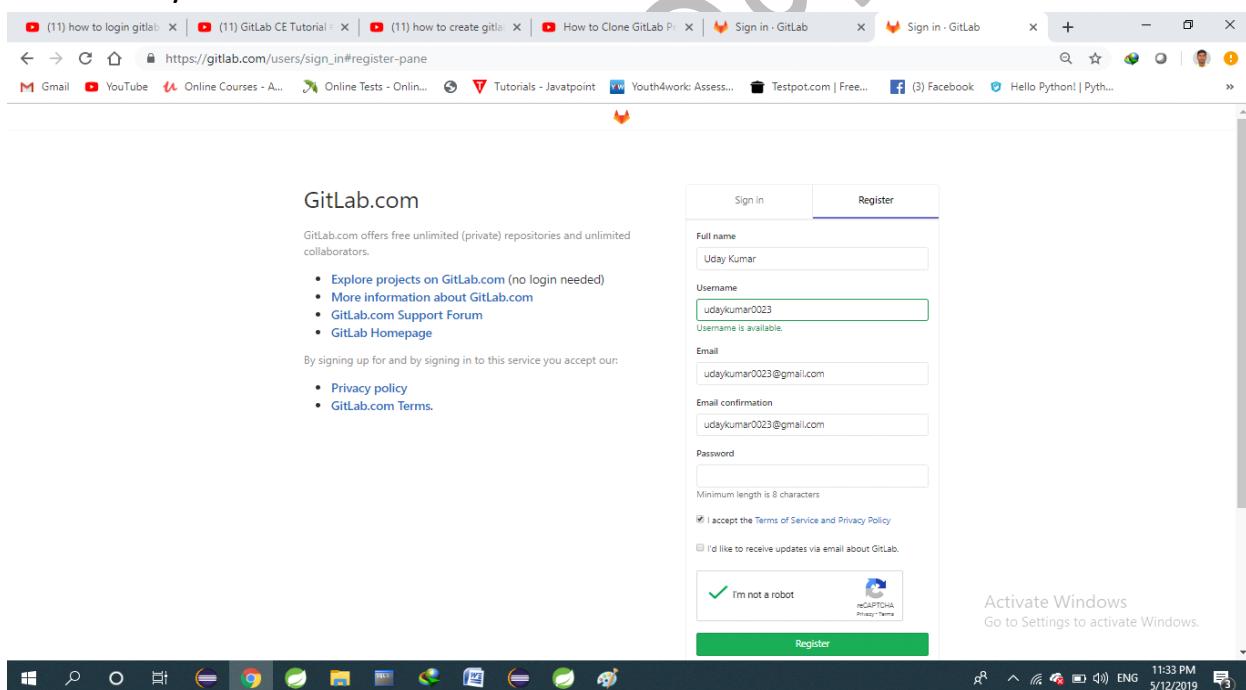
[https://www.youtube.com/watch?v=\\_3I5OelRtk8](https://www.youtube.com/watch?v=_3I5OelRtk8)

**Step#1:-** Goto <https://about.gitlab.com> links

**Step#2:-** Click on Register Menu



**Step#3:-** Fill the details and click on Register Button else login with github or Google Account If you have.



**Step#4:-** Click on Request new Confirmation Email

Almost there...

Please check your email to confirm your account

No confirmation email received? Please check your spam folder or

[Request new confirmation email](#)

Explore Help About GitLab

**Step#5:- Enter Email address and click on Resend Button.**

GitLab.com

GitLab.com offers free unlimited (private) repositories and unlimited collaborators.

- Explore projects on GitLab.com (no login needed)
- More information about GitLab.com
- GitLab.com Support Forum
- GitLab Homepage

Resend confirmation instructions

Email  
udaykumar0023@gmail.com

Resend

Already have login and password? [Sign in](#)

By signing up for and by signing in to this service you accept our:

- Privacy policy
- GitLab.com Terms

**Step#6:- Login to Gmail account and click on Confirm your Account.**

Gmail Search mail

Inbox 8

Compose

Starred Snoozed Important Sent Drafts 8 Categories Social 15 Updates 18 Forums 1

Confirmation instructions

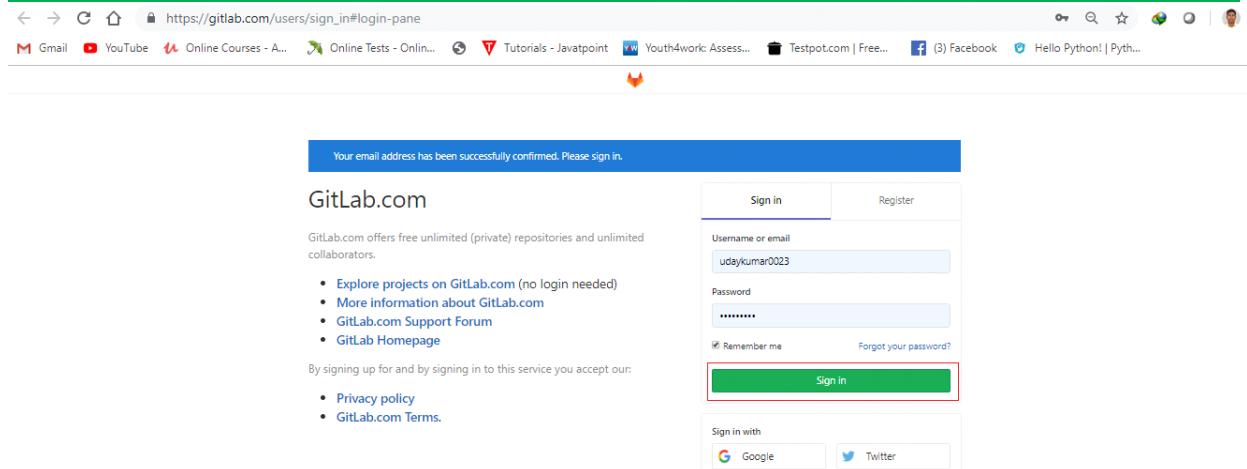
GitLab <gitlab@mg.gitlab.com> to me 11:36 PM (8 minutes ago)

Thanks for signing up to GitLab!

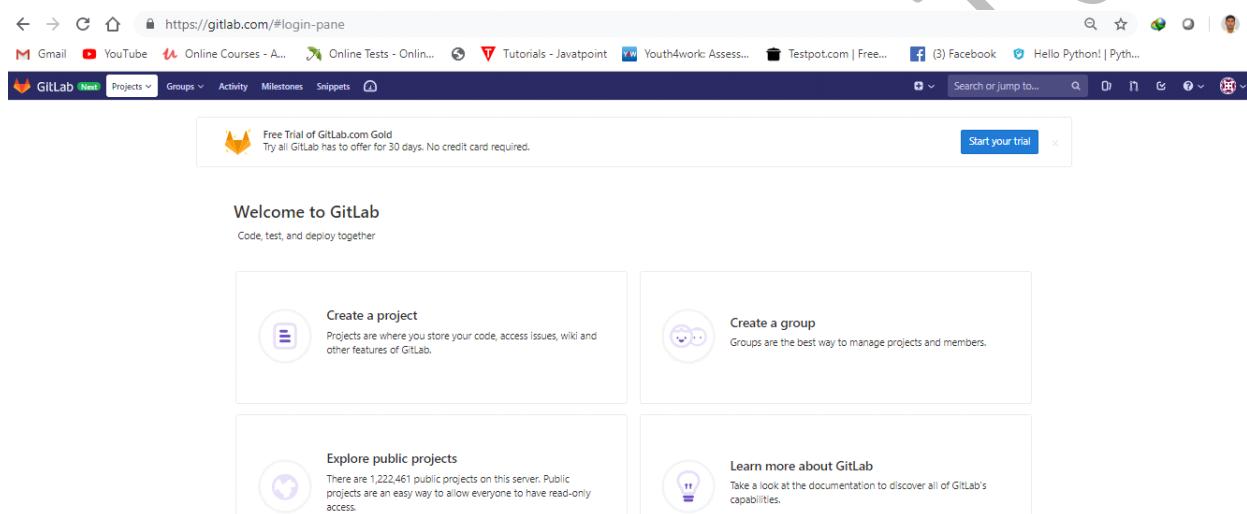
To get started, click the link below to confirm your account.

[Confirm your account](#)

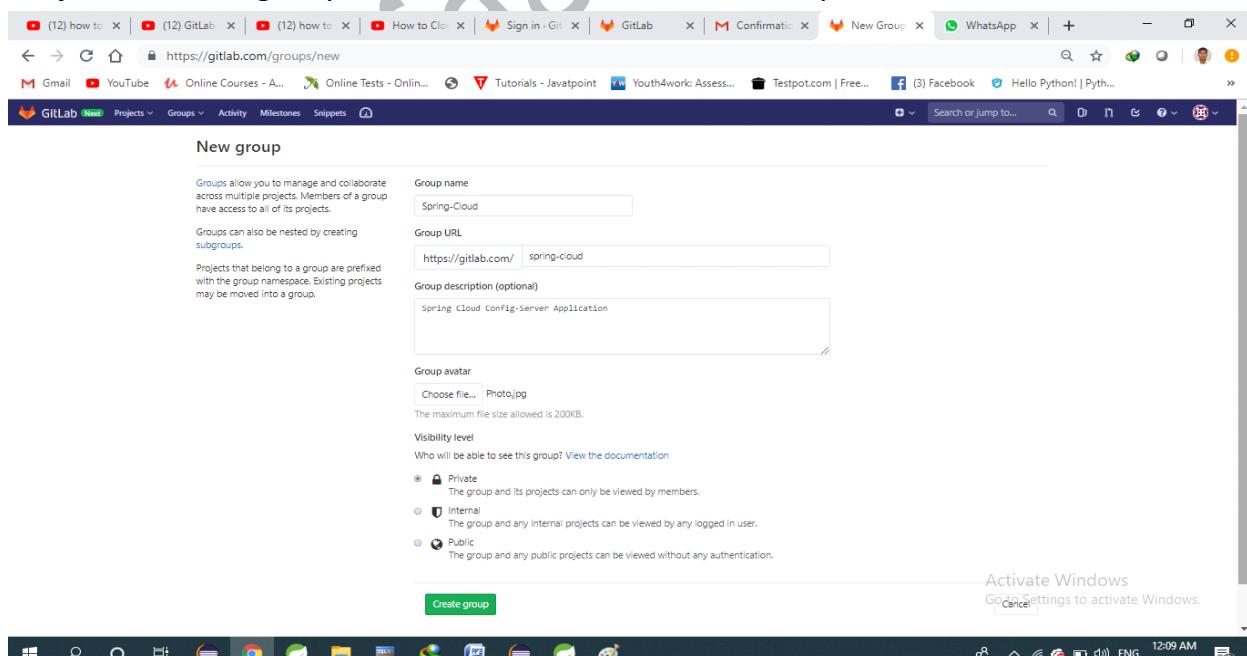
**Step#7:- Provide your valid Credentials and click on Sign in Button.**



## Step#8:-Final screen After Login into Gitlab account, Click on Create Group.



## Step#9:- Fill the group name and click on Create Group



## Step#10:-After successful Creating a new Group screen looks like.

The screenshot shows the GitLab web interface. At the top, there's a navigation bar with links like Gmail, YouTube, Online Courses, etc. Below it, the main header says 'GitLab' and 'Projects'. A sidebar on the left lists 'Overview', 'Details', 'Activity', 'Contribution Analytics', 'Issues' (0), 'Merge Requests' (0), 'Kubernetes', 'Members' (highlighted with a red box), and 'Settings'. The main content area shows a success message: 'Group 'SpringCloudconfigserver' was successfully created.' Below this, there's a summary card for 'SpringCloudconfigserver' with a Group ID of 5190981. It includes sections for 'Subgroups and projects', 'Shared projects', and 'Archived projects'. A search bar at the bottom allows searching by name or last created.

### Step#11:- Click on Member Menu to add the new member into your project

This screenshot shows the 'Members' section of the 'SpringCloudconfigserver' group. The 'Members' button in the sidebar is highlighted with a red box. A success message 'Users were successfully added.' is displayed. Below it, there's a form to 'Add new member to SpringCloudconfigserver'. It includes fields for 'Search for a user', 'Role' (set to 'Guest'), 'Expiration date' (set to '2019-05-15'), and a 'Add to group' button. The 'Existing members' table lists three users: Aishwarya Venkatesh, Uday Kumar, and ram kumar, each with their access details and expiration dates.

=>Select Member from list, Role Permissions like (Guest, Reporter, Developer, Maintainer, Owner) and Expire date and finally click on Add to Group.

### Step#12:- Click on Group name (**SpringCloudconfigserver**) and then after click on New Project to create a project.

This screenshot shows the 'Details' page for the 'SpringCloudconfigserver' group. The group name is highlighted with a red box. The 'New project' button in the top right corner is also highlighted with a red box. The page contains a summary card for 'SpringCloudconfigserver' with a Group ID of 5190981, sections for 'Subgroups and projects', 'Shared projects', and 'Archived projects', and a search bar at the bottom.

## Step#13:- Give a Project name and Click on Create Project.

## Step#14:- It will show the all details.

Create a new repository

```
git clone https://gitlab.com/springcloudconfigserver/spring-cloud-eureka-server.git
cd spring-cloud-eureka-server
touch README.md
git add README.md
git commit -m "add README"
git push -u origin master
```

Push an existing folder

```
cd existing_folder
git init
git remote add origin https://gitlab.com/springcloudconfigserver/spring-cloud-eureka-server.git
git add .
git commit -m "Initial commit"
git push -u origin master
```

Push an existing Git repository

```
cd existing_repo
git remote rename origin old-origin
git remote add origin https://gitlab.com/springcloudconfigserver/spring-cloud-eureka-server.git
git push -u origin --all
git push -u origin --tags
```

## Step15:- Click on new file to create a file into repository

The screenshot shows a GitLab project named 'Spring-Cloud-Eureka-Server'. The left sidebar has 'Project' selected. The main area displays a message: 'The repository for this project is empty. You can create files directly in GitLab using one of the following options.' Below this are four buttons: 'New file' (highlighted with a red box), 'Add README', 'Add CHANGELOG', and 'Add CONTRIBUTING'. Further down, there's a 'Command line instructions' section with a note about uploading files via command line.

## Step16:- Provide the details in file and then click on commit changes.

The screenshot shows a GitLab repository page for 'configserverex'. The left sidebar has 'Repository' selected. In the main area, a file named 'application.properties' is shown. The content of the file is:

```
server.port=8888
eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka
```

Below the file content are standard GitLab file operations: Find file, Blame, History, Permalink, Edit, Web IDE, Replace, and Delete.

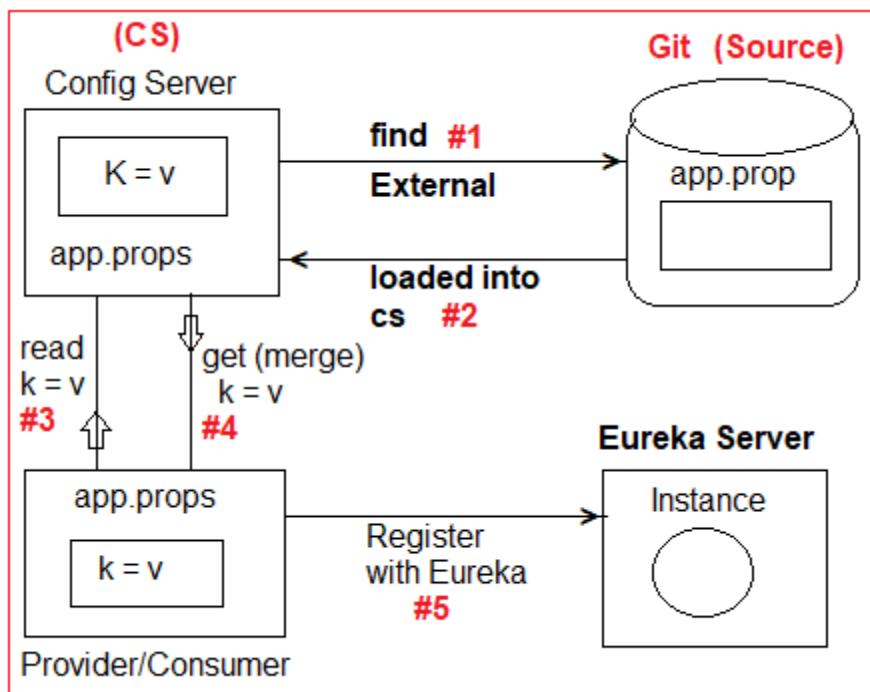
### Config Server with Provider/Consumer Execution flow [External Source]:--

=>On Startup ConfigServer (CS) Application it will goto External Source (Git) and read \_\_.properties (or) \_\_.yml file having key=value pairs.

=>Then Provider/Consumer Application (On Startup) will try to read k=v from Config Server and merge with our application.properties.

=>If same key is found in both Config Server and Provider/Consumer App, then 1<sup>st</sup> priority is : Config Server.

=>After fetching all required props then Provider gets registered with Eureka Server.



=>Every Microservices must have below dependency in pom.xml.

```

<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-config</artifactId>
</dependency>

```

=>It behaves as Config Client connected to Config Server.

=>Above Config Client dependency will search forConfig Server in a default location given as : (key = value)

**spring.cloud.config.uri=http://localhost:8888**

=>To modify above location (IP or PORT) provide this key in Config Client setup using bootstrap.properties.

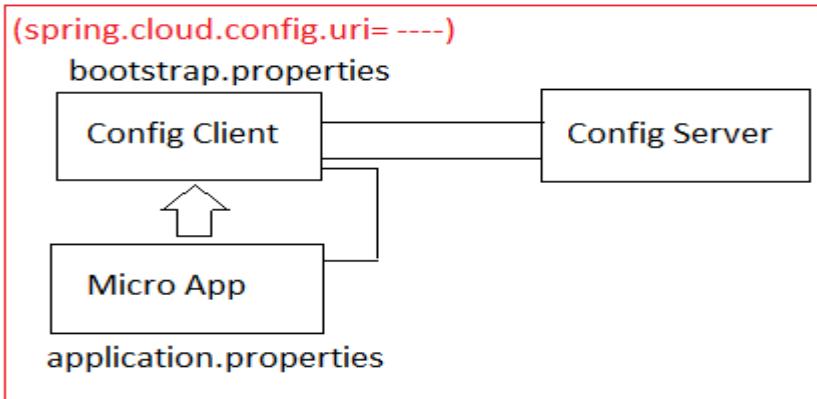
### **Q>What is bootstrap.properties in Spring Boot (cloud) Programming?**

Ans:-- Our Project (child Project) contains input keys in application.properties, in same way Parent Project also maintains one Properties file named as : **bootstrap.properties** which will be loaded before our application.properties.

### **Q>What is the difference between application.properties file and bootstrap.properties file?**

Ans:-- application.properties file is used to provide input to our application, where bootstrap.properties file is used to provide **Input** to parent project (or Configuration Setup) which gets run before our application.

**Execution Order:--**      1>bootstrap.properties      2>application.properties



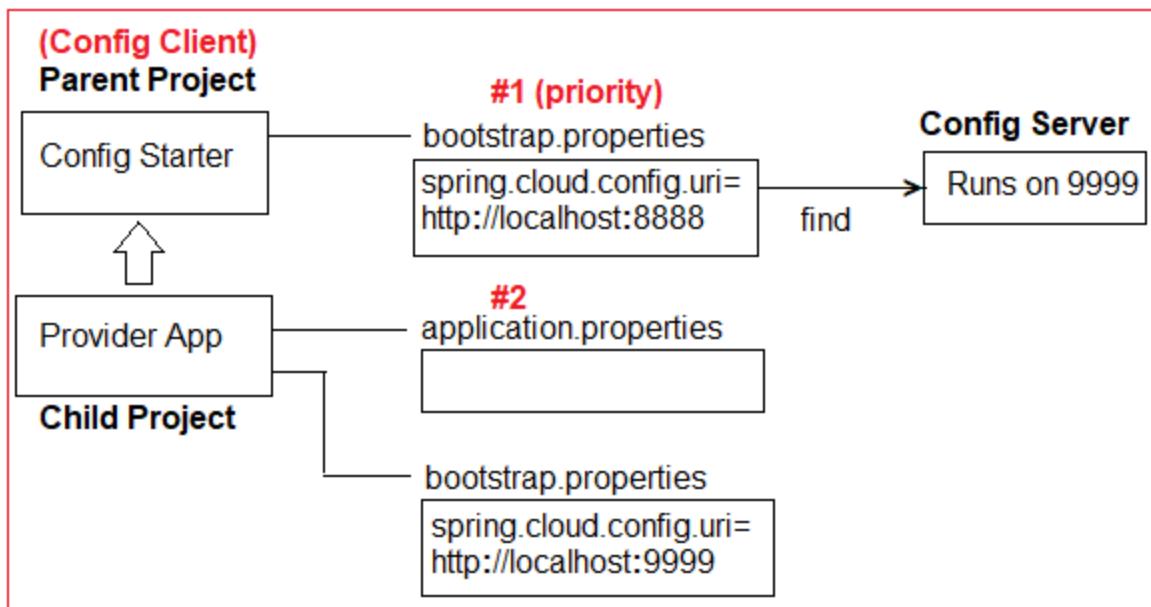
**Execution Order:--** Parent Project-loads bootstrap.properties /.yml or Project-loads application.properties /.yml

->We can override this file in our project to provide key-values to be loaded by Parent Project.

\*\*By default Config Server runs on <http://localhost:8888> even config client also takes this as default location.

\*\*To modify this IP/PORT use bootstrap.properties file in Our Project.

\*\*In this bootstrap.properties file override key : **spring.cloud.config.uri** to any other location where Config Server is running.



### Coding Changes for Config Server and Provider/Consumer Apps:--

**Step#1:-** Open application.properties file in Config Server Project and modify port number.

```
server.port =9999
```

**Step#2:-** In Provider/Consumer Project, create file bootstrap.properties under src/main/resources

**Step#3:-** Add key=value in bootstrap.properties file

```
spring.cloud.config.uri=http://localhost:8888
```

=>By default Config client will not get updates or modification from Config Server after starting client + microservice.

=>Only first time config Client fetch the data even to get new Modification after starting Client (without-Restart) use Annotation : @RefreshSocpe.

=>At Config Server application add below dependency in pom.xml.

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

=>In application.properties (in Config Server) provide Native or External configuration details. This file is also called as Link file.

server.port=8888

spring.cloud.config.uri=http://localhost:8888

spring.cloud.config.server.git.uri=https://gitlab.com/udaykumar0023/configserverex

**Q>Which class will load bootstrap.properties file for config Server URI fetching?**

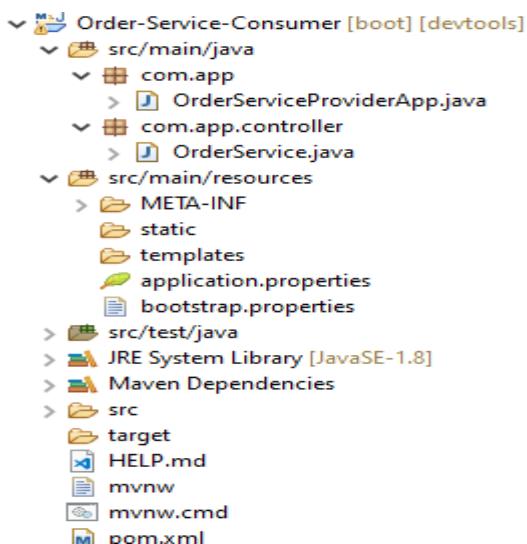
**Ans:**-- ConfigServicePropertySourceLocator will read config server input by default from <http://localhost:8888>.

=>It is only **first execution step** in provider Consumer Project.

### Provider/Consumer Application:--

Create any Microservice application with Dependencies : Web, Eureka Discovery, Config- Client.

### 13. Folder Structure of Microservice with bootstrap:--



#### Step#1:- Starter class

```

package com.app;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
import org.springframework.cloud.context.config.annotation.RefreshScope;

@SpringBootApplication
@EnableDiscoveryClient
@RefreshScope //Get updates from Config Server Automatically
public class OrderServiceProviderApp {

```

```

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

```

**Step#2:- Controller class.**

```

package com.app.controller;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/order")
public class OrderService {

    @Value("${my.app : Hello Default One}")
    private String port;

```

```

    @GetMapping("/status")
    public String getOrderStatus() {
        return "FINISHED ::"+port;
    }
}

```

**Step#3:-1>application.properties:--**

```

server.port=7800
spring.application.name=ORDER-PROVIDER
eureka.client.serviceUri.defaultZone=http://localhost:8761/eureka
eureka.instance.instance-id=${spring.application.name}:${random.value}
management.endpoints.web.exposure.include=*

```

**2>bootstrap.properties:--**

```
spring.cloud.config.uri=http://localhost:8888
```

**Execution Order:--**

1. Eureka Server
2. Config Server
3. MicroServices (Order)

=>Enter URL : <http://192.168.100.27:7800/order/status>

=>To enable **refresh** concept for microservices application(Order-APP), follow below steps.

\*\*\*INSIDE ORDER-APP only.

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

=>In application.properties file add

```
management.endpoints.web.exposure.include=*
```

=>At Controller class level, add **RefreshScope**

- >Run : EurekaConfig Server, CART-APP.
- >Enter URL for CartApp.
- >Now modify value in Hibhub file.
- >Open postman and makes POST request.

<b>POST</b>	<a href="http://localhost:7800/actuator/refresh">http://localhost:7800/actuator/refresh</a>	<b>SEND</b>

=>After showing success msg, Goto OrderApp URL & Refresh.

#### **NOTE:--**

- \*\*\*Actuator refresh is ready-made service given by Spring boot that fetch the data from Config Server location to ourApp.
- =>It must be made as POST type request onl, using any http client (Ex: POSTMAN).

## **8. Fault Tolerance API :--**

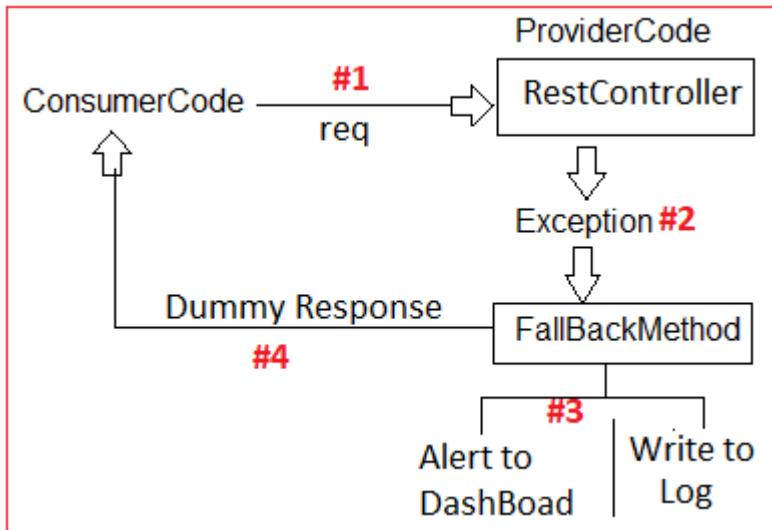
If any Microservice is continuously throwing Exceptions, then logic must not be executed every time also must be finished with smooth termination. Such Process is called as Fault Tolerance.

=>Fault Tolerance is achieved using fallBackMethod and CircuitBreaker Library in distributed system.

=>Fallback or Circuit breakers are used to handle exception which is occurred in Producer Application. It avoid Improper response and Shutdown termination of process.

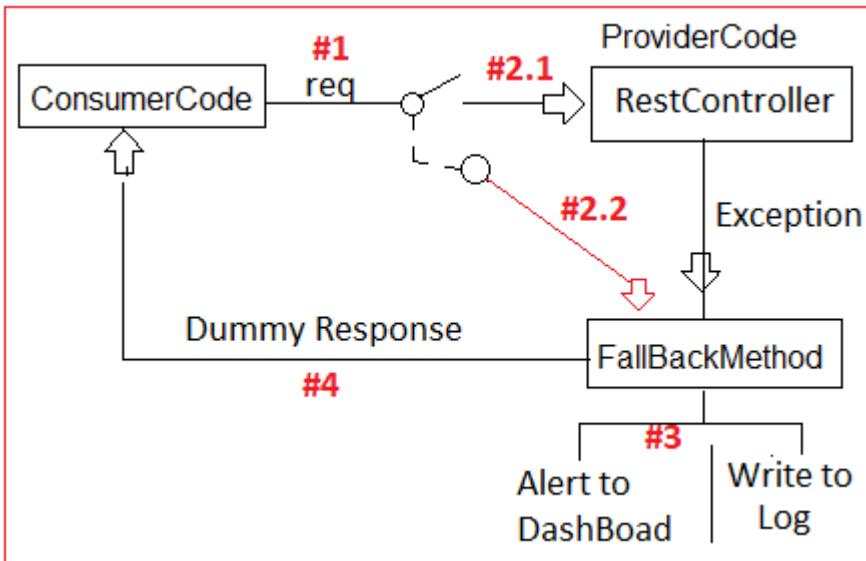
a. **fallBackMethod**:- If Microservice is throwing exception then service method execution is redirected to another supportive method (**fallBackMethod**) which gives dummy output and “alerts to DashBoard” (to Dev, MS, Admin teams). Writes to Log files, Email to admin etc.. is known as Fallback method, It makes termination of process in smooth manner.

#### FallBackMethod Process:--



b. **CircuitBreaker**:- If Producer App (Service) is throwing exceptions continuously then circuit breaker stops executing Producer code and Exception flow directly linked to fallback method. This is called as Opening a Circuit. It means Avoid invalid process or incomplete process execution and return “**Dummy Message**”. After some time gap (or) no. of request given to fallback, again re-checks once, still same continue to fallBackMethod else execute Microservice (Producer).

#### CircuitBreaker:--



**Q>What is the difference between try-catch and Fallback with CircuitBreaker(CB)?**

Ans:--In try-catch always try code is executed even if exception is regular, whereas Fallback with **CircuitBreaker** will stop execution of actual logic if exception is regular.

#### Hystrix:--

It is an API (set of classes and interfaces) given by Netflix to handle Proper Execution and Avoid repeated exception logic (Fault Tolerance API) in Microservice Programming.

- =>It is mainly used in production Environment not in Dev Environment.
- =>Hystrix supports FallBack and CircuitBreaker process.
- =>It provides Dashboard for UI to view current request flow and Status. (View problems and other details in Services).

#### Working with Hystrix:--

**Step#1:-** Create one Microservice Application with dependencies web, eureka

Discovery client, Hystrix.

groupId : com.app

artifactId : Spring\_Cloud\_Hystrix\_Server

**Netflix Hystrix dependency:--** Add below dependencies in pom.xml

```

<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
</dependency>

```

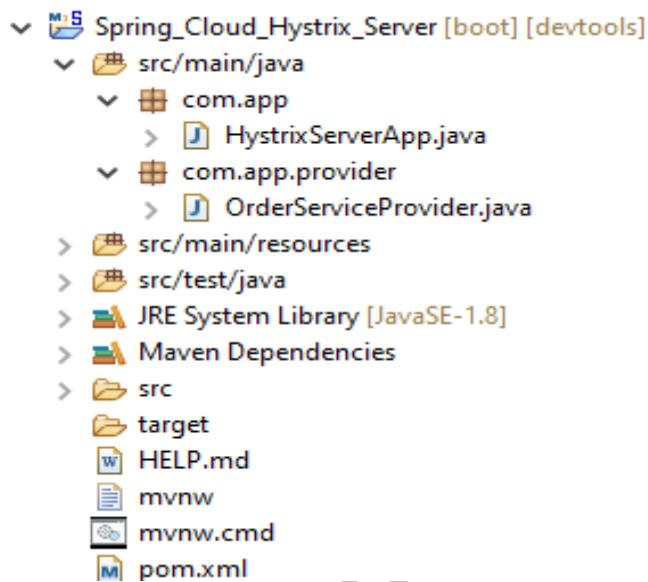
**Step#2:**-- At starter class level apply annotation `@EnableCircuitBreaker` (or) `@EnableHystrix`.

=>`@EnableCircuitBreaker` will find concept at runtime using pom.xml dependency.  
ex: Hystrix, Turbine etc... where as `@EnableHystrix` will execute only Hystrix CircuitBreaker.

**Step#3:**-- Define one RestController with actual Service and fallback method and apply Annotation : `@HystrixCommand` with Details like fallBackMethod, commandKey...

**NOTE:**-- Fallback method return type must be same as Actual Service method

#### #14. Folder Structure of Hystrix (CircuitBreaker) Implementation:--



#### pom.xml:--

```

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.1.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
  
```

```
</parent>
<groupId>com.app</groupId>
<artifactId>Spring_Cloud_Hystrix_Server</artifactId>
<version>1.0</version>
<name>Spring_Cloud_Hystrix-Server</name>
<description>Demo project for Hystrix Service</description>

<properties>
    <java.version>1.8</java.version>
    <spring-cloud.version>Greenwich.SR1</spring-cloud.version>
</properties>

<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

```

<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>${spring-cloud.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
</project>

```

**Step#1:- application.properties file:--**

server.port=9800  
spring.application.name=ORDER-PROVIDER  
eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka

**Step#2:- Apply below any one annotation at Starter class.**

@EnableCircuitBreaker  
@EnableHystrix

**HystrixServerApplication.java:--**

```

package com.app;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
import org.springframework.cloud.netflix.hystrix.EnableHystrix;

```

```
@SpringBootApplication
@EnableDiscoveryClient
@EnableHystrix
public class HystrixServerApp {
    public static void main(String[] args) {
        SpringApplication.run(HystrixServerApp.class, args);
    }
}
```

**Step#3:- Define RestController with FallbackMethod (OrderServiceProvider.java).**

```
package com.app.provider;
import java.util.Random;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
import com.netflix.hystrix.contrib.javanica.annotation.HystrixCommand;

@RestController
public class OrderServiceProvider {

    @GetMapping("/show")
    @HystrixCommand(fallbackMethod="showFallBack")
    //parameter must be same as fallBackMethod name
    public String showMsg() {
        System.out.println("From service");
        if (new Random().nextInt(10)<=10) {
            throw new RuntimeException("DUMMY");
        }
        return "Hello From Provider";
    }
    //fallBack method
    public String showFallBack() {
        System.out.println("From ballback");
        return "From FallBack method";
    }
}
```

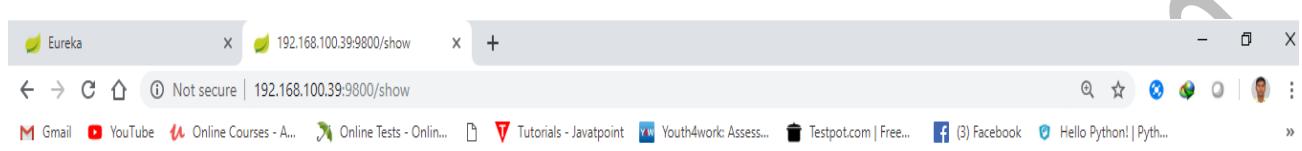
**NOTE:-- Fallback method returnType must be same as service method return type.**

## **Execution Process:--**

#### **Step#4:- Run Eureka Server and Order Provider.**

**Step#5:-** Goto Eureka and click on **ORDER PROVIDER** Instance and enter URL path :/show (<http://192.168.100.39:9800/show>)

=>Referesh multiple times to, see console.



From FallBack method

## **OUTPUT SCREEN OF “HYSTRIX-SERVICE-APP”:-**

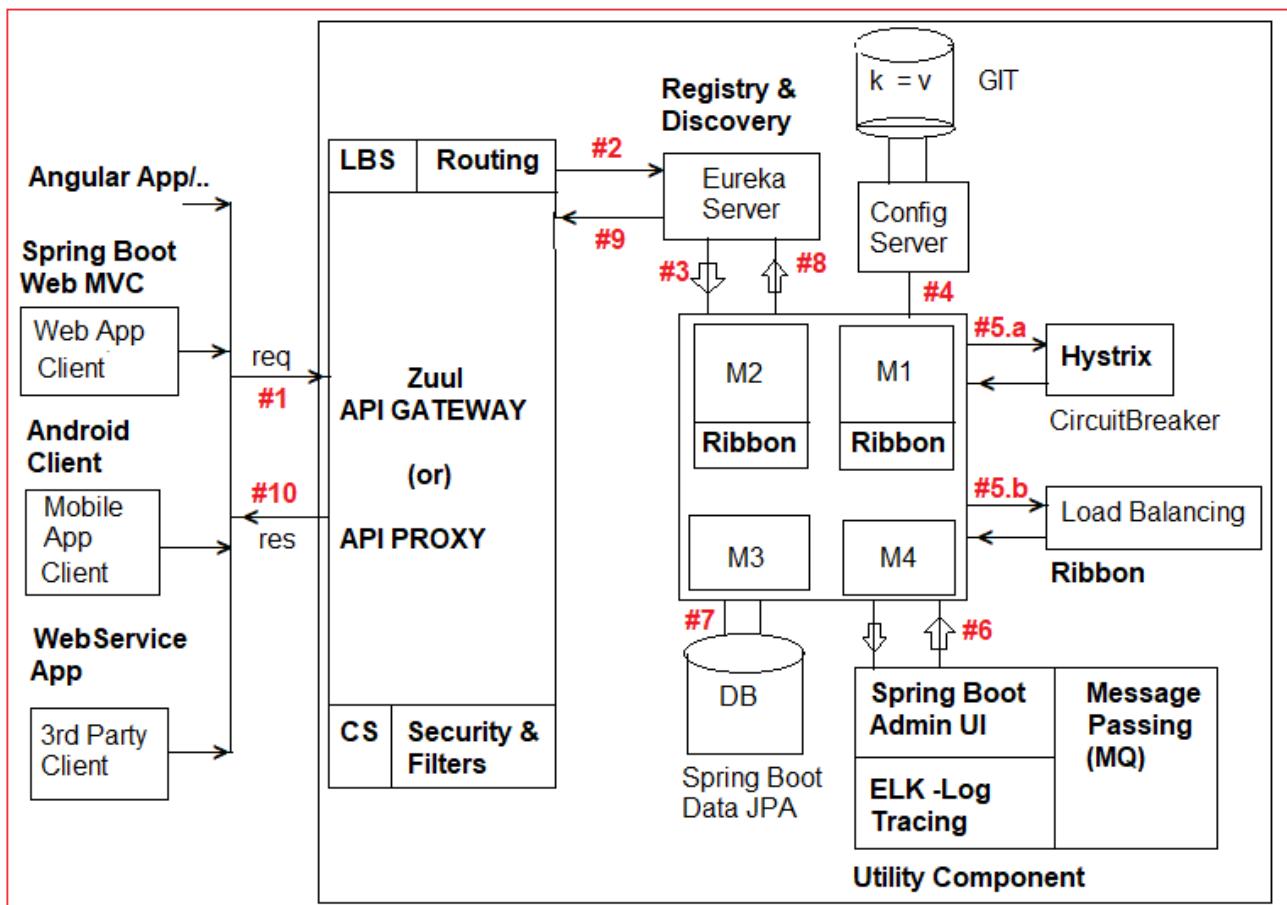
```
2019-04-21 10:19:29.114 INFO 13632 --- [freshExecutor-0] com.netflix.discovery.DiscoveryClient : Single vip registry refresh property : nu
2019-04-21 10:19:29.126 INFO 13632 --- [freshExecutor-0] com.netflix.discovery.DiscoveryClient : Force full registry fetch : false
2019-04-21 10:19:29.130 INFO 13632 --- [freshExecutor-0] com.netflix.discovery.DiscoveryClient : Application is null : false
2019-04-21 10:19:29.130 INFO 13632 --- [freshExecutor-0] com.netflix.discovery.DiscoveryClient : Registered Applications size is zero : tr
2019-04-21 10:19:29.130 INFO 13632 --- [freshExecutor-0] com.netflix.discovery.DiscoveryClient : Application version is -1: false
2019-04-21 10:19:29.134 INFO 13632 --- [freshExecutor-0] com.netflix.discovery.DiscoveryClient : Getting all instance registry info from t
2019-04-21 10:19:29.559 INFO 13632 --- [freshExecutor-0] com.netflix.discovery.DiscoveryClient : The response status is 200
2019-04-21 10:19:59.597 INFO 13632 --- [freshExecutor-0] com.netflix.discovery.DiscoveryClient : Disable delta property : false
2019-04-21 10:19:59.597 INFO 13632 --- [freshExecutor-0] com.netflix.discovery.DiscoveryClient : Single vip registry refresh property : nu
2019-04-21 10:19:59.597 INFO 13632 --- [freshExecutor-0] com.netflix.discovery.DiscoveryClient : Force full registry fetch : false
2019-04-21 10:19:59.597 INFO 13632 --- [freshExecutor-0] com.netflix.discovery.DiscoveryClient : Application is null : false
2019-04-21 10:19:59.597 INFO 13632 --- [freshExecutor-0] com.netflix.discovery.DiscoveryClient : Registered Applications size is zero : tr
2019-04-21 10:19:59.597 INFO 13632 --- [freshExecutor-0] com.netflix.discovery.DiscoveryClient : Application version is -1: false
2019-04-21 10:19:59.597 INFO 13632 --- [freshExecutor-0] com.netflix.discovery.DiscoveryClient : Getting all instance registry info from t
2019-04-21 10:19:59.597 INFO 13632 --- [freshExecutor-0] com.netflix.discovery.DiscoveryClient : The response status is 200
2019-04-21 10:20:00.074 INFO 13632 --- [freshExecutor-0] com.netflix.discovery.DiscoveryClient : Initializing Spring DispatcherServlet 'd'
2019-04-21 10:20:42.409 INFO 13632 --- [nio-9800-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Servlet 'dispatcherServlet'
2019-04-21 10:20:42.409 INFO 13632 --- [nio-9800-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 120 ms
From service
From ballback
```

Activate Windows

**NOTE:**-- `@HystrixCommand(..)` used to provide fallback method details. This is used to indicate Hystrix is enable for this method. (Not for all methods in RestController).

=>FallBack Method name can be any one. But Return Type must be same as

## Spring Cloud Netflix Microservice Design:-



MQ = Message Queues

CS = Config Server

LBS = Load Balancing Server

ELK = Elastic Search –Logstash – Kibana

### 9. API PROXY / API GATEWAY:-

In one Application there will be multiple Microservices defined.

=>Every Microservice might be running in different servers (IP:PORT) and ports, even connected with load Balancing (Multiple IP:PORTs).

=>Microservices URLs are not directly given to client Applications (Mobile, Web, 3<sup>rd</sup> party) as there will be multiple addresses exist.

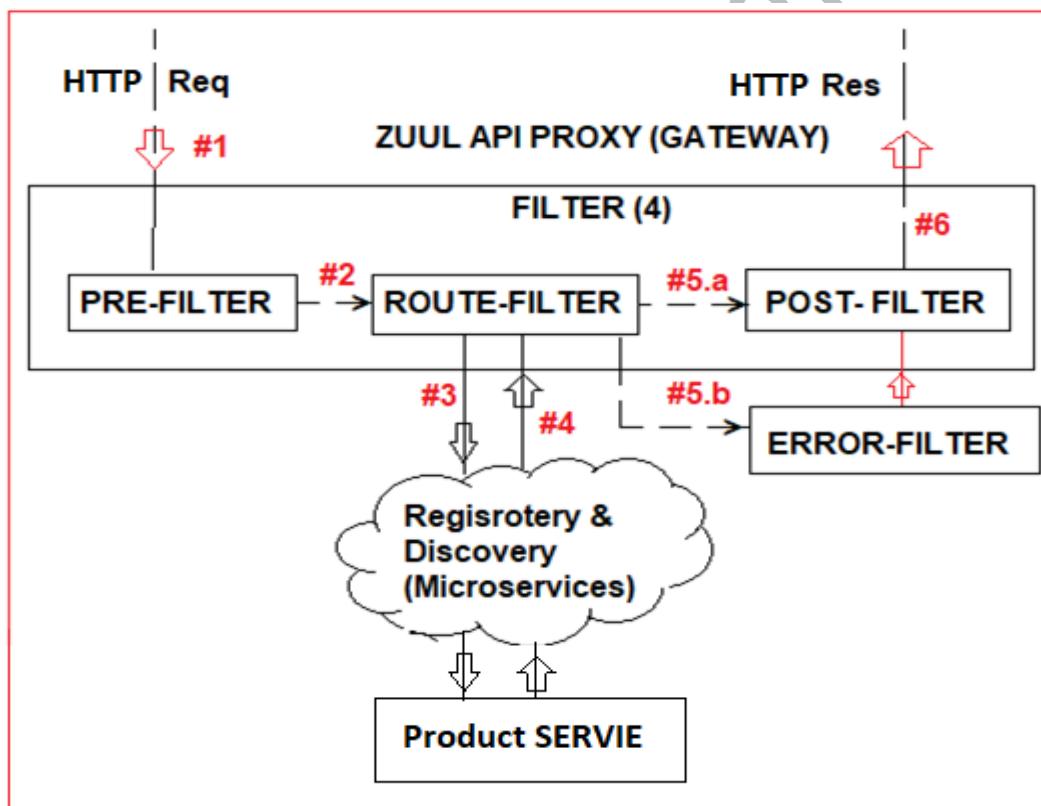
=>So, finally all microservices are grouped (masked) using Gateway which controls every request.

=>By using **unique PATH** (API), Gateway executes Microservice and given response back to client.

### \*\*\*Nature of Gateway:--

- >Single-Entry, one-Exist & one-URL.
  - >One time Authentication (SSO = Single Sing on) One Security system for all Microservices.
  - >**Dynamic Routing**: Supporting with Eureka for finding Microservices using Ribbon. (Find Execution path b/w multiple Microservices).
  - >Avoid Direct URL to client (Avoid CROS origin Request/response Format).
  - >Global data support (XML/JSON data Exchange).
  - >Supports Filtering.
- =>Spring Cloud Netflix ZUUL behaves as API PROXY for Microservices Application which supports “Integration with any type of client component” (Web, mobile, 3<sup>rd</sup> party, webservices... etc).

### Zuul (API PROXY) working flow:--



### Zuul API (PROXY-SERVER) GATEWAY:--

Zuul Server is a Netflix component used to configure “Routing for Microservices”. It is an API gateway provided by Netflix and integrated with Spring Cloud.

=>Netflix ZUUL supports dynamic routing, monitoring, Security and behaves as Consumer to all Microservices.

=>ZUUL is HTTP Client and server i.e. for UI/3<sup>rd</sup> party Clients it is a server for Microservices it is a client.

=>Zuul behaves as Consumer Application with R&D (Eureka) server to get all Microservices serviceIds with Instances.

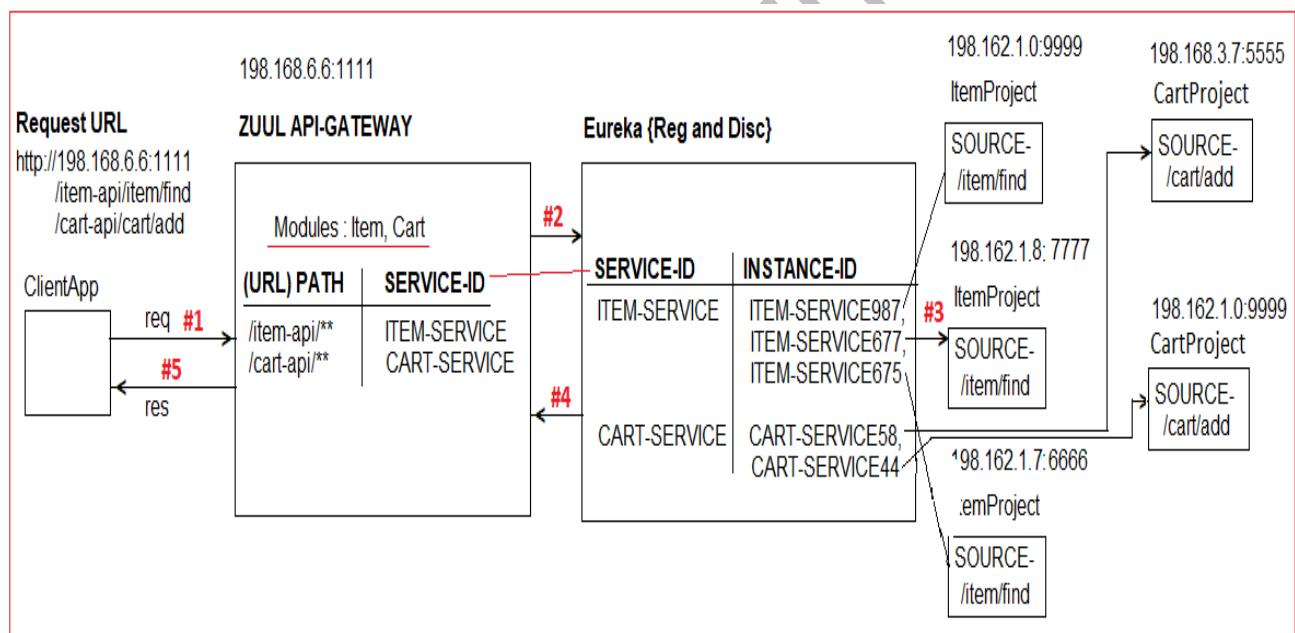
=>Using keys like :

```
zuul.routes.<module>.path=
zuul.routes.<module>.serviceId=
```

=>Before creating Zuul Server, we should have already created:

- a. Eureka Server Project
- b. Microservices (Ex: PRODUCT-SERVICE, CUSTOMER-SERVICE, STUDENT-SERVICE...) (with load balance implementation)

### ZUUL EUREKA-MICROSERVICE WORK FLOW:--



=>Client make HTTP request to ZUUL server (ZUUL IP: PORT)

=>Client URL should have api (route) of Microservice provided by ZUUL. Based on this route (api) ZUUL reads “SERVICE-ID”.

=>Now, zuul behaves as consumer app to R&D fetch register and compare the service-ID, using request made by client.

=>ZUUL uses R&Ds LBSR (Load Balancing Service Register) to choose one instance-Id (URI) of Microservice based on load factor.

=>Now zuul makes Http call to RestController using class level and method level path given by client.

=>Response of RestController (global data) is given back to client (done by zuul).

- >Here, Zuul Server behaves as Entry and Exit Point.
- >It hides all details like Eureka Server and Service Instances from Client,
- >Zuul Provides only **ZUUL-URL** and Paths of Services only.
- >Zuul takes care of Client (request) Load balancing even.
- >Provides Routing based on API (PATH/URL).
- >Zuul must be registered with Eureka Server.

=>In Zuul Server Project, we should provide module details like path, service-Id using application.properties (or .yml)

=>If two modules are provided in ZUUL then, only module name gets changed in keys. Consider below example:

MODULE	PATH	SERVICE-ID
Product	/prod-api/**	PROD-SERVICE
Student	/std-api/**	STD-SERVICE

#### application.properties:--

```
zuul.routes.product.path=/item-api/*
zuul.routes.product.service-id=ITEM-SERVICE
zuul.routes.student.path=/std-api/*
zuul.routes.student.service-id=STD-SERVICE
```

#### application.yml:--

```
zuul:
  routes:
    product:
      path: /item-api/*
      service-id: ITEM-SERVICE
    student:
      path: /std-api/*
      service-id: STD-SERVICE
```

**ZUUL Proxy:--** Netflix Zuul Server behaves as,

a. **>Server:--** When end client component made HTTP request to Zuul.

**b.>Consumer:**-- To identify one service Instance based on Service-Id and communicate using feign + Ribbon (Code generated at runtime based on ServiceId chosen).

=>In simple Zuul is “Server+Consumer”.

\*\*\*To behaves as consumer, Zuul must get registered with Eureka.

### Working with ZUUL Filters:--

=>Filters are used to validate request and construct valid response. In simple we can also call as “**PRE-POST**” Processing Logic.

=>ZUUL Filter provides even extra types like **ROUTE FILTERS** and **ERROR FILTERS**.

=>ZUUL supports 4 types of filters implementation for Gateway service. Those are

**1.>PRE-FILTER:**- Work on request head body.

**2.>POST-FILTER:**- Work on response head/body.

**3.>ROUTE-FILTER:**- Work on Service Instance object (What is URI, host/port, SSL valid or not)

**4.>ERROR FILTER:**- Executed only if exception is raised (Write exception to log, email...).

=>In general our code (logic) is called as process. To execute extra code before our logic and filter our logic, before choosing logic instance after throwing exception logic filters are used

->When client made request to ZUUL then Pre Filter gets called automatically.

->After validating, request is dispatched to Route Filter.

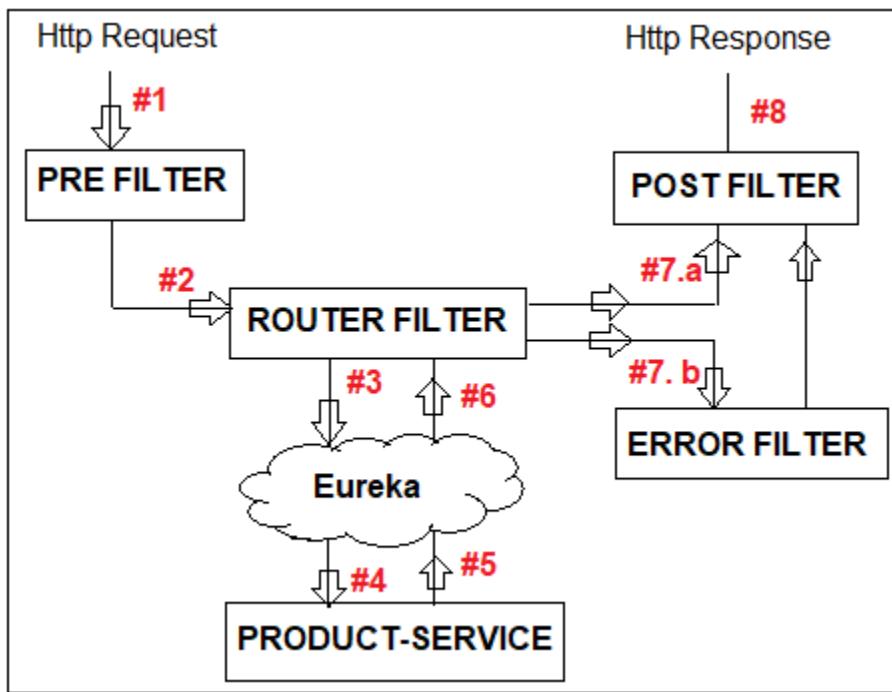
->Route Filter is like 2<sup>nd</sup> level validation at Required SERVICE level.

->Route Filter will request to one Microservice based on Service-Id.

->If microservice is not executed properly (i.e throwing exception) then Error Filter is called.

->Finally Post Filter works on Http Response (adds Headers, Encode data, Provide info to client... etc) in case of either success or failure.

Diagram:::



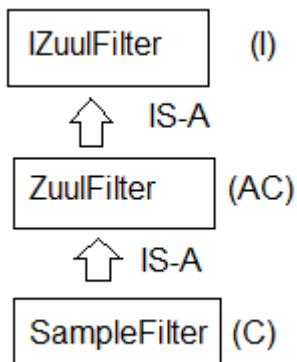
=>Here, Filter is a class must extends one abstract class “ZuulFilter” provided by Netflix API.

=>We can define multiple filters in one Application. Writing Filters are **optional**.

=>While creating filter class we must provide Filter **Order** (0, 1, 2, 3...) and Filter Type (“pre”, “route”, “error”, “post”)

=>Two filters of same type can have same Order which indicates any Execution order is valid.

=>We can enable and disable Filters using its flags (true/false).



=>To indicate Filter Types, we use its equal constants (public static final String variables), provided as.

TYPE	CONSTANT
Pre	PRE_TYPE
Route	ROUTE_TYPE
Error	ERROR_TYPE
Post	POST_TYPE

=>All above Constants are defined in FilterConstants (C) as global variables (public static final String).

=>Write one class in ZUUL Server and extends ZuulFilter Abstract class, override below method (4) in your filter class.

**a>shouldFilter():--** Must be set to 'true' If value is set to false then filter will not be executed.

**b>run():--** Contains Filter logic Executed once when filter is called.

**c>filterType():--** Provide Filter Constant Must be one Type (pre, post, route, error).

**d>filterOrder():--** Provide order for Filter Any int type number like 0, 56, 78.

Ex:- \*\*\*Create below class under ZuulServer Application.

```
package com.app.filter;
```

```
@Component
```

```
public class FilterType extends ZuulFilter {
```

```
    /**To enable or Disable current filter Enable(true) or Disable Filter(false)**/
```

```
    public boolean shouldFilter() {
```

```
        return true;
```

```
}
```

```
    /**Define Filter Logic Here**/
```

```
    public Object run() throws ZuulException {
```

```
        System.out.println("FROM POST FILTER");
```

```
        return null;
```

```
}
```

```
    /**Specify Filter Type (Pre, Post...)**/
```

```
    public String filterType() {
```

```
        return FilterConstants.POST_TYPE;
```

```
}
```

```
    /**Provider Filter Order for Execution if same type of filters are used**/
```

```
    public int filterOrder() {
```

```
        return 0;
```

```
}
```

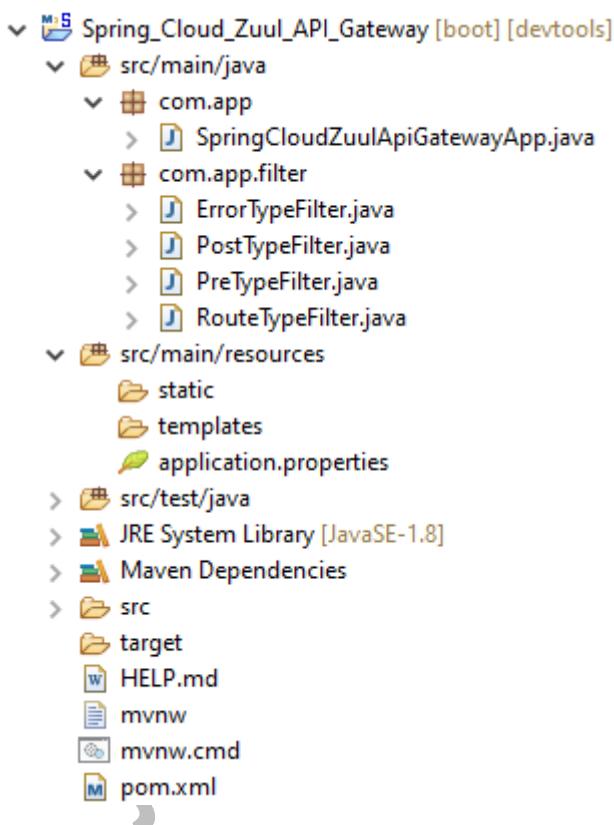
=>FilterConstants is a class having static final variables like PRE\_TYPE, POST\_TYPE, ROUTE\_TYPE, and ERROR\_TYPE (**public static final String PRE\_TYPE="pre"**).  
=>FilterConstants even provide global standard constant details HTTP PORT =80, HTTPS PORT:443.

**Step#1:-** Create Spring starter Project as : **ZUUL SERVER** with dependencies : Web, Zuul, Eureka Discovery.

### Zuul Dependency:--

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-zuul</artifactId>
</dependency>
```

### #15. Folder Structure of Zuul Server (Gateway Server):--



### pom.xml:--

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.1.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.app</groupId>
    <artifactId>Spring_Cloud_Zuul_API_Gateway</artifactId>
    <version>1.0</version>
    <name>Spring_Cloud_Zuul_API_Gateway</name>
    <description>Demo project for Zuul API Gateway</description>

    <properties>
        <java.version>1.8</java.version>
        <spring-cloud.version>Greenwich.SR1</spring-cloud.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-zuul</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
```

```

<scope>runtime</scope>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
</dependencies>
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>${spring-cloud.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
</project>

```

**Step#2:-** application.properties should have below details like:

```

#zuul.routes.<module>.path=/<module>-api/**
#Service Id name must be Eureka Server Service Id name
#zuul.routes.<module>.service-id=[SERVICE-ID]

```

server.port=8558

eureka.client.service-url.default-zone=http://localhost:8761/eureka

```
spring.application.name=ZUUL-PROXY
#Each Microservice url and Service Id configuration
zuul.routes.item.path=/item-api/**
zuul.routes.item.service-id=ITEM-SERVICE
zuul.routes.cart.path=/cart-api/**
zuul.routes.cart.service-id=CART-PROVIDER
```

**Step#3:-** In Zuul Server Project, at starter class level add annotation :

**@EnableZuulProxy.**

```
package com.app;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
import org.springframework.cloud.netflix.zuul.EnableZuulProxy;

@SpringBootApplication
@EnableDiscoveryClient
@EnableZuulProxy
public class SpringCloudZuulApiGatewayApp {
    public static void main(String[] args) {
        SpringApplication.run(SpringCloudZuulApiGatewayApp.class, args);
        System.out.println("Zuul Api Gateway Server is executed");
    }
}
```

**Step#4:- Implement Filters like :**

PRE, ROUTE, POST, ERROR

using one abstract class : ZuulFilter (AC) and use FilterConstants (C).

**#1. PreTypeFilter.java:--**

```
package com.app.filter;
import org.springframework.cloud.netflix.zuul.filters.support.FilterConstants;
import org.springframework.stereotype.Component;
import com.netflix.zuul.ZuulFilter;
import com.netflix.zuul.exception.ZuulException;
```

```
@Component
public class PreTypeFilter extends ZuulFilter{
    /**Enable(true) or Disable Filter(false)*/
    public boolean shouldFilter() {
        return true;
    }
    /**Define Filter Logic Here*/
    public Object run() throws ZuulException {
        System.out.println("FROM PRE FILTER");
        return null;
    }
    /**Specify Filter Type*/
    public String filterType() {
        return FilterConstants.PRE_TYPE;
    }
    /**Provider Filter Order for Execution*/
    public int filterOrder() {
        return 0;
    }
}
```

**#2. RouteTypeFilter.java:--**

```
package com.app.filter;
import org.springframework.cloud.netflix.zuul.filters.support.FilterConstants;
import org.springframework.stereotype.Component;
import com.netflix.zuul.ZuulFilter;
import com.netflix.zuul.exception.ZuulException;
```

```
@Component
public class RouteTypeFilter extends ZuulFilter {
    /**Enable(true) or Disable Filter(false)*/
    public boolean shouldFilter() {
        return true;
    }
    /**Define Filter Logic Here*/
    public Object run() throws ZuulException {
```

```
        System.out.println("FROM ROUTE FILTER");
        return null;
    }
    /**Specify Filter Type*/
    public String filterType() {
        return FilterConstants.ROUTE_TYPE;
    }
    /**Provider Filter Order for Execution*/
    public int filterOrder() {
        return 0;
    }
}
```

### #3. ErrorTypeFilter.java:--

```
package com.app.filter;
import org.springframework.cloud.netflix.zuul.filters.support.FilterConstants;
import org.springframework.stereotype.Component;
import com.netflix.zuul.ZuulFilter;
import com.netflix.zuul.exception.ZuulException;

@Component
public class ErrorTypeFilter extends ZuulFilter{

    /**Enable(true) or Disable Filter(false)*/
    public boolean shouldFilter() {
        return true;
    }
    /**Define Filter Logic Here*/
    public Object run() throws ZuulException {
        System.out.println("FROM ERROR FILTER");
        return null;
    }
    /**Specify Filter Type*/
    public String filterType() {
        return FilterConstants.ERROR_TYPE;
    }
    /**Provider Filter Order for Execution*/
}
```

```
public int filterOrder() {  
    return 0;  
}  
}
```

**#4. PostTypeFilter.java:--**

```
package com.app.filter;  
import org.springframework.cloud.netflix.zuul.filters.support.FilterConstants;  
import org.springframework.stereotype.Component;  
import com.netflix.zuul.ZuulFilter;  
import com.netflix.zuul.exception.ZuulException;  
  
@Component  
public class PostTypeFilter extends ZuulFilter{  
  
    /**Enable(true) or Disable Filter(false)**/  
    public boolean shouldFilter() {  
        return true;  
    }  
    /**Define Filter Logic Here**/  
    public Object run() throws ZuulException {  
        System.out.println("FROM POST FILTER");  
        return null;  
    }  
    /**Specify Filter Type**/  
    public String filterType() {  
        return FilterConstants.POST_TYPE;  
    }  
    /**Provider Filter Order for Execution**/  
    public int filterOrder() {  
        return 0;  
    }  
}
```

**Execution Process:--**

->http://desktop-ch8lpuh:8558/cart-api/cart/info

[Raghu Sir]

[NareshIT, Hyd]

-><http://desktop-ch8lpuh:8558/cart-api/cart/list>  
-><http://desktop-ch8lpuh:8558/item-api/item/info>



## **Console SCREEN of Wateway:-**

## 10. Spring Boot Message Queue (MQ):--

In case of real-time application large data needs to be transferred and processes.

Like Google Server to Amazon, Facebook, etc...

=>Data (Message) Transfer can be done using Message Queues.

**Message Queues are used for:-**

- 1.>Data Streaming
  - 2.>Web Activities
  - 3.>Log Aggregation
  - 4.>Command Oriented Components.

**a.>Data Streaming:**-- Sending continuous data between two applications is called data streaming. In real-time large data from files, networks, and databases etc...

Ex:-GoogleMap updates for swiggy IRCTC Train updates where is my Train, Flights enquiry and booking status.

**b.>Web Activities:**-- Sharing search information to partner applications (Know what users are searching and provide related links and ADS in client websites).

Ex:- Google search data given to Cricbuzz, Amazon.

**c.>Log Aggregation:**-- Sending log files data from current server to other servers to find Exception and warning details.

**d.>Command Oriented Components:**-- It is a trigger based service connected to multiple components works based on command.

Ex:- If User books an Order from Amazon then send a message to his mobile (Mobile service component), send an email to user (email service component), send invoice to printer (Remote printer service component).

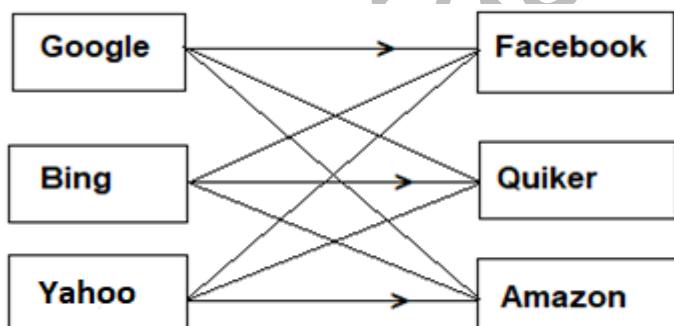
=>Above components are executed on command save.

#### **Distributed Messaging (MQ):--**

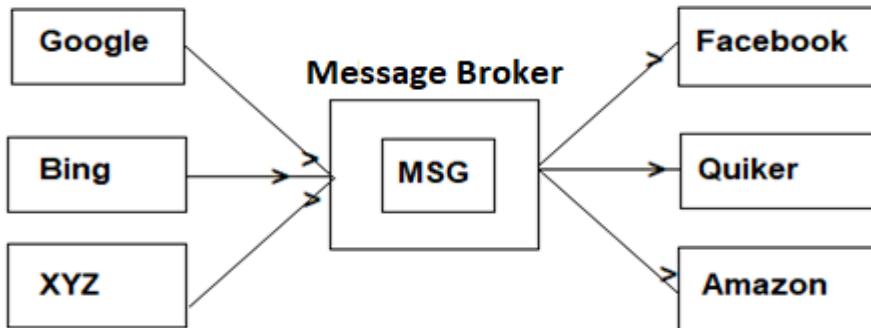
=>It is a process of Sending message from Source Application to one or more Destination application.

=>It is not a concept of request and response. It is implemented using MQ.

=>In case of multiple clients share data without MQ, files look like this:



**A design with Message Queue (MQ):--**This is simplified by MQ which is given below with one mediator also called as Message Broker.



=>MQ can be implemented using Language based Technologies (or API).

Ex:-- JMS (Java Message Service)

=>Global MQ (between any type of client) Advanced Message Queuing protocol (AMQP)

=>Apache ActiveMQ, Rabbit MQ, Apache Atrims are different Service providers (Broker software's) for JMS.

=>Apache KAFKA is a service Provider (Broker software) for AMQP).

This are Two Types of MQ:

- a.>Basic Message Queuing Protocol (BMQP).
- b.>Advanced Message Queuing Protocol (AMQP).

**a>Basic Message Queuing Protocol(BMQP):--** It is also called as language specific MQ. In case of Java it is called as JMS (Java message Service). It is language dependent.

#### Java Message Server:--

It is used to implement Basic Messaging process between application to send receive data. It uses one Broker Software i.e.e called as MOM (message Oriented Middleware).

=MOM contains special type memory (Container) called as Destination that holds Messages.

=>Every application connected to MOM is called as **Client**.

=>There are two types of client Consumer and Producer.

#### Spring Boot with Apache ActiveMQ:--

Java JMS is simplified using Spring Boot which reduces writing basic configuration for ConnectionFactory, Connection, Session, Destination Creation, Send/Receive Message etc...

JMS supports 2 types of client. Those are.

- a.>Producer (Client) : Sends Message to MOM
- b.>Consumer (Client) : Read Message from MOM

=>Messages are exchanged using **Message Broker** called as **MOM** (Message Oriented Middleware). Apache ActiveMQ is MOM Software.

### Types of Communications in JMS:--

a.>**P2P (Peer-To-Peer)** : -- Sending 1 message to one consumer. In this case destination type is called as “QUEUE”.

b.>Pub/Sub (Publish and Subscribe):-- Sending 1 message (same copy) to Multiple consumers. Here, Destination type is “TOPIC”.

\*\*\* JMS supports two types of Communications.

### NOTE:--

a.>Destination is a Special memory created in MOM which holds messages.

b.>Here **Queue** Destination is used for P2P. **Topic** Destination is used for Pub/Sub.

### Limitation of JMS:--

a.>Used between Java Applications.

b.>Message (Data) size should be smaller.

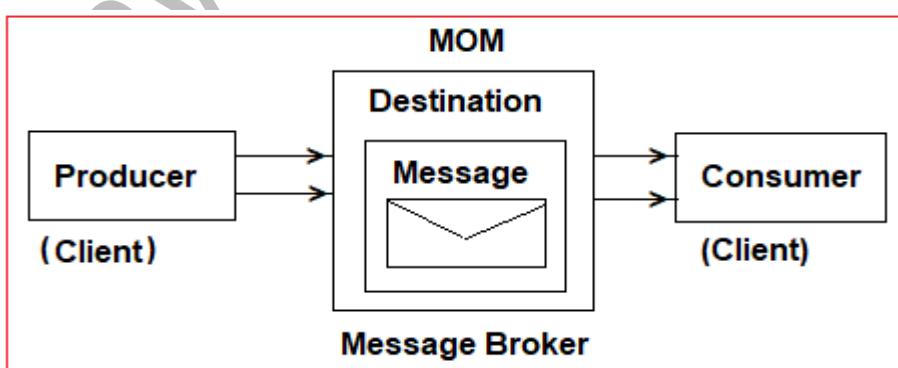
c.>Only one MOM (one instance) runs at a time.

d.>Large data takes lot of time to process.

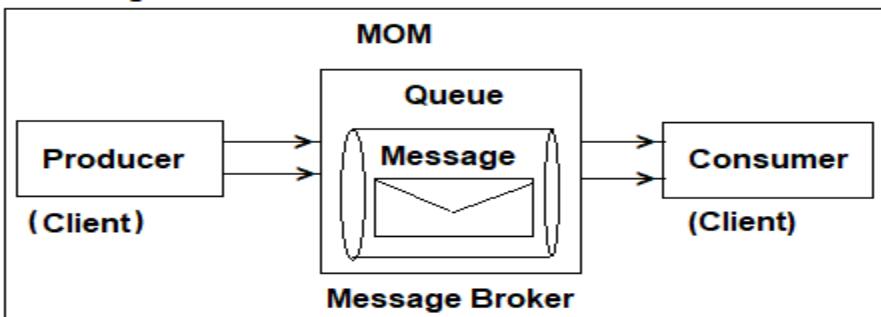
e.>If Pub/Sub model is implemented with more consumers then process will be very slow.

f.>Data may not be delivered (data lose) in case of MOM Stops or Restart.

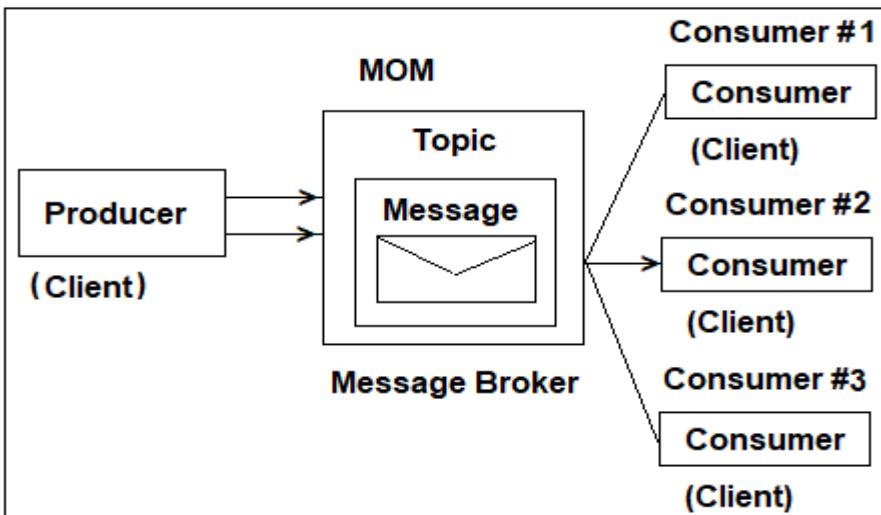
### Generic Design of JMS:--



### 1. Peer to Peer : P2P (Queue):--



## 2. Pub/Sub : Publish and Subscribe (Topic):--



### Steps to Implement ActiveMQ:-

**Step#1:-** Create one Spring boot starter application (Client Application) using dependencies: ActiveMQ (JMS) (or add below dependency in pom.xml)

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

### pom.xml:-

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
```

```
<modelVersion>4.0.0</modelVersion>
```

```
<parent>
```

```
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-parent</artifactId>
<version>2.1.1.RELEASE</version>
<relativePath /> <!-- lookup parent from repository -->
</parent>
<groupId>com.app</groupId>
<artifactId>Spring-Cloud-JMS-ActiveMQ</artifactId>
<version>1.0</version>
<name>Spring-Cloud-JMS-ActiveMQ</name>
<description>Demo project for Spring JMS Implementation</description>
<properties>
    <java.version>1.8</java.version>
</properties>
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-activemq</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
<build>
    <plugins>
        <plugin>
```

```

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

```

**Step#2:-** In application.properties file provide common keys like MQ brokerUrl, User, Password in all Producer and Consumer Application.

->If we do not specify any type then it behaves as **P2P (Queue)**. To make it as Pub/Sub (Topic).

**application.properties**-- Add same properties in Producer & Consumer application.

spring.activemq.broker-url=tcp://localhost:61616

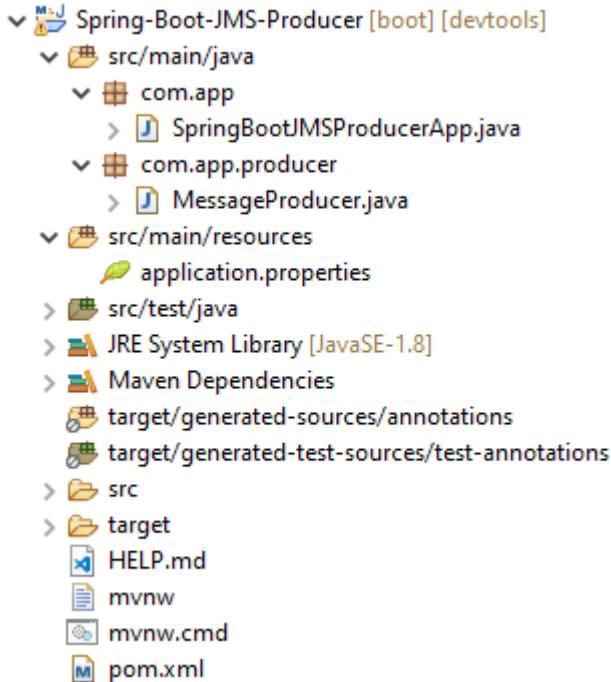
spring.activemq.user=admin

spring.activemq.password=admin

spring.jms.pub-sub-domain=false

**Step#3:-** If application is Producer type then use JmsTemplate object and call send() which will send message to MoM.

## #16. Folder Structure of JMS Producer:--



### #1. Starter class for Producer (SpringBootActiveMqProducerApp.java):--

```
package com.app;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringBootJMSProducerApp {
    public static void main(String[] args)
    {
        SpringApplication.run(SpringBootJMSProducerApp.class, args);
        System.out.println("JMS Producer Executed :");
    }
}
```

**#2 MessageProducer.java:--**

```
package com.app.producer;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.jms.core.JmsTemplate;
import org.springframework.stereotype.Component;

@Component
public class MessageProducer implements CommandLineRunner {

    @Autowired
    private JmsTemplate template;

    @Override
    public void run(String... args) throws Exception {
        template.send("my-tpca", (ses) -> ses.createTextMessage("AAAAAAA"));
        System.out.println("sent from Producer");
    }
}
```

**Step#4:-** If Application is Consumer type then define one Listener class using destination.

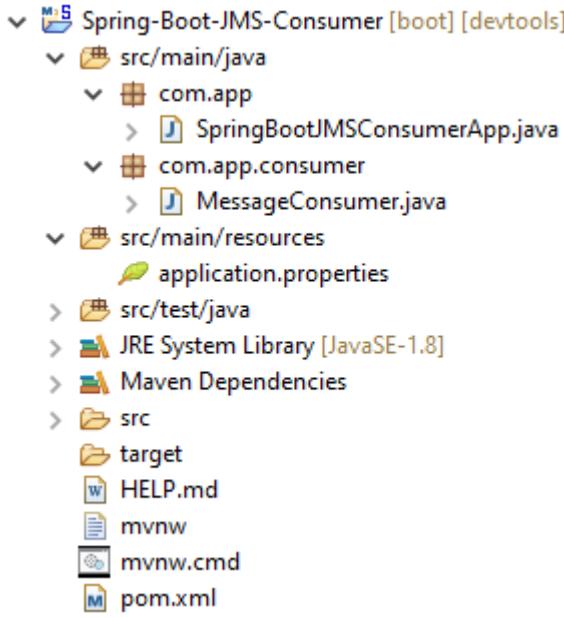
Use code : `@JmsListener(destination="-----")`

=>It must be Enabled using code: `@EnableJms`

=>In case of JmsTemplate (C) **@EnableJms** is not required.

**NOTE:**-- Create multiple consumer app in same or different workspaces and set.  
 spring.jms.pub-sub-domain=true  
 In application.properties file (or .yml).

### #16.a Folder Structure of JMS Consumer1 Application:--



### #1. Starter class (SpringBootJMSConsumer.java):--

```

package com.app;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringBootJMSConsumerApp {

    public static void main(String[] args) {
        SpringApplication.run(SpringBootJMSConsumerApp.class, args);
        System.out.println("JMS Consumer Executed :");
    }
}
  
```

### #2. Consumer class(MessageConsumer.java):--

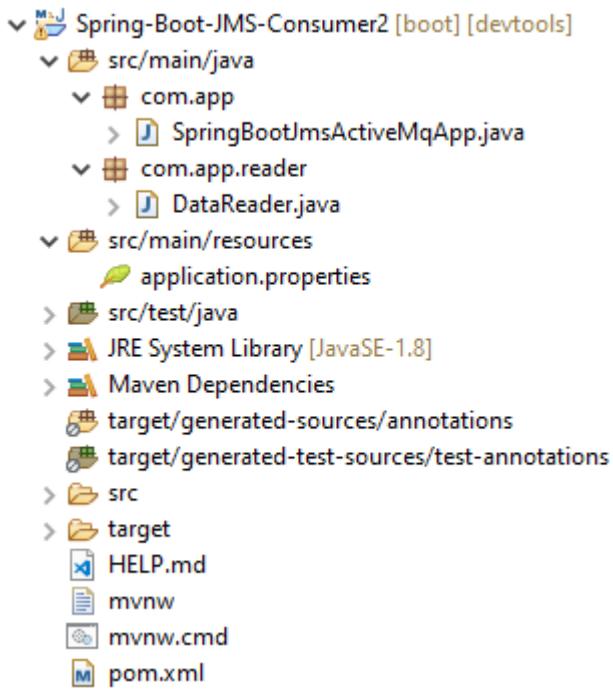
```

package com.app.consumer;
import org.springframework.jms.annotation.EnableJms;
  
```

```
import org.springframework.jms.annotation.JmsListener;
import org.springframework.stereotype.Component;
```

```
@EnableJms //optional in case of @JmsListener
@Component
public class MessageConsumer {
    @JmsListener(destination = "my-tpca")
    public void readMessage(String msg)
    {
        System.out.println("from consumer");
        System.out.println("msg is:" +msg);
    }
}
```

### #16.b Folder structure of JMS Consumer2:-



### #2:- Starter class (SpringBootJmsActiveMQApp.java):-

```
package com.app;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class SpringBootJmsActiveMqApp {
```

```
public static void main(String[] args) {  
    SpringApplication.run(SpringBootJmsActiveMqApp.class, args);  
    System.out.println("JMS Executed :");  
}  
}
```

**#2:- Reader class (DataReader.java):--**

```
package com.app.reader;  
import org.springframework.jms.annotation.EnableJms;  
import org.springframework.jms.annotation.JmsListener;  
import org.springframework.stereotype.Component;  
  
@EnableJms //optional in case of @JmsListener  
@Component  
public class DataReader {  
  
    @JmsListener(destination="my-tpca")  
    public void getMsg(String msg) {  
        System.out.println("consumer#222");  
        System.out.println("Message is:=>" +msg);  
    }  
}
```

**NOTE:--** send (String destinationName, Message (Creator MSG));  
=>This method is used to send message from producer Application to MOM.  
=>if destination not exist in MOM, Creates new one else uses same.

**Execution Order:--**

- 1.>Active MQ Server.
- 2.>Producer Application Starter class (One Provider App).
- 3.>Consumer Application Starter class (One or Multiple Consumer Application).

**Download and Setup for ActiveMQ:--**

=>Get into dependency and click version

Example: <activemq.version>5.15</activemq> download Activemq same version.

Download Link:-- Go to below location

<https://activemq.apache.org/components/classic/download/>

->Click on : apache-activemq-5.15.9-bin.zip

->Extract to one folder

->Goto .. F:\Study Software\JMS ApacheMQ\apache-activemq-5.15.9\bin\win64  
(\apache-activemq-5.15.9\bin\win64)

### Execution process:-

#### Step#1:- Double click on activemq.bat file

```

ActiveMQ
wrapper | --> Wrapper Started as Console
wrapper | Launching a JVM...
jvm 1 | Wrapper (Version 3.2.3) http://wrapper.tanukisoftware.org
jvm 1 | Copyright 1999-2006 Tanuki Software, Inc. All Rights Reserved.
jvm 1 |
jvm 1 | Java Runtime: Oracle Corporation 1.8.0_171 C:\Program Files\Java\jre1.8.0_171
jvm 1 |   Heap sizes: current=125952k free=116632k max=932352k
jvm 1 |   JVM args: -Dactivemq.home=... -Dactivemq.base=... -Djavax.net.ssl.keyStorePassword=password -Djavax.net.ssl.trustStorePassword=password -Djavax.net.ssl.keyStore=../conf/broker.ks -Djavax.net.ssl.trustStore=../conf/broker.ts -Dcom.sun.management.jmxremote -Dorg.apache.activemq.UseDedicatedTaskRunner=true -Djava.util.logging.config.file=logging.properties -Dactivemq.conf=../conf -Dactivemq.data=../data -Djava.security.auth.login.config=../conf/login.config -Xmx1024m -Djava.library.path=../bin/win64 -Dwrapper.key=P6ZsAKBzNItqPV_D -Dwrapper.port=32001 -Dwrapper.jvm.port.min=31000 -Dwrapper.jvm.port.max=31999 -Dwrapper.pid=508 -Dwrapper.version=3.2.3 -Dwrapper.native_library=wrapper -Dwrapper.cpu.timeout=10 -Dwrapper.jvmid=1
jvm 1 | Extensions classpath:
jvm 1 | [..]\lib\camel\..\lib\optional\..\lib\web\..\lib\extra
jvm 1 | ACTIVEMQ_HOME: ...
jvm 1 | ACTIVEMQ_BASE: ...
jvm 1 | ACTIVEMQ_CONF: ...
jvm 1 | ACTIVEMQ_DATA: ...
jvm 1 | Loading message broker from: xbean:activemq.xml
jvm 1 | INFO | Refreshing org.apache.activemq.xbean.XBeanBrokerFactory$1@5313a462: startup date [Sun Jun 16 00:06:45 TST 2019]; root of context hierarchy
jvm 1 | INFO | Using Persistence Adapter: KahaDBPersistenceAdapter[F:\Study Software\JMS ApacheMQ\apache-activemq-5.15.9\bin\win64..\data\kahadb]
jvm 1 | INFO | Database ..\data\kahadb\lock is locked by another server. This broker is now in slave mode waiting a lock to be acquired

```

#### Step#2:- Goto browser and type URL: <http://localhost:8161/admin>



Home | Queues | Topics | Subscribers | Connections | Network | Scheduled | Send

Support

#### Welcome!

Welcome to the Apache ActiveMQ Console of **localhost** (ID:DESKTOP-CH8LPUH-49743-1560194454645-0:1)

You can find more information about Apache ActiveMQ on the [Apache ActiveMQ Site](#)

#### Broker

Name	localhost
Version	5.15.9
ID	ID:DESKTOP-CH8LPUH-49743-1560194454645-0:1
Uptime	4 days 23 hours
Store percent used	0

#### Queue Views

- Graph
- XML

#### Topic Views

- XML

#### Subscribers Views

- XML

#### Useful Links

- Documentation
- FAQ
- Downloads

Step#3:- Click on Queues (P2P)/TOPIC (Pub-Sub) Menu option to see message list.

The screenshot shows the Apache ActiveMQ Admin interface. At the top, there's a navigation bar with links to Home, Queues, Topics, Subscribers, Connections, Network, Scheduled, and Send. Below the navigation is a search bar for Queue Name and a 'Create' button. To the right is the Apache Software Foundation logo. On the left, under 'Queues:', there's a table with one row for 'my-tpca'. The columns are: Name, Number Of Pending Messages, Number Of Consumers, Messages Enqueued, Messages Dequeued, Views, and Operations. The 'Operations' column contains links for 'Browse Active Consumers', 'Active Producers', 'Send', 'Purge', and 'Delete'. On the right side of the interface, there are three sections: 'Queue Views' (Graph, XML), 'Topic Views' (XML), and 'Subscribers Views' (XML).

=>In case of Interface is provided with abstract method then we need to provide implementation using.

- 1>Child class
- 2>Anonymous inner class
- 3>Lambda Expression

=>Example#1:--

### #1. Interface:--

```
public interface MessageCreator {
    Message creatMessage(Session s) throws JMSException;
}
```

### #2.a. Implement class and Create Object:--

```
public Test implements MessageCreator {
    public Message createMessage(Session s) throws JMS Exception {
        return s.createMessage("Hello...")
    }
}
```

### #2.b Anonymous inner class:--

Syntax:--

```
new interfaceName () {
    //Override all methods
}
```

Code:--

```
new MessageCreator () {
    public Message createMessage(Session s) throws JmsException {
        return s.createMessage("hello...");
    }
}
```

**#2.c Lambda Expression:--**

```
messageCreator mc = (s) -> {  
    return s.createTestMessage("Hello....");  
}
```

Or

```
Message.create mc = s ->s.createMessage("Hello...");
```

**11. Spring Boot Apache Kafka Integration:--** Apache kafka is used for

=>Apache kafka supports AMQP for large data transfer for MQ applications over network.

=>Supports Real-time Data Streaming. It means read continuous and large data from external source like Float Files, Database, networks, etc...

=>It supports data reading/writing from

- a. Databases
- b. Flat File System
- c. Applications (Data Streams).

=>Kafka supports integration with any type of application (language Independent + Plugin required, default JAVA).

=>Kafka supports basic concepts of MQ like

- 1. Data Streaming
- 2. Web Activities
- 3. Log-Aggregations
- 4. Command-Components

=>Kafka follows below concept even,

a.>Kafka uses Message Broker also called as **Broker Server** which supports MQ operations.

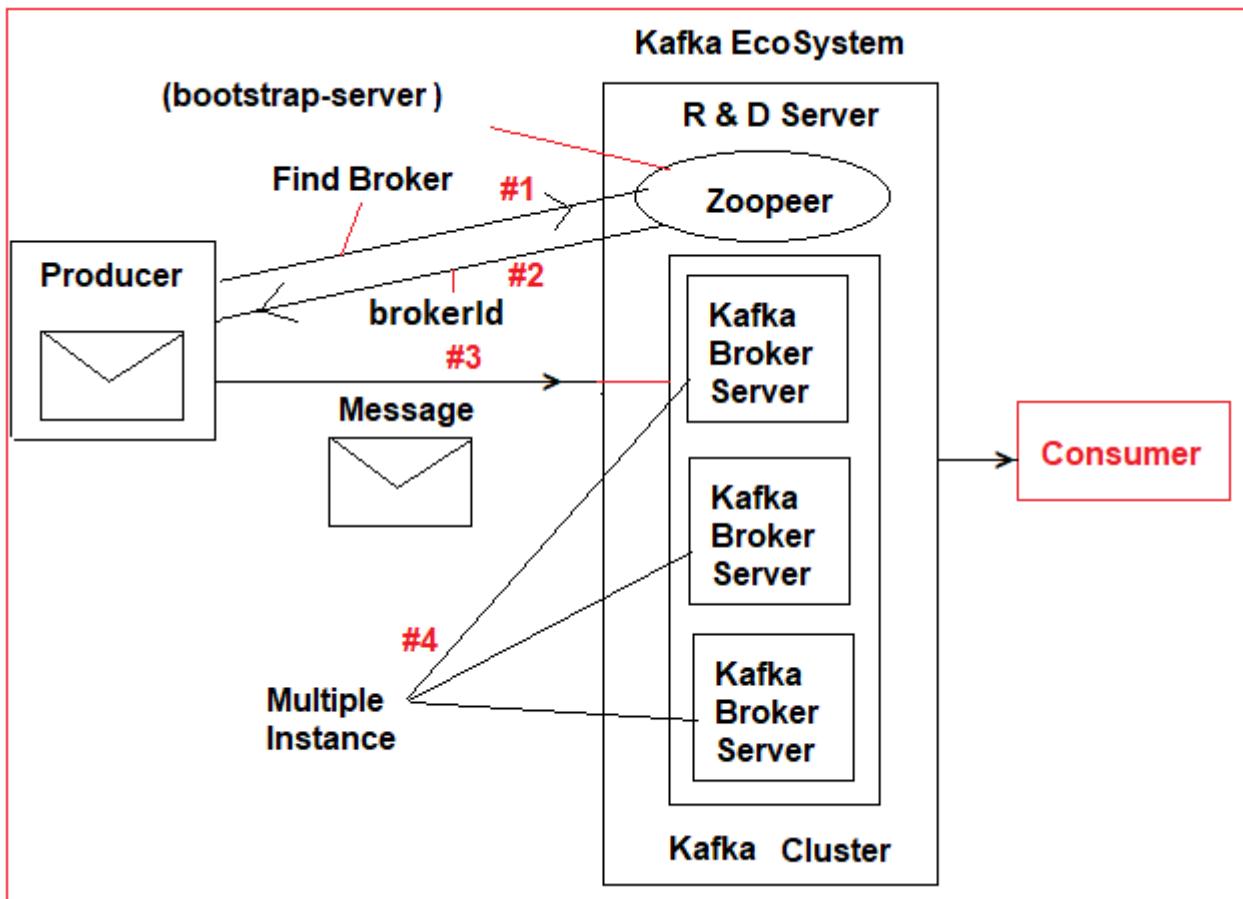
b.>Kafka supports **Load Balancing** for Broker Software to avoid more traffic, i.e. called as **Kafka cluster**. In general cluster is a group of Broker servers (1 to n).

c.>Kafka supports only **Topics** (Pub/Sub Model) and it is only default also.

d.>Kafka Cluster Auto-Scaling is done by **Bootstrap server** (AKA Zookeeper). It behaves as R&D Server.

- e.>Data is sent to Topic in  $\langle K, V \rangle$  format. K=TopicName (Destination), V=Data.
- f.>All these Broker instances must be Registered with Registry and Discovery (R&D) Server. Kafka comes with default “**Zookeeper R&D Server**”.
- g.> This complete Kafka Software is called as **Kafka EcoSystem** (Kafka Eco-System = Kafka Cluster + Bootstrap Server).
- h.> Data Partitions concept is used by kafka to send large data.
- i.>Message Brokers will persist the message (save into their memory) to avoid data lose in case of consumer non-available or broker is down.
- j.>Kafka works as Protocol independent i.e. works for TCP, FTP, SMTP, HTTP... etc)

### Kafka EcoSystem Diagram:--



### Execution Flow:--

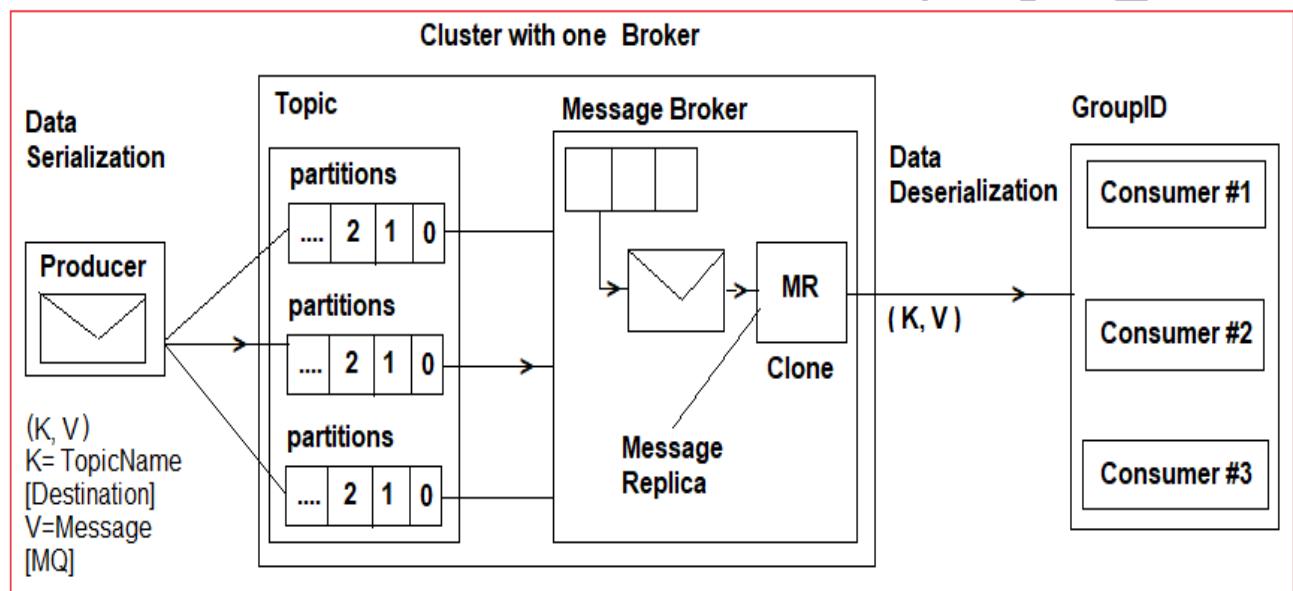
- =>Producer Application should get Message Broker details from R & D Server (zookeeper) known as bootstrap-server).
- =>Producer gets unique-id (InstanceId) of Message Broker server and sends message to Broker. Producer use **KafkaTemplate<K, V>** to send data to one Broker server.
- =>Message Broker will send this message to one or multiple consumers.

=>Producer sends data  $\langle k, V \rangle$  format in Serialized (Converting to binary/Characters formats). Here K=Destination (Topic name) and V= Message.

=>Every Message will be partitioned into Multiple parts in Topic (Destination) to avoid large data sending, by making into small and equal parts (some time size may vary).

=>Broker reads all partitions data and creates its replica (Clone/Mirror obj) to send message to multiple consumers based on **Topic** and **Group-Id**.

=>At Consumer side Deserialization must be applied on K, V to read data. Consumer should also be linked with bootstrap-server to know its broker.



=>Partitions are used to breakdown large message into multiple parts and send same to multiple brokers to make data destination in parallel.

**Message Replica:**-- It creates multiple copies to one message to publish one message to multiple Consumers.

#### Kafka Producer and Consumer Setup Details:--

=>For Producer Application we should provide details in application.properties (or .yml).

=>Those are

bootstrap-servers=localhost:9092

key-serializer=StringSerializer

value-serializer=StringSerializer

=>By using this Spring Boot creates instance of “KafkaTemplate<K, V>” then we can call send(k, v) method which will send data to Consumer.

->Here : K=Topic Name, V= Data/Message

=>For Consumer Application we should provide details in application.properties (or .yml)

=>Those are

bootstrap-servers=localhost:9092

key-deserializer=StringDeserializer

value-deserializer=StringDeserializer

group-id=MyGroupId

=>By using this Spring Boot configures the Consumer application, which must be implemented using : @KafkaListener(topics="—", groupId="—")

### **Kafka Download and Setup:--**

Apache Kafka comes with Unix based software called as Scala 2.x

=>Download link (<https://kafka.apache.org/downloads>)

=>Choose one mirror=>Extract once (.tgz -> .tar)

=>Extract one more time (.tar -> folder)

=>Open folder and create --.bat file for bootstrap Server (Zookeeper).

#### **\*\*\* bat files in kafka to be created\*\*\***

1>Server.bat

=>Starts Kafka Server (Message Broker)

.\bin\windows\zookeeper-server-start.bat .\config\zookeeper.properties

2>cluster.bat

=>Starts Zoopeer with Kafka Cluster design

.\bin\windows\kafka-server-start.bat .\config\server.properties

### **Coding Steps:--**

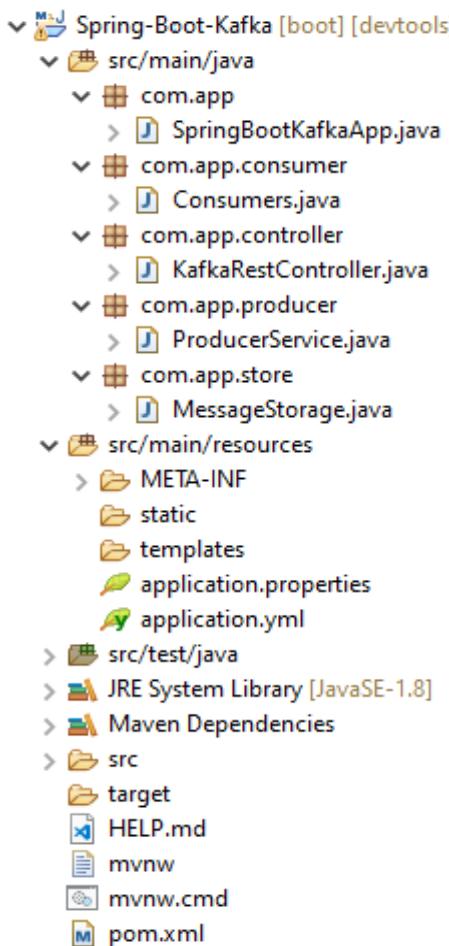
**Step#1:-** Choose spring apache kafka Dependency while Creating project (Which gives Integration of Spring with Kafka).

```
groupId : com.app
artifactId : SpringBootKafkaApp
version : 1.0
```

### Kafka Dependency:--

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

### #17. Folder Structure of Kafka:--



**Step#2:-** add key= value pairs in application (.properties/ .yml) file.

### application.properties:--

```
server.port: 9988
#Producer properties
```

```
my-app-topicname: sampletopic
spring.kafka.producer.bootstrap-servers: localhost:9092
spring.kafka.producer.key-serializer:
org.apache.kafka.common.serialization.StringSerializer
spring.kafka.producer.value-serializer:
org.apache.kafka.common.serialization.StringSerializer
```

#### #Consumer properties

```
spring.kafka.consumer.bootstrap-servers: localhost:9092
spring.kafka.consumer.key-deserializer:
```

```
org.apache.kafka.common.serialization.StringDeserializer
```

```
spring.kafka.consumer.value-deserializer:
```

```
org.apache.kafka.common.serialization.StringDeserializer
```

```
spring.kafka.consumer.group-id: group-id
```

#### properties.yml:--

```
server:
```

```
  port: 9988
```

```
my:
```

```
  app:
```

```
    topicname: sampletopic
```

```
spring:
```

```
  kafka:
```

```
    producer:
```

```
      bootstrap-servers: localhost:9092
```

```
      key-serializer: org.apache.kafka.common.serialization.StringSerializer
```

```
      value-serializer: org.apache.kafka.common.serialization.StringSerializer
```

```
    consumer:
```

```
      bootstrap-servers: localhost:9092
```

```
      key-deserializer: org.apache.kafka.common.serialization.StringDeserializer
```

```
      value-deserializer: org.apache.kafka.common.serialization.StringDeserializer
```

```
      group-id: groupId
```

#### Step#3:- Define MessageStorage class

```
package com.app.store;
import java.util.ArrayList;
import java.util.List;
import org.springframework.stereotype.Component;
```

```

@Component
public class MessageStorage {
    private List<String> list= new ArrayList<String>();

    public void put(String message) {
        list.add(message); //Write the logic to store data in database
    }
    public String getAll() {
        return list.toString();
    }
}

```

**Step#4:- Define Consumer class**

```

package com.app.consumer;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.stereotype.Component;
import com.app.store.MessageStorage;

```

```
@Component
```

```
public class Consumer {
```

```
    @Autowired
```

```
    private MessageStorage storage;
```

```
    @KafkaListener (topics="${my.app.topicname}", groupId="groupId")
```

```
    public void consume (String message) {
```

```
        storage.put(message);
```

```
    } }
```

**Step#5:- Define Producer code**

```
package com.app.producer;
```

```
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.beans.factory.annotation.Value;
```

```
import org.springframework.kafka.core.KafkaTemplate;
```

```
import org.springframework.stereotype.Component;
```

```
@Component
```

```
public class Producer {  
  
    @Value("${my.app.topicname}")  
    private String topic;  
  
    @Autowired  
    private KafkaTemplate<String, String> template;  
  
    public void sendMessage(String message) {  
        template.send(topic, message);  
    }  
}
```

**Step#6:- Define KafkaRestController class**

```
package com.app.controller;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RequestParam;  
import org.springframework.web.bind.annotation.RestController;  
import com.app.producer.Producer;  
import com.app.store.MessageStorage;
```

```
@RestController
```

```
public class KafkaRestController {
```

```
    @Autowired
```

```
    private Producer producer;
```

```
    @Autowired
```

```
    private MessageStorage storage;
```

```
    @RequestMapping("/send")
```

```
    public String readInMessage(@RequestParam String message)  
    {
```

```
        producer.sendMessage(message);
```

```
        return "message sent!!";
```

```
}
```

```

    @RequestMapping("/view")
    public String viewMessage() {
        return storage.getAll();
    }
}

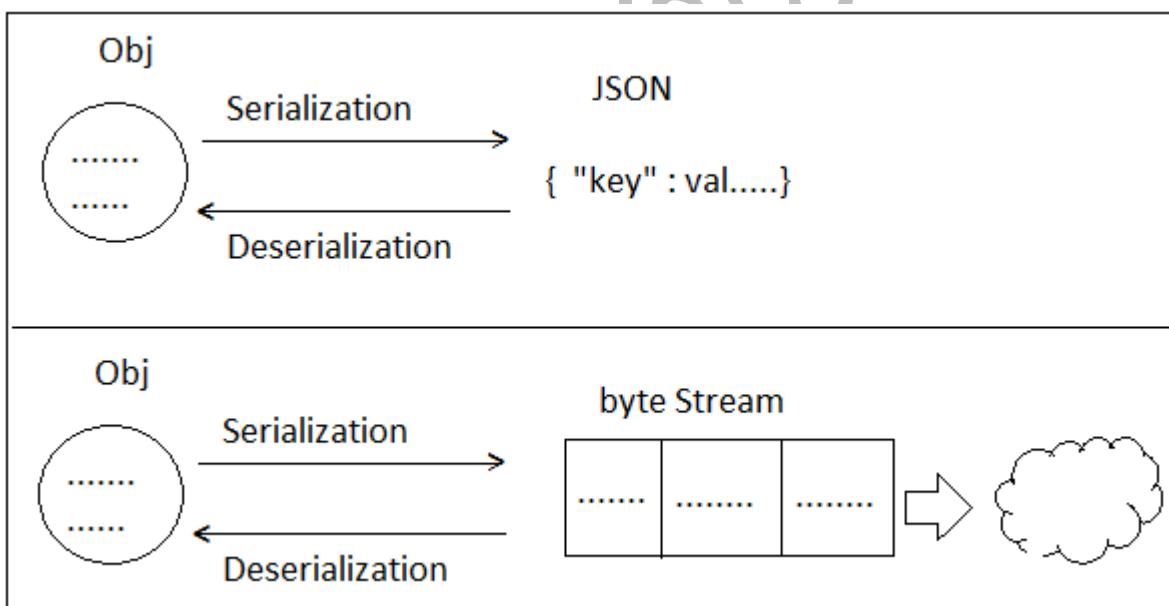
```

**Coding order:--**

1. application.properties:--
2. MessageStorage.java
3. ConsumerService.java
4. ProducerRestController.java
5. KafkaRestController.java

**NOTE:--** Use KafkaTemplate <K, V> at producer application to send message(V) to given Topic (K).

**NOTE:--** Use @KafkaListener (topics="....", groupId="....") at consumer side to read message (V) using topic (K) with groupId(G).

**Execution Order:--**

- 1>Start the Zookeeper Server
- 2>Start kafka-server (Cluster)
- 3>Start Application Starter class (Provider, Consumer)

## #1:- Start the Zookeeper Server

## #2:-- Start the kafka-Server (Cluster)

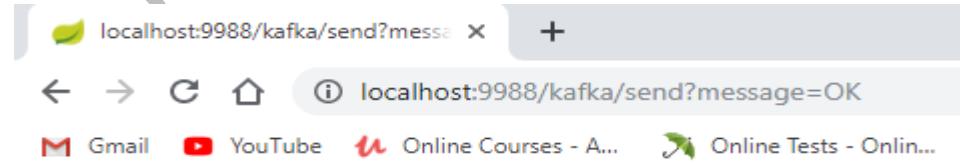
```
C:\Windows\System32\cmd.exe -l bin\windows\kafka-server-start.bat .\config\server.properties

Microsoft Windows [Version 10.0.18362.10015]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Uday\Downloads\kafka_2.12-2.2.0.\bin\windows\kafka-server-start.bat .\config\server.properties
[2019-08-28 09:31:18,359] INFO Registered kafka:type=kafka.Log4jController MBean (kafka.utils.Log4jControllerRegistration$)
[2019-08-28 09:31:22,046] INFO starting (kafka.server.KafkaServer)
[2019-08-28 09:31:22,049] INFO Connecting to zookeeper on localhost:2181 (kafka.server.KafkaServer)
[2019-08-28 09:31:22,242] INFO [ZooKeeperClient] Initializing a new session to localhost:2181. (kafka.zookeeper.ZooKeeperClient)
[2019-08-28 09:31:22,268] INFO Client environment:zookeeper.version=4.13-2d71af4dbe22557fd74f9a9b15a7478f03, built on 06/29/2018 00:39 GMT (org.apache.zookeeper.ZooKeeper)
[2019-08-28 09:31:22,269] INFO Client environment:host.name=DESKTOP-CH8LPUH (org.apache.zookeeper.ZooKeeper)
[2019-08-28 09:31:22,270] INFO Client environment:java.version=1.8.0_171 (org.apache.zookeeper.ZooKeeper)
[2019-08-28 09:31:22,271] INFO Client environment:java.vendor=Oracle Corporation (org.apache.zookeeper.ZooKeeper)
[2019-08-28 09:31:22,271] INFO Client environment:java.home=C:\Program Files\Java\jre1.8.0_171 (org.apache.zookeeper.ZooKeeper)
[2019-08-28 09:31:22,272] INFO Client environment:java.class.path=C:\Program Files\Apache Software Foundation\Tomcat 8.0\lib\server-api.jar;C:\kafka-2.2.0-src\bin\;C:\Users\Uday\Downloads\kafka_2.12-2.2.0\libs\activation-1.1.1.jar;C:\Users\Uday\Downloads\kafka_2.12-2.2.0\libs\apollo-alliance-repackaged-2.5.0-b42.jar;C:\Users\Uday\Downloads\kafka_2.12-2.2.0\libs\commons-lang3-3.8.1.jar;C:\Users\Uday\Downloads\kafka_2.12-2.2.0\libs\connect-2.2.0.jar;C:\Users\Uday\Downloads\kafka_2.12-2.2.0\libs\connect-basic-auth-extension-2.2.0.jar;C:\Users\Uday\Downloads\kafka_2.12-2.2.0\libs\connect-file-2.2.0.jar;C:\Users\Uday\Downloads\kafka_2.12-2.2.0\libs\connect-jms-2.2.0.jar;C:\Users\Uday\Downloads\kafka_2.12-2.2.0\libs\connect-runtime-2.2.0.jar;C:\Users\Uday\Downloads\kafka_2.12-2.2.0\libs\connect-transforms-2.2.0.jar;C:\Users\Uday\Downloads\kafka_2.12-2.2.0\libs\guava-20.0.jar;C:\Users\Uday\Downloads\kafka_2.12-2.2.0\libs\hk2-api-2.5.0-b42.jar;C:\Users\Uday\Downloads\kafka_2.12-2.2.0\libs\hk2-locator-2.5.0-b42.jar;C:\Users\Uday\Downloads\kafka_2.12-2.2.0\libs\hk2-locator-2.5.0-b42.jar;C:\Users\Uday\Downloads\kafka_2.12-2.2.0\libs\jackson-core-2.9.8.jar;C:\Users\Uday\Downloads\kafka_2.12-2.2.0\libs\jackson-databind-2.9.8.jar;C:\Users\Uday\Downloads\kafka_2.12-2.2.0\libs\jackson-datatype-jdk8-2.9.8.jar;C:\Users\Uday\Downloads\kafka_2.12-2.2.0\libs\jackson-jaxrs-base-2.9.8.jar;C:\Users\Uday\Downloads\kafka_2.12-2.2.0\libs\jackson-module-jaxb-annotations-2.9.8.jar;C:\Users\Uday\Downloads\kafka_2.12-2.2.0\libs\jaxassist-3.2-0.CR2.jar;C:\Users\Uday\Downloads\kafka_2.12-2.2.0\libs\javax.annotation-api-1.2.jar;C:\Users\Uday\Downloads\kafka_2.12-2.2.0\libs\javax.inject-1.jar;C:\Users\Uday\Downloads\kafka_2.12-2.2.0\libs\javax.inject-2.5.0-b42.jar;C:\Users\Uday\Downloads\kafka_2.12-2.2.0\libs\javax.servlet-api-3.1.0.jar;C:\Users\Uday\Downloads\kafka_2.12-2.2.0\libs\javax.ws.rs-api-2.1.1.jar;C:\Users\Uday\Downloads\kafka_2.12-2.2.0\libs\javax.ws.rs-api-2.1.1.jar;C:\Users\Uday\Downloads\kafka_2.12-2.2.0\libs\jaxb-api-2.3.0.jar;C:\Users\Uday\Downloads\kafka_2.12-2.2.0\libs\jersey-client-2.27.jar;C:\Users\Uday\Downloads\kafka_2.12-2.2.0\libs\jersey-common-2.27.jar;C:\Users\Uday\Downloads\kafka_2.12-2.2.0\libs\jersey-container-servlet-2.27.jar;C:\Users\Uday\Downloads\kafka_2.12-2.2.0\libs\jersey-container-servlet-core-2.27.jar;C:\Users\Uday\Downloads\kafka_2.12-2.2.0\libs\jersey-servlet-2.27.jar
```

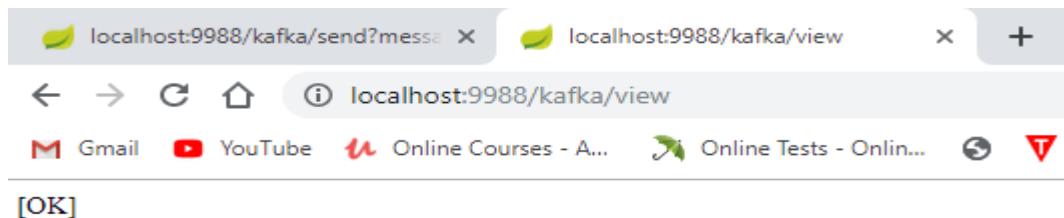
3#:- \*\*\*Run Starter class and enter URLs:-

1. <http://localhost:9988/kafka/send?message=OK>



Message sent...!!OK

2. <http://localhost:9988/kafka/view>



## 12. Spring Boot with Apache Camel:--

**Routing:**-- It is a process of sending large data from One Application (Source) to another Application (Destination).

=>A source/destination can be File System (.xml, .txt, .csv, .xlsx, .json,...etc), Database (Oracle DB, MySQL DB) or Message Queues using JMS (Active MQ) etc.

=>Apache Camel is open Source and Light weight “**Conditional based Routing Engine**” which supports filtering and processing. It behaves like mediator software between multiple source and destination.

=>Apache Camel is language independent software implemented in Java but also supports integration with different language like Hadoop, Bigdata, PHP, Python, & JavaScript... etc.

=>Compared to Spring Batch Integration tool Apache camel is a light weight tool.

=>Camel supports reading data from different sources even like HTTP, FTP, JMS protocols based.

=>It supports data processing (conversions, calculations, like XML--->JSON, JSON--->Object, Object--->XML, and XML ---> EXCEL etc.s

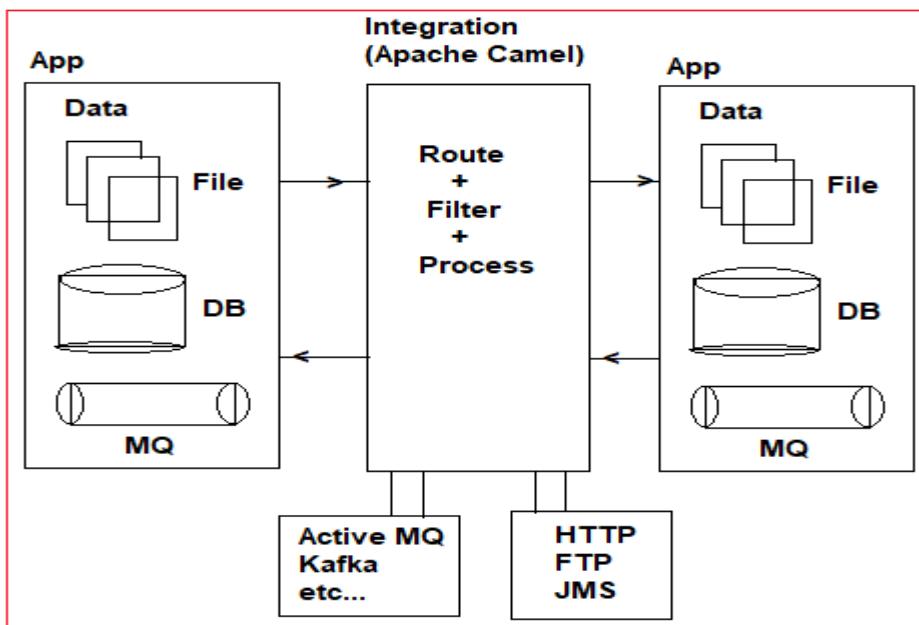
=>It has light weight engine, compared to spring Batch it is faster.

=>Camel mainly supports 3 operations:

a.>Routing :-- Data Transfer

b.>Processing :-- Data Conversion

c.>Filtering:-- Conditional checking



=>Camel supports easy coding using “EIP” (Enterprise Integration Pattern).

=>Format looks like :

```

[source-details]
.[filter-modes]
.[processing]
.[destination]

```

### Implementing Camel Routing in Spring boot:--

**Step#1:-** Create one Starter project with Apache Camel dependency. Or else add below mention dependency in pom.xml.

**#1:-** In pom.xml, we must add below dependency which supports Spring boot integration.

```

<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-spring-boot-starter</artifactId>
    <version>2.23.2</version>
</dependency>

```

### pom.xml:--

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.1.RELEASE</version>
        <relativePath /> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.app</groupId>
    <artifactId>Spring-Cloud-Camel-EIP</artifactId>
    <version>1.0</version>
    <name>Spring-Cloud-Camel-EIP</name>
    <description>Demo project for Camel Integration</description>

    <properties>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
        </dependency>
        <dependency>
            <groupId>org.apache.camel</groupId>
            <artifactId>camel-spring-boot-starter</artifactId>
            <version>2.23.2</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

```

```

    </dependencies>

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

```

**Step#2:-** Camel avoid main method (thread) execution control and run as independent while working with Spring boot, our starter class is main method. So we should add key=value in properties file as

**application.properties:--**

camel.springboot.main-run-controller=true

**Step#3:-** Define one RouteBuilder class and configure details of routing, filtering and processing.

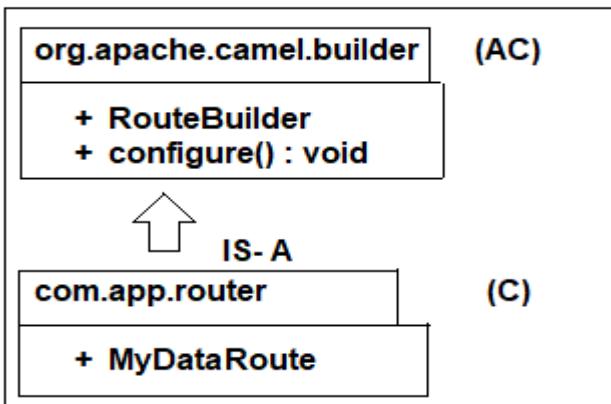
=>To implement this we need to write one Router class using “**RoutingBuilder (AC)**” provided by Apache camel having one abstract method: **configure()** which contains coding format like:

```

from (SourceLocation)
    .[filter] . [process].
    .to (DestinationLocation)

```

=>Here Location can be URL/Local File System DB, JMS (Message Queues)... etc.

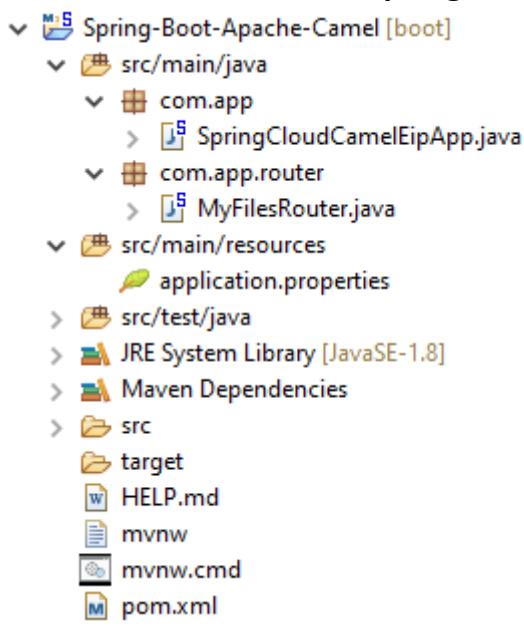
**Coding Steps:--**

**Step#1:-** Create Spring Boot starter application with dependencies : Apache Camel

GroupId : com.app

ArtifactId : Spring-Boot-Apache-Camel

Version : 1.0

**#18. Folder Structure of Spring Boot Integration with Apache Camel:--**

**Step#2:-** open application.properties (.yml) and add main method-control key as true.

**application.properties:--**

camel.springboot.main-run-controller=true

**application.yml:**

camel:

springboot:

main-run-controller: true

**Step#3:- Define Router class with EIP pattern with file transfer logic.**

```
package com.app.router;
import org.apache.camel.builder.RouteBuilder;
import org.springframework.stereotype.Component;
```

@Component

```
public class MyFilesRouter extends RouteBuilder {
    public void configure() throws Exception {
        //with static location
        from ("file:D:\\source").to("file:D:\\destination");
    }
}
```

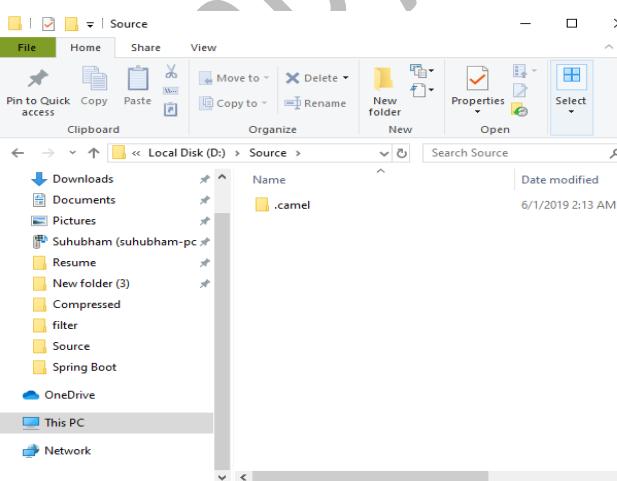
**Step#4:- Create two folder : Source and Destination in any Drive (“D: drive”).**

**Step#5:- Start Application and place files in “D:/source” which will be copied automatically into “D:/destination”.**

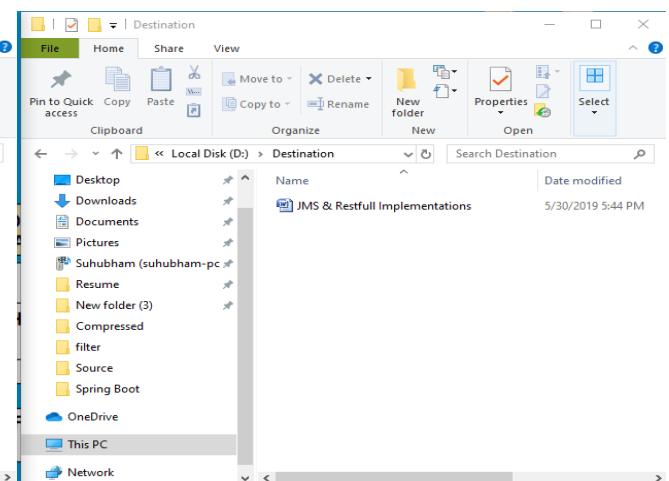
**NOTE:**--In source file .camel folder is created automatically which contains Copied file and in Destination folder only File will copied but no any folder will be created.

## Output Screen :-

### Folder#1: Source



### Folder#2: Destination



**1. EIP Patterns by Apache Camel:**-- EIP stand for “Enterprise Integration Patterns” used to define short form code for.

- >Data Routing
- >Data Filtering
- >Data Processing

**#1:- from ("file:source") .to("file:destination");**

=>It will copy files from source to destination by taking files in backup folder in source with name **.camel**.

=>It supports even same file sending with new data (Operation to Override Program).

**#2:- from ("file:source?noop=true") .to("file:destination");**

=>To avoid sending same files with different (Modified) data again “No operation to override program” should set as true.

**#3:- from ("{{source}}").to("{{destination}}");**

=>Here **{{location}}** indicates dynamic location that is given as input passing using Properties/Yml files, System args, command line inputs etc.

=>It means locations provided at runtime. For above example add below key in properties file.

application.properties:--

```
camel.springboot.main-run-controller=true
my.source=file:D://Source?noop=true
my.destination=file:D://Destination
```

**Example EIPs:--**

**#1>Dynamic Location:--**

Location (source/destination) can be passed at runtime using properties/yml files, config server, system arguments... etc.

=>To indicate Location (URL file System, DB, MQ...) comes at runtime use format:  
**{}{{location}}**

**Code changes:--**

**a. applictaion.properties:--**

```
server.port=2344
```

```
camel.springboot.main-run-controller=true  
my.source=file:D://Source?noop=true  
my.destination=file:D://Destination
```

b. Router class code

```
from("{{my.source}}").to("{{my.destination}}");
```

Ex:--

```
package com.app.router;  
import org.apache.camel.builder.RouteBuilder;  
import org.springframework.stereotype.Component;
```

```
@Component
```

```
public class MyFilesRouter extends RouteBuilder {  
    public void configure() throws Exception {  
        //With Dynamic Location  
        from("{{my.source}}").to("{{my.destination}}");  
    }  
}
```

## #2 Data Processing Using Apache Camel:--

In realtime data will be converted or modified based on requirements.

Like XML--->JSON, JSON--->Simple Text, XML--->CSV, CSV--->SQL Format etc..

i.e data converted from one format to another format which can be done using 3 supporting interfaces. Those are:

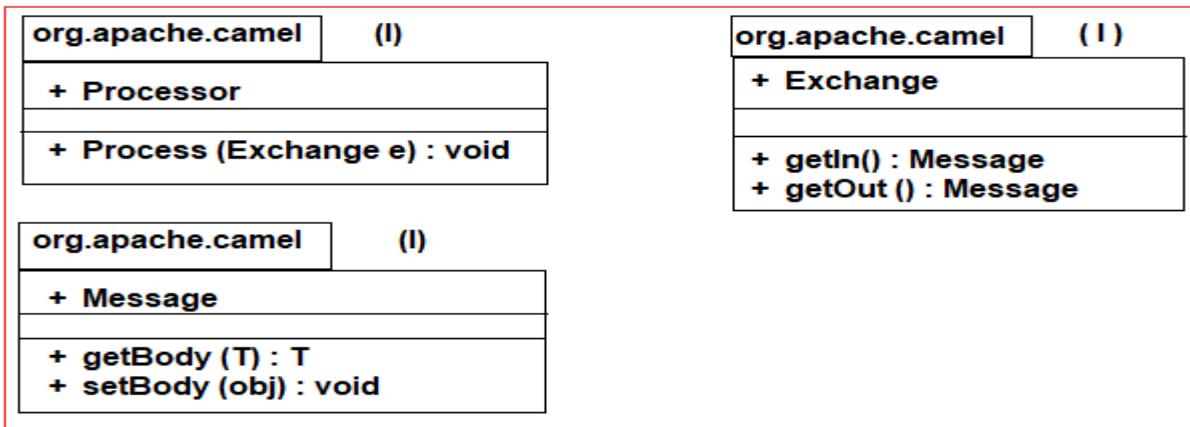
1>Processor (I)      2>Exchange (I)      3>Message (I)

=>Apache Camel has provided “Processor (I)” defined in package org.apache.camel which has only one abstract method (Functional Interface) i.e process (...).

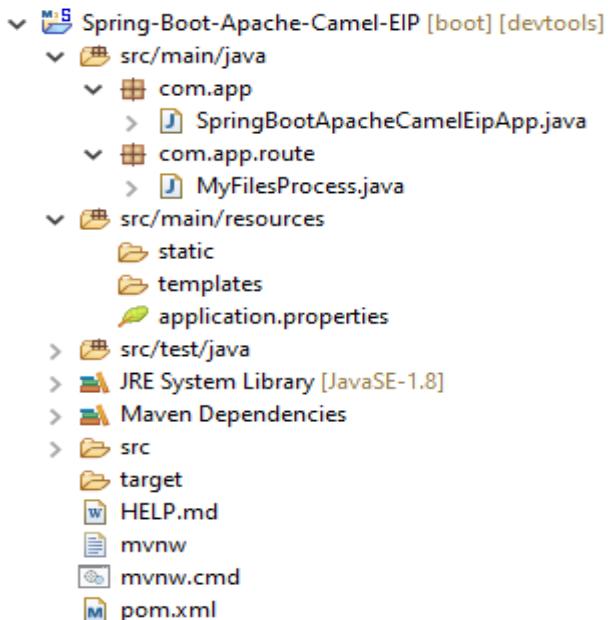
=>Processor (I) takes help of Exchange (I) to read In message and to set out message.

=>Exchange depends on Message (I) which has message body (get/set method).

=>Our files data is stored in Message (I) format. To read this data set, get methods are provided.



### #19. Folder Structure of Spring Boot Apache Camel-EIP:--



#### Process#1 Write inside Router (Impl) class:--

```

package com.app.route;

import org.apache.camel.Exchange;
import org.apache.camel.Message;
import org.apache.camel.Processor;
import org.apache.camel.builder.RouteBuilder;
import org.springframework.stereotype.Component;

@Component
public class MyFilesProcess extends RouteBuilder {

    public void configure() throws Exception {
        from ("file:D:/Source").process(new Processor()
    }
}

```

```

public void process (Exchange ex) throws Exception
{
    //#1 Read Input Message
    Message m = ex.getIn();
    //#2 Read Body from message
    String body = m.getBody(String.class);
    //#3 do processing
    body ="modified ::"+body;
    //#4 Writer data to out message
    Message m2 = ex.getOut();
    //#5 Set body (Data) to out message
    m2.setBody(body);
}
})
.to("file:D:/Destination?fileName=myFile.txt");
}
}

```

**\*\*\*Note:-** Here fileName indicates new name for modified message. It can also be passed at run time.

=>fileName must be provided in-case of processor is used, else file generated with ID which has no extension details.

### Process#2 Using Lambda Expression## code:--

```

package com.app.route;
import org.apache.camel.builder.RouteBuilder;
import org.springframework.stereotype.Component;

@Component
public class MyFilesProcess extends RouteBuilder {
    public void configure() throws Exception
    {
        from("file:D:/source")
        .process(ex->{
            String body=ex.getIn().getBody(String.class);
            body="New AA modified ::"+body;
            ex.getOut().setBody(body);
        })
        .to("file:D:/Destination?fileName=myFile.txt");
    }
}

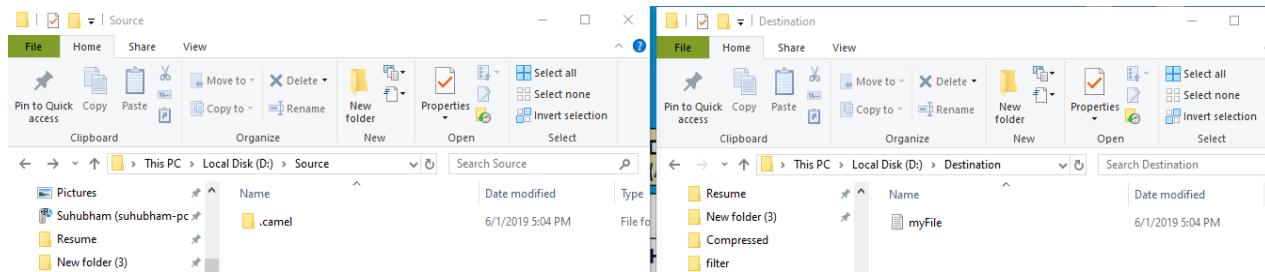
```

```

        })
        .to("file:D:/Destination?fileName=myFile.txt");
    }
}

```

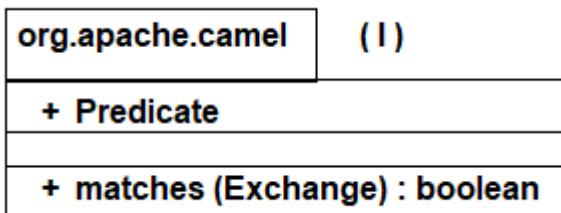
### Output Screen:--



### #3 Filters using Predicates:--

Predicate is a type expression which returns either true or false based on condition checking. Predicate is a Boolean expression after executing a task.

=>If true next level steps are executed else false execution stopped here only.



=>ValueBuilder (C) is used to execute required Predicates, using method: contains(), startWith(), isNull(), ...etc.

=>We can get ValueBuilder (C) object using methods body(), header().

=>header() indicates checking on file name, extension, location ... etc.

=>body() indicates file data/ content check

**Ex#1:-** file name having word sample

```

package com.app.route;
import org.apache.camel.Exchange;
import org.apache.camel.builder.RouteBuilder;
import org.springframework.stereotype.Component;

```

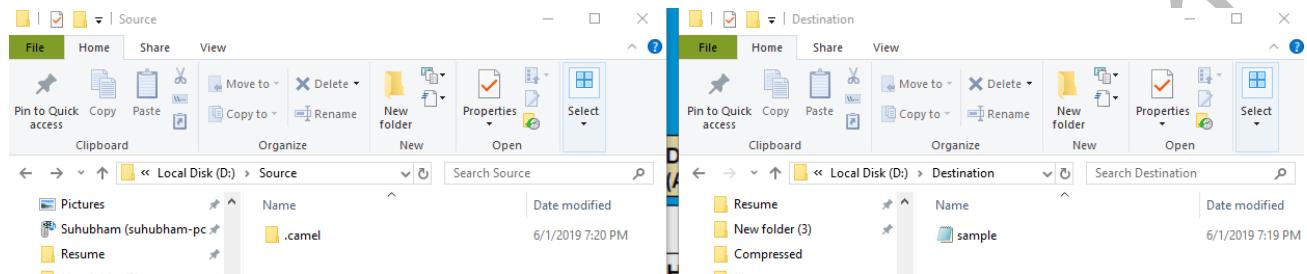
@Component

```
public class MyFilesProcess extends RouteBuilder {
```

```

public void configure() throws Exception {
    from("file:D:/Source").filter(header(Exchange.FILE_NAME)
        .contains("sample"))
        .to("file:D:/Destination");
}

```

**Output:--**

**Ex#2:-** File body starts with word java

```

package com.app.route;
import org.apache.camel.builder.RouteBuilder;
import org.springframework.stereotype.Component;

```

```
@Component
```

```
public class MyFilesProcess extends RouteBuilder
{
```

```

    public void configure() throws Exception {
        from("file:D:/source")
            .filter(body().startsWith("java"))
            .to("file:D:/destination");
    }
}
```

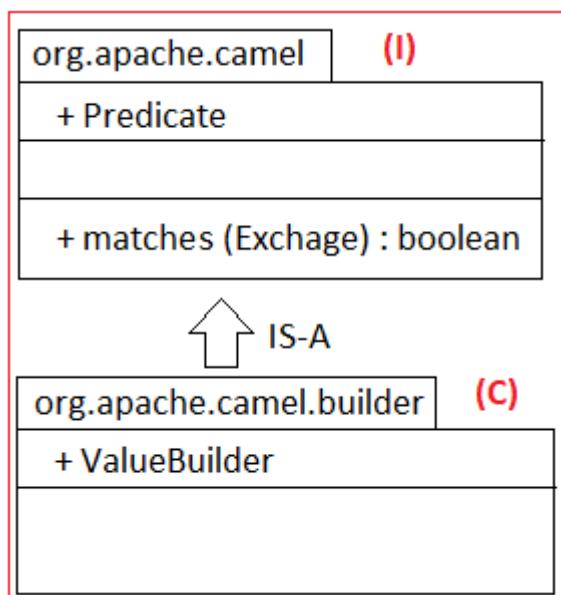
**Output Screen file contents:--**

sample - Notepad  
File Edit Format View Help  
**java Is a Programming Language.**

**Conditional based Routing [Choose –when-otherwise]:--**

=>Apache camel supports data routing based on predicates (Type expression) which support verify conditions.

- =>A predicate is a Boolean expression after executing a task.
- =>To verify inputs files use methods header() and body() given by RouteBuilder.
- =>Here header () is used to verify file details like “filename”, extension, filesize, ... etc.
- =>Here body () is used to verify file data like “fileContains”, startWith, endWith ... etc.
- =>To read message (head + body) camel provides Exchange object.
- =>To Handle switch-case concept for data routing use choose-when-other.



=>By using ValueBuilder (C) we can do filter() and choice() based predicates execution.

a. **filter:**-- This is used to check one given boolean expression if true execute next step, else does nothing.

#### RouteBuilder Impl Class for Filter:-

**Example#1:**--

```

package com.app.route;
import org.apache.camel.Exchange;
import org.apache.camel.builder.RouteBuilder;
import org.springframework.stereotype.Component;

@Component
public class MyFilesProces extends RouteBuilder {
    public void configure() throws Exception {
        from("file:D:/Source")
            .filter(header(Exchange.FILE_NAME)

```

```

        //.startsWith("Employee")
        //.endsWith("-data.xml")
        .contains("employee"))
        .to("file:D:/Destination");

    }
}

```

**Example#2:--**

```

package com.app.route;
import org.apache.camel.builder.RouteBuilder;
import org.springframework.stereotype.Component;

```

```

@Component
public class MyFilesProces extends RouteBuilder {
    public void configure() throws Exception {
        from("file:D:/Source")
            .filter(body().startsWith("Hello"))
            .to("file:D:/Destination");
    }
}

```

**b. choice():--** This is used to execute multiple condition checks in a order, It behaves like switch-case concept.

=>We can provide multiple when() with predicates even with process(..). Finally otherwise() is executed if all conditions are fail.

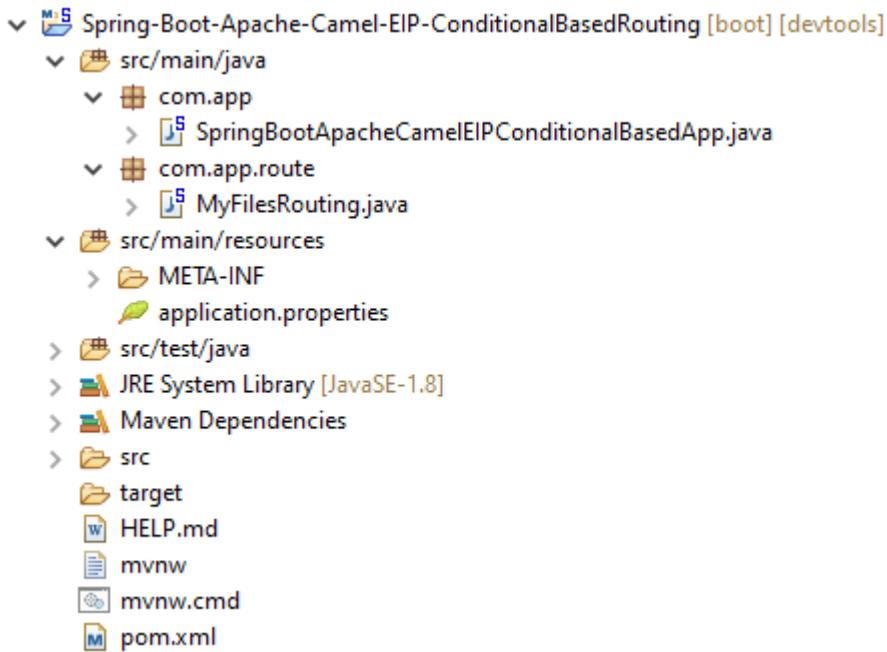
**Format looks like:--**

```

from("source")
    .choice()
        .when(condition#1).to("destinatation#1")
        .when("condition#2").to("destinatation#2")
        ....
        otherwise().to("destinatation#n")

```

## #20. Folder Structure of ConditionalBased Routing:--



### #1 Starter class (`SpringBootApacheCamelEIPConditionalBasedApp`):--

```
package com.app;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringBootApacheCamelEIPConditionalBasedApp {

    public static void main(String[] args)
    {
        SpringApplication.run(SpringBootApacheCamelEIPConditionalBasedApp.class, args);
        System.out.println("Spring Boot Camel Executed :");
    }
}
```

### #2 `MyFilesRouting.java`:--

```
package com.app.route;
import org.apache.camel.builder.RouteBuilder;
import org.springframework.stereotype.Component;

@Component
public class MyFilesRouting extends RouteBuilder {
```

```

@Override
public void configure() throws Exception {
    from("file:D:/Source")
    .choice()
    .when(body().startsWith("java")).to("file:D:/Destination?fileName=a.txt")
    .when(body().startsWith("xml")).to("file:D:/Destination?fileName=b.txt")
    .when(body().startsWith("json")).to("file:D:/Destination?fileName=c.txt")
    .otherwise().to("file:D:/Destination?fileName=d.txt");
}

```

**Input file contains:--**

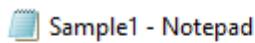
**#1** If file body contains start with java then create a new file a.txt in destination.  
Where as body contain is case sensitive.



File Edit Format View Help

**java Is a Programming Language.**

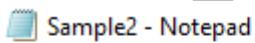
**#2** If file body contains start with xml then create a new file b.txt in destination. Where as body contain is case sensitive.



File Edit Format View Help

**xml is a global data format.**

**#3** If file body contains start with json then create a new file c.txt in destination.  
Where as body contain is case sensitive.



File Edit Format View Help

**json is a global data format.**

**#4** If file body contains is not start with **java, xml, json** then create a new file d.txt in destination.



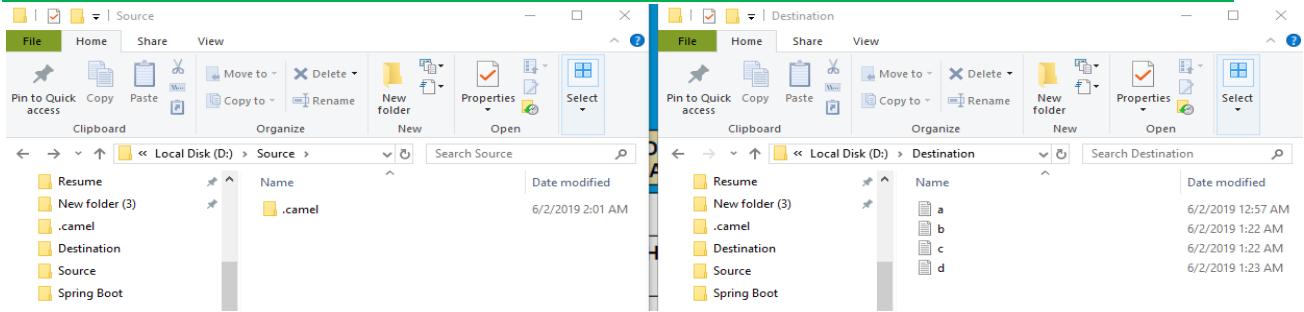
File Edit Format View Help

**Raja is a Good Boy.**

**Output Screen:--**

**Source File:--**

**Destination File:--**



## 2. Apache Camel Intergradations with Active MQ :-

=>Camel supports read/write data from MQ source using EIP patterns and MQ Broker.  
=>Patterns used to communicate with MQ using camel is : jms<type>:<destination>  
=>Here type can be queue or topic. To use this we need to define protocol with prefix "jms", given as

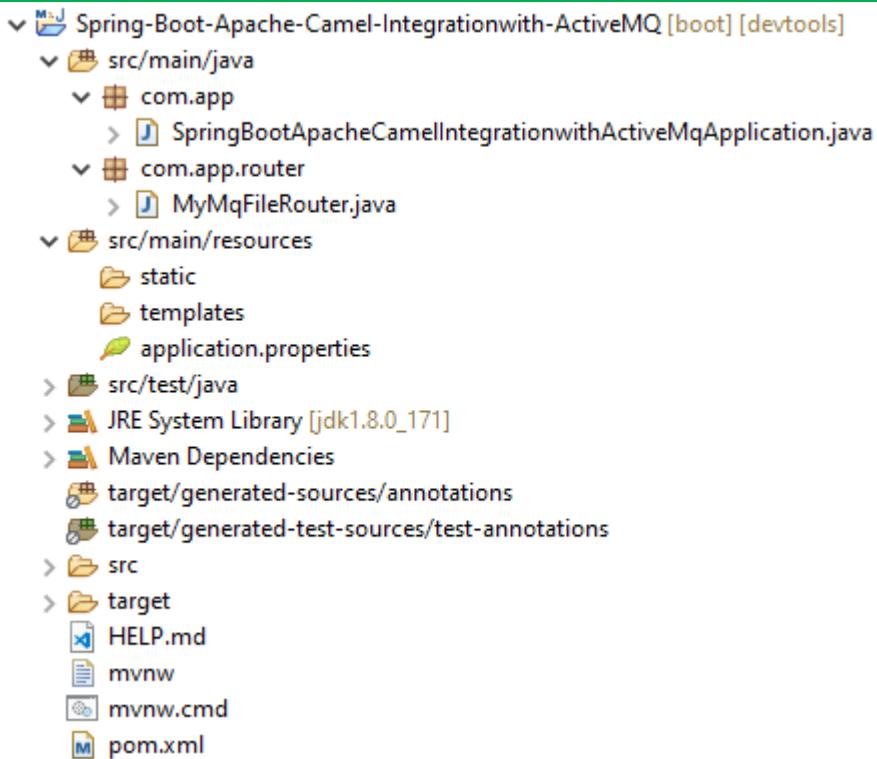
Example : 1.> jms:queue:info (queue Name)  
2.> jms:topic:news (topic Name)

=>For this coding along with ActiveMQ and Camel Dependencies we should add ActiveMQ-pool and camel-jms integration dependencies.

**Step#2:-** Create project and add below dependencies in pom.xml file

```
<dependency>
    <groupId>org.apache.activemq</groupId>
    <artifactId>activemq-pool</artifactId>
</dependency>
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-jms</artifactId>
    <version>2.24.0</version>
</dependency>
```

## #21. Folder Structure of Apache Camel Intergradations with ActiveMQ:--



### Step#3:- Provide camel and ActiveMQ properties

#### **application.properties**

```
camel.springboot.main-run-controller=true
spring.activemq.broker-url=tcp://localhost:61616
spring.activemq.user=admin
spring.activemq.password=admin
```

### Step#4:- Define Routers

```
package com.app.router;
import org.apache.camel.builder.RouteBuilder;
import org.springframework.stereotype.Component;

@Component
public class MyMqFileRouter extends RouteBuilder {

    @Override
    public void configure() throws Exception {
        //1. Sending file Data from Source to Message Queue
        /* from("file:D:/Source?fileName=mydata.txt") .to("jms:queue:outdata"); */
    }
}
```

```
//2. Sending Data from MessageQueue to Destination folder
from("jms:queue:indata")
.to("file:D:/Destination?fileName=mydata.txt");
}
}
```

#### Execution order:--

- Run ActiveMQ
- Run Starter class
- Create queue "abc"
- Click on send to link
- Enter message and send button
- Open file and view data

**Step#5:-** Run starter class and start ActiveMQ using bat

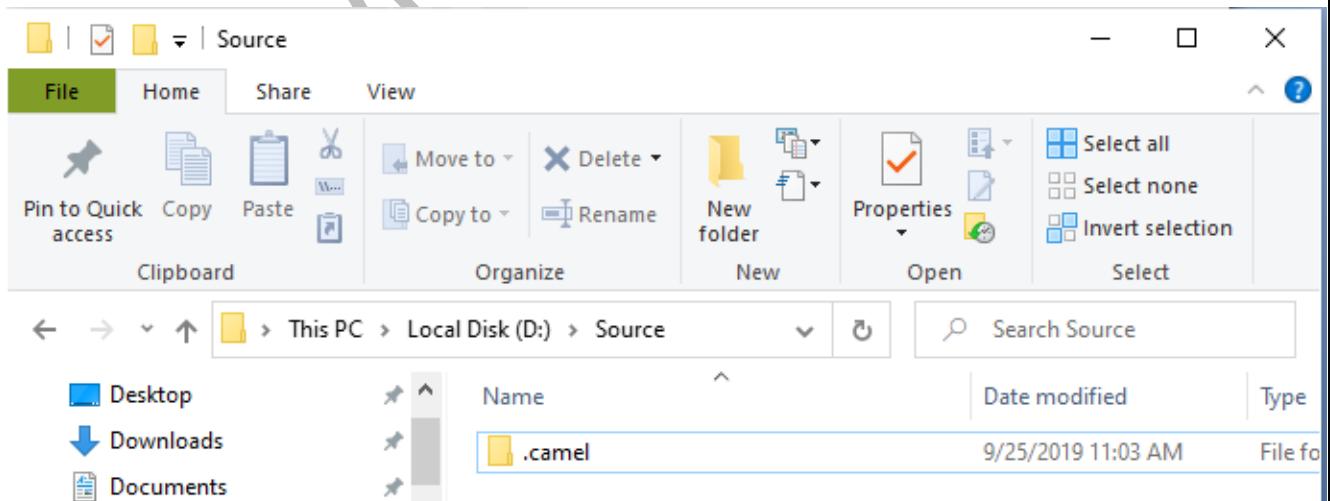
**Step#6:-** Login to MQ (<http://localhost:8161/admin>)

=>Click on menu Queue.

=>Enter Queue name and click create [2 queues : outdata, indata]

#### #1:- Sending data from file to MessageQueue:--

=>Goto Specified file location like(D:/Source) and copy any file or keep any file before starting Spring starter class.



**NOTE:--** If you keep any file before, after sending it will be automatically converted to ".camel" folder.

## #2:- Screen for send data from MessageQueue to Destination:--

The screenshot shows the ActiveMQ Admin interface at [localhost:8161/admin/queues.jsp](http://localhost:8161/admin/queues.jsp). The 'Queues' tab is selected. A search bar at the top has 'indata' entered. Buttons labeled '#2' and '#3' are overlaid on the interface.

Name	Number Of Pending Messages	Number Of Consumers	Messages Enqueued	Messages Dequeued	Views	Operations
indata	0	0	0	0	<a href="#">Browse Active Consumers</a> <a href="#">Active Producers</a>	<a href="#">Send To Purge</a> <a href="#">Delete</a> #4
outdata	0	0	0	0	<a href="#">Browse Active Consumers</a> <a href="#">Active Producers</a>	<a href="#">Send To Purge</a> <a href="#">Delete</a>

=>Click on “send To” option on “indata” queue, Enter message and press send.

=>File will be copied to destination folder.

The screenshot shows the ActiveMQ Admin interface at [localhost:8161/admin/send.jsp?JMSDestination=indata&JMSDestinationType=queue](http://localhost:8161/admin/send.jsp?JMSDestination=indata&JMSDestinationType=queue). The 'Send a JMS Message' form is displayed. The 'Message body' field contains the text '#5 Enter Some Message'.

=>Goto inside D:/Destinatation folder and see the output

### 3. ApacheCamel Integration with JDBC:--

Camel supports reading data or writing data using JDBC to any database like MySQL, POSTGRES, H2, ORACLE etc... for this we need to provide dataSource Object as input to Camel.

=>Here DataSource is an Interface defined in Javax.sql package we need to configure its implementation class object.

=>By using one dataSource and SQL query we can fetch data from Database table.

->It will be converted to List<Map<String, Object>> [K=Key=String type, V=Value=Object Type].

=>ResultSet data is converted to List<Map> format by Camel internally, example.

empTab			List<Map<String, Object>>												
<b>eid</b> <b>ename</b> <b>esal</b>															
10	Uday	236.9	<table border="1"> <tr> <td>eid</td><td>10</td></tr> <tr> <td>ename</td><td>Uday</td></tr> <tr> <td>esal</td><td>236.9</td></tr> <tr> <td> </td><td> </td></tr> <tr> <td> </td><td> </td></tr> <tr> <td> </td><td> </td></tr> </table>	eid	10	ename	Uday	esal	236.9						
eid	10														
ename	Uday														
esal	236.9														
11	Venkat	556.6	<table border="1"> <tr> <td>eid</td><td> </td></tr> <tr> <td>ename</td><td> </td></tr> <tr> <td>esal</td><td> </td></tr> <tr> <td> </td><td> </td></tr> <tr> <td> </td><td> </td></tr> <tr> <td> </td><td> </td></tr> </table>	eid		ename		esal							
eid															
ename															
esal															
12	Neha	345.7	<table border="1"> <tr> <td>eid</td><td> </td></tr> <tr> <td>ename</td><td> </td></tr> <tr> <td>esal</td><td> </td></tr> <tr> <td> </td><td> </td></tr> <tr> <td> </td><td> </td></tr> <tr> <td> </td><td> </td></tr> </table>	eid		ename		esal							
eid															
ename															
esal															

## Ex#1: Camel SQL-MQ Integration

### 22. Folder Structure of Camel JDBC (SQL)-MQ :-

```

    <img alt="Project tree diagram for ApacheCamelIntegrationWithJDBC project" data-bbox="127 107 580 415"/>
    ApacheCamelIntegrationWithJDBC [boot] [devtools]
    +-- src/main/java
        +-- com.app
            +-- ApacheCamelIntegrationWithJdbc1Application.java
        +-- com.app.config
            +-- AppConfig.java
        +-- com.app.route
            +-- MyBuilder.java
    +-- src/main/resources
        +-- application.properties
    +-- src/test/java
    +-- JRE System Library [JavaSE-1.8]
    +-- Maven Dependencies
    +-- src
    +-- target
    +-- HELP.md
    +-- mvnw
    +-- mvnw.cmd
    +-- pom.xml
  
```

**Step#1:-** Create project using Apache Camel, Apache ActiveMQ, MySQL connector.

**Step#2:-** Add Extra MQ Dependencies and camel-jdbc, spring-jdbc, mysqlversion/Oracle based.

#### pom.xml:-

```

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.7.RELEASE</version>
    <relativePath /> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.app</groupId>
  <artifactId>Apache-Camel-Integration-JDBC</artifactId>
  
```

```
<version>1.0</version>
<name>ApacheCamelIntegrationWithJDBC1</name>
<description>Camel Integration with JDBC App</description>

<properties>
    <java.version>1.8</java.version>
</properties>

<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-activemq</artifactId>
    </dependency>
    <dependency>
        <groupId>org.apache.camel</groupId>
        <artifactId>camel-spring-boot-starter</artifactId>
        <version>2.24.0</version>
    </dependency>
    <dependency>
        <groupId>org.apache.camel</groupId>
        <artifactId>camel-jdbc</artifactId>
        <version>2.24.0</version>
    </dependency>
    <dependency>
        <groupId>org.apache.camel</groupId>
        <artifactId>camel-jms</artifactId>
        <version>2.24.0</version>
    </dependency>
    <dependency>
        <groupId>org.apache.activemq</groupId>
        <artifactId>activemq-pool</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-jdbc</artifactId>
    </dependency>
```

```

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
    <optional>true</optional>
</dependency>

<dependency>
    <groupId>com.oracle</groupId>
    <artifactId>ojdbc6</artifactId>
    <version>11.2.0</version>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
</dependencies>

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

```

### Step#3:- Starter class

```

package com.app;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class ApacheCamelIntegrationWithJdbc1Application {

```

```

public static void main(String[] args) {
    SpringApplication.run(ApacheCamelIntegrationWithJdbc1Application.class, args);
    System.out.println("Starter class Executed..:");
}
}

```

**Step#4:-** Define one DataSource Object to connect with Database.

```

package com.app.config;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.jdbc.datasource.DriverManagerDataSource;

@Configuration
public class AppConfig {

    @Bean
    public DriverManagerDataSource dsObj() {
        DriverManagerDataSource d = new DriverManagerDataSource();
        d.setDriverClassName("oracle.jdbc.driver.OracleDriver");
        d.setUrl("jdbc:oracle:thin:@localhost:1521:xe");
        d.setUsername("system");
        d.setPassword("system");
        return d;
    }
}

```

**Step#5:-** Define One RouteBuilder class with logic

```

package com.app.route;
import org.apache.camel.builder.RouteBuilder;
import org.springframework.stereotype.Component;

@Component
public class MyBuilder extends RouteBuilder {

    public void configure() throws Exception {
        from("timer:timer1?repeatCount=1&period=10s")
            .setBody(constant("select * from emptab"))
            .to("jdbc:dataSource")
    }
}

```

```

    .process(ex-> {
        Object ob=ex.getIn().getBody();
        System.out.println(ob.toString());
        ex.getOut().setBody(ob.toString());
    })
    .to("jms:queue:two");
}

```

**application.properties:--**

camel.springboot.main-run-controller=true  
 spring.activemq.broker-url=tcp://localhost:61616  
 spring.activemq.user=admin  
 spring.activemq.password=admin

**Execution Order:--**

- #1:-- Create table and insert data.
- #2:-- Run ActiveMQ
- #3:-- Run Starter class of Project.
- #4:-- Goto ActiveMQ (<http://localhost:8161/admin>), Click on Queue to see the details.

Name	Number Of Pending Messages	Number Of Consumers	Messages Enqueued	Messages Dequeued	Views	Operations
indata	2	0	2	0	<a href="#">Browse Active Consumers</a> <a href="#">Active Producers</a> <a href="#">atom</a> <a href="#">rss</a>	<a href="#">Send To</a> <a href="#">Purge</a> <a href="#">Delete</a>

**NOTE:--**

=>use protocol “timer://...” which takes input like repeatCount (no. of iterations) and period (time gap) in mill sec.

**Example---**

```
from("timer:timer1?repeatCount=1&period=10s")
    .setBody(constant("select * from emptab"))
    .to("jdbc:dataSource")
```

**Q>SQL V/s JDBC in Camel?**

=>Sql Uses Iterator to fetch data from List<Map> and returns Object by Object where as JDBC gets data in List<Map> returns same to next step.

#1:-timer://anyName?period=1000

->It is repeated for every 1Sec

#2:-timer://anyName?period=10s

->It is repeated for every 10Sec

#3:-timer://anyName?repeatCount=4&period=10m

->It is executed for every 10 minutes and 4 times only

Task#1:-- Use sql: insert fetch data from MQ

Task#2:-- use jdbc: select read List<Map> convert to csv file format.

### 13. Open Authorization (OAuth 2.x):--

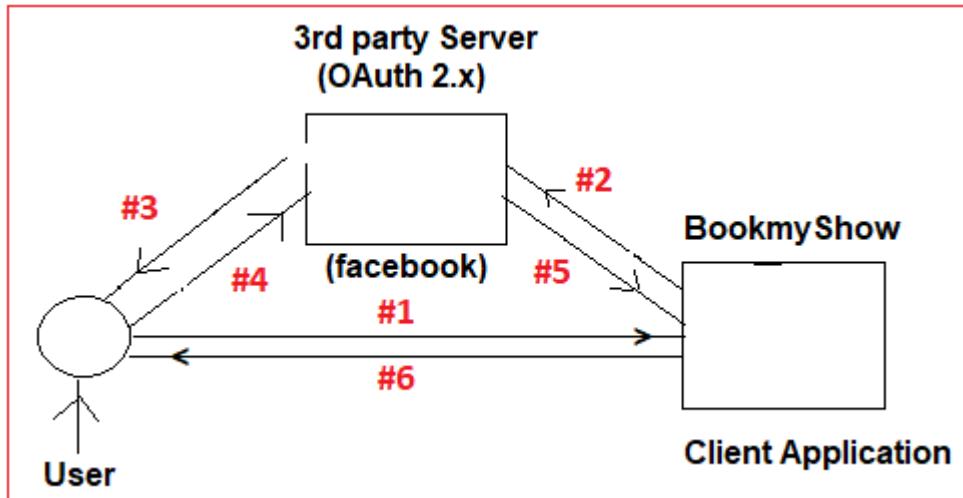
A & A = Authentication and Authorization

=>It is a process of secure one application user data. Here,

->Authentication means “IDENTITY OF USER”, like username, password, otp ...etc.

=>Authorization means “ROLE OF USER” what you can do? Like ADMIN ROLE, END USER, EMPLOYEE ROLE... (It is like permissions/grant).

**OAuth 2.x:**-- It is standard and framework which provides 3<sup>rd</sup> party security services to client application which are registered, on behalf of end user.



1.>Browser making request to client Application.

2.>Client asking permission to third party.

3.>Third party Application asking confirmation (Grant) to end user.

4.>User confirmation.

5.>Data shared from 3<sup>rd</sup> party Application to Client App.

6.>Client gives response to end user.

=>These 3<sup>rd</sup> party Applications are also called as “Authorization and Resource Servers”.

->Authorization and Resource Server examples are:- Google, Facebook, Github, Twitter, Linkedin ... etc.

->Example client Applications that are using OAuth2 are BookMyshow, redbus, yatra, makemytrip, avast, zomato.. etc.

=>OAuth 2.x standard is widely used in small/medium scale/daily used, business application.

=>OAuth2 Provide SSO (Single Sign on) for multiple applications acting as a one Service.

=>\*\*\*OAuth2 may not be used for high level and Large scale applications (Ex:- Banking Creditcard, Stock Market, finance... etc). These Spring Security ORM is used mostly.

### #1:-- Client Register with Authorization Server

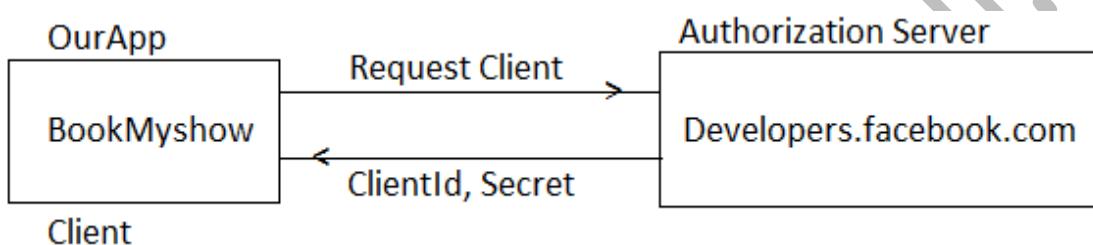
Ex: BookMyShow---Register-with--->Facebook Server

=>Here, every Client Application needs to be register with **AuthorizationServers** First.

=>Client Application gets clientId and clientSecret (like Password) on Successful register at AuthorizationServer.

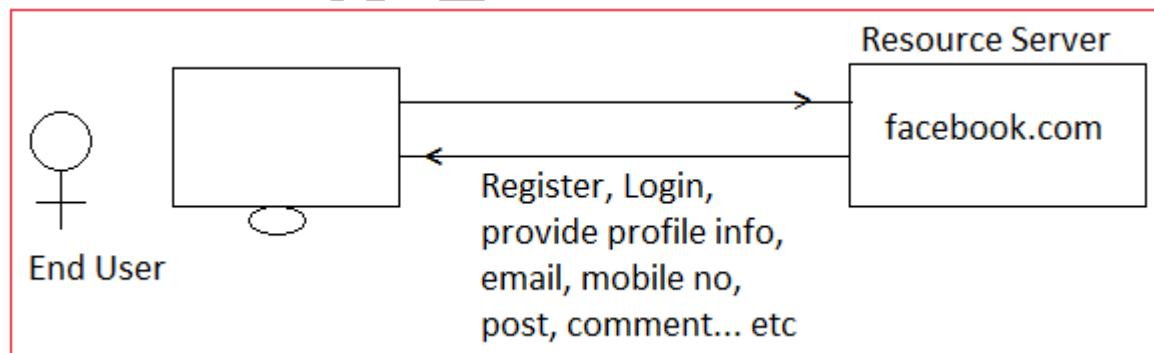
=>This is like one time setup between Client Application and Authorization Server.

#### #1. Register Client Application with Authorization Server.

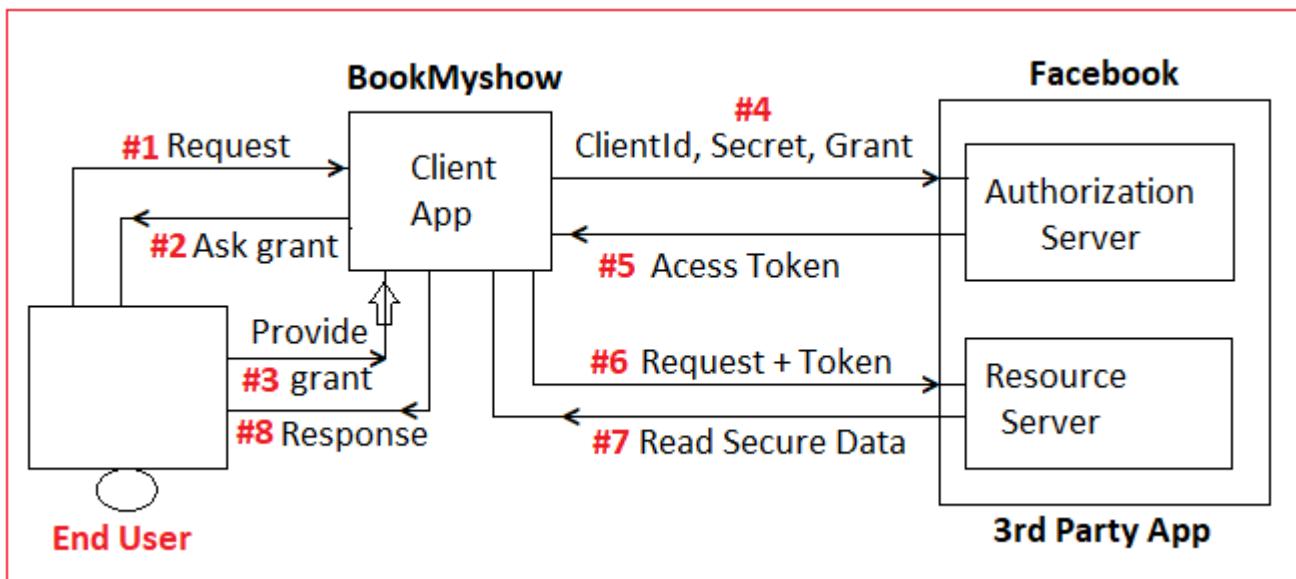


### #2:-- End User must be Register with Resource Server:--

=>User must be register and Login with 3<sup>rd</sup> party **Resource Servers** (Like facebook.com).  
 =>User need to provide profile information, basic data, comments, posts, photos... etc.  
 =>User Id and password will never be shared with Client Application. Only Users Public and General information is shared with Client Application.



### Request Work flow of OAuth2:-



1. End user makes request using browser to client application (BookMyshow) request for “verify using 3<sup>rd</sup> party service” ex: Facebook, Google... etc.
2. Client Application will ask for Grant from end user, which confirms that access user data.
3. End user has to provide Grant (Permission) to access data.
4. Client makes request to Authorization server using ClientId, secret, user grant.
5. Auth server verifies details and goes to token Management process.  
=A unique number is generated, called as Token which works for User+Client combination.
6. Now, client application makes request to resource Server using Access Token.
7. Resource server returns end user secure data to client.
8. Finally, Client App process the end user request and gives response.

### OneTime setup for OAuth2:-

**Step#1:-** Choose any one (or more) 3<sup>rd</sup> party “Authorization & Resource Server”.

Ex:-- facebook, Google cloud platform (GCP) Github, Linkedin, Twitter... etc.

**Step#2:-** Here choosing Facebook as 3<sup>rd</sup> party server for open Authorization link is:  
<https://developers.facebook.com/>

**Step#3:-** Define one new (client) Application in FB Authorization server which generates ClientId (AppId) and secrete (App Secret)

=>Goto facebook developer page

=>Click on top right corner “My Apps”

[Raghu Sir]

[NareshIT, Hyd]

The screenshot shows the Facebook Developers homepage. At the top, there's a navigation bar with links to Products, Docs, More, and My Apps. The 'My Apps' link is highlighted with a red box. Below the navigation, there's a large banner for 'Facebook for Developers' with the tagline 'Empowering creators, developers, and businesses to build for the future.' A button labeled 'Explore Products' is visible. On the left side of the main content area, there's a photograph of a person standing outdoors in a field, holding a laptop and looking at a piece of equipment on a tripod. The equipment includes a solar panel and a small computer unit.

=>Choose “Add New App”.

This screenshot shows the same Facebook Developers page as the previous one, but the right sidebar is more prominent. It displays a list of apps under the heading 'Uday's first app'. The first item is 'Uday's first app' with a gear icon. Below it is a large red box highlighting the 'Add New App' button, which has a plus sign icon. To the right of the sidebar, there are links for Requests, Developer Settings, Company Settings, and Log Out of Facebook. The central content area features a banner for the 'Facebook Developer Conference' with the text 'That's a wrap for F8! Watch the keynote sessions on demand' and a 'Experience F8 Online' button.

=>Provide Display name (ex: SpringCloudTestApp) and email id :  
udaykumar0023@gmail.com.

This screenshot shows the 'Create a New App ID' form. The form is titled 'Create a New App ID' and has a sub-instruction 'Get started integrating Facebook into your app or website'. It contains two input fields: 'Display Name' with the value 'SpringCloudTestApp' and 'Contact Email' with the value 'udaykumar0023@gmail.com'. At the bottom of the form, there's a note 'By proceeding, you agree to the Facebook Platform Policies' and two buttons: 'Cancel' and 'Create App ID'. The background of the page shows the same landscape photograph and developer conference banner as the previous screenshots.

- =>Click on “Create App ID”
- =>Complete Security verification
- =>Click on “Dashboard”
- =>Click on “facebook Login” setup button
- =>Click on “Settings” >>Basic

APP ID: 1304393836394202

Status: In Development

My Products

Facebook Login

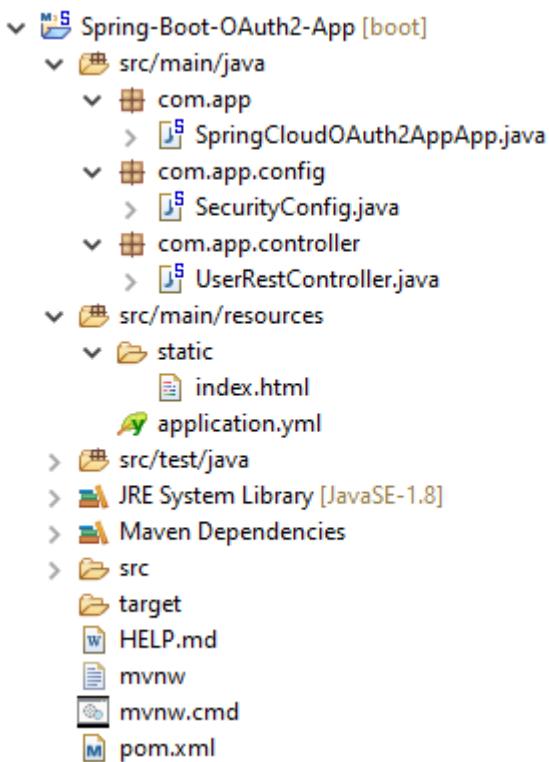
=>Basic AppId (ClientId) and App Secret (ClientSecret).

App ID	App Secret
1304393836394202	a7b20d9d6896953a21b9a1e951ad29c

**Step#4:-** Create one SpringBoot app with dependencies “Security” & “Cloud OAuth2”.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-oauth2</artifactId>
</dependency>
```

### #23. Folder Structure of OAuth2 Application:--



### pom.xml:--

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
```

```
<version>2.1.1.RELEASE</version>
<relativePath/> <!-- lookup parent from repository -->
</parent>
<groupId>com.app</groupId>
<artifactId>Spring-Cloud-OAuth2-App</artifactId>
<version>1.0</version>
<name>Spring-Cloud-OAuth2-App</name>
<description>Spring Cloud OAuth2 Implementation</description>
<properties>
    <java.version>1.8</java.version>
    <spring-cloud.version>Greenwich.SR1</spring-cloud.version>
</properties>
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-oauth2</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

```

<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>${spring-cloud.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
</project>

```

**Step#5:-** Create application.properties / application.yml file using below key value pairs.

**application.properties:-**

server.port: 9898

**#Auth Server details**

security.oauth2.client.clientId: 1304393836394202

security.oauth2.client.clientSecret: a7b20d9d68986953a21b9a1e951ad29c

security.oauth2.client.accessTokenUri:

[https://graph.facebook.com/oauth/access\\_token](https://graph.facebook.com/oauth/access_token)

security.oauth2.client.userAuthorizationUri: <https://www.facebook.com/dialog/oauth>

security.oauth2.client.tokenName: oauth\_token

security.oauth2.client.authenticationScheme: query

security.oauth2.client.clientAuthenticationScheme: form

**#Resource Server details**

security.oauth2.resource.userInfoUri: <https://graph.facebook.com/me>

**application.yml::**

server:

port: 9898

security:

oauth2:

**#Auth Server details**

client:

clientId : 1304393836394202

clientSecret : a7b20d9d68986953a21b9a1e951ad29c

accessTokenUri : https://graph.facebook.com/oauth/access\_token

userAuthorizationUri : https://www.facebook.com/dialog/oauth

tokenName: oauth\_token

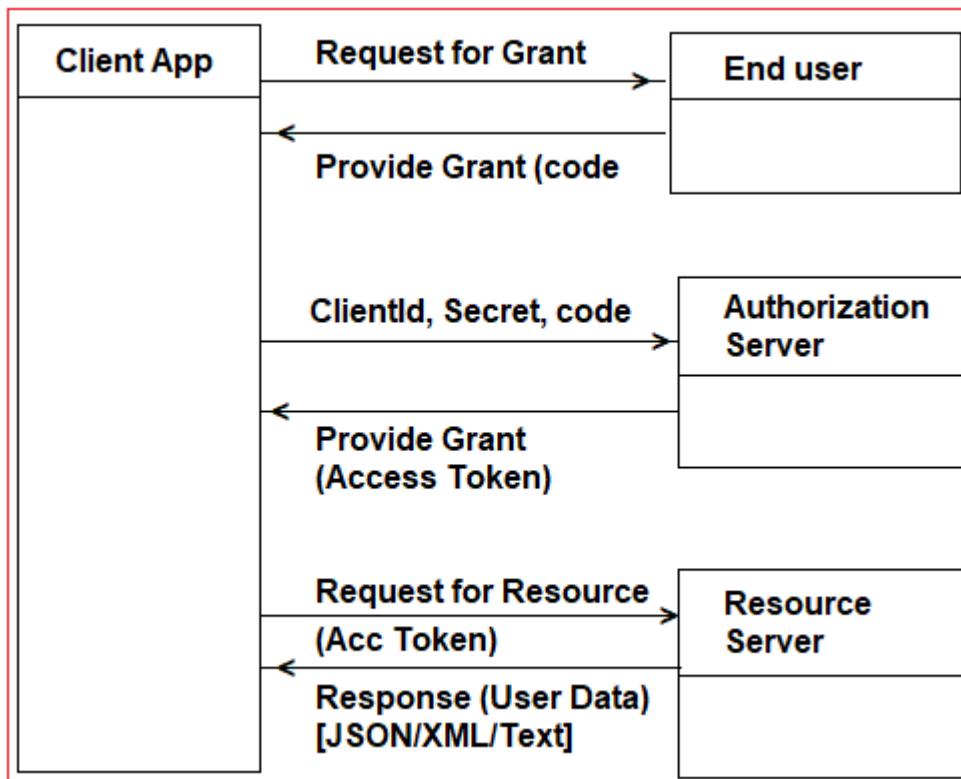
authenticationScheme: query

clientAuthenticationScheme : form

**#Resource Server details**

resource:

userInfoUri: https://graph.facebook.com/me

**Request Execution flow ::**

**NOTE:**-- ApInit is provided by Boot, no need to write.

Components Involved : ClientApp, User(Browser), Auth Server (with Token Generator and Resource Server (User data in XML/JSON)).

**Step#6:-** Define SecurityConfig class (SecurityInit is not required, Handled by spring Boot only).

=>Here Authentication Details not required configuring as we are using 3<sup>rd</sup> party security services.

=>In Authorization config specify which URLs needs type “**Every One Access**” [PermitAll].

**SecurityConfig.java:**--

```
package com.app.config;
import org.springframework.boot.autoconfigure.security.oauth2.client.EnableOAuth2Sso;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

@Configuration
@EnableOAuth2Sso
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
            .antMatchers("/", "/login").permitAll()
            .anyRequest().authenticated();
    }
}
```

**Step#7:-** Define RestController for User (or Any)

```
package com.app.controller;
import java.security.Principal;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
```

```
@RestController
```

```
public class UserRestController {
```

```
    @RequestMapping("/user")
    public Principal showUser(Principal p)
    {
        return p;
    }
    @RequestMapping("/home")
    public String showData()
    {
        return "Hello";
    }
}
```

**Step#8:- Define one UI Page**

ex:-- index.html under src/main/resource, in static folder

**Index.html:--**

```
<html>
<body>
<h1>Welcome to Login Page</h1>
<a href="user">Facebook</a>
<hr/>
<form action="#" method="post">
User : <input type="text">
Pwd : <input type="password">
<input type="submit">
</form>
</body>
</html>
```

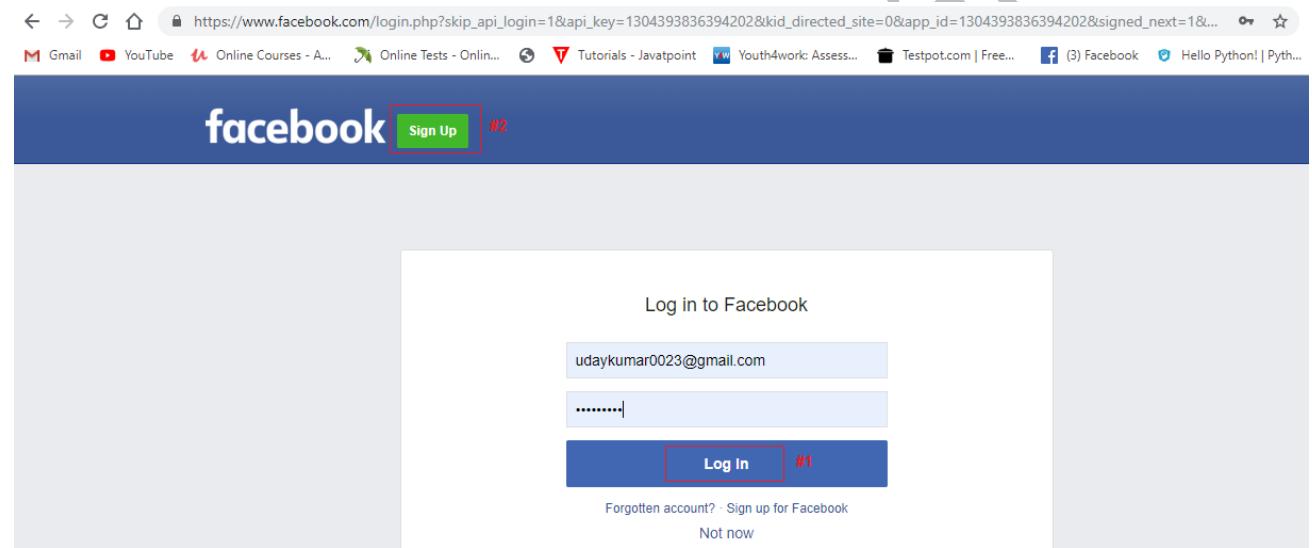
**Step#9:- Run application and Enter URL <http://localhost:9898>**

**Step#10:-** Click on facebook (#1 Process) Link and accept name (or) enter details or Else follow Second Process.



A screenshot of a web browser showing a login page. At the top, there is a navigation bar with links to Gmail, YouTube, Online Courses, Online Tests, Tutorials - Javatpoint, Youth4work, Testpot.com, and others. Below the navigation bar, the main title is "Welcome to Login Page". There is a red-bordered button labeled "Facebook #1 Process". Below it, there are input fields for "User" (containing "udaykumar0023@gmail.com") and "Pwd" (containing "\*\*\*\*\*"). To the right of the Pwd field is a "Submit" button and the text "#2 Process".

=>Click on **Login** if you have Facebook account else Click on **Sign Up** Button.



A screenshot of a web browser showing the Facebook login page at https://www.facebook.com/login.php. The page has a blue header with the Facebook logo and a green "Sign Up" button. The main form is titled "Log in to Facebook" and contains fields for "Email/Phone number" (with "udaykumar0023@gmail.com") and "Password" (with "\*\*\*\*\*"). Below the password field is a red-bordered "Log In" button with the text "#1". At the bottom of the form, there are links for "Forgotten account? · Sign up for Facebook" and "Not now".

Task :-- Google Cloud Console

=>Create app and get ClientId, Secret.

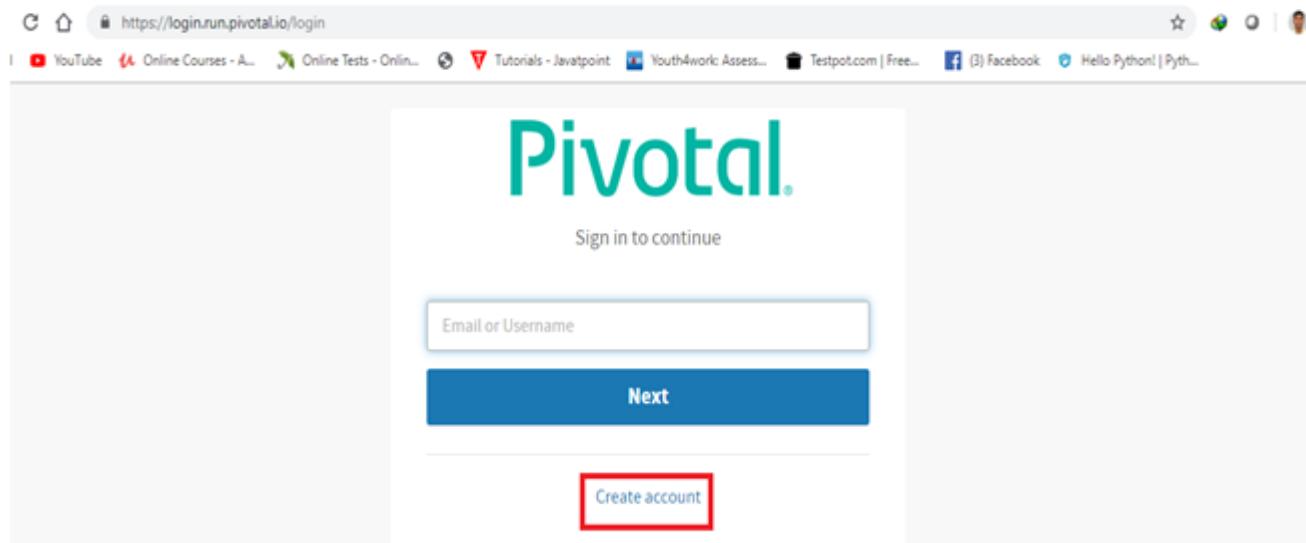
## 14. Pivotal Cloud Foundry (PCF) :-

### Spring Boot PCF deployment:-

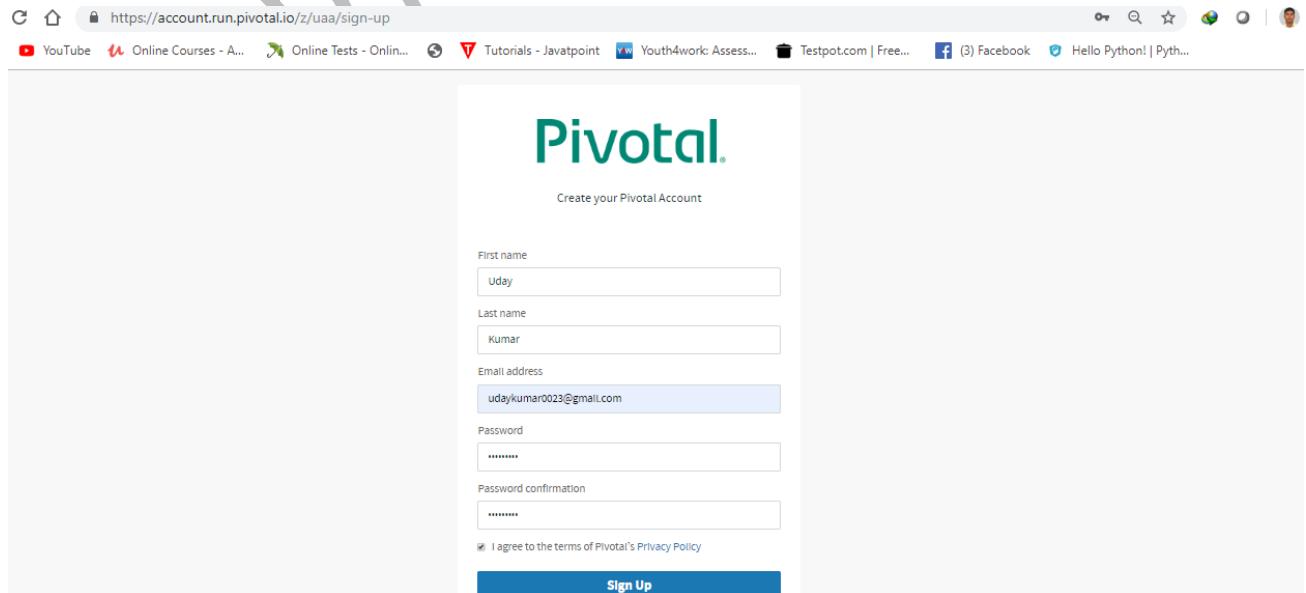
Pivotal Cloud Foundry is a Cloud Deployment Server provides service to Spring Boot and Microservices Applications mainly with all kinds of Environment setup like Databases, server, log tracers etc.

#### #PCF Account Setup#

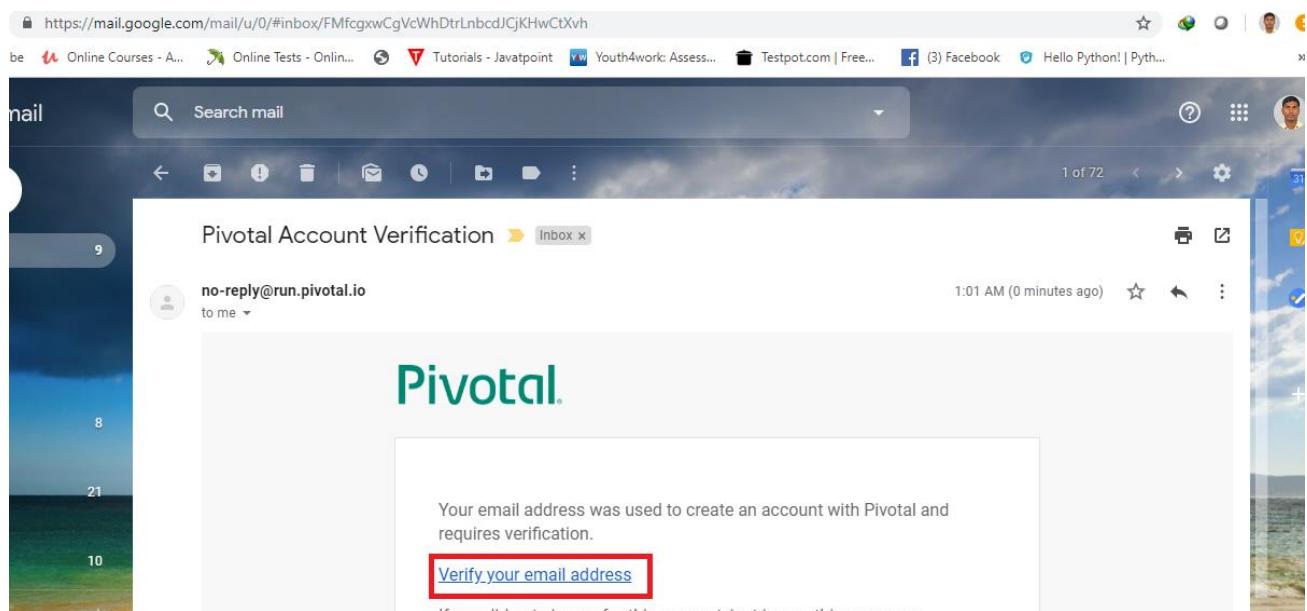
**Step#1:-** Goto <https://login.run.pivotal.io/login> And click on “Create Account”.



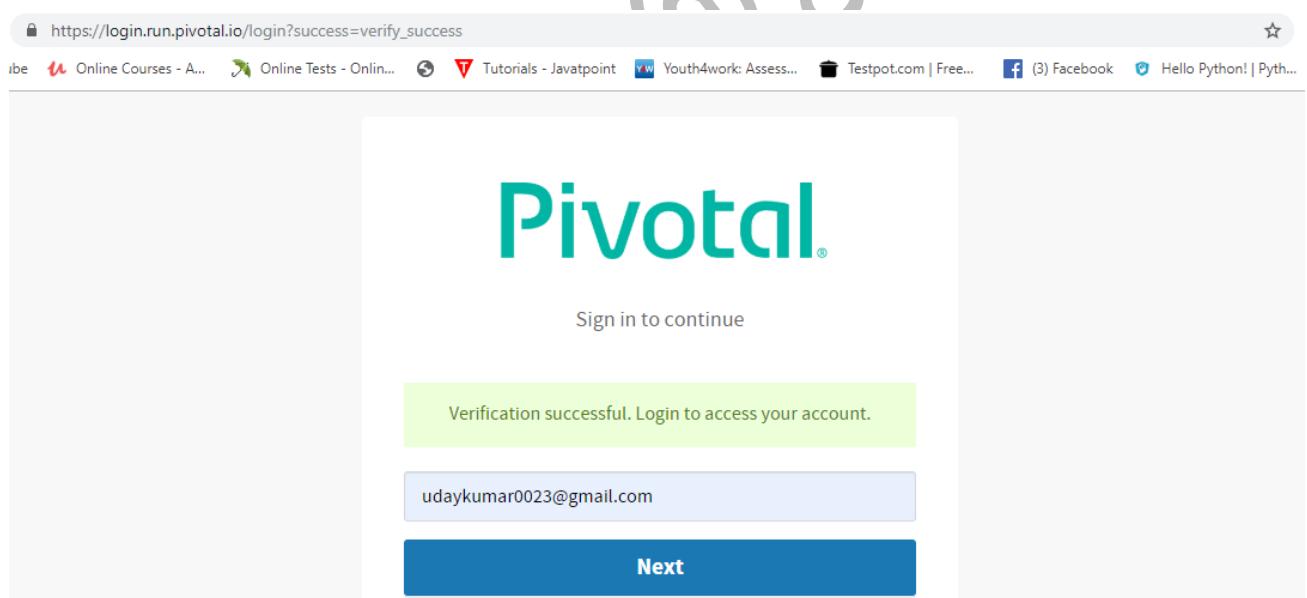
**Step#2:-** Enter details like name, email, password and conform password & Click on Sign Up.



### Step#3:- Goto Email Account and verify PCF link



### Step#4:- Login to PCF account (Above URL)



=>After login Click on “pivotal web service”

### Step#5:- Provide initial details like

=>Company name

The screenshot shows the Pivotal Web Services sign-up interface. On the left sidebar, there are links for Marketplace, Tools, Docs, Support, Blog, and Status. At the top right, there are three buttons: 'SIGN UP' (with a circled 1), 'CLAIM YOUR TRIAL' (with a circled 2), and 'CREATE ORG' (with a circled 3). The main content area has a heading 'Sign Up for your free trial'. It includes a welcome message: 'Welcome to Pivotal Web Services! Complete these steps to access your account.' Below this, there is a 'Username' field containing 'udaykumar0023@gmail.com'. A 'Company' field contains 'org.verizon'. A checkbox labeled 'I have read and agree to the Terms of Service for Pivotal Web Services' is checked. A blue button at the bottom right says 'Next: Claim Your Trial'.

=>Enter Mobile number and verify

The screenshot shows the 'Claim Your Free Trial' page. The left sidebar has links for Create a New Org, Marketplace, Tools, Docs, Support, Blog, and Status. At the top right, there are three buttons: 'SIGN UP' (with a circled 1), 'CLAIM YOUR TRIAL' (with a circled 2), and 'CREATE ORG' (with a circled 3). The main content area has a heading 'Claim Your Free Trial'. It includes a note: 'We require SMS or voice call verification for claiming free trials. Please select which method you prefer and you will receive your code momentarily.' To the right, a note states: 'Your number is only used for claiming your free trial, and will never be distributed to third-parties or used for marketing purposes.' Another note says: 'Users are limited to one free trial org per user account. If you have any issues or questions, please contact support@run.pivotal.io.' Below the note, there is a 'Verification Method' dropdown set to 'SMS', a 'Country' dropdown set to 'India', and a 'Mobile Number' input field containing '9092576623'. A blue button at the bottom right says 'Send me my code'.

**[Raghu Sir]**

**[NareshIT, Hyd]**

=>Enter OTP which is send on give Mobile

The screenshot shows a web browser window for Pivotal Web Services. The URL is [https://console.run.pivotal.io/pws/sign\\_up/verification\\_attempt/new?method=voice\\_call&resent=true](https://console.run.pivotal.io/pws/sign_up/verification_attempt/new?method=voice_call&resent=true). The page title is "Pivotal Web Services". On the left, there's a sidebar with links like "Create a New Org", "Marketplace", "Tools", "Docs", "Support", "Blog", and "Status". The main content area has a heading "Claim Your Free Trial". It contains a note about phone number restrictions in India and a form where the user has entered the code "739682". There are buttons for "SIGN UP", "CLAIM YOUR TRIAL", and "CREATE ORG". To the right, there's a note about the number being used for the trial and another note about users being limited to one trial org per account.

=>Organization (org) name > finish

The screenshot shows a web browser window for Pivotal Web Services. The URL is <https://console.run.pivotal.io/organizations/new>. The page title is "newOrg - Pivotal Web Services". The sidebar on the left includes "Home", "Marketplace", "Tools", "Docs", "Support", "Blog", and "Status". The main content area has a heading "Create an Org" and a form where the user has entered "Vzot-Authorization-Service" into the "Org (or Project) Name" field. There are "CREATE ORG" and "CANCEL" buttons at the bottom right. A note below the input field explains what an organization account is and a note below that says the name can be changed later.

## Step#6:- Download and setup PCF CLI

- >Click on “Tools” option
- >Choose OS and Bit and Download Software

Pivotal Web Services

Home

Marketplace #1

**Tools**

Docs

Support

Blog

Status

GIVE FEEDBACK

Search apps, services, spaces, & orgs

## Tools

Cloud Foundry CLI for pushing and managing apps, creating and binding services, and more. For more info visit the cf documentation.

Download and Install the CLI #2

Download for Windows 64 bit

CLI Basics

Login to the CLI

```
$ cf login -a api.run.pivotal.io
```

Get help in the CLI

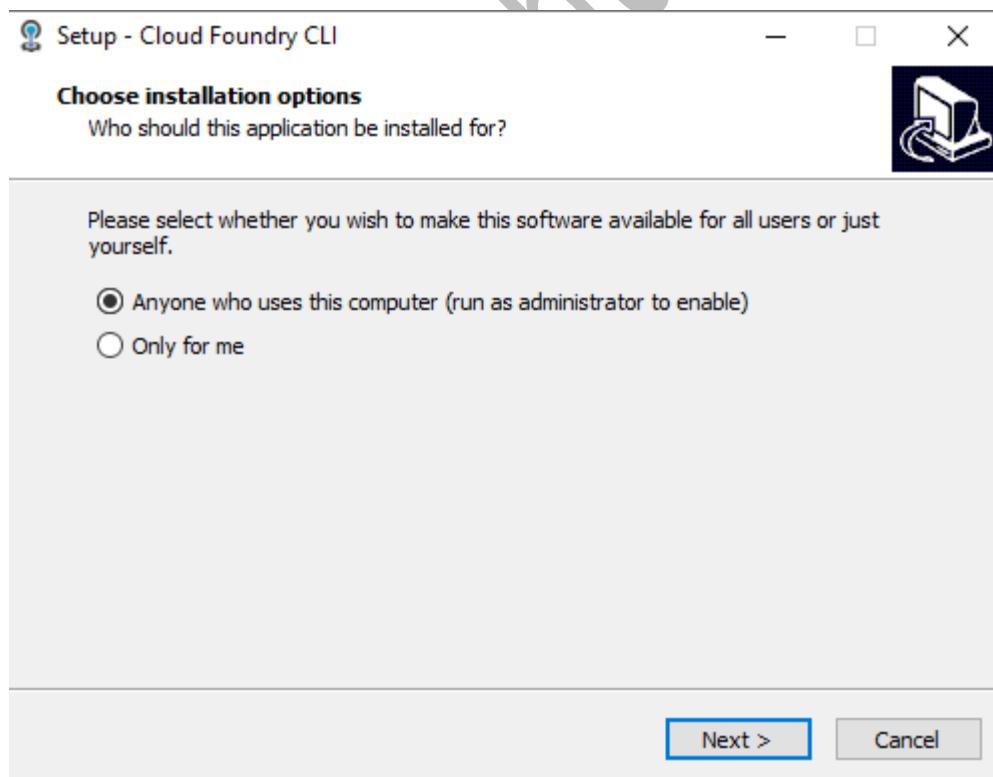
```
$ cf help
```

Push an application

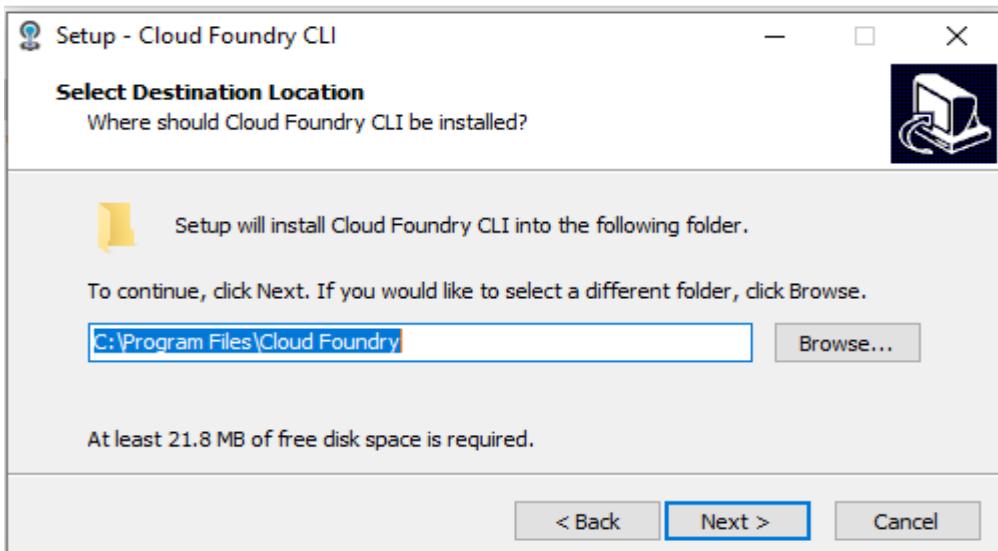
```
$ cf push your_app
```

[View the getting started tutorial](#)

- >Extract ZIP into Folder
- >Click on “cf\_installer.exe”.



=>Click on Next



=>>next > next > Finish

### Step#7:- Login and Logout Commands

=>Goto cmd prompt

1>Login command is

cf login -a api.run.pivotal.io

Email >

Password >

2>Logout command is

cf logout

```
Command Prompt
Microsoft Windows [Version 10.0.18342.8]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Uday>cf login -a api.run.pivotal.io
API endpoint: api.run.pivotal.io

Email> udaykumar0023@gmail.com

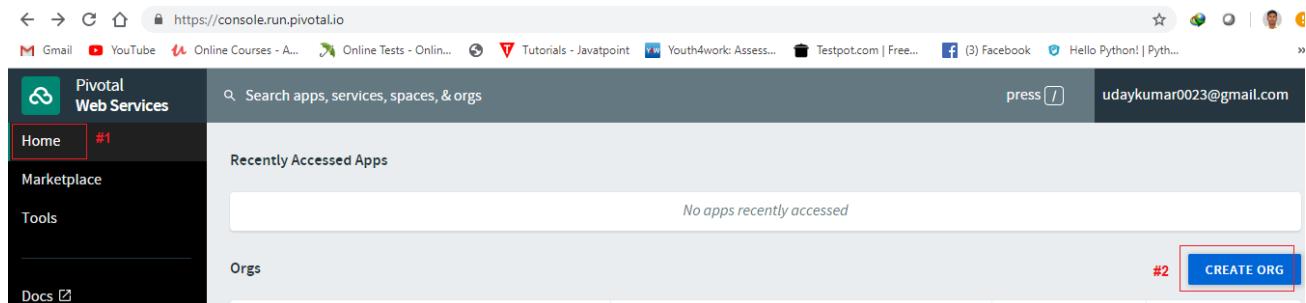
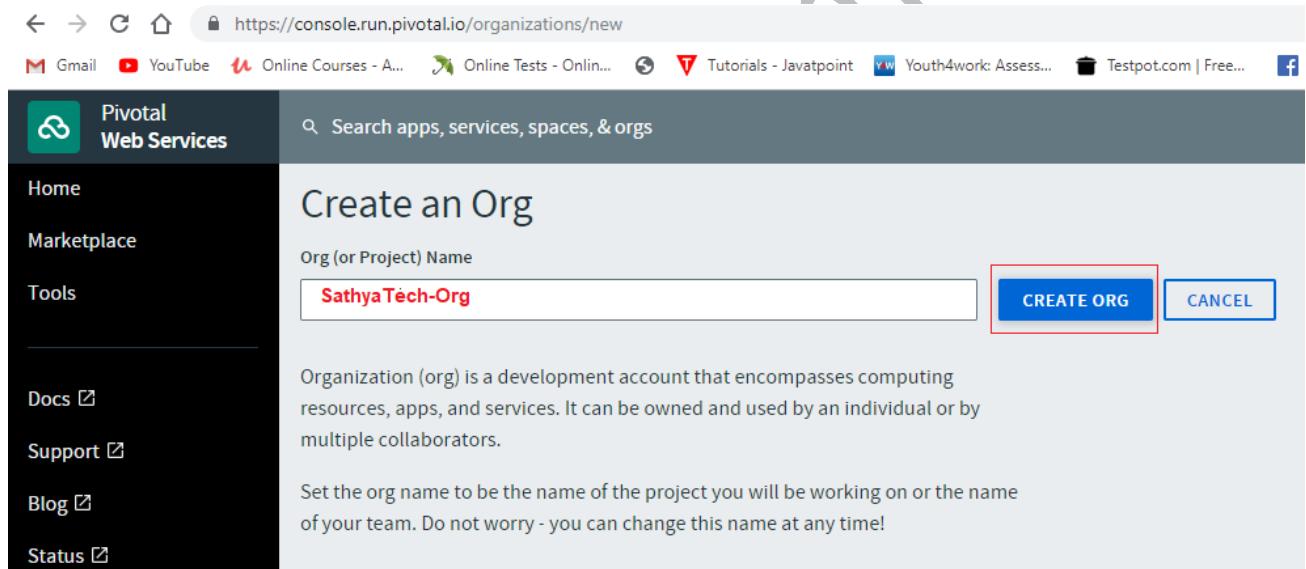
Password>
Authenticating...
OK

Targeted org Vzot-Authorization-Service

Targeted space development

API endpoint: https://api.run.pivotal.io (API version: 2.134.0)
User: udaykumar0023@gmail.com
Org: Vzot-Authorization-Service
Space: development

C:\Users\Uday>cf logout
Logging out udaykumar0023@gmail.com...
OK
```

**Step#8:- Create Org and space in PCF****=>Login to PCF using browser****=>Click on home > Click on “CREATE ORG” button****=>Enter org name ex : App-org****=>Click on “CREATE ORG”.**

**\*\* An org “sample-org” is created with default space (workspace) “development” is created.**

**->A space (workspace) is a folder or location where project is stored and running.**

**Step#9:- Create one Spring Boot Starter Project in STS.****=>File > new > Spring Starter Project****=>Enter Details > choose Dependencies (Ex: web)****=>Finish.**

**Step#10:-** Write Code for RestControllers, Services, Dao, and properties/yml files etc.

**Step#11:-** Clean Application

=>Right click > Run As > Maven Clean

...Wait for status BUILD SUCCESS...

=>This step will clear all old folders and files which are in “project-name/target” folder.

**Step#12:-** Do setup of JDK in workspace

>Windows > Preferences

>search with “Installed JRE”

>Click on “Add” > browse for location

Ex: “C:\Program Files\Java\jdk1.8.0\_201”

>Choose new and delete old JRE

>Apply and close

**Step#13:-** Update JRE to Project

=>Right click on Project > build path

=>Configure build path > Choose JRE System Lib.

=>Edit > select workspace JRE/JDK

=>Apply and close.

**Step#14:-** Install Application (build project)

=>Right click > Run As > Maven install

..wait for few minutes, still BUILD SUCCESS..

\*\*(If failed, then Update Maven Project, select force Update and finish.)

Repeat step#11, 13 and then 14)

**Step#15:-** Refresh Project to see updates in “target” folder.

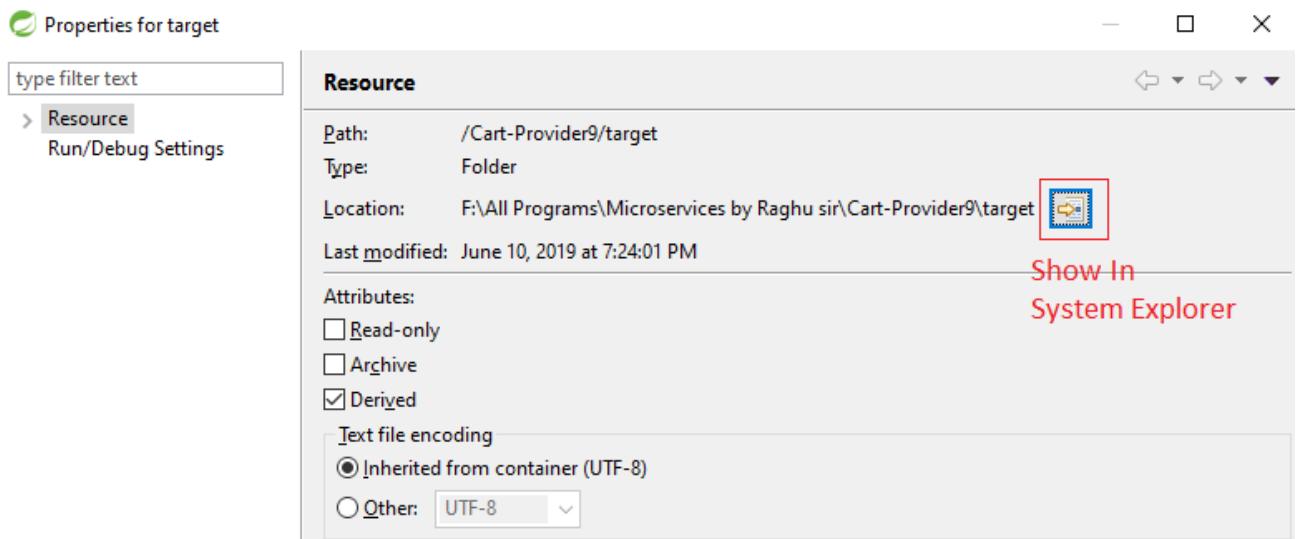
=>Right click on Project > refresh (F5)

.. then..

=>Right click on target folder

=>Properties (or Alt + Enter)

=>click on “Show In System Explorer”



=>Open target folder > find --.jar file  
Ex:-- (SpringBootSampleApp-1.0.jar)

**Step#16:-** Open commandprompt and login to PCF from here

=>shift + mouse right click  
=>choose “open cmd window here”  
=>Login to PCF

1>cmd/>cf login –a api.run.pivotal.io (F:\All Programs\Microservices by Raghu sir\Cart-Provider9\target>cf login -a api.run.pivotal.io)  
=>Enter Email Id and password (which is used for login pcf account)

2>Select ORG in pivotal cloud (Where Vzot-Authoriztion-Service is a Org in PCF account)  
Ex:-- cf target -o Vzot-Authorization-Service ( F:\All Programs\Microservices by Raghu sir\Cart-Provider9\target>cf target -o Vzot-Authorization-Service)

3>Push Application to created org & space

Cmd/>cf push [Project-name] –p [JARNAME].jar

Syntax:-- cf push app –p SpringBootSampleApp-1.0.jar

Ex:-- cmd>F:\All Programs\Microservices by Raghu sir\Cart-Provider9\target>cf push Cart-Provider9 -p Cart-Provider9-1.0.jar

.. wait for 5 minutes to upload project..

=>Screen for Process 1-3 command (Login-Push Application)

```
F:\All Programs\Microservices by Raghu sir\Cart-Provider9\target>cf login -a api.run.pivotal.io
API endpoint: api.run.pivotal.io

Email> udaykumar0023@gmail.com

Password>
Authenticating...
OK

Select an org (or press enter to skip):
1. SathyaTech-org
2. Vzot-Authorization-Service

F:\All Programs\Microservices by Raghu sir\Cart-Provider9\target>cf target -o Vzot-Authorization-Service
api endpoint: https://api.run.pivotal.io
api version: 2.136.0
user: udaykumar0023@gmail.com
org: Vzot-Authorization-Service
space: development

F:\All Programs\Microservices by Raghu sir\Cart-Provider9\target>cf push Cart-Provider9 -p Cart-Provider9-1.0.jar
Pushing app Cart-Provider9 to org Vzot-Authorization-Service / space development as udaykumar0023@gmail.com...
Getting app info...
Updating app with these attributes...
  name:           Cart-Provider9
  path:          F:\All Programs\Microservices by Raghu sir\Cart-Provider9\target\Cart-Provider9-1.0.jar
  command:        JAVA_OPTS="-agentpath:$PWD/.java-buildpack/open_jdk_jre/bin/jvmkill-1.16.0_RELEASE=printHeapHist
essorCount=$(nproc) -Djava.ext.dirs=$PWD/.java-buildpack/container_security_provider:$PWD/.java-buildpack/open_jdk_jre/b
uildpack/java_security/java.security $JAVA_OPTS" && CALCULATED_MEMORY=$(($PWD/.java-buildpack/open_jdk_jre/bin/java-bu
ildpack/memory-calculator-memory-calculator.sh -loadedClasses=18393 -poolType=metaspace -stackThreads=250 -vmOptions="$JAVA_OPTS") && echo JVM Memory
="$JAVA_OPTS $CALCULATED_MEMORY" && MALLOC_ARENA_MAX=2 SERVER_PORT=$PORT eval exec $PWD/.java-buildpack/open_jdk_jre/bi
k.boot.loader.JarLauncher
  disk quota:    1G
  health check type: port
  instances:    1
  memory:       1G
  stack:         cflinuxfs3
  routes:
    cart-provider9.cfapps.io

Updating app Cart-Provider9...
Mapping routes...
Comparing local files to remote cache...
Packaging files to upload...

Waiting for app to start...

name:           Cart-Provider9
requested state: started
routes:          cart-provider9.cfapps.io
last uploaded:   Mon 10 Jun 23:32:19 IST 2019
stack:          cflinuxfs3
buildpacks:
  client-certificate-mapper=1.8.0_RELEASE container-security-provider=1.16.0_RELEASE
  java-buildpack=      -offline-https://github.com/cloudfoundry/java-buildpack.git#3f4eee2
  jvmkill-agent=1.16.0_RELEASE open-jdk-...
  $PWD/. org.springframework.boot.loader.JarLauncher

      state   since          cpu     memory      disk      details
#0   running  2019-06-10T18:02:42Z  167.5%  189.9M of 1G  147.5M of 1G
```

**Step#17:-** Once success then goto PCF WebConsole (browser) and click on SPACE "development"

=>To see total apps and running app.

=>Click on Route (URL) to execute app

=>Enter Paths to URL ex: /cart/info ...

**Step#18:-** Logout PCF from cmd prompt

Cmd/> cf logout

\*\*\* BUILD FAIL at CLEAN

Then Force Update maven Project

=>Project > alt + F5 > choose Force update

=>Apply and close [finish]

\*\*\* BUILD FAIL at INSTALL

Then update JDK new version, in place of JRE

\*\*\* showing Cached Errors

=>Goto .m2 folder and delete that folder

=>come back to project > force update project

=>Create Project with MySQL Db with CURD operations and upload to PCF

=>Goto Market place and choose mysql provider enter details

Instance name : myappsql

Choose or/space:

>choose plan > finish

=>Come to Home >Click on org > space > service

=>click on MySQL services > Bind App >

=>select Our Project > bind > re-start app

=>Enter URL in browser and execute.

**Note:**-- Every log line contains four fields:

- ❖ Timestamp
- ❖ Log type
- ❖ Channel
- ❖ Message

### 15. Spring Boot Actuator:-

It provides “**Production Read Endpoints**” which are used to give extra information of Production Server.

=>In simple readymade service used for production server to “**Track and Trace Extra information**” from server.

=>It must be enabled by Programmer and used by Admin/Production Team.

**Production Server**:- A server having application that provide service to end user is known as **Production server**.

=>In case of production, we need some times information of it like,

a>DB Details

b>Beans (Objects) Created

c>Memory (Head, ThreadDump, ...etc) details.

d>Cache Managements...

e>Is app Connected to any Remote networks or tools (Printer, Scanner...)

f>Logfiles and Log Level updates messages etc...

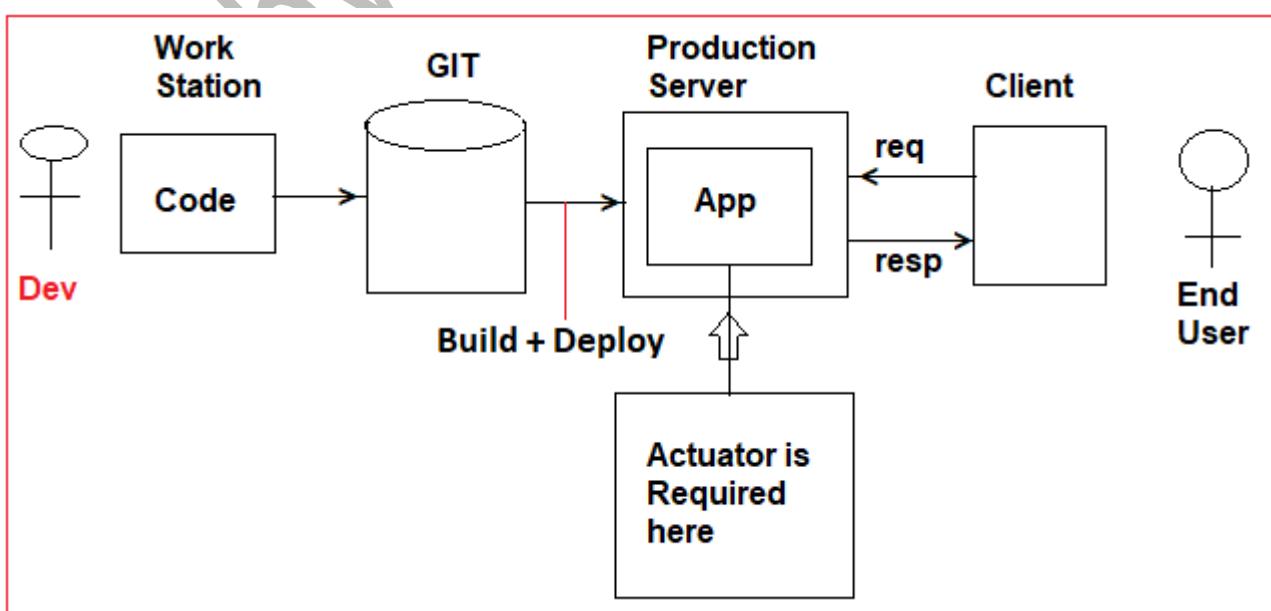
### Actuator Dependencies:-

```
<dependency>
```

```
    <groupId>org.springframework.boot</groupId>
```

```
    <artifactId>spring-boot-starter-actuator</artifactId>
```

```
</dependency>
```



=>By default Actuator enabled 2 endpoints those are :

1>/actuator/health

2>/actuator/info

=>To enable all endpoints use code **management.endpoints.web.exposure.include=\***  
[In application.properties.]

### **Coding Step:--**

**Step#1:-** Create one starter project with one web and Actuator dependencies.

**Step#2:-** pom.xml:--

spring-boot-starter-web

spring-boot-starter-actuator

**Step#3:-** Define one Rest controller

**Step#4:-** application.properties

server.port=9988

management.endpoints.web.exposure.include=\*

**Step#5:-** Run application and enter URL

1>http://localhost:9988/actuator/info

2>http://localhost:9988/actuator/health

3>http://localhost:9988/actuator/beans

4>http://localhost:9988/actuator/env

Etc. (few more: /httptrace, /threaddump, /caches)

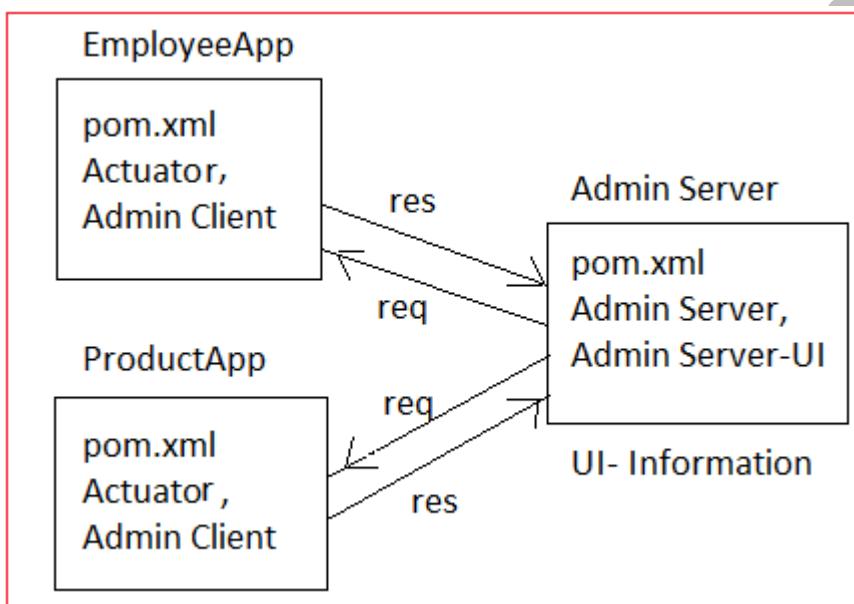
## 16. Spring Admin UI (Codecentric):--

Only Actuator if we implement in MicroServices then all details we should check manually. These all are made “**ADMIN UI- Track and Trace**” using Spring Boot Admin UI application.

=>Actuator provides all endpoints we need to enter URL using Http client (browser/POSTMAN) manually, which is time consuming process and big task to admin team.

=>To avoid this manual approach use admin setup for actuators, which is provided by 3<sup>rd</sup> party service “**de.codecentric**”?

=>For this every Microservices should have **actuator** and **admin client** dependencies and one Application with **admin server** and **admin-server-ui** dependencies.



### Spring boot Admin Server setup:--

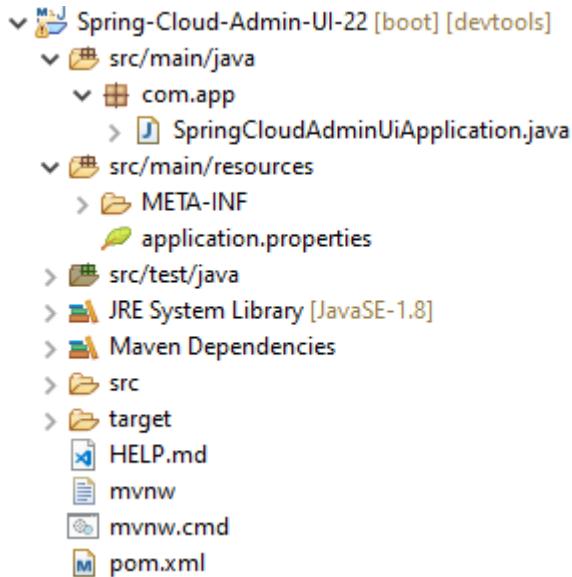
**Step#1:-** Create Spring Boot Starter app with dependencies : admin server and ui.

```

<dependency>
    <groupId>de.codecentric</groupId>
    <artifactId>spring-boot-admin-starter-server</artifactId>
</dependency>
<dependency>
    <groupId>de.codecentric</groupId>
    <artifactId>spring-boot-admin-server-ui</artifactId>
</dependency>

```

## #24. Folder Structure of Admin UI Dashboard:--

**pom.xml:--**

```

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.1.RELEASE</version>
        <relativePath /> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.app</groupId>
    <artifactId>Spring-Cloud-Admin-UI</artifactId>
    <version>1.0</version>
    <name>Spring-Cloud-Admin-UI</name>
    <description>Spring Cloud Admin Dashboard Implementation</description>
    <properties>
        <java.version>1.8</java.version>
        <spring-boot-admin.version>2.1.4</spring-boot-admin.version>
    </properties>
    <dependencies>
  
```

```
<dependency>
    <groupId>de.codecentric</groupId>
    <artifactId>spring-boot-admin-starter-server</artifactId>
</dependency>
<dependency>
    <groupId>de.codecentric</groupId>
    <artifactId>spring-boot-admin-server-ui</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-config</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-web</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
</dependencies>
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>de.codecentric</groupId>
            <artifactId>spring-boot-admin-dependencies</artifactId>
            <version>${spring-boot-admin.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
```

```

        </dependencies>
    </dependencyManagement>
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>

```

**Step#2:-** At starter class level provide annotation `@EnableAdminServer`

```

package com.app;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import de.codecentric.boot.admin.server.config.EnableAdminServer;

```

```

@SpringBootApplication
@EnableAdminServer
public class SpringCloudAdminUiDashboardsApp {
    public static void main(String[] args) {
        SpringApplication.run(SpringCloudAdminUiDashboardsApp.class, args);
    }
}

```

**Step#3:-** Provide details in application.properties

```
server.port=9999
```

### Spring Boot (Any Microservice) Client App:--

**Step#1:-** Create Spring Boot starter app with dependencies : web, admin client, Actuator.

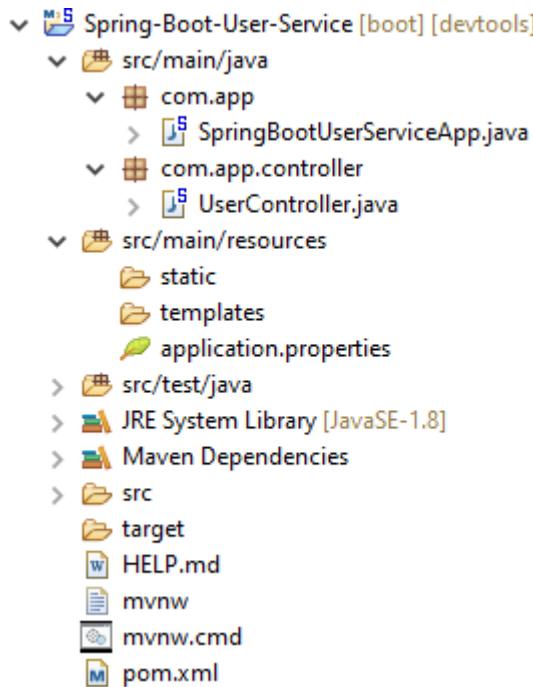
#### Admin client dependency:--

```

<dependency>
    <groupId>de.codecentric</groupId>
    <artifactId>spring-boot-admin-starter-client</artifactId>
</dependency>

```

## #25. Folder structure of User-Service:--

**pom.xml:--**

```

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.1.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.app</groupId>
  <artifactId>Spring-Boot-User-Service</artifactId>
  <version>1.0</version>
  <name>Spring-Boot-User-Service</name>
  <description>Demo project for Spring JMS Implementation</description>

  <properties>
    <java.version>1.8</java.version>
  
```

```
<spring-boot-admin.version>2.1.5</spring-boot-admin.version>
<spring-cloud.version>Greenwich.SR1</spring-cloud.version>
</properties>

<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>de.codecentric</groupId>
        <artifactId>spring-boot-admin-starter-client</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>${spring-cloud.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>

```

```

<dependency>
    <groupId>de.codecentric</groupId>
    <artifactId>spring-boot-admin-dependencies</artifactId>
    <version>${spring-boot-admin.version}</version>
    <type>pom</type>
    <scope>import</scope>
</dependency>
</dependencies>
</dependencyManagement>

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

```

**Step#2:-** provide Server details actuator endpoints in application.properties file.

```

server.port=6666
spring.boot.admin.client.url=http://localhost:9999
management.endpoints.web.exposure.include=*
spring.application.name=USER-PROVIDER
eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka

```

**Step#3:-** Starter class of User Microservice

```

package com.app;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

@SpringBootApplication
@EnableDiscoveryClient

```

```
public class SpringBootUserServiceApp {
```

```
    public static void main(String[] args) {
        SpringApplication.run(SpringBootUserServiceApp.class, args);
        System.out.println("User Microservice Executed...");
    }
}
```

**Step#4:- Define one RestController (even service, DAL...)**

```
package com.app.controller;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.bind.annotation.RequestMapping;
```

```
@RestController
@RequestMapping("user")
public class UserController {
    @RequestMapping("show")
    public String showUser() {
        System.out.println("Hello User");
        return null;
    }
}
```

**Execution Order:--**

- 1>Run Admin Server
- 2>Then run ClientApps (SpringBootUserServiceApp)
- =>Enter URL : <http://localhost:9999>

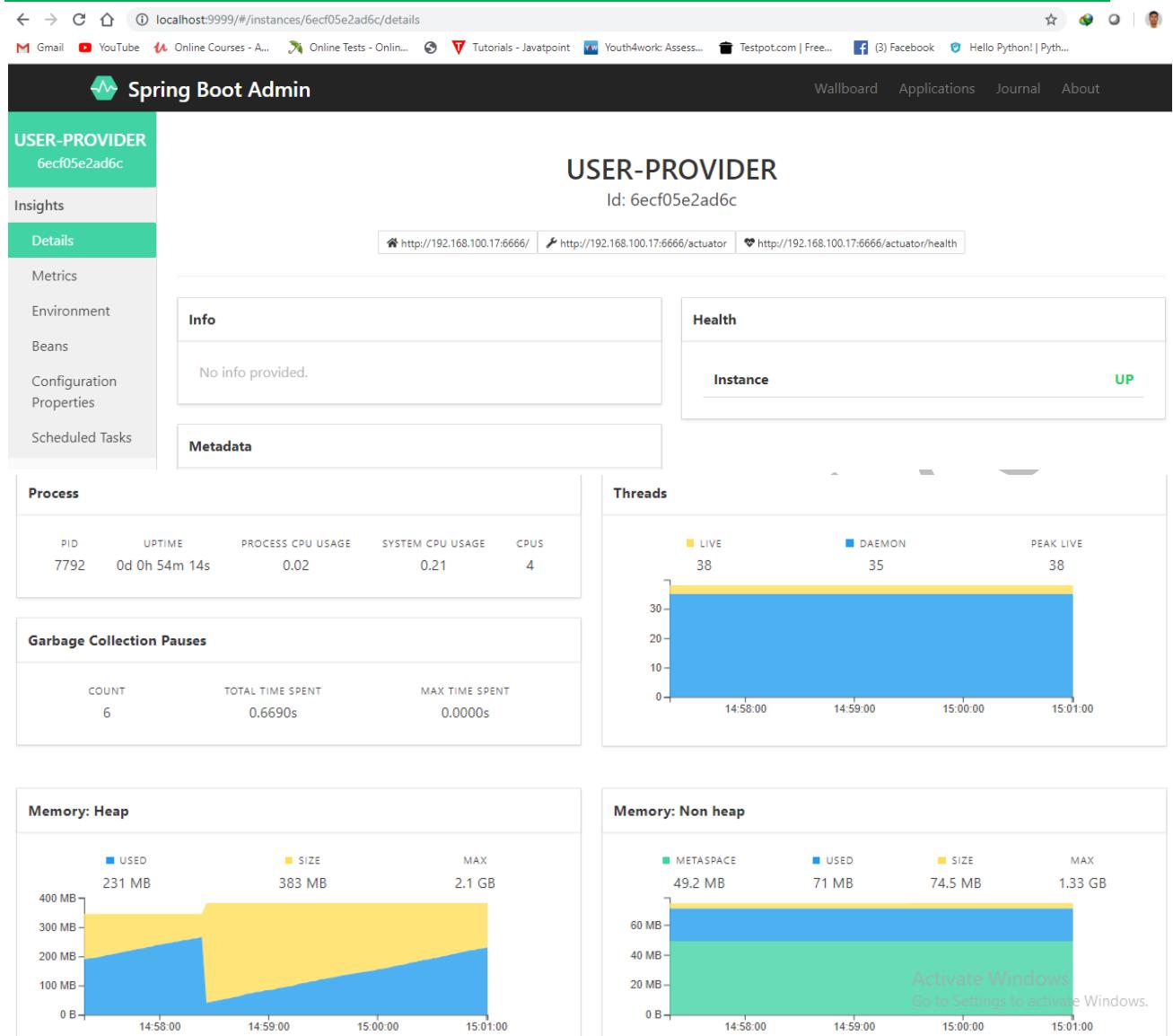
**Output Screen:--**

The screenshot shows the Spring Boot Admin dashboard. At the top, there's a navigation bar with links like Gmail, YouTube, Online Courses, Online Tests, Tutorials - Javatpoint, Youth4work: Assess..., Testpot.com | Free..., Facebook, and Hello Python! | Pyth... Below the navigation is a dark header bar with the text "Spring Boot Admin" and links for Wallboard, Applications, Journal, and About.

The main content area has three columns: APPLICATIONS, INSTANCES, and STATUS. The APPLICATIONS column shows the number 1. The INSTANCES column shows the number 1. The STATUS column shows "all up".

Below these columns is a table with two rows. The first row is a header with columns "UP" and "NAME". The second row contains a single entry for "USER-PROVIDER". This entry includes a green checkmark icon, the text "48m", and the URL "http://192.168.100.17:6666/". The entire row for "USER-PROVIDER" is highlighted with a red border.

=>Click on “User Provider” Service to see full details.



**NOTE:-- Add bellow all dependencies into all microservices.**

**1>Add Discovery client dependency in Microservice (Consumer, Producer, Config Server, Zuul Server) to register with Eureka Server.**

<dependency>

    <groupId>org.springframework.cloud</groupId>

    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>

</dependency>

**2>Add Config-Client dependency in (Consumer, Provider, Zuul Server) to read properties from Config server.**

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-config-client</artifactId>
</dependency>
```

**3>To Register with Admin Server add below dependency in every application (Consumer, Provider even if Eureka Server, Config Server, ZUUL Server).**

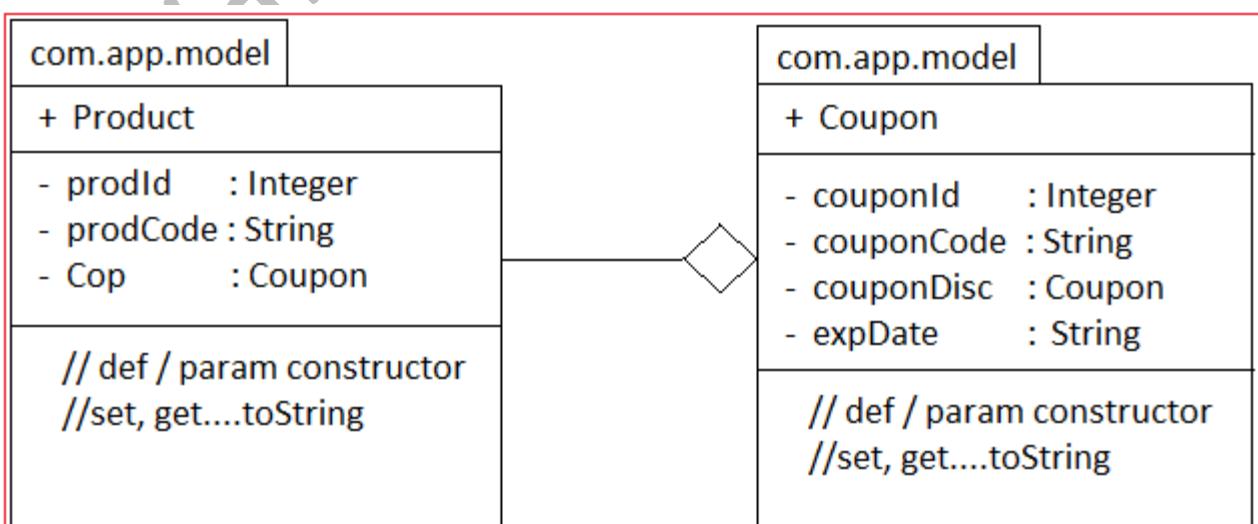
```
<dependency>
    <groupId>de.codecentric</groupId>
    <artifactId>spring-boot-admin-starter-client</artifactId>
</dependency>
```

**4>To Provide Production ready Endpoint add actuator in Consumer Application.**

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

### Microservice Implementation Task:---

Product----->>>Coupon



**NOTE:**--1.>Implement two Micro-Service

1> CouponServiceProvider

2> ProductServiceProvider

2.>CouponService ProductShould take JSON from RestClient (POSTMAN) and create sample in Database.

{

```
"couponId" : 8765,
"couponCode" : "INDAUG15",
"couponDisc" : 15,
"expDate" : 17/08/2019
```

}

=>Operation (Methods) : [Use Data JPA]

a>Create Coupon and Success Message

b>Get Coupon (full Object) byCuponCode, byCouponId

3.>Create Product using Coupon code only

{

```
"prodId" : 999,
"prodCode" : "PENDRIVE",
"prodCost" : 876.56,
"coupon" : {"code" : "INDAUG15"}
```

}

### Operations:--

a.>Verify coupon code is valid or not also get Coupon Object.

b.>Calculate finalAmt = prodCost – discAmount

=>Store all details in database.

c.>Fetch all Product, fetch all Coupons

=>ProductRestController ----->CouponRestConsumer(Feign Client)

4.>Coupon App, Product App should be registered with eureka Server.

5.>Common properties keys must be located from config Server.

6.>Implement Hystrix for Product save (...).

7.>Enable Load Balancing for Coupon App (Multiple Instances).

8.>Enable Zuul API Gateway for both Apps.

**16. MicroService Task:-- Task with Eureka, Config server, Zuul server, Swagger, Caching, Feign Client, Validation Of Data, custom error messages.**

### 1. EurekaServer Folder Structure:--

```

    SpringCloud_Task-EurekaServer [boot]
      src/main/java
        com.app
          SpringCloudTaskEurekaServerApp.java
      src/main/resources
        application.properties
      src/test/java
      JRE System Library [JavaSE-1.8]
      Maven Dependencies
      src
      target
        HELP.md
        mvnw
        mvnw.cmd
      pom.xml
  
```

#### Step#1:- application.properties

```

server.port=8761
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
  
```

#### Step#2:-Eureka Server Starter class

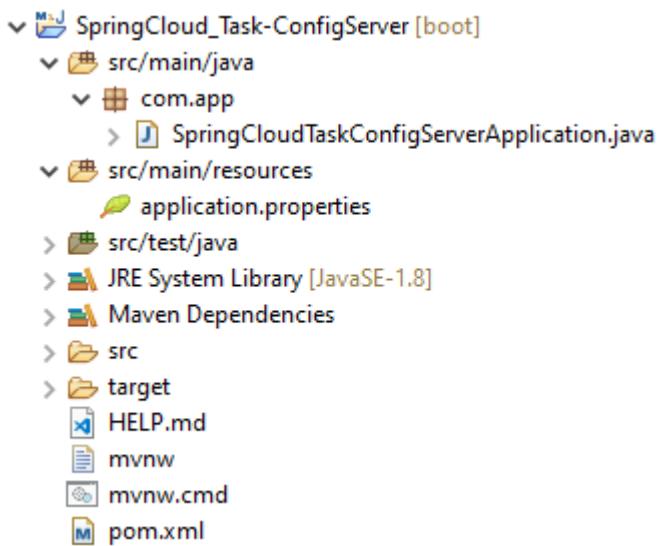
```

package com.app;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@SpringBootApplication
@EnableEurekaServer
public class SpringCloudTaskEurekaServerApp {

    public static void main(String[] args) {
        SpringApplication.run(SpringCloudTaskEurekaServerApp.class, args);
        System.out.println("Eureka Server Starter...!!!");
    }
}
  
```

## 2. ConfigServer Folder Structure:--



### 1>application.properties

```

# Recommended port
server.port = 8888
# External config file link
spring.cloud.config.server.git.uri=https://github.com/saurabh-vaish/MicroserviceTask
#spring.cloud.config.server.git.uri=https://gitlab.com/udaykumar0023/configserverex

```

### 2>Starter class of Config Server

```

package com.app;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.config.server.EnableConfigServer;

@SpringBootApplication
@EnableConfigServer
public class SpringCloudTaskConfigServerApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringCloudTaskConfigServerApplication.class, args);
        System.out.println("Config Server Executed...!!!!");
    }
}

```

**External Config Server (Git) application.properties file:--**

server.port=8888

eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka

## database connection (Oracle)

spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver

spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe

spring.datasource.username=system

spring.datasource.password=system

## connection pooling

### max connections

spring.datasource.hikari.maximum-pool-size=12

### if no connection allocated

spring.datasource.hikari.connection-timeout=20000

### life time of sql

spring.datasource.hikari.max-lifetime=1200000

### idle connections

spring.datasource.hikari.minimum-idle=5

### if no activity then idle

spring.datasource.hikari.idle-timeout=30000

### if no activity then idle

spring.datasource.hikari.auto-commit=true

## jpa properties

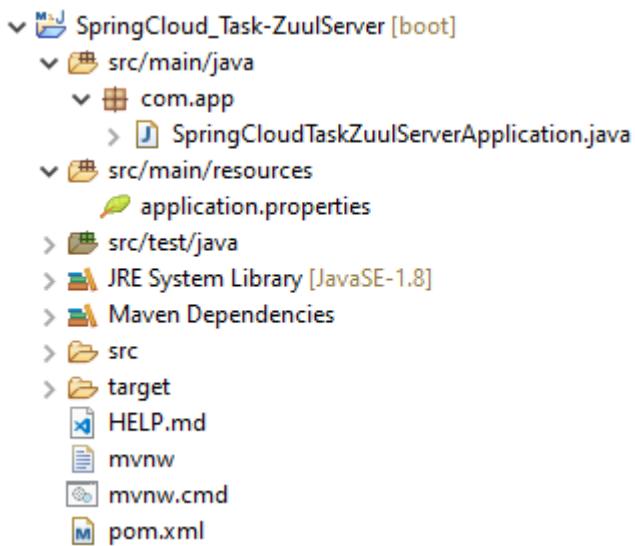
spring.jpa.hibernate.ddl-auto=update

spring.jpa.properties.hibernate.show\_sql=true

spring.jpa.properties.hibernate.format\_sql=true

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.Oracle10gDialect

### 3. ZuulServer Folder Structure:--



#### 1>application.properties:--

```
# zuul server port
server.port=6565
# service id for eureka server
spring.application.name=ZUUL_PROXY
# service url
eureka.client.service-url.default-zone=http://localhost:8761/eureka
# routing for coupan
zuul.routes.coupan.serv.service-id=COUPAN-APP
zuul.routes.coupan.serv.path=/coupan-api/**
# routing for product
zuul.routes.productserv.service-id=PRODUCT-APP
zuul.routes.productserv.path=/prod-api/**
```

#### 2>Starter class of Zuul Server

```
package com.app;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.zuul.EnableZuulProxy;

@SpringBootApplication
@EnableZuulProxy
public class SpringCloudTaskZuulServerApplication {
```

```

    public static void main(String[] args) {
        SpringApplication.run(SpringCloudTaskZuulServerApplication.class, args);
        System.out.println("Zuul Server Executed...!!!");
    }
}

```

#### 4. CouponService Folder Structure:--

SpringCloud\_Task-CouponProvider [boot] [devtools]

- src/main/java
  - com.app
    - SpringCloudTaskCouponProviderApp.java
  - com.app.config
    - CacheConfig.java
    - SwaggerConfig.java
  - com.app.exception
    - ApiError.java
    - CouponNotFoundException.java
    - ValidationError.java
  - com.app.model
    - Coupon.java
  - com.app.repo
    - CouponRepository.java
  - com.app.rest.controller
    - CouponController.java
    - ExceptionController.java
  - com.app.service
    - CouponService.java
  - com.app.service.impl
    - CouponServiceImpl.java
  - com.app.validator
    - CouponValidator.java
- src/main/resources
  - application.properties
- src/test/java
- JRE System Library [JavaSE-1.8]
- Maven Dependencies
- target/generated-sources/annotations
- target/generated-test-sources/test-annotations
- src
- target
  - HELP.md
  - mvnw
  - mvnw.cmd
  - pom.xml

pom.xml:--

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.7.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.app</groupId>
  <artifactId>SpringCloud_Task-CouponProvider</artifactId>
  <version>1.0</version>
  <name>SpringCloud_Task-CouponProvider</name>
  <description>Microservices project for Spring Boot</description>

  <properties>
    <java.version>1.8</java.version>
    <spring-cloud.version>Greenwich.SR2</spring-cloud.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
```

```
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-config</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-hystrix-dashboard</artifactId>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.46</version>
    <!-- $NO-MVN-MAN-VER$ -->
    <scope>runtime</scope>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
    <optional>true</optional>
</dependency>
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
</dependency>
```

```
<!-- Swagger -->
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.7.0</version>
</dependency>
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.7.0</version>
</dependency>
<!-- hazel cast caching -->
<dependency>
    <groupId>com.hazelcast</groupId>
    <artifactId>hazelcast</artifactId>
</dependency>
<dependency>
    <groupId>com.hazelcast</groupId>
    <artifactId>hazelcast-spring</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>com.oracle</groupId>
    <artifactId>ojdbc6</artifactId>
    <version>11.2.0</version>
</dependency>
</dependencies>

<dependencyManagement>
    <dependencies>
        <dependency>
```

```

<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-dependencies</artifactId>
<version>${spring-cloud.version}</version>
<type>pom</type>
<scope>import</scope>
</dependency>
</dependencies>
</dependencyManagement>
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
</project>

```

**1>application.properties:--**

```

server.port=7777
# service Id
spring.application.name=COUPON-APP
# service-url
eureka.client.service-url.default-zone=http://localhost:8761/eureka
#load balancing
eureka.instance.instance-id=${spring.application.name}:${random.value}
# no need to write for config server port 8888
#spring.cloud.config.uri=http://localhost:8888
#spring.profiles.include=dev

```

**2>Starter class:--**

```

package com.app;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cache.annotation.EnableCaching;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;

```

```
import org.springframework.cloud.netflix.hystrix.EnableHystrix;  
@SpringBootApplication  
@EnableEurekaClient  
@EnableCaching  
@EnableHystrix  
public class SpringCloudTaskCoupanProviderApp {  
  
    public static void main(String[] args) {  
        SpringApplication.run(SpringCloudTaskCoupanProviderApp.class, args);  
        System.out.println("Coupon Provider Executed...!!");  
    }  
}
```

**3>Model class:--**

```
package com.app.model;  
import java.io.Serializable;  
import java.util.Date;  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.Id;  
import javax.persistence.Table;  
import javax.persistence.Temporal;  
import javax.persistence.TemporalType;  
import com.fasterxml.jackson.annotation.JsonIgnore;  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Entity  
@Table(name="coupon_micro")  
public class Coupon implements Serializable  
{  
    private static final long serialVersionUID = 1L;
```

```
@Id  
@Column(name="coupon_id")  
private Integer couponId;  
  
@Column(name="coupon_code", length=25)  
private String couponCode;  
  
@Column(name="coupon_discount")  
private Double couponDisc;  
  
@Column(name="coupon_expiry")  
@Temporal(TemporalType.DATE)  
private Date expDate;  
  
@JsonIgnore  
@Column(name="coupon_valid")  
private Boolean isValid = true;  
}
```

#### 4>Repository Interface:--

```
package com.app.repo;  
import java.util.Date;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.jpa.repository.Query;  
import com.app.model.Coupon;  
  
public interface CouponRepository extends JpaRepository<Coupon, Integer>  
{  
    public Coupon findByCouponCode(String code);  
  
    @Query("from com.app.model.Coupon as c where c.couponCode =:code  
        and c.expDate >=:date ")  
    public Coupon findByCouponCodeAndExpDate(String code, Date date);  
}
```

**5>Service Interface:--**

```
package com.app.service;
import java.util.List;
import com.app.model.Coupon;
public interface CouponService {

    public Coupon saveCoupon(Coupon coupon);
    public Coupon updateCoupon(Coupon coupon);
    public Coupon getCouponById(Integer id);

    public Coupon get_coupon_by_code(String code);
    public void deleteCouponById(Integer id);
    public List<Coupon> getAllCoupons();

    public boolean isExist(Integer id);
    public boolean isExpired(String code);
}
```

**6>ServiceImpl class:--**

```
package com.app.service.impl;
import java.util.Date;
import java.util.List;
import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.cache.annotation.CacheEvict;
import org.springframework.cache.annotation.CachePut;
import org.springframework.cache.annotation.Cacheable;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import com.app.model.Coupon;
import com.app.repo.CouponRepository;
import com.app.service.CouponService;

@Service
public class CouponServiceImpl implements CouponService
```

```
{  
    @Autowired  
    private CouponRepository repo;  
  
    @Override  
    @Transactional  
    public Coupon saveCoupon(Coupon coupon) {  
        return repo.save(coupon);  
    }  
  
    @Override  
    @Transactional  
    @CachePut(value = "coupon-cache",key="#coupon.couponId")  
    public Coupon updateCoupon(Coupon coupon) {  
        return repo.save(coupon);  
    }  
  
    @Override  
    @Transactional(readOnly = true)  
    @Cacheable(value = "coupon-cache",key="#couponId")  
    public Coupon getCouponById(Integer couponId)  
    {  
        Optional<Coupon> c= repo.findById(couponId);  
        return c.isPresent()?c.get():null;  
    }  
  
    @Override  
    @Transactional(readOnly = true)  
    public Coupon getCouponByCode(String code) {  
        return repo.findByCouponCode(code);  
    }  
  
    @Override  
    @Transactional  
    @CacheEvict(value = "coupon-cache",key="#couponId")  
    public void deleteCouponById(Integer couponId) {  
        repo.deleteById(couponId);  
    }
```

```

    }

    @Override
    @Transactional(readOnly = true)
    public List<Coupon> getAllCoupons() {
        return repo.findAll();
    }

    @Override
    public boolean isExpired(String code) {
        Coupon c= repo.findByCouponCodeAndExpDate(code, new Date());
        return (c!=null)?true:false;
    }

    @Override
    public boolean isExist(Integer id) {
        System.out.println(id);
        return repo.existsById(id);
    }
}

```

### 7>Validator class:--

```

package com.app.validator;
import java.util.Date;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import org.springframework.util.StringUtils;
import org.springframework.validation.Errors;
import org.springframework.validation.Validator;
import com.app.exception.ApiError;
import com.app.model.Coupon;

```

```

@Component
public class CouponValidator implements Validator{

    @Autowired
    private ApiError api=new ApiError();

```

```
@Override
public boolean supports(Class<?> clazz) {
    return clazz.equals(Coupon.class);
}

@Override
public void validate(Object target, Errors errors) {
    Coupon c = (Coupon) target;

    if(StringUtils.isEmpty(c.getCouponId().toString()))
    {
        errors.reject("couponId");
        api.getValidationErrors().add("Please Provide Coupon Id");
    }
    if(StringUtils.isEmpty(c.getCouponCode()))
    {
        errors.reject("couponCode");
        api.getValidationErrors().add("Please Provide Coupon Code ");
    }

    if(StringUtils.isEmpty(c.getCouponDisc().toString()))
    {
        errors.reject("couponDisc");
        api.getValidationErrors().add("Please Provide Coupon Discount ");
    }
    if(c.getCouponDisc()<0)
    {
        errors.reject("couponDisc");
        api.getValidationErrors().add("Please Provide Valid Discount ");
    }

    if(StringUtils.isEmpty(c.getExpDate()))
    {
        errors.reject("expDate");
        api.getValidationErrors().add("Please Provide Coupon Expiray Date");
    }
}
```

```

        }
        if(c.getExpDate().before(new Date()))
        {
            errors.reject("expDate");
            api.getValidationErrors().add("Please Provide Valid Expiray Date");
        }
    }
}

```

**8>Exception Class:--****1. API Class:--**

```

package com.app.exception;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import org.springframework.stereotype.Component;
import lombok.AllArgsConstructorConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
@Data
@AllArgsConstructor
@NoArgsConstructor
@Component
public class ApiError {

    private String errorCode;
    private String errorDesc;
    private Date errorDate;
    private List<String> validationErrors =new ArrayList<>();
}

```

**2. Exception Validation class:--**

```
package com.app.exception;
```

```
public class ValidationError extends RuntimeException {
```

```

private static final long serialVersionUID = -5925942273970967113L;
public ValidationError(String s) {
    super(s);
}

```

### 3. CouponNotFoundException class:--

```

package com.app.exception;

public class CouponNotFoundException extends RuntimeException {
    private static final long serialVersionUID = 2627003438670611967L;

    public CouponNotFoundException(String s) {
        super(s);
    }
}

```

### 9>Controller class:--

#### 1. CouponController:--

```

package com.app.rest.controller;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.validation.Errors;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.app.exception.CouponNotFoundException;
import com.app.exception.ValidationError;
import com.app.model.Coupon;
import com.app.service.CouponService;
import com.app.validator.CouponValidator;
import com.netflix.hystrix.contrib.javanica.annotation.HystrixCommand;

```

@RestController

```
@RequestMapping("/rest/coupon")
public class CouponController {

    @Autowired
    private CouponService service;

    @Autowired
    private CouponValidator validator;

    @PostMapping("/save")
    public ResponseEntity<String> saveCoupon(@RequestBody Coupon
                                                coupon, Errors errors)
    {
        validator.validate(coupon, errors);

        if(!errors.hasErrors()) {
            service.saveCoupon(coupon);
            return new ResponseEntity<String>("Coupon has been added
                                                successfully",HttpStatus.OK);
        } else {
            throw new ValidationError("Please Provide Valid Details");
        }
    }

    @GetMapping("/check/{code}")
    public String checkExpiredOrNot(@PathVariable String code) {
        return service.isExpired(code)?"Not Expired":"Expired";
    }

    @GetMapping("/getOne/{id}")
    @HystrixCommand(fallbackMethod = "getCouponException")
    public Coupon getOneCoupon(@PathVariable Integer id)
    {
        Coupon c = service.getCouponById(id);

        if(c!=null)
        {
            return c;
        } else {
            throw new CouponNotFoundException("No Coupon Found");
        }
    }
}
```

```
}

@GetMapping("/getByCode/{code}")
//@HystrixCommand(fallbackMethod = "getCouponCodeException")
public Coupon getCouponByCode(@PathVariable String code)
{
    System.out.println(code);
    Coupon c = service.getCouponByCode(code);

    if(c!=null)
    {
        return c;
    } else {
        throw new CouponNotFoundException("No Coupon Found");
    }
}

// fallback method (method name must be same as fallbackMethod name)
public Coupon getCouponException(Integer id)
{
    throw new CouponNotFoundException("No Coupon Found");
}

public Coupon getCouponCodeException(String code)
{
    throw new CouponNotFoundException("No Coupon Found");
}

@GetMapping("/all")
public List<Coupon> getAllCoupons()
{
    List<Coupon> list =service.getAllCoupons();
    System.out.println(list);
    return list;
}

@PostMapping("/update")
public ResponseEntity<String> updateCoupon(@RequestBody Coupon
                                            coupon,Errors errors)
{
    if(service.isExist(coupon.getId()))
```

```

    {
        validator.validate(coupon, errors);

        if(!errors.hasErrors())
        {
            service.saveCoupon(coupon);
            return new ResponseEntity<String>("Coupon has been
                updated successfully", HttpStatus.OK);
        } else {
            throw new ValidationError("Please Provide Valid Details");
        }
    } else {
        throw new CouponNotFoundException("No Coupon Found");
    }
}

@ResponseBody
@DeleteMapping("/delete/{id}")
public ResponseEntity<String> deleteCoupon(@PathVariable Integer id)
{
    if(service.isExist(id))
    {
        service.deleteCouponById(id);
        return new ResponseEntity<String>("Coupon Deleted
            Successfully", HttpStatus.OK);
    } else {
        throw new CouponNotFoundException("No Coupon Found");
    }
}

```

## 2. ExceptionController class:--

```

package com.app.rest.controller;
import java.util.Date;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RestControllerAdvice;
import com.app.exception.ApiError;
import com.app.exception.CouponNotFoundException;

```

```
import com.app.exception.ValidationError;

@RestControllerAdvice
public class ExceptionController {

    @Autowired
    private ApiError api;

    @ExceptionHandler(value=CouponNotFoundException.class)
    public ResponseEntity<ApiError> noCouponFound()
    {
        ApiError error = new ApiError();

        error.setErrorCode("400");
        error.setErrorDesc("No Coupon Found ");
        error.setErrorDate(new Date());
        return new ResponseEntity<ApiError>(error,HttpStatus.BAD_REQUEST);
    }

    @ExceptionHandler(value=ValidationError.class)
    public ResponseEntity<ApiError> validationError()
    {
        api.setErrorCode("400");
        api.setErrorDesc("Please Provide Valid Details");
        api.setErrorDate(new Date());
        return new ResponseEntity<ApiError>(api,HttpStatus.BAD_REQUEST);
    }
}
```

**10. Config Class:--****1. CacheConfig class:--**

```
package com.app.config;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import com.hazelcast.config.Config;
import com.hazelcast.config.EvictionPolicy;
import com.hazelcast.config.MapConfig;
```

```

import com.hazelcast.config.MaxSizeConfig;
import com.hazelcast.config.MaxSizeConfig.MaxSizePolicy;
@Configuration
public class CacheConfig {

    @Bean
    public Config chacheConfig()
    {
        return new Config()                                // for creating cache
            .setInstanceName("hazelcast-cahe")           // setting cache name
            .addMapConfig(                                // Setting MapConfig module
                new MapConfig()
                    .setName("coupon-cache")             // MapConfig Name
                    .setTimeToLiveSeconds(2000)          // max time for object to present in
memory if no activity done
                    .setMaxSizeConfig(new
MaxSizeConfig(200,MaxSizePolicy.FREE_HEAP_SIZE)) // size of objects in MapConfig
                    .setEvictionPolicy(EvictionPolicy.LRU)
            );
    }
}

```

## 2. Swagger Implementation:--

```

package com.app.config;
import static springfox.documentation.builders.PathSelectors.regex;
import static springfox.documentation.builders.RequestHandlerSelectors.basePackage;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import springfox.documentation.builders.ApiInfoBuilder;
import springfox.documentation.service.ApiInfo;
import springfox.documentation.service.Contact;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spring.web.plugins.Docket;
import springfox.documentation.swagger2.annotations.EnableSwagger2;

@Configuration

```

```
@EnableSwagger2
```

```
public class SwaggerConfig {  
    @Bean  
    public Docket configSwagger() {  
  
        return new Docket(DocumentationType.SWAGGER_2)  
            .select()  
            .apis(basePackage("com.app.rest.controller"))  
            .paths(regex("/rest.*"))  
            .build()  
            // optional  
            .apiInfo(apiInfo());  
    }  
    // it is optional only to provide additional details  
    private ApiInfo apiInfo()  
    {  
        /* using builder factory ApiInfoBuilder that will make easy to  
         construct object and return ApiInfo*/  
        return new ApiInfoBuilder()  
            .title("My Project Swagger")  
            .description("This is swagger implementation for rest")  
            .contact(new Contact("Saurabh vaish","www.github.com/saurabh-vaish",  
                "saurabh.vaish1993@gmail.com"))  
            .license("Apache 2.0")  
            .licenseUrl("http://www.apache.org/licenses/LICENSE-2.0.html")  
            .version("1.0")  
            .build();  
    }  
}
```

## 5>ProductService Folder Structure Application:--

```
✓ SpringCloud_Task-ProductProvider [boot] [devtools]
  ✓ src/main/java
    ✓ com.app
      > SpringCloudTaskProductProviderApp.java
    ✓ com.app.client
      > CouponRestConsumer.java
    ✓ com.app.config
      > CacheConfig.java
      > SwaggerConfig.java
    ✓ com.app.exception
      > ApiError.java
      > ProductNotFoundException.java
      > ValidationError.java
    ✓ com.app.model
      > Coupon.java
      > Product.java
    ✓ com.app.repo
      > ProductRepository.java
    ✓ com.app.rest.controller
      > ExceptionController.java
      > ProductController.java
    ✓ com.app.service
      > ProductService.java
    ✓ com.app.service.impl
      > ProductServiceImpl.java
    ✓ com.app.validator
      > ProductValidator.java
  ✓ src/main/resources
    application.properties
  > src/test/java
  > JRE System Library [JavaSE-1.8]
  > Maven Dependencies
    target/generated-sources/annotations
    target/generated-test-sources/test-annotations
  ✓ src
    ✓ main
    ✓ test
  > target
    HELP.md
    mvnw
    mvnw.cmd
    Mylog.log
    Mylog.log.2019-08-15.0.gz
    pom.xml
```

**1. pom.xml:--**

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.7.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.app</groupId>
  <artifactId>SpringCloud_Task-ProductProvider</artifactId>
  <version>1.0</version>
  <name>SpringCloud_Task-ProductProvider</name>
  <description>Microservices project for Spring Boot</description>

  <properties>
    <java.version>1.8</java.version>
    <spring-cloud.version>Greenwich.SR2</spring-cloud.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
```

```
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-config</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
    <optional>true</optional>
</dependency>
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
</dependency>
<!-- hazel cast caching -->
<dependency>
    <groupId>com.hazelcast</groupId>
    <artifactId>hazelcast</artifactId>
</dependency>
<dependency>
```

```
<groupId>com.hazelcast</groupId>
<artifactId>hazelcast-spring</artifactId>
</dependency>
<!-- Swagger -->
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.7.0</version>
</dependency>
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.7.0</version>
</dependency>
<!--Data Base Use any one-->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.46</version><!--$NO-MVN-MAN-VER$ -->
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>com.oracle</groupId>
    <artifactId>ojdbc6</artifactId>
    <version>11.2.0</version>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
</dependencies>

<dependencyManagement>
    <dependencies>
        <dependency>
```

```

<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-dependencies</artifactId>
<version>${spring-cloud.version}</version>
<type>pom</type>
<scope>import</scope>
</dependency>
</dependencies>
</dependencyManagement>

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

```

## **2>application.properties:--**

```

server.port=9999
# service id
spring.application.name=PRODUCT-APP
# service-url
eureka.client.service-url.default-zone=http://localhost:8761/eureka
#load balancing
eureka.instance.instance-id=${spring.application.name}:${random.value}
# no need to write for config server port 8888
#spring.cloud.config.uri=http://localhost:8888
# loggin keys
logging.level.root=info
logging.level.com.app=DEBUG
logging.file=Mylog.log
logging.file.max-size=5MB
logging.file.max-history=5

```

#Database details

```
spring.datasource.driverClassName=oracle.jdbc.driver.OracleDriver  
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe  
spring.datasource.username=system  
spring.datasource.password=system
```

#JPA details

```
spring.jpa.hibernate.ddl-auto=create  
spring.jpa.show-sql=true  
spring.jpa.database-platform=org.hibernate.dialect.Oracle10gDialect
```

### **3>Starter class:--**

```
package com.app;  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.cache.annotation.EnableCaching;  
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;  
import org.springframework.cloud.netflix.hystrix.EnableHystrix;  
import org.springframework.cloud.openfeign.EnableFeignClients;  
  
@SpringBootApplication  
@EnableEurekaClient  
@EnableCaching  
@EnableHystrix  
@EnableFeignClients  
public class SpringCloudTaskProductProviderApp {  
    public static void main(String[] args) {  
        SpringApplication.run(SpringCloudTaskProductProviderApp.class, args);  
        System.out.println("Product Provider Executed...!!");  
    }  
}
```

### **4>Model class:--**

#### **1. Coupon class:--**

```
package com.app.model;  
import com.fasterxml.jackson.annotation.JsonIgnore;
```

```
import lombok.AllArgsConstructorConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
public class Coupon {  
  
    private Integer couponId;  
    private String couponCode;  
  
    private Double couponDisc;  
    private String expDate;  
  
    @JsonIgnore  
    private Boolean applied=false;  
}
```

## 2. Product class:--

```
package com.app.model;  
import java.io.Serializable;  
import javax.persistence.Entity;  
import javax.persistence.Id;  
import javax.persistence.Table;  
import javax.persistence.Transient;  
import com.fasterxml.jackson.annotation.JsonIgnore;  
import lombok.AllArgsConstructorConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Entity  
@Table(name="product_micro")  
public class Product implements Serializable {
```

```
private static final long serialVersionUID = 1L;

@Id
private Integer proId;
private String prodCode;
private Double prodCost;

@JsonIgnore
private Double finalCost;

@Transient
private Coupon coupon;
}
```

**5>Repository Interface:--**

```
package com.app.repo;
import org.springframework.data.jpa.repository.JpaRepository;
import com.app.model.Product;

public interface ProductRepository extends JpaRepository<Product, Integer> { }
```

**6>Service Interface:--**

```
package com.app.service;
import java.util.List;
import com.app.model.Product;

public interface ProductService {

    public Product saveProduct(Product product);
    public Product updateProduct(Product product);
    public Product getProductById(Integer id);
    public void deleteProductById(Integer id);
    public List<Product> getAllProducts();

    public boolean isExist(Integer id);
```

```
        public Double calculateDiscount(Double cost,Double disc);  
    }
```

### 7>ServiceImpl Class:--

```
package com.app.service.impl;  
import java.util.List;  
import java.util.Optional;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.cache.annotation.CacheEvict;  
import org.springframework.cache.annotation.CachePut;  
import org.springframework.cache.annotation.Cacheable;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
import com.app.model.Product;  
import com.app.repo.ProductRepository;  
import com.app.service.ProductService;
```

```
@Service
```

```
public class ProductServiceImpl implements ProductService {
```

```
    @Autowired
```

```
    private ProductRepository repo;
```

```
    @Override
```

```
    @Transactional
```

```
    public Product saveProduct(Product product) {
```

```
        return repo.save(product);
```

```
}
```

```
    @Override
```

```
    @Transactional
```

```
    @CachePut(value = "product-cache",key="#product.prodId")
```

```
    public Product updateProduct(Product product) {
```

```
        return repo.save(product);
```

```
}
```

```
@Override  
@Transactional(readOnly = true)  
@Cacheable(value = "product-cache",key="#prodId")  
public Product getProductById(Integer prodId) {  
    Optional<Product> c= repo.findById(prodId);  
    return c.isPresent()?c.get():null;  
}  
  
@Override  
@Transactional  
@CacheEvict(value = "product-cache",key="#prodId")  
public void deleteProductById(Integer prodId) {  
    repo.deleteById(prodId);  
}  
  
@Override  
@Transactional(readOnly = true)  
public List<Product> getAllProducts() {  
    return repo.findAll();  
}  
  
@Override  
public boolean isExist(Integer id) {  
    return repo.existsById(id);  
}  
  
@Override  
public Double calculateDiscount(Double cost, Double disc) {  
    Double dCost = cost*disc/100.0;  
    Long value = Math.round(cost-dCost);  
    return value.doubleValue();  
}  
}
```

**8>Validator Class:--**

```
package com.app.validator;
```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import org.springframework.util.StringUtils;
import org.springframework.validation.Errors;
import org.springframework.validation.Validator;
import com.app.exception.ApiError;
import com.app.model.Product;

@Component
public class ProductValidator implements Validator{

    @Autowired
    private ApiError api;

    @Override
    public boolean supports(Class<?> clazz) {
        return clazz.equals(Product.class);
    }

    @Override
    public void validate(Object target, Errors errors) {

        Product p = (Product) target;

        if(StringUtils.isEmpty(p.getProdId().toString()))  {
            errors.reject("prodId");
            api.getValidationErrors().add("Please Provide Product Id");
        }

        if(StringUtils.isEmpty(p.getProdCode())) {
            errors.reject("prodCode");
            api.getValidationErrors().add("Please Provide Product Code ");
        }

        if(StringUtils.isEmpty(p.getProdCost().toString())) {
            errors.reject("prodCost");
            api.getValidationErrors().add("Please Provide Product Cost ");
        }
    }
}
```

```
        }  
    }  
}
```

### 9>Exception Class:--

#### 1. API Error class:--

```
package com.app.exception;  
import java.util.ArrayList;  
import java.util.Date;  
import java.util.List;  
import org.springframework.stereotype.Component;  
import lombok.AllArgsConstructorConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;
```

```
@Data  
@AllArgsConstructor  
@NoArgsConstructor  
@Component  
public class ApiError {
```

```
    private String errorCode;  
    private String errorDesc;  
    private Date errorDate;  
  
    private List<String> validationErrors = new ArrayList<>();  
}
```

#### 2. Validation class:--

```
package com.app.exception;
```

```
public class ValidationError extends RuntimeException {
```

```
    private static final long serialVersionUID = -5925942273970967113L;
```

```
    public ValidationError(String s) {  
        super(s);
```

```

        }
    }
}
```

**3. ProductNotFoundException:--**

```

package com.app.exception;

public class ProductNotFoundException extends RuntimeException {

    private static final long serialVersionUID = 1L;

    public ProductNotFoundException(String s) {
        super(s);
    }
}
```

**10>ClientRestController class:--**

```

package com.app.client;
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import com.app.model.Coupon;

@FeignClient("COUPON-APP")
public interface CouponRestConsumer {

    @GetMapping("/rest/coupon/getByCode/{code}")
    public Coupon getCouponByCode(@PathVariable String code);

    @GetMapping("/rest/coupon/check/{code}")
    public String checkExpiredOrNot(@PathVariable String code);
}
```

**11>Controller class:--****1. ProductController:--**

```

package com.app.rest.controller;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
```

```
import org.springframework.http.ResponseEntity;
import org.springframework.validation.Errors;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.app.client.CouponRestConsumer;
import com.app.exception.ProductNotFoundException;
import com.app.exception.ValidationError;
import com.app.model.Coupon;
import com.app.model.Product;
import com.app.service.ProductService;
import com.app.validator.ProductValidator;

@RestController
@RequestMapping("/rest/product")
public class ProductController {

    @Autowired
    private ProductService service;

    @Autowired
    private CouponRestConsumer consumer;

    @Autowired
    private ProductValidator validator;

    @PostMapping("/save")
    public ResponseEntity<String> saveProduct(@RequestBody Product
                                                product, Errors errors)
    {
        validator.validate(product, errors);

        if(!errors.hasErrors())
        {
            String res= consumer.checkExpiredOrNot (product.getCoupon().getCouponCode());
            System.out.println(res);
        }
    }
}
```

```

Coupon c = consumer.getCouponByCode(product.getCoupon().getCouponCode());
System.out.println(c);
    if(c==null)
    {
        return new ResponseEntity<String>("Coupon Not Found",HttpStatus.OK);
    }
    else if(res.equals("Expired"))
    {
        return new ResponseEntity<String>("Coupon has been expired !",HttpStatus.OK);
    } else {
        Double dis = c.getCouponDisc();
        if(product.getCoupon().getApplied()) // if coupon applied
        {
            dis=0.0;
        }
    }

Double cost = product.getProdCost(); System.out.println(cost+","+dis);
product.setFinalCost(service.calculateDiscount(cost, dis));
service.saveProduct(product);

return new ResponseEntity<String>("Product has been added successfully",
HttpStatus.OK);
    }
} else {
    throw new ValidationError("Please Provide Valid Details");
}
}

@GetMapping("/apply")
public ResponseEntity<Double> applyCoupon(Double cost,Double disc)
{
    return new ResponseEntity<Double>(service.calculateDiscount(cost,
disc),HttpStatus.OK);
}

@GetMapping("/getOne/{id}")
//@HystrixCommand(fallbackMethod = "getProductException")
public ResponseEntity<Product> getOneProduct(@PathVariable Integer id)
{
    Product p = service.getProductById(id);System.out.println(p);
}

```

```
if(p!=null)
{
    return new ResponseEntity<Product>(p,HttpStatus.OK);
} else {
    throw new ProductNotFoundException("No Product Found");
}
}

// fallback method
public ResponseEntity<Product> getProductException(Integer id)
{
    throw new ProductNotFoundException("No Product Found");
}

@GetMapping("/all")
public List<Product> getAllProducts()
{
    return service.getAllProducts();
}

@PostMapping("/update")
public ResponseEntity<String> updateProduct(@RequestBody Product
                                              product, Errors errors)
{
    if(service.isExist(product.getProdId()))
    {
        validator.validate(product, errors);

        if(!errors.hasErrors())
        {
            service.saveProduct(product);
            return new ResponseEntity<String>("Product has been
updated successfully",HttpStatus.OK);
        } else {
            throw new ValidationError("Please Provide Valid Details");
        }
    } else {
        throw new ProductNotFoundException("No Product Found");
    }
}
```

```

    @DeleteMapping("/delete/{id}")
    public ResponseEntity<String> deleteProduct(@PathVariable Integer id)
    {
        if(service.isExist(id)) {
            service.deleteProductById(id);

        return new ResponseEntity<String>("Product Deleted Successfully",HttpStatus.OK);
        } else {
            throw new ProductNotFoundException("No Product Found");
        }
    }
}

```

**12>Config class:--****1. Config class:--**

```

package com.app.config;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import com.hazelcast.config.Config;
import com.hazelcast.config.EvictionPolicy;
import com.hazelcast.config.MapConfig;
import com.hazelcast.config.MaxSizeConfig;
import com.hazelcast.config.MaxSizeConfig.MaxSizePolicy;

```

@Configuration

```
public class CacheConfig {
```

@Bean

```
public Config chacheConfig()
```

```
{
```

```
    return new Config() // for creating cache
```

```
        .setInstanceName("hazelcast-cahe") // setting cache name
```

```
        .addMapConfig() // Setting MapConfig module
```

```
            new MapConfig()
```

```
                .setName("product-cache") // MapConfig Name
```

```
                .setTimeToLiveSeconds(2000) // max time for object to present in
```

memory if no activity done

```
.setMaxSizeConfig(new MaxSizeConfig(200,MaxSizePolicy.FREE_HEAP_SIZE))  
// size of objects in MapConfig  
    .setEvictionPolicy(EvictionPolicy.LRU)  
};  
}  
}
```

## 2. Swagger Config:--

```
package com.app.config;  
import static springfox.documentation.builders.PathSelectors.regex;  
import static springfox.documentation.builders.RequestHandlerSelectors.basePackage;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import springfox.documentation.builders.ApiInfoBuilder;  
import springfox.documentation.service.ApiInfo;  
import springfox.documentation.service.Contact;  
import springfox.documentation.spi.DocumentationType;  
import springfox.documentation.spring.web.plugins.Docket;  
import springfox.documentation.swagger2.annotations.EnableSwagger2;  
  
@Configuration  
@EnableSwagger2  
public class SwaggerConfig {  
  
    @Bean  
    public Docket configSwagger()  
    {  
        return new Docket(DocumentationType.SWAGGER_2)  
            .select()  
            .apis(basePackage("com.app.rest.controller"))  
            .paths(regex("/rest.*"))  
            .build()  
  
            // optional  
            .apiInfo(apiInfo());  
    }  
}
```

```
// it is optional only to provide additional details
private ApiInfo apiInfo()
{
    return new ApiInfoBuilder() // using builder factory ApiInfoBuilder that
    will make easy to construct object and return ApiInfo
        .title("My Project Swagger")
        .description("this is swagger implementation for rest")
    .contact(new Contact("saurabh vaish","http://www.github.com/saurabh-vaish",
        "saurabh.vaish1993@gmail.com"))
        .license("Apache 2.0")
        .licenseUrl("http://www.apache.org/licenses/LICENSE-2.0.html")
        .version("1.0")
    .build();
}
}
```

**Execution Order:--**

- 1.>Eureka Server
- 2.>Config Server
- 3.>Zuul Server
- 4.>CouponProvider Application Starter class
- 5.>ProductProvider Application Starter class

**OUTPUT SCREEN STEP BY STEP:--**

1>Eureka Server (type <http://localhost:8761> in browser to see Eureka Dashboard):--

The screenshot shows the Spring Eureka dashboard. At the top, there's a header with the Eureka logo and navigation links for HOME and LAST 1000 SINCE STARTUP. Below the header, the 'System Status' section displays environment details like 'Environment: test' and 'Data center: default'. To the right, a table provides real-time metrics: Current time (2019-10-09T09:47:37 +0530), Uptime (00:14), Lease expiration enabled (false), Renews threshold (10), and Renews (last min) (10). A red warning message at the bottom states: 'EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSER THAN THRESHOLD AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE.' The 'DS Replicas' section lists three registered instances: COUPON-APP, PRODUCT-APP, and ZUUL\_PROXY, each with their respective AMIs and availability zones.

## 2>CouponProvider Application Swagger SCREEN:--

The screenshot shows the Swagger UI for the CouponProvider application. The top navigation bar includes links for Online Courses, Online Tests, Tutorials, Youth4work, Testpot.com, Facebook, and Hello Python. The main title is 'My Project Swagger'. It displays information about the creator (Saurabh vaish), GitHub link, developer contact, and Apache 2.0 license. Below this, the 'coupon-controller' section is shown under 'Coupon Controller'. It lists various REST operations with their HTTP methods, URLs, and descriptions:

coupon-controller : Coupon Controller		ShowHide   List Operations   Expand Operations
<code>GET</code>	/rest/coupon/all	getAllCoupons
<code>GET</code>	/rest/coupon/check/{code}	checkExpiredOrNot
<code>DELETE</code>	/rest/coupon/delete/{id}	deleteCoupon
<code>GET</code>	/rest/coupon/getByCode/{code}	getCouponByCode
<code>GET</code>	/rest/coupon/getOne/{id}	getOneCoupon
<code>POST</code>	/rest/coupon/save	saveCoupon
<code>POST</code>	/rest/coupon/update	updateCoupon

[ BASE URL: / , API VERSION: 1.0 ]

1. Swagger screen for **Save** Operation for CouponProvider Application:--

**POST** /rest/coupon/save saveCoupon

**Response Class (Status 200)**  
string

**Response Content Type** `*/*` ▾

**Parameters**

Parameter	Value	Description	Parameter Type	Data Type
coupon	<pre>{   "couponCode": "INDAUG15",   "couponDisc": 30,   "couponId": 111,   "expDate": "2019-10-05T18:59:24.763Z" }</pre>	coupon	body	<a href="#">Model</a>   <a href="#">Example Value</a> <pre>{   "couponCode": "string",   "couponDisc": 0,   "couponId": 0,   "expDate": "2019-09-27T18:59:24.763Z" }</pre>

Parameter content type: `application/json` ▾

**Response Messages**

HTTP Status Code	Reason	Response Model	Headers
201	Created		
401	Unauthorized		
403	Forbidden		
404	Not Found		

[Try it out!](#) [Hide Response](#)

## 2. Swagger screen for **update** Operation for CouponProvider Application:--

**POST** /rest/coupon/update updateCoupon

**Response Class (Status 200)**  
string

**Response Content Type** `*/*` ▾

**Parameters**

Parameter	Value	Description	Parameter Type	Data Type
coupon	<pre>{   "couponCode": "INDAUG15",   "couponDisc": 20,   "couponId": 111,   "expDate": "2019-09-30T18:01:22.687Z" }</pre>	coupon	body	<a href="#">Model</a>   <a href="#">Example Value</a> <pre>{   "couponCode": "string",   "couponDisc": 0,   "couponId": 0,   "expDate": "2019-09-27T18:59:24.789Z" }</pre>

Parameter content type: `application/json` ▾

**Response Messages**

HTTP Status Code	Reason	Response Model	Headers
201	Created		
401	Unauthorized		
403	Forbidden		
404	Not Found		

[Try it out!](#) [Hide Response](#)

### 3. Swagger screen for **delete** Operation for CouponProvider Application:--

**DELETE** /rest/coupon/delete/{id} deleteCoupon

Response Class (Status 200)  
string

Response Content Type `*/*`

**Parameters**

Parameter	Value	Description	Parameter Type	Data Type
<b>id</b>	<b>111</b>	<b>id</b>	path	integer

**Response Messages**

HTTP Status Code	Reason	Response Model	Headers
204	No Content		
401	Unauthorized		
403	Forbidden		

**Try it out!** [Hide Response](#)

### 4. POSTMAN screen for **GETALL** Operation for CouponProvider Application:--

GET <localhost:5432/rest/coupon/all> Params Send

Pretty Raw Preview JSON

```

1  [
2    {
3      "couponId": 122,
4      "couponCode": "INDAUG15",
5      "couponDisc": 15,
6      "expDate": "2019-09-28"
7    },
8    {
9      "couponId": 111,
10     "couponCode": "INDAUG15",
11     "couponDisc": 30,
12     "expDate": "2019-10-06"
13   },
14   {
15     "couponId": 110,
16     "couponCode": "INDAUG15",
17     "couponDisc": 30,
18     "expDate": "2019-09-30"
19   }
20 ]
  
```

### 5. POSTMAN screen for **GetByCode** Operation for CouponProvider Application:--

## 5.1. If code Not available

The screenshot shows the POSTMAN interface with a GET request to `http://desktop-ch8lpuh:5432/rest/coupon/getByCode/INDAUG15`. The 'Authorization' tab is selected, showing 'No Auth'. The 'Body' tab is selected, displaying a JSON response:

```

1 {
2   "errorCode": "400",
3   "errorDesc": "No Coupon Found",
4   "errorDate": "2019-09-28T07:28:33.424+0000",
5   "validationErrors": []
6 }

```

The status bar at the bottom right indicates `Status: 400 Bad Request`.

## 5.2. If Code available

The screenshot shows the POSTMAN interface with a GET request to `http://desktop-ch8lpuh:5432/rest/coupon/getByCode/INDAUG15`. The 'Authorization' tab is selected, showing 'No Auth'. The 'Body' tab is selected, displaying a JSON response:

```

1 {
2   "couponId": 122,
3   "couponCode": "INDAUG15",
4   "couponDisc": 15,
5   "expDate": "2019-09-28"
6 }

```

The status bar at the bottom right indicates `Status: 200 OK`.

## 6. POSTMAN screen for CHECK CODE VALIDITY Operation for CouponProvider Appl:--

### 6.1. If Code is not expire.

The screenshot shows the POSTMAN interface with a GET request to `http://desktop-ch8lpuh:5432/rest/coupon/check/INDAUG15`. The 'Authorization' tab is selected, showing 'No Auth'. The 'Body' tab is selected, displaying a Text response:

```

1 Not Expired

```

The status bar at the bottom right indicates `Status: 200 OK`.

### 6.2. If code is expire

GET <http://desktop-ch8lpuh:5432/rest/coupon/check/INDAUG17>

Params Send

Authorization Headers Body Pre-request Script Tests

Type: No Auth

Body Cookies Headers (3) Test Results Status: 200 OK

Pretty Raw Preview Text Expired

1 Expired

7. Postman screen for **getOne** by couponId Operation for CouponProvider Appl:--

GET <http://desktop-ch8lpuh:5432/rest/coupon/getOne/122>

Params Send

Authorization Headers Body Pre-request Script Tests

Type: No Auth

Body Cookies Headers (3) Test Results Status: 200 OK

Pretty Raw Preview JSON Expired

```

1 {
2   "couponId": 122,
3   "couponCode": "INDAUG15",
4   "couponDisc": 15,
5   "expDate": "2019-09-28"
6 }
```

3. ProductService Swagger Screen Shot:--

[localhost:9999/swagger-ui.html#/](http://localhost:9999/swagger-ui.html#/)

My Project Swagger

this is swagger implementation for rest

Created by saurabh vaish  
See more at <http://www.github.com/saurabh-vaish>  
[Contact the developer](#)  
[Apache 2.0](#)

**product-controller : Product Controller**

Show/Hide | List Operations | Expand Operations

GET	/rest/product/all	getAllProducts
GET	/rest/product/apply	applyCoupon
DELETE	/rest/product/delete/{id}	deleteProduct
GET	/rest/product/getOne/{id}	getOneProduct
POST	/rest/product/save	saveProduct
POST	/rest/product/update	updateProduct

[ BASE URL: / , API VERSION: 1.0 ]

## 1. Postman screen for **Save** Operation for ProductProvider Application:--

### i. When Coupon is valid

POST localhost:9999/rest/product/save

Params | Send

Params Authorization Headers (9) Body Pre-request Script Tests Cookies Code

Body (raw) JSON (application/json)

```
1 ▾ {
2 ▾   "coupan": {
3     "coupanCode": "AUG16"
4   },
5   "prodCode": "PRO012",
6   "prodCost": 1000,
7   "prodId": 2
8 }
```

Status: 200 OK Time: 613 ms Size: 159 B

Pretty Raw Preview Auto

1 Product has been added successfully

### ii. If Coupon is expired

POST  Params Send Save

Authorization Headers (1) Body Pre-request Script Tests Code

form-data x-www-form-urlencoded raw binary JSON (application/json)

```

1 {
2   "coupon": {
3     "couponCode": "INDAUG15"
4   },
5   "prodCode": "M30",
6   "prodCost": 1000,
7   "prodId": 50
8 }

```

Body Cookies Headers (3) Test Results Status: 200 OK Time: 1186 ms

Pretty Raw Preview Text

Activate Windows Go to Settings to activate Windows.

1 Coupon has been expired !

## 2. Postman screen for **Update** by ProductId Operation for ProductProvider Appl:--

POST  Params Send

Params Authorization Headers (9) Body Pre-request Script Tests Cookies Code

none form-data x-www-form-urlencoded raw binary GraphQL BETA JSON (application/json)

```

1 {
2   "coupan": {
3     "coupanCode": "AUG16"
4   },
5   "prodCode": "PROD12",
6   "prodCost": 1600,
7   "prodId": 2
8 }

```

Body Cookies Headers (3) Test Results Status: 200 OK Time: 225 ms Size: 161 B

Pretty Raw Preview Auto

1 Product has been updated successfully

## 3. Postman screen for **Delete** by productId Operation for ProductProvider Appl:--

POST  Params Send

Params Authorization Headers (8) Body Pre-request Script Tests Cookies Code

Query Params

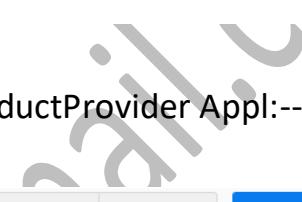
KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (3) Test Results Status: 200 OK Time: 364 ms Size: 152 B

Pretty Raw Preview Auto

1 Product Deleted Successfully

## 4. Postman screen for **getOne** by productId Operation for ProductProvider Appl:--



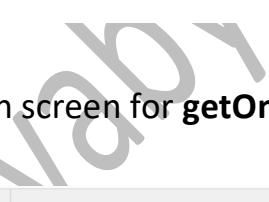
Postman screenshot showing a GET request to `localhost:9999/rest/product/getOne/11`. The response status is 400 Bad Request with the following JSON body:

```

1 [
2   "errorCode": "400",
3   "errorDesc": "No Product Found",
4   "errorDate": "2019-08-16T17:50:46.425+0000",
5   "validationErrors": []
6 ]

```

## 5. Postman screen for `getAll` record Operation for ProductProvider Appl:--



Postman screenshot showing a GET request to `localhost:9999/rest/product/all`. The response status is 200 OK with the following JSON body:

```

1 [
2   {
3     "prodId": 1,
4     "prodCode": "PROD11",
5     "prodCost": 100,
6     "coupan": null
7   },
8   {
9     "prodId": 2,
10    "prodCode": "PROD12",
11    "prodCost": 1000,
12    "coupan": null
13  }
14 ]

```

## 6. Postman screen for `getOne` ValidationError ProductProvider Appl:--



Postman screenshot showing a GET request to `localhost:9999/rest/product/getOne/1`. The response status is 400 Bad Request with the following JSON body:

```

1 [
2   "errorCode": "400",
3   "errorDesc": "No Product Found",
4   "errorDate": "2019-08-16T17:50:46.425+0000",
5   "validationErrors": []
6 ]

```

# \*\*\*MOCKITO\*\*\*

## Spring Boot UnitTesting using JUnit+Mocking:--

**Mock:**-- It is a component which acts like Client to make request.

=>Behaves as Container.

=>Supports object creation and Destroy.

=>Uses Proxy design Pattern.

=>Supports HTTP calls, No need of using any client (**RestTemplate**, **HttpClient**... etc).

Mocking is implemented using

1. EasyMock
2. Mockito

=>Spring boot uses JUnit 4 + Mockito 2 Integration for UnitTesting of Applications.

=>Here Mockito supports,

- a. HTTP Request Creation.
- b. Making HTTP Client (GET/POST...).
- c. Execute code using Proxies.
- d. GetResult and store in HTTP Response objects.

=>Here JUnit is used for

- a. Writing Test cases with annotations.
- b. Verify Result using assert methods (Return PASS/FALL).

**Step#1:-** Create one Spring Starter Project and define RestController with methods and URLs added.

**Step#2:-** Define UnitTesting class (Test case) under src/test/java.

- a. Autowire MockMvc (C) Object[acts as Http Client and supports container communication].
- b. Call perform method to make Http Request but construct Request object at same time that returns as **MockHttpServletRequest** (use **RequestBuilder** static methods and call).

Ex:- `mockMvc.perform(post("/product/save").header("Content-Type", "application/json").content("{\"prodId\":101,...}")).andReturn();`

c. Here `andReturn()` submits `HttpRequest` above one looks like.

Http Request

POST /product/save
ContentType : application/json
{"prodId": 101}

( MockHttpServletRequest )

d. Both Request and Response objects are stored in “`MvcResult`” as,

- a. `MockHttpServletRequest`
- b. `MockHttpServletResponse`.

It looks like

MvcResult

Http Request	HttpResponse
POST /product/save	Http 1.1 200 Ok
ContentType : application/json	Content-Type=text/plain.....
{"prodId": 101}	Hello App
( MockHttpServletRequest )	MockHttpServletResponse

e. Enable this Mock and Test process using Annotations :

```
@RunWith(SpringRunner.class)
@WebMvcTest
```

**Code:-- RestController:--**

```
package com.app.controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping(name="/emp")
public class EmployeeRestController {

    @GetMapping("/data")
    public String getData() {
        return "Hello";
    }
}
```

**Code:- Test class:--**

```
package com.app;
import static org.junit.Assert.assertEquals;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.mock.web.MockHttpServletResponse;
import org.springframework.test.context.junit4.SpringRunner;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.MvcResult;

@RunWith(SpringRunner.class)
@WebMvcTest
@SpringBootTest
public class JUnitMockitoApplicationTests {
```

```
@Autowired
```

```
private MockMvc mockMvc;
```

```
@Test
```

```
public void testEmpData() throws Exception {
```

```
    MvcResult result=mockMvc.perform(get("/emp/data")).andReturn();
```

```
//MockHttpServletRequest req =result.getRequest();
```

```
MockHttpServletResponse resp =result.getResponse();
```

```
assertEquals("Hi", resp.getContentAsString());
```

```
}
```

```
}
```

=>Right click => Run as => JUnit Test

## Spring Boot with JUnit and Mockito:--

### ##Testing : ReST CURD Application ##

=>To implement Unit Testing, we need to follow 4 steps. Given below,

1. Construct Http Request using **RequestBuilders**.
2. Execute Http Request using **MockMvc**
3. Store Response along with Request in **MvcResult**.
4. Use assert API (JUnit) to verify actual details with expected values.

**Step#1:-** To construct Request Object use RequestBuilder and call static method (Http Method Type).

```
MockMvcRequestBuilders.post(...);
```

=>It returns HttpServletRequest object.

```
MockMvcServletRequestBuilder
```

### Ex#1 (GET):-

Request

GET	/emp/findOne/21
	H
	B

```
MockHttpServletRequestBuilder request =
MockMvcBuilders.get ("/emp/findOne/21");
```

### Ex#2 (POST):--

HttpServletRequest

POST /emp/save	H
Content-Type:application/json	B
{....}	

=>Equal code of above Diagram is

```
MockHttpServletRequestBuilder request = MockMvcBuilders
    .post("/emp/save")
    .header("Content-Type", "application/json")
    //contentType("application/json")
    //contentType(MediaType.APPLICATION_JSON)
    .content("{...}");
```

### Ex#3 (DELETE):-

HttpRequest

DELETE /emp/remove/101	H
	B

=>Equal code

```
MockHttpServletRequestBuilder request =
MockMvcBuilders.delete("/emp/remove/101");
```

**EX#4 (PUT):-**

HttpRequest

PUT /emp/update	H
Content-Type: application/xml	B
<emp> </emp>	

=>Equal code

```
MockHttpServletRequestBuilder request = MockMvcRequestBuilders
    .post("/emp/save")
    .contentType("application/xml")
    .content("<emp>...</emp>");
```

**Step#2:-** Execute Request call using MockMvc.

```
MvcResult result = mockMvc.perform(request).andReturn();
```

**Step#3:-** Get Request/Response Object from Mvc.

```
MockHttpServletRequest req = result.getRequest();
MockHttpServletResponse resp = result.getResponse();
```

**Project Creation step by Step:-**

Step#1:- Define one CURD application using one database and RestController.

Step#2:- Check all operations using POSTMAN Screen.

Step#3:- Define one test profile for properties or yml.

=>application-test.properties

Step#4:- Define one class under src/test/java folder and apply annotations.

Step#5:- Apply below annotations over test class.

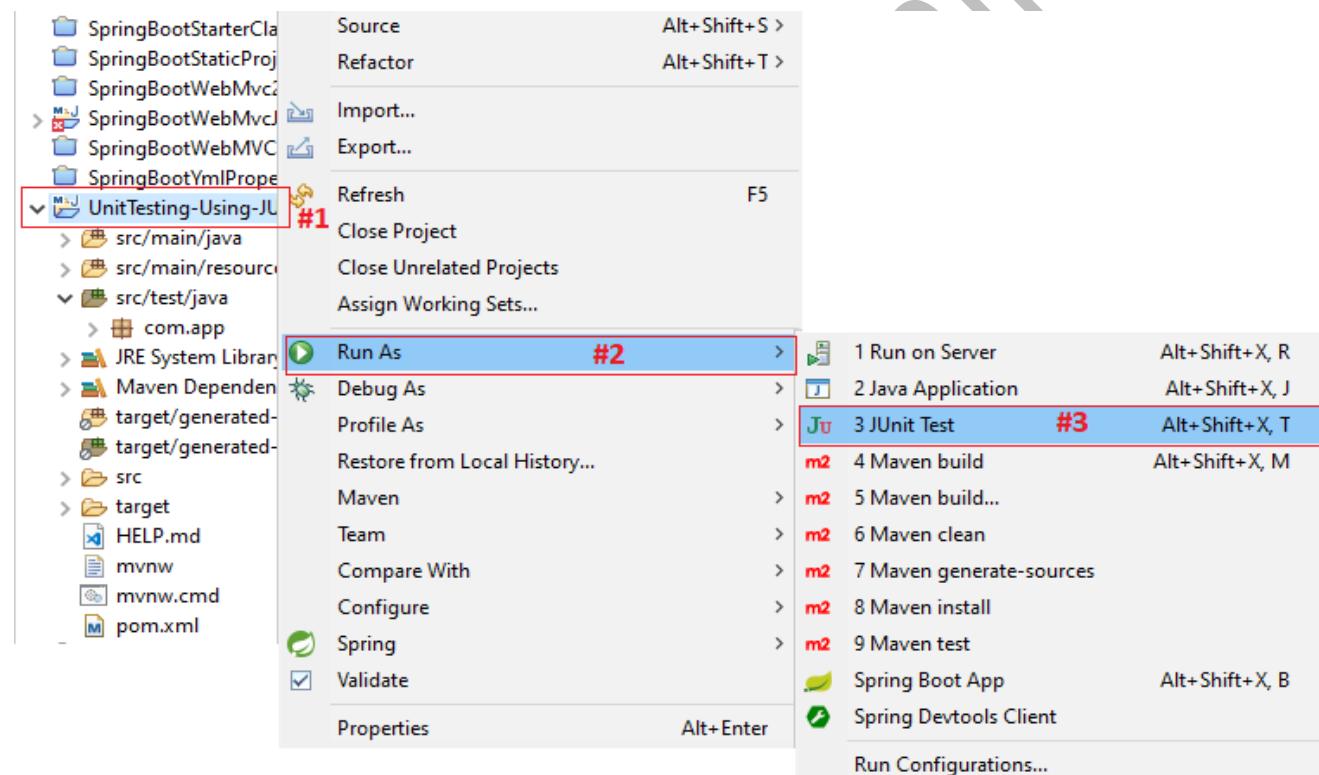
```
@SpringBootTest(webEnvironment=WebEnvironment.MOCK)
@AutoConfigureMockMvc
@TestPropertySource("classpath:application-test.properties")
```

Step#6:- Use MockMvc dependency (autowire) in Test class.

Step#7:- Define @Test methods under Test class.

Step#8:- Run Test class using JUnit Test.

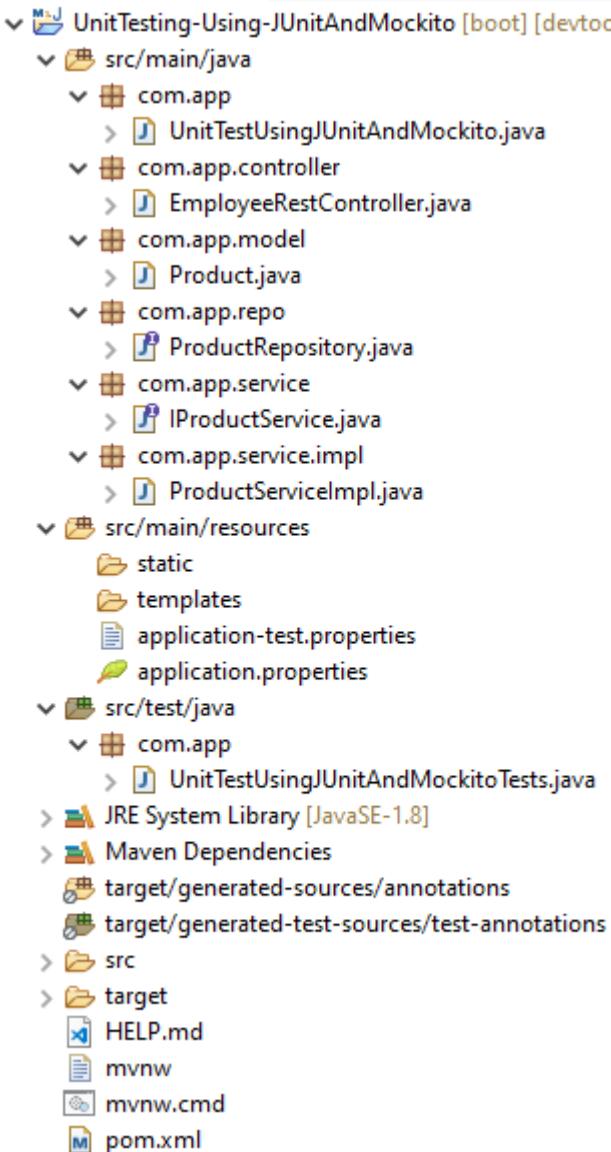
=>Right click on Project > Run As > JUnit Test.



#### NOTE:--

- 1>@TestPropertySource:- It is used to load properties/yml file into UnitTest
- 2>@AutoConfigureMockMvc:- It Define all Mock Beans related to environment  
(Ex: Datasource, ConnectionPool, Cache...).
- 3>@SpringBootTest:- It define beans and injects them based on relations (Objects for: RestControllers, Services, Repos...etc).
- 4>@WebMvcTest:- Works only for @RestControllers without service and other dependencies.

## 1. Folder Structure of JUnit + Mockito Testing:--



### pom.xml:--

```

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.8.RELEASE</version>
    <relativePath /> <!-- lookup parent from repository -->
  
```

```
</parent>
<groupId>com.app</groupId>
<artifactId>UnitTesting-Using-JUnitAndMockito</artifactId>
<version>1.0</version>
<name>UnitTesting-Using-JUnitAndMockito</name>
<description>Spring Boot Test Application</description>

<properties>
    <java.version>1.8</java.version>
</properties>

<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <!-- <optional>true</optional> -->
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>com.oracle</groupId>
        <artifactId>ojdbc6</artifactId>
        <version>11.2.0</version>
    </dependency>
    <!-- <dependency>
        <groupId>org.mockito</groupId>
        <artifactId>mockito-all</artifactId>
        <scope>test</scope>
    </dependency> -->
```

```

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
    <optional>true</optional>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
</dependencies>
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
</project>

```

#### Coding Step by Step:--

**Step#1: In “application.properties” & “application-test.properties” add bellow code:-**

server.port=2019

##DataSource##

spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver

spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe

spring.datasource.username=system

spring.datasource.password=system

#Spring Data JPA

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=create

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.Oracle10gDialect

spring.jpa.properties.hibernate.format\_sql=true

spring.jpa.open-in-view=true

**Step#2: Model class (Product.class):--**

```
package com.app.model;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;
import lombok.Data;

@Entity
@Data
@Table
public class Product {

    @Id
    @GeneratedValue
    private Integer proId;
    private String prodCode;
    private Double prodCost;
    private String vendorCode;
}
```

**Step#3: Repository Interface:--**

```
package com.app.repo;
import org.springframework.data.jpa.repository.JpaRepository;
import com.app.model.Product;

public interface ProductRepository extends JpaRepository<Product, Integer> { }
```

**Step#4: Service Interface:--**

```
package com.app.service;
import java.util.List;
import com.app.model.Product;

public interface IProductService {

    public Integer saveProduct(Product p);
```

```
    public void deleteProduct(Integer proId);
    public Product getProductById(Integer proId);
    public List<Product> getAllProducts();
    public boolean isProductExist(Integer id);
}
```

**Step#5: ServiceImpl class:--**

```
package com.app.service.impl;
import java.util.List;
import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.app.model.Product;
import com.app.repo.ProductRepository;
import com.app.service.IProductService;
```

```
@Service
```

```
public class ProductServiceImpl implements IProductService{
```

```
    @Autowired
```

```
    private ProductRepository repo;
```

```
    public Integer saveProduct(Product p) {
        p=repo.save(p);
        Integer proId=p.getProId();
        return proId;
    }
```

```
    public void deleteProduct(Integer proId) {
        repo.deleteById(proId);
    }
```

```
    public Product getProductById(Integer proId) {
        Optional<Product> p=repo.findById(proId);
        if(p.isPresent()) {
            return p.get();
```

```
        }else {
            return new Product();
        }
    }

    public List<Product> getAllProducts() {
        List<Product> prods=repo.findAll();
        return prods;
    }

    @Override
    public boolean isProductExist(Integer id) {
        return repo.existsById(id);
    }
}
```

**Step#6: RestController:--**

```
package com.app.controller;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.app.model.Product;
import com.app.service.IProductService;

@RestController
@RequestMapping("/product")
public class EmployeeRestController {

    @Autowired
    private IProductService service;
```

**//1. Post Method**

```

    @PostMapping("/register")
    public ResponseEntity<?> saveProduct(@RequestBody Product product) {
        ResponseEntity<?> response=null;
        try {
            Integer stdId=service.saveProduct(product);
            response = new ResponseEntity<String>(stdId+"-Inserted", HttpStatus.OK);
        } catch (Exception e) {
            e.printStackTrace();
            response = new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
        }
        return response;
    }

```

**//2. Get Method**

```

    @GetMapping("/get/{id}")
    public ResponseEntity<?> showOneProducts(@PathVariable Integer id) {
        ResponseEntity<?> response=null;
        boolean exist=service.isProductExist(id);
        if(exist) {
            Product s=service.getProductById(id);
            response=new ResponseEntity<Product>(s, HttpStatus.OK);
        } else {
            response=new ResponseEntity<>(HttpStatus.NO_CONTENT);
        }
        return response;
    }

```

**//3. Get Method for all data**

```

    @GetMapping("/all")
    public ResponseEntity<?> showAllProducts() {
        ResponseEntity<?> response=null;
        List<Product> Products=service.getAllProducts();
        if(Products!=null && !Products.isEmpty()) {
            response=new ResponseEntity<List<Product>>(Products, HttpStatus.OK);
        } else {
            response=new ResponseEntity<>(HttpStatus.NO_CONTENT);
        }
    }

```

```
        }
        return response;
    }
```

#### //4. Delete Method

```
@DeleteMapping("/delete/{id}")
public ResponseEntity<?> deleteProduct(@PathVariable Integer id) {
    ResponseEntity<?> response=null;
    boolean exist=service.isProductExist(id);
    if(exist) {
        service.deleteProduct(id);
    response=new ResponseEntity<String>(id+"-Removed", HttpStatus.OK);
    } else {
        response=new ResponseEntity<String>("Product NOT FOUND",
            HttpStatus.BAD_REQUEST);
    }
    return response;
}
```

#### //5. Edit method

```
@PutMapping("/edit")
public ResponseEntity<?> editProduct(@RequestBody Product product) {
    ResponseEntity<?> response=null;
    Integer id=product.getProdId();
    boolean exist=service.isProductExist(id);
    if(exist) {
        service.saveProduct(product);
    response = new ResponseEntity<String>(id+"-Updated", HttpStatus.OK);
    }else {
        response = new ResponseEntity<String>("Product NOT FOUND",
            HttpStatus.BAD_REQUEST);
    }
    return response;
}
```

Step#7:JUnitAndMockitoTests class:--

```
package com.app;
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertNotNull;
import static org.springframework.test.web.servlet.request.
MockMvcRequestBuilders.delete;
import static org.springframework.test.web.servlet.request.
MockMvcRequestBuilders.get;
import static org.springframework.test.web.servlet.request.
MockMvcRequestBuilders.post;
import static org.springframework.test.web.servlet.request.
MockMvcRequestBuilders.put;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.
AutoConfigureMockMvc;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.mock.web.MockHttpServletResponse;
import org.springframework.test.context.TestPropertySource;
import org.springframework.test.context.junit4.SpringRunner;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.MvcResult;

@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment= SpringBootTest.WebEnvironment.MOCK)
@AutoConfigureMockMvc
@TestPropertySource(locations = "classpath:application-test.properties")
public class UnitTestUsingJUnitAndMockitoTests {

    @Autowired
    private MockMvc mockMvc;
```

### //1. Post Method case

```

    @Test
    public void testProductSave() throws Exception {
        MvcResult result = mockMvc.perform(post("/product/register")
            .contentType(MediaType.APPLICATION_JSON)
            .content("{\"prodCode\":\"ABCD\", \"prodCost\":88.55, \"vendorCode\":\"V11\"}")
            .andReturn());
        MockHttpServletResponse resp = result.getResponse();
        assertEquals(HttpStatus.OK.value(), resp.getStatus());
        System.out.println(resp.getContentAsString());
        assertNotNull(resp.getContentAsString());
    }

```

### //3. Put Method Test Case

```

    @Test
    public void testProductPut() throws Exception {
        MvcResult result = mockMvc.perform(put("/product/edit")
            .contentType(MediaType.APPLICATION_JSON)
            .content("{\"prodId\":1, \"prodCode\":\"ABCDE\", \"prodCost\":38.55, \"vendorCode\":\"V14\"}"))
            .andReturn();
        MockHttpServletResponse resp = result.getResponse();
        assertEquals(HttpStatus.OK.value(), resp.getStatus());
        System.out.println(resp.getContentAsString());
        assertNotNull(resp.getContentAsString());
    }

```

### //2. Get Method Test Case

```

    @Test
    public void testProductGet() throws Exception {
        MvcResult result = mockMvc.perform(get("/product/get/4")).andReturn();
        MockHttpServletResponse resp = result.getResponse();
        assertEquals(HttpStatus.OK.value(), resp.getStatus());
        assertNotNull(resp.getContentAsString());
    }

```

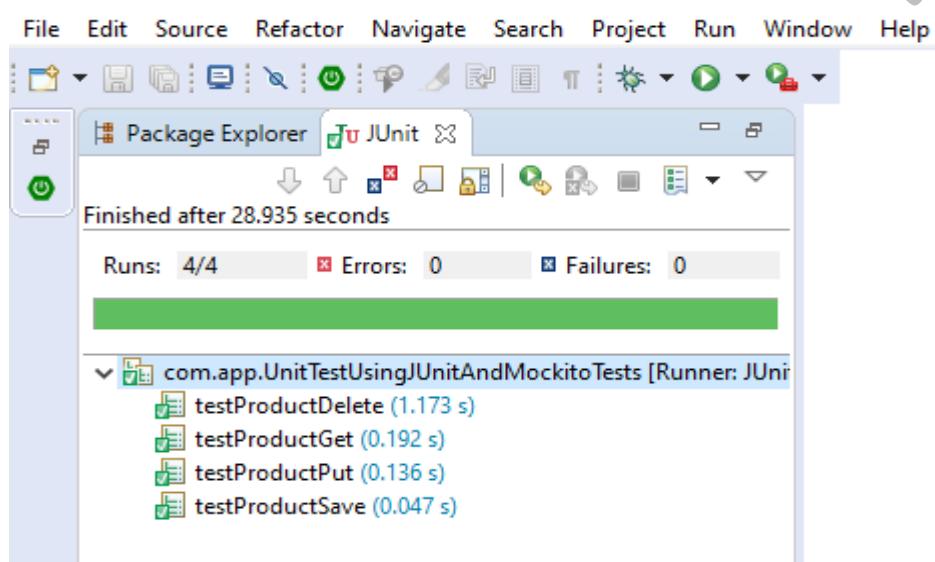
## //4. Delete Method Test Case

```

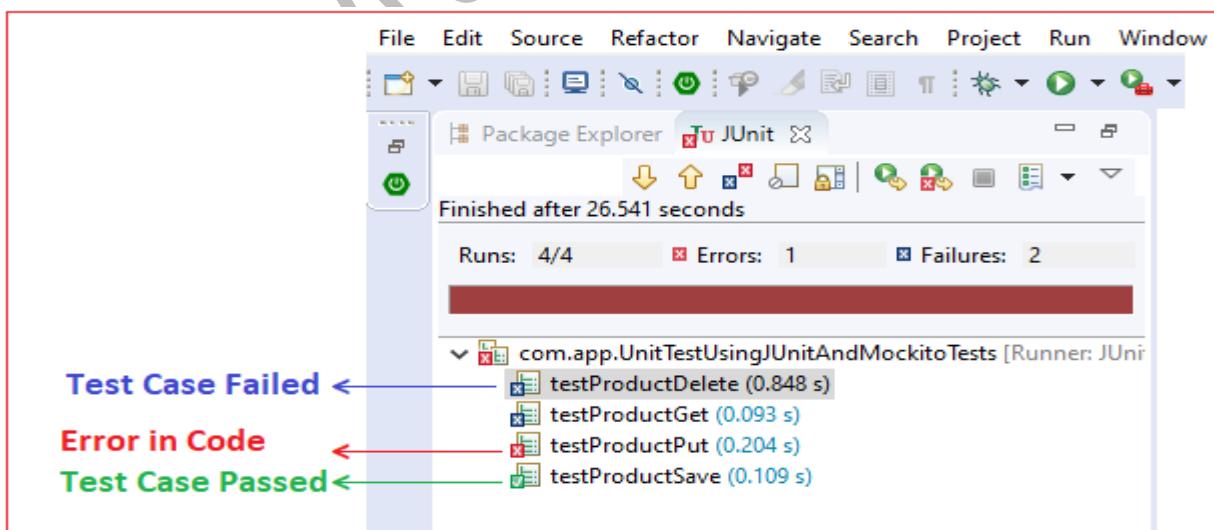
    @Test
    public void testProductDelete()throws Exception {
        MvcResult result=mockMvc.perform(delete("/product/delete/5")).andReturn();
        MockHttpServletResponse resp=result.getResponse();
        assertEquals(HttpStatus.OK.value(), resp.getStatus());
        System.out.println(resp.getContentAsString());
        assertNotNull(resp.getContentAsString());
    }
}

```

## 1&gt;Output SCRENN Short of JUnit:--



## 2&gt;Output Screen for all test cases.



**NOTE:--** 1>Green Color(testProductSave) indicate Test cases passed.

2>Blue color(testProductDelete,testProductGet) indicates test cases failed.

3>Red color(testProductPut) indicate error in test class specific method code.

## **\*\*\*Gradle\*\*\***

**Gradle:**-- Gradle is a build tool. It works based on Groovy Language.

=>It gets Jars related to Project.

=>It will be connected to parent projects as plugin.

=>Provides Java Application support.

=>Connected to multiple server for jars fetching Ex:- Jcenter, mavenCentral etc.

=>Supports compile our file.

=>Creates .jar/.war file.

=>\*\*Maven is XML based Build tool where Gradle is Language Script.

=>Both are used for same purpose, Gradle is faster a bit compared to Maven.

=>\*\*Gradle is a task based Scripting. It means, for every work/setup we need to write one task. These are pre defined in Gradle.

**Format looks like:**

```
task {  
    //details  
}
```

-----Example Task-----

```
plugins{ ..... }  
bootJar { ..... }  
repositories { ..... }  
dependencies { ..... }
```

**Gradle setup in STS 4:**--

=> File -> New -> Other ->

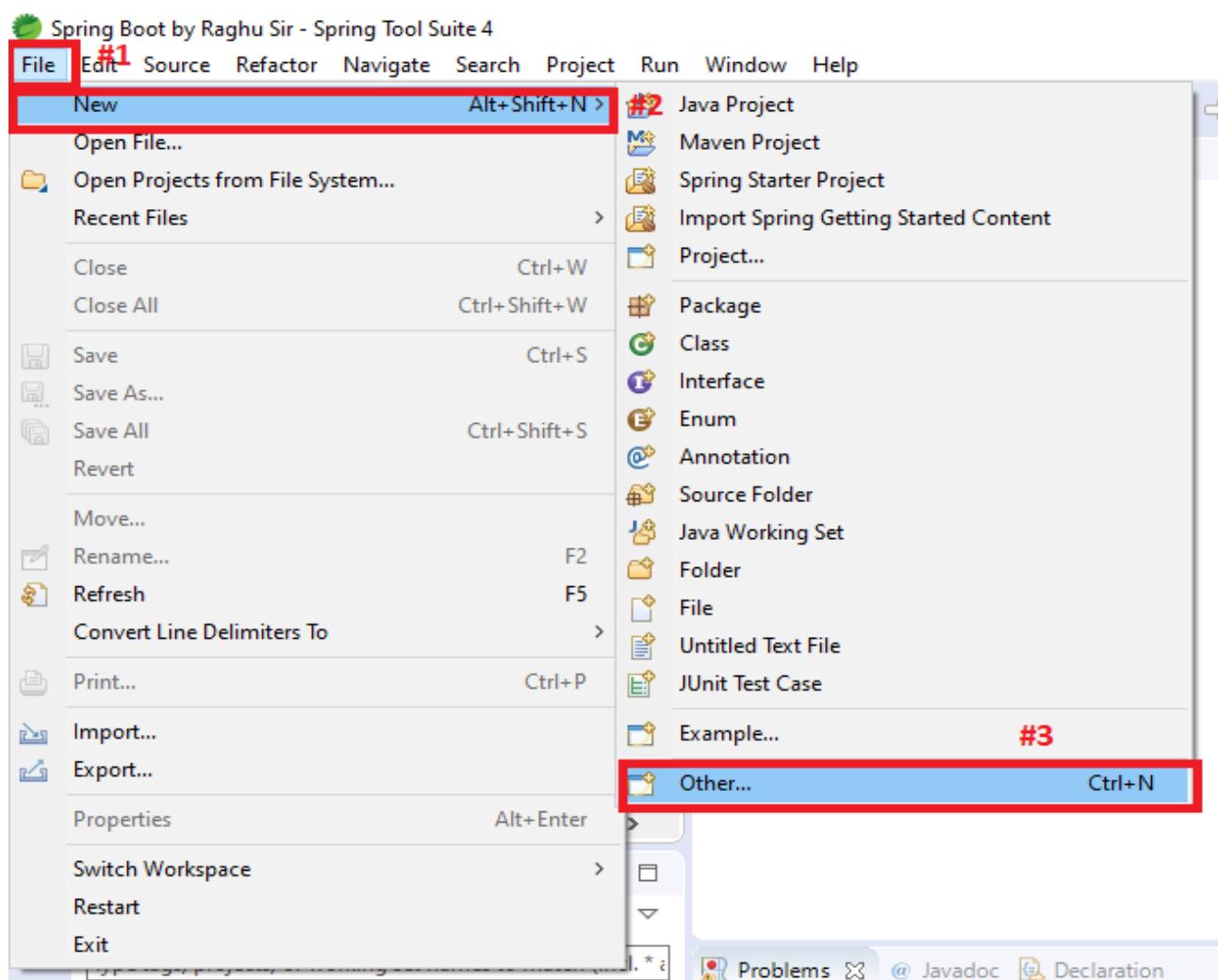
=>Search using Gradle ->

=>Enter Project Name ->

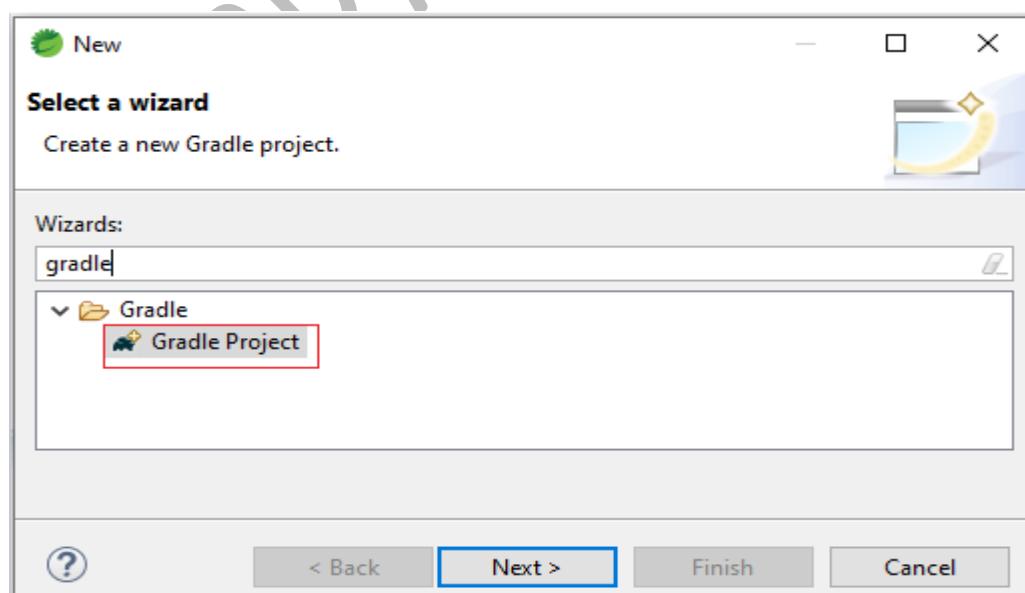
=>Next/Finish.

## 1. Screen shot step by step to create Gradle Project:-

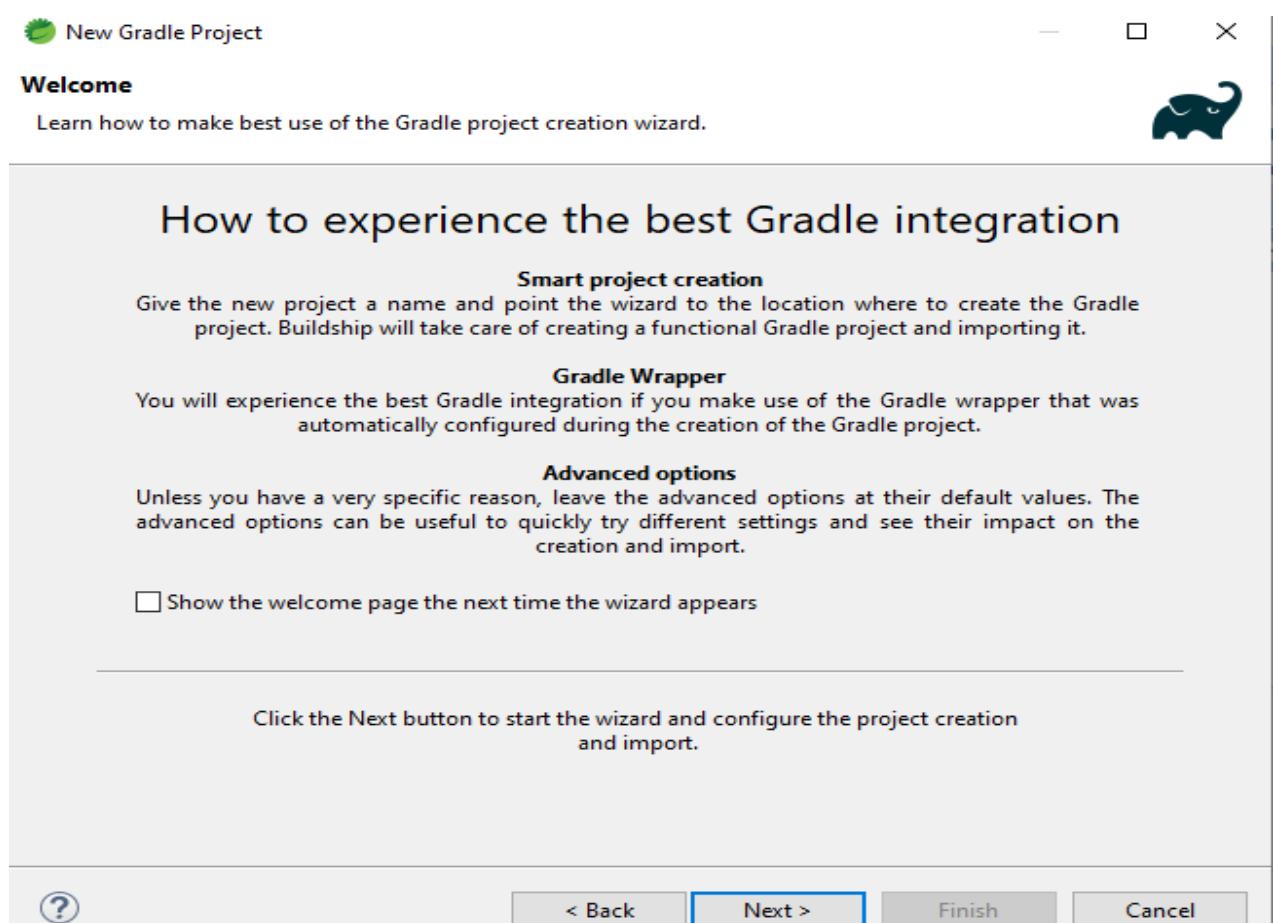
**Step#1:- Click on File > New > Other**



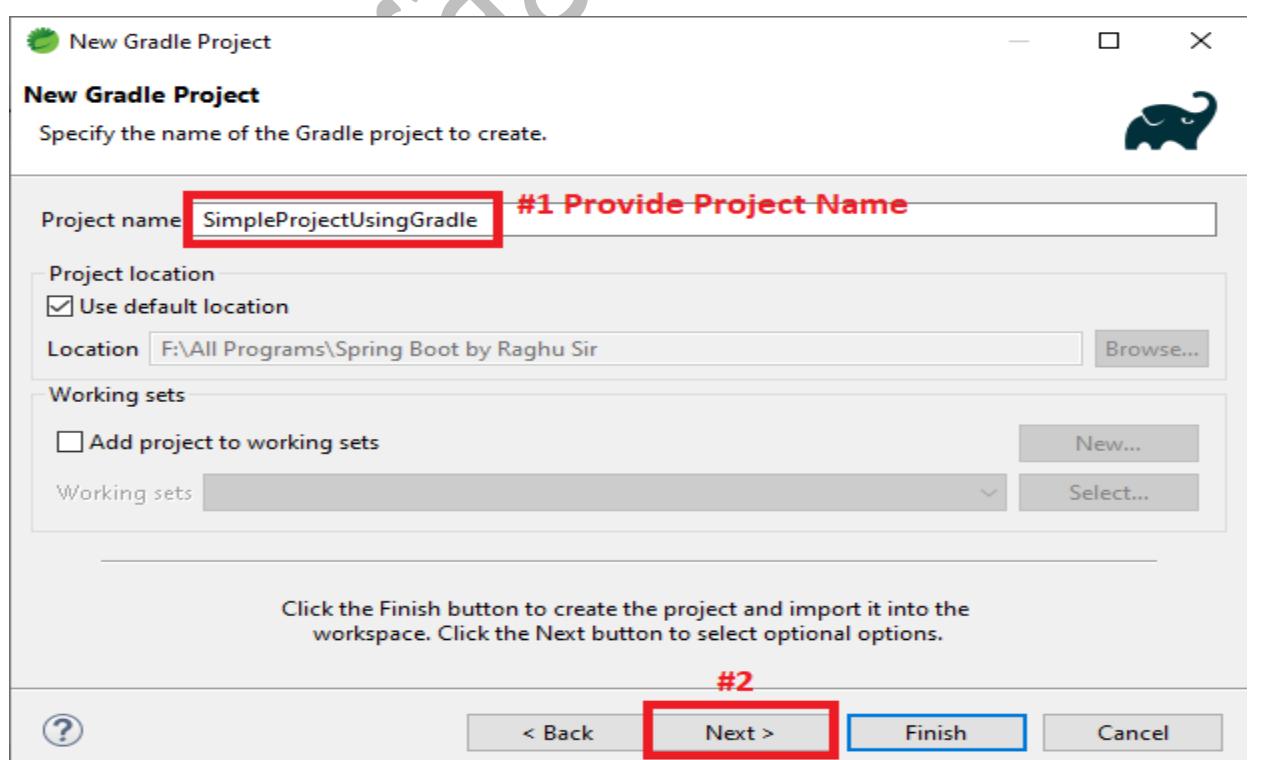
**Step#2:- Search with Gradle > Select “Gradle Project” and click on Next.**



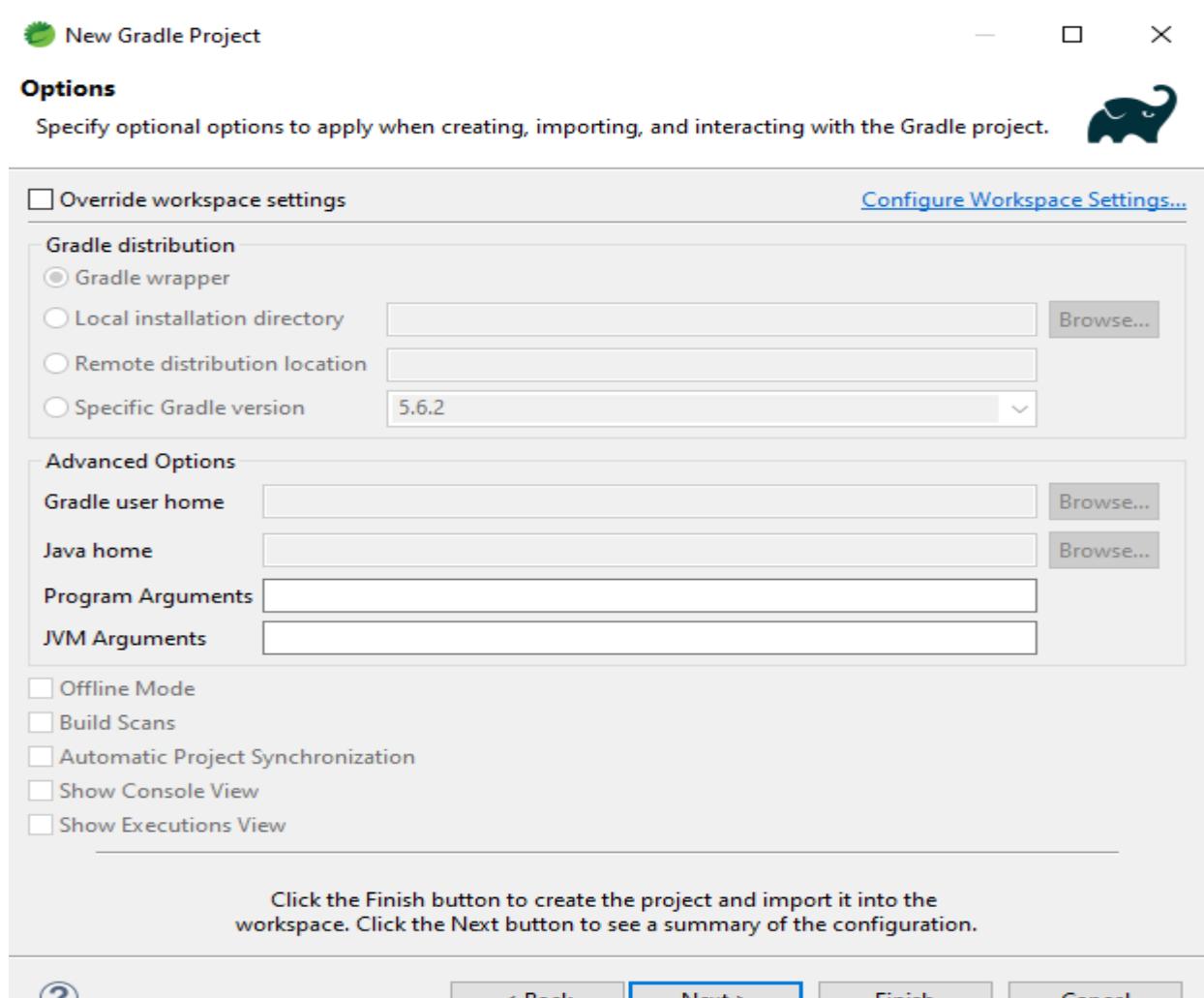
**Step#3:- Click on Next.**



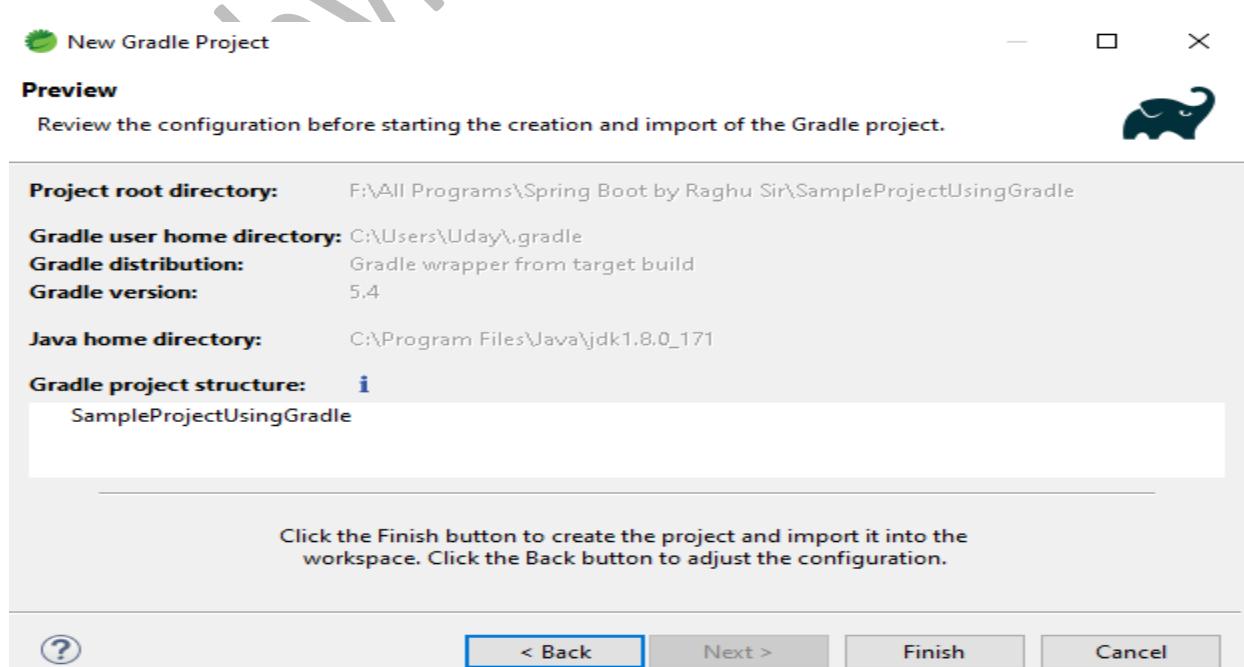
**Step#4:- Provide Project Name and Click on Next/Finish.**



**Step#5:- Click on Next.**

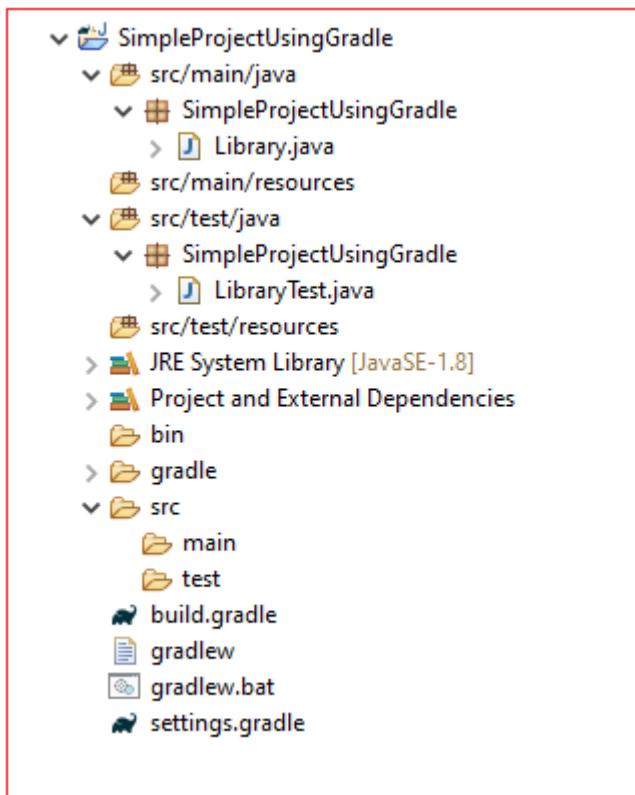


**Step#6:- Click on Next.**



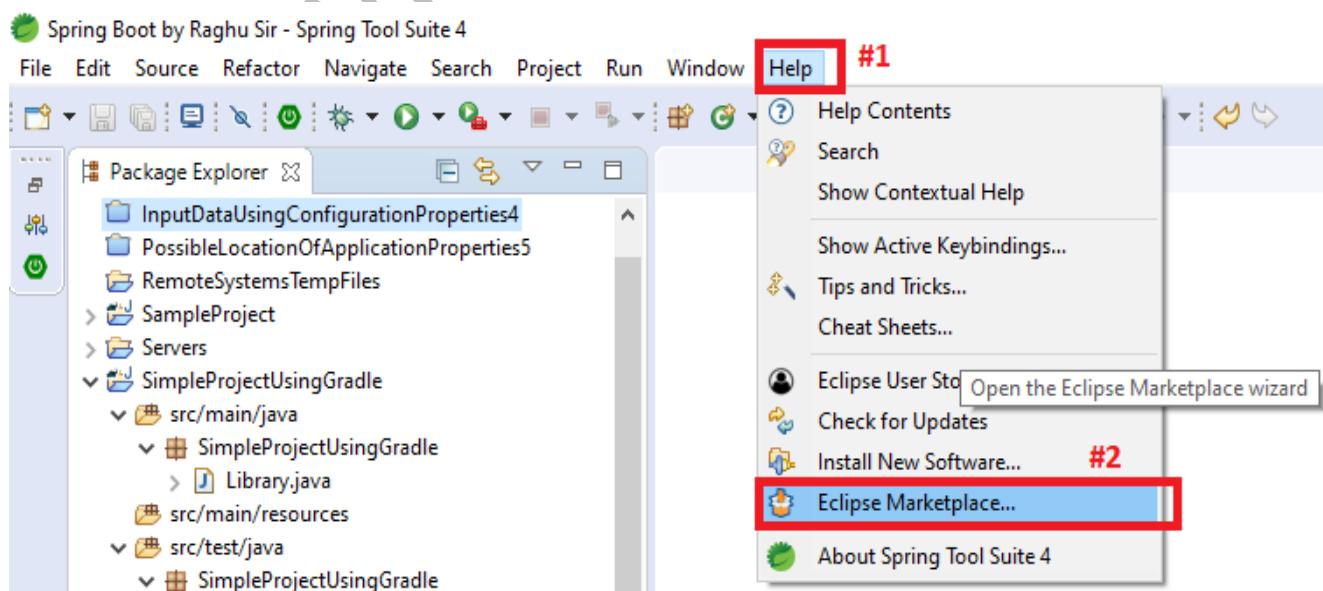
**Step#7:-** Click on Finish.

=>After click on finish Menu final project is created like below.

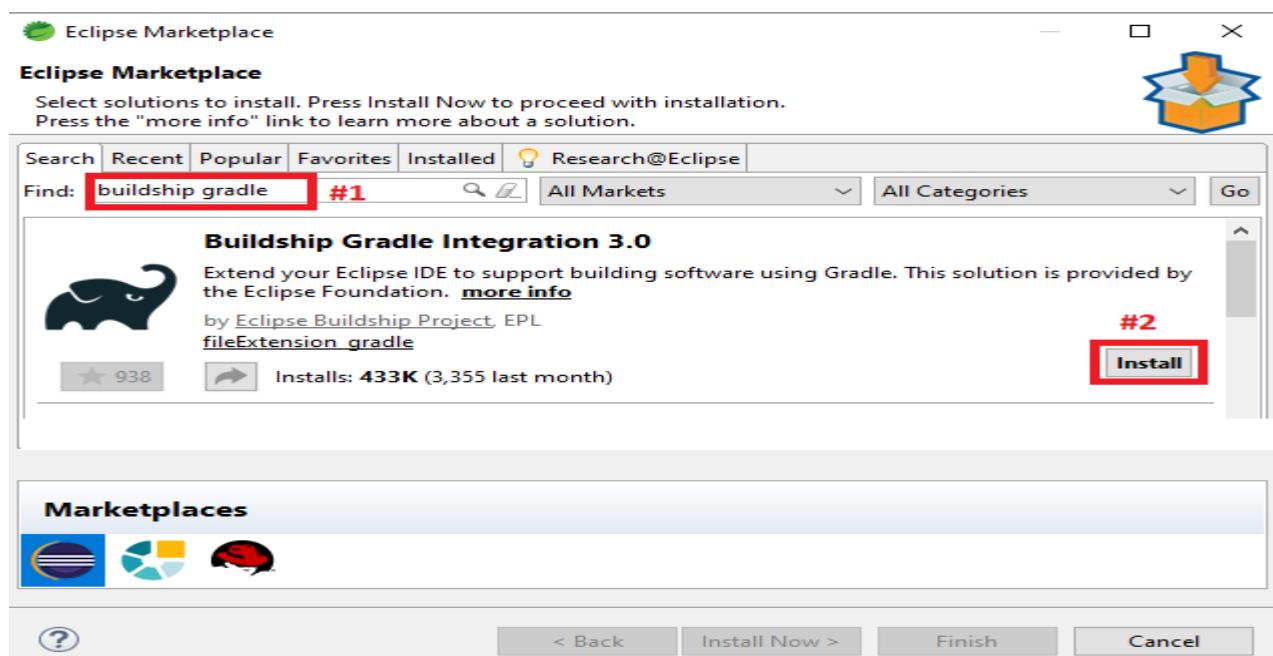


**NOTE:**--\*\*\*If Gradle option is not available then goto Eclipse Market place and download “Buildship Gradle Integration”.

**Step#1:-** Goto Eclipse/STS help Menu and click on “Eclipse Marketplace”.



**Step#2:-** Search with “Buildship Gradle” and click on Install, After install restart the System.



=>Project is created using folder src/main/java, src/test/java ... etc.

=>build.gradle is a file which contains all task details. Those are executed by gradlew (or gradlew.bat).

**task: dependencies:**-- This task is used to provide Jars related to project using one scope.

**Dependency contains:-**

Scope

Group

Name

Version

**Ex:-**

compile

group : 'org.springframework',

name : 'spring-context',

version : '5.1.8.RELEASE'

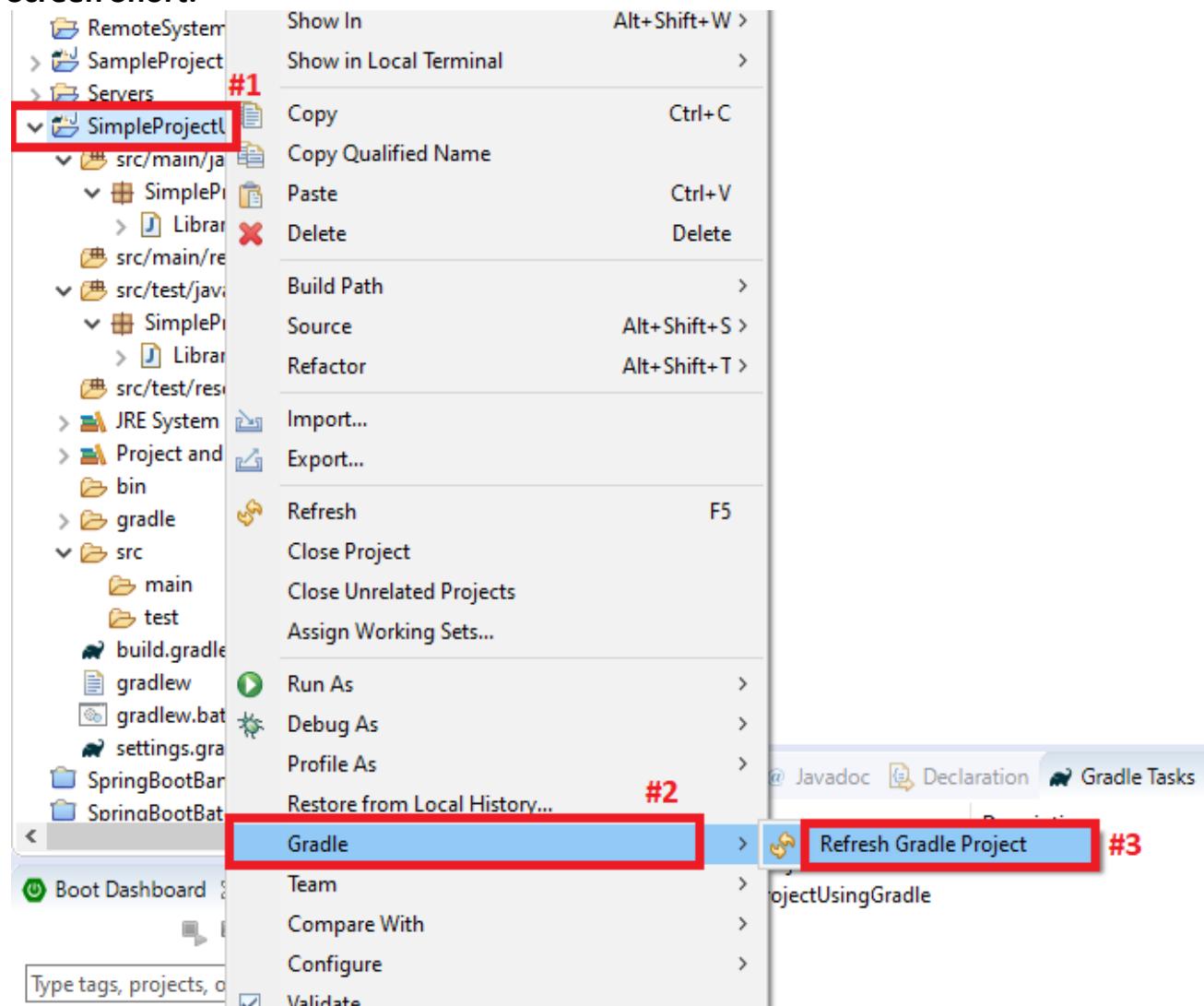
=>We can also write in shot format as  
scope 'group:name:version'

**Ex:-** testCompile 'junit:junit:4.10'

**\*\*Refresh Project (To get updated External dependencies contain(jar))**

=>Right click on Project > Gradle > Refresh Gradle Project

### Screen Short:--



### Scopes in Gradle:--

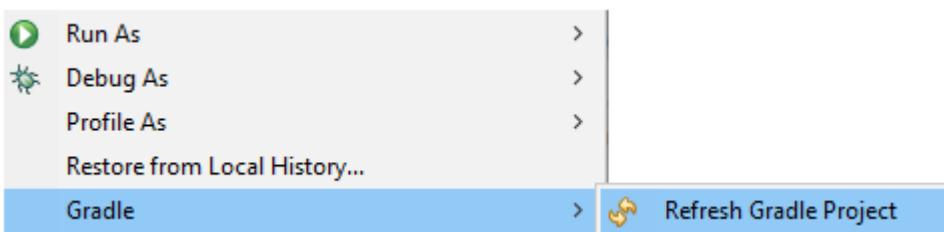
- a. compile
  - b. testCompile
  - c. providedCompile
  - d. runtime
  - e. api //same like system in maven
- 
- a. compile:-- Jar will be available from Application compile time onwards.
  - b. testCompile:-- Jar used at UnitTesting
  - c. providedCompile:-- Jar given by 3<sup>rd</sup> party (server, f/w containers...).
  - d. runtime:-- Jar used/loaded at runtime
  - e. api:-- Jar used by Application API & documentation (loaded from system with high priority)

**2. Configuring Gradle:**-- You need to add the configuration for your application to become "WEB Application". And can be run directly on Eclipse + Tomcat Plugin.

```
apply plugin: 'java'  
apply plugin: 'war'  
apply plugin: 'com.bmuschko.tomcat'  
apply plugin: 'eclipse-wtp'  
  
repositories {  
    jcenter()  
}  
  
sourceCompatibility = 1.8  
targetCompatibility = 1.8  
  
dependencies {  
    compile 'com.google.guava:guava:23.0'  
    testCompile 'junit:junit:4.12'  
    providedCompile "javax.servlet:javax.servlet-api:3.1.0"  
}  
  
dependencies {  
    def tomcatVersion = '8.0.53'  
    tomcat "org.apache.tomcat.embed:tomcat-embed-core:${tomcatVersion}",  
    "org.apache.tomcat.embed:tomcat-embed-loggingjuli:${tomcatVersion}",  
    "org.apache.tomcat.embed:tomcat-embed-jasper:${tomcatVersion}"  
}  
  
buildscript {  
    repositories {  
        jcenter()  
    }  
}  
  
dependencies {  
    classpath 'com.bmuschko:gradle-tomcat-plugin:2.5'  
}
```

**NOTE:**-- Each time there is a change in build.gradle you need to update the project, using the tool of Gradle.

=>Right click on Project > Gradle > Refresh Gradle Project



### 3. Add Folder To Application:--

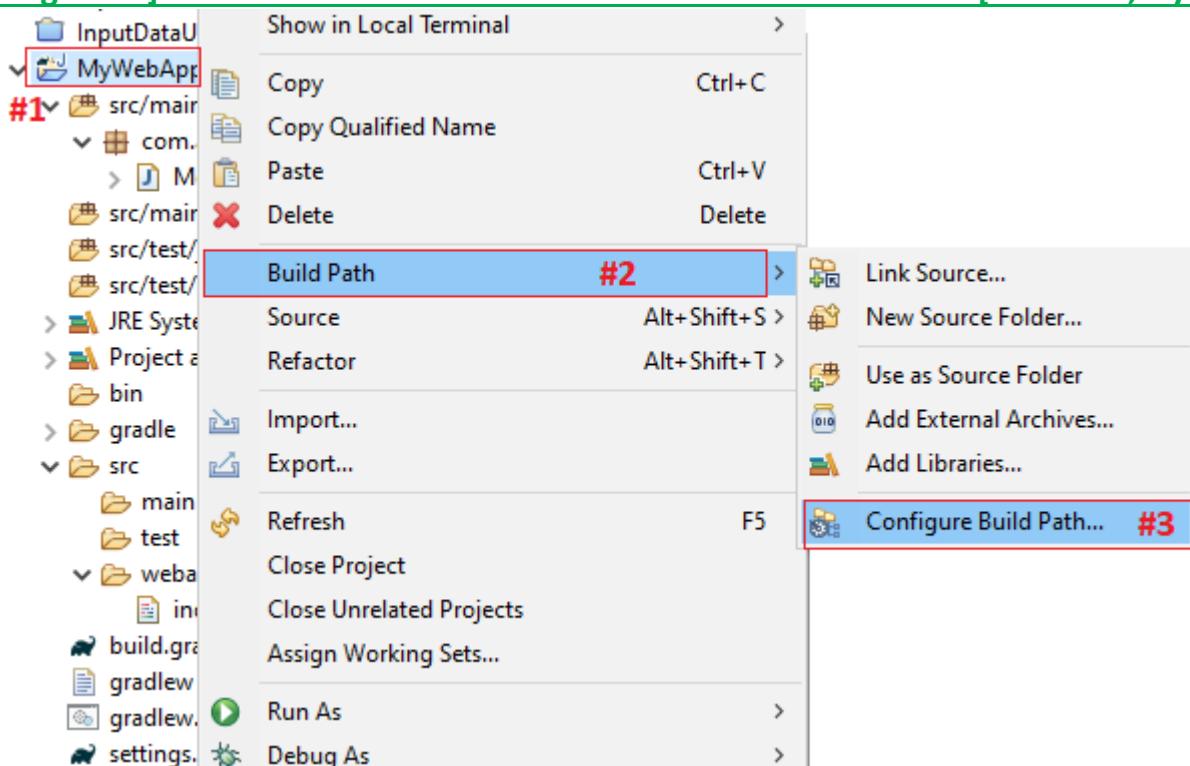
- >In "src/main" folder, you need to create 2 sub folders
- >Those are "resources" and "webapp".
- >src/main/java: This folder has java sources.
- >src/main/resources: This folder can hold property files and other resources
- >src/main/webapp: This folder holds jsp and other web application content.

#### Create folder:--

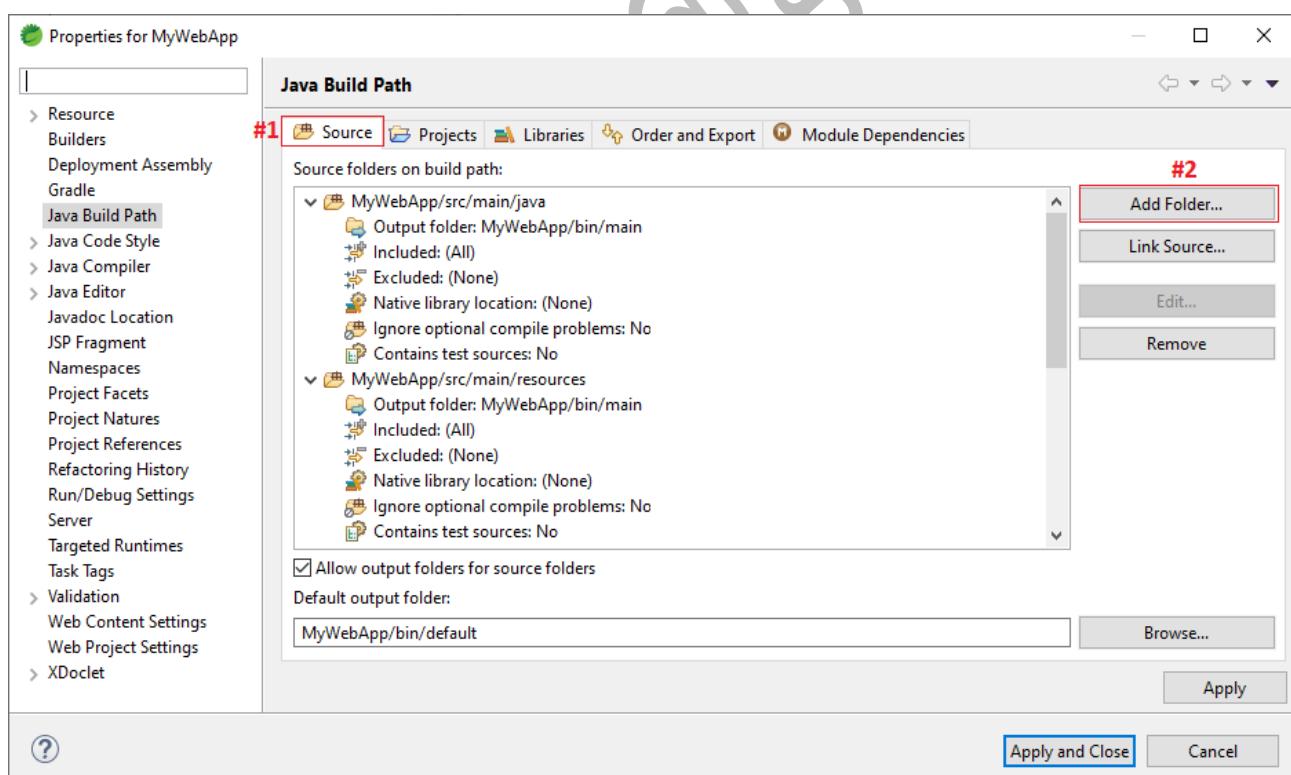
- =>Right click on Project > build path > Configure Build path >
- >Source > add folder > new folder >
- Enter name ex: src/main/resources -> finish.

#### Screen Shot:--

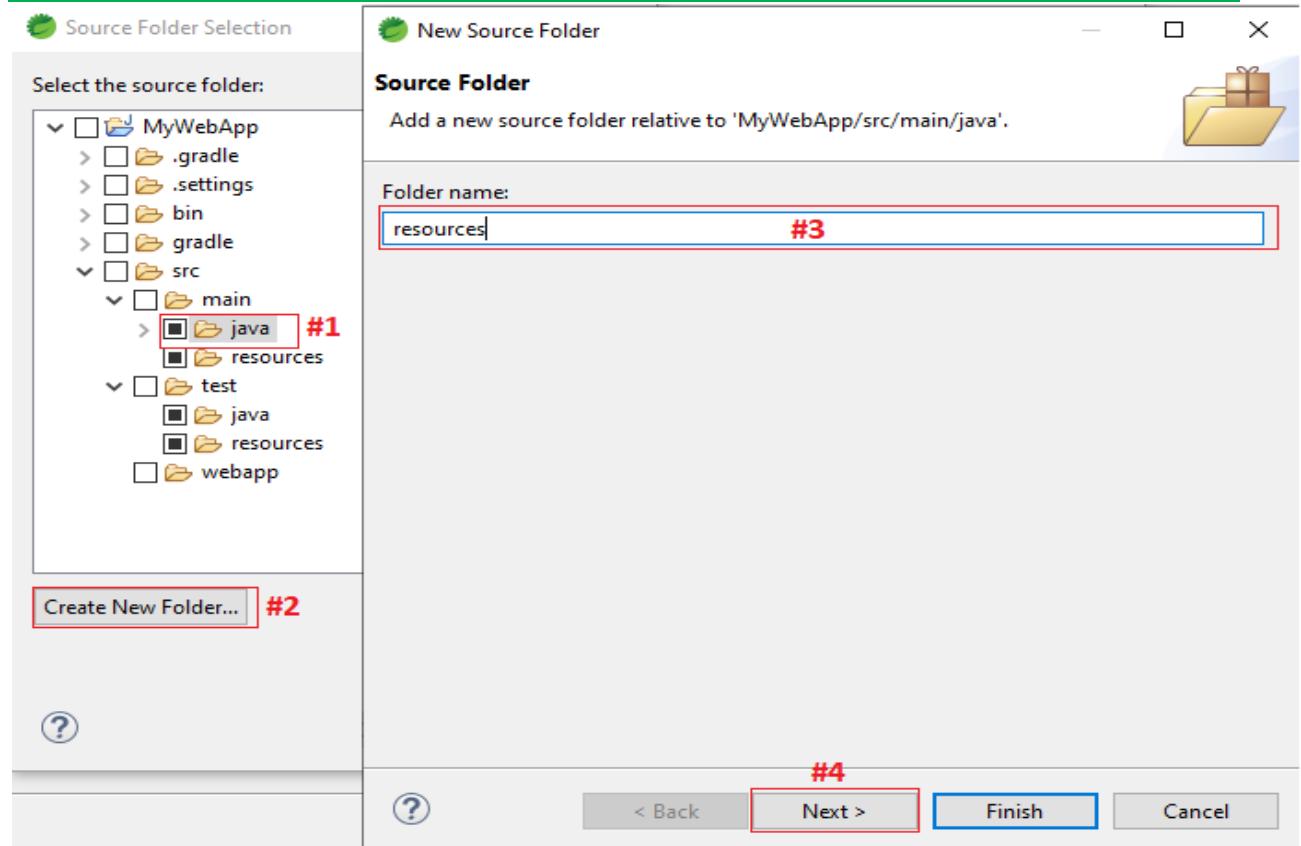
**Step#1:-** Right click on Project > Build Path > Configure Build Path



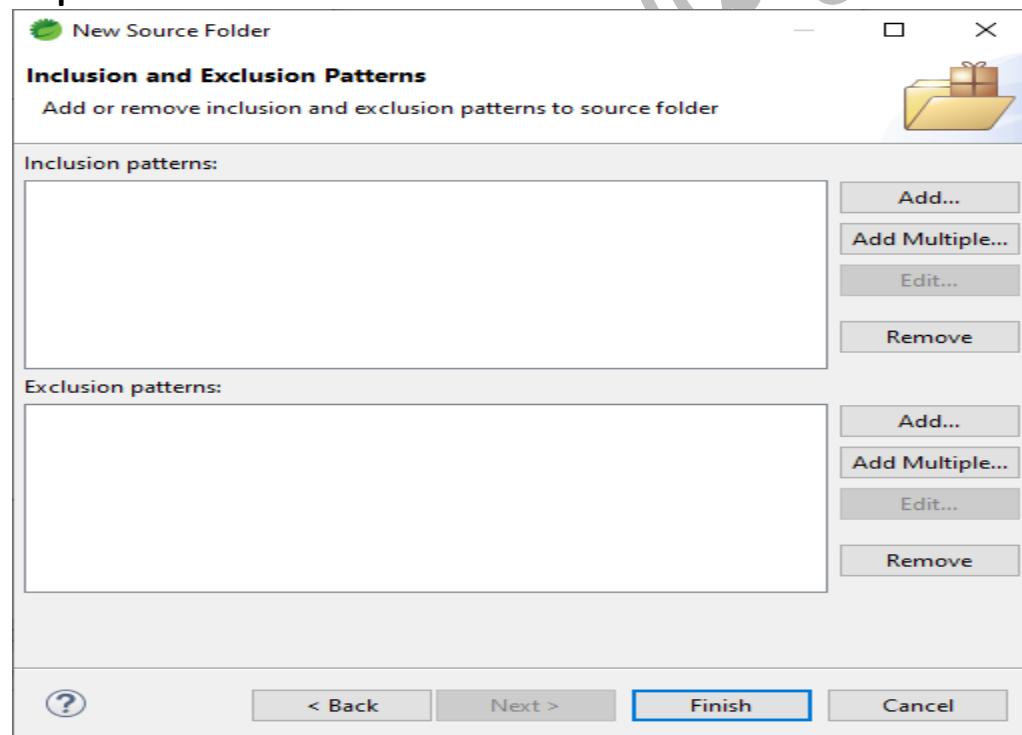
**Step#2:- Click on Source Option > Click on “Add Folder”.**



**Step#3:- Select folder in which you want to create folder > Click on Create New Folder > Enter Folder Name like (resources) > click on Next.**



**Step#4:- Click on Finish**



#### 4. Write Code:--

1>Service class, jsp, html etc..

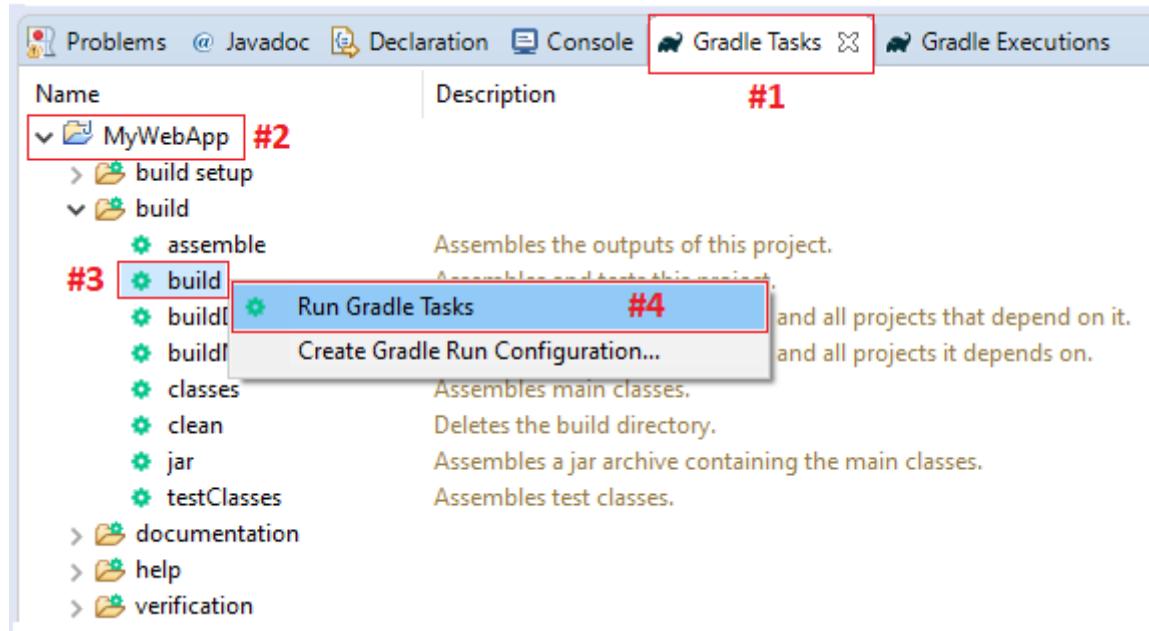
#### 5. Execute Gradle Build Task:--

=>Open “Gradle Task” view >

=>Expand Project > Choose build >

=>Right click on build > Run Gradle Tasks

Screen Shot:--



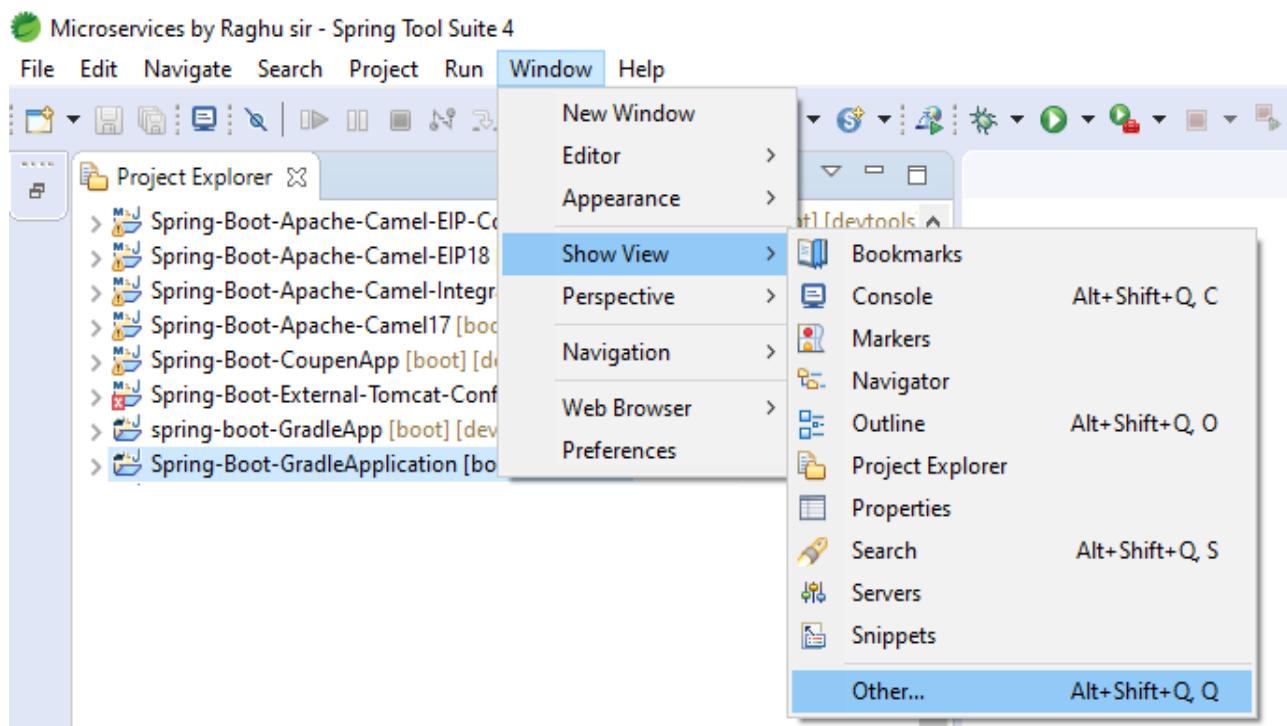
**NOTE:--** If “Gradle Tasks” and “Gradle Execution” is not showing bellow then

=>Goto “Window” Menu > Show View > Other...>

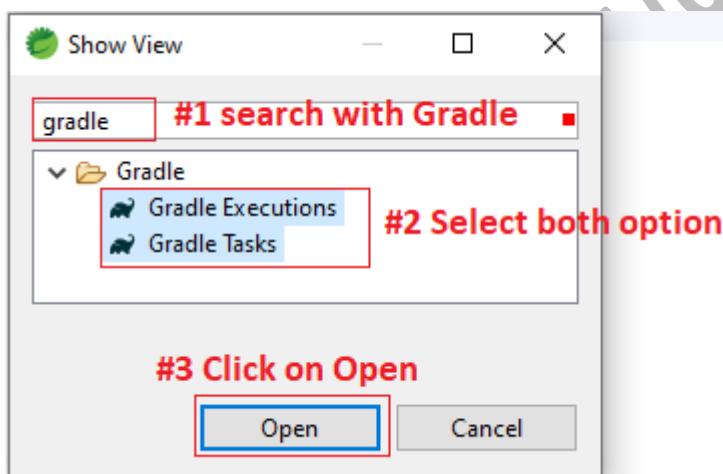
=>Search with Gradle > Select both option >

=>Click on Open Button.

Screnn#1:-

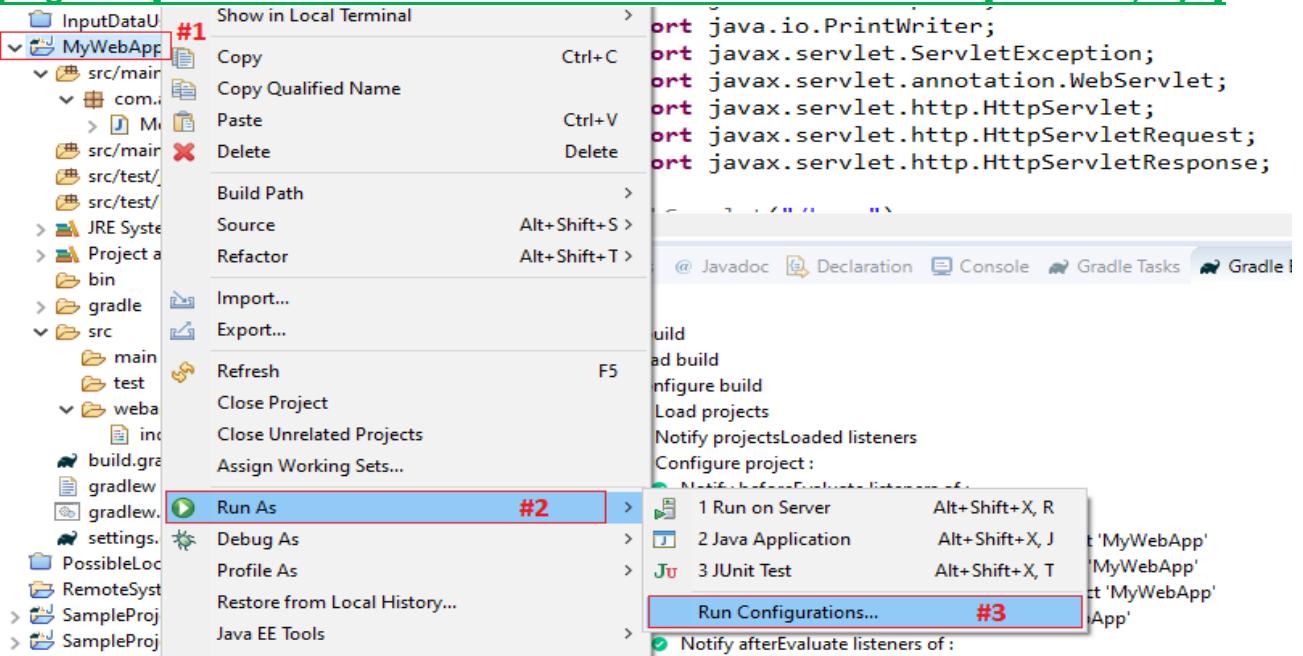


Screen#2:-

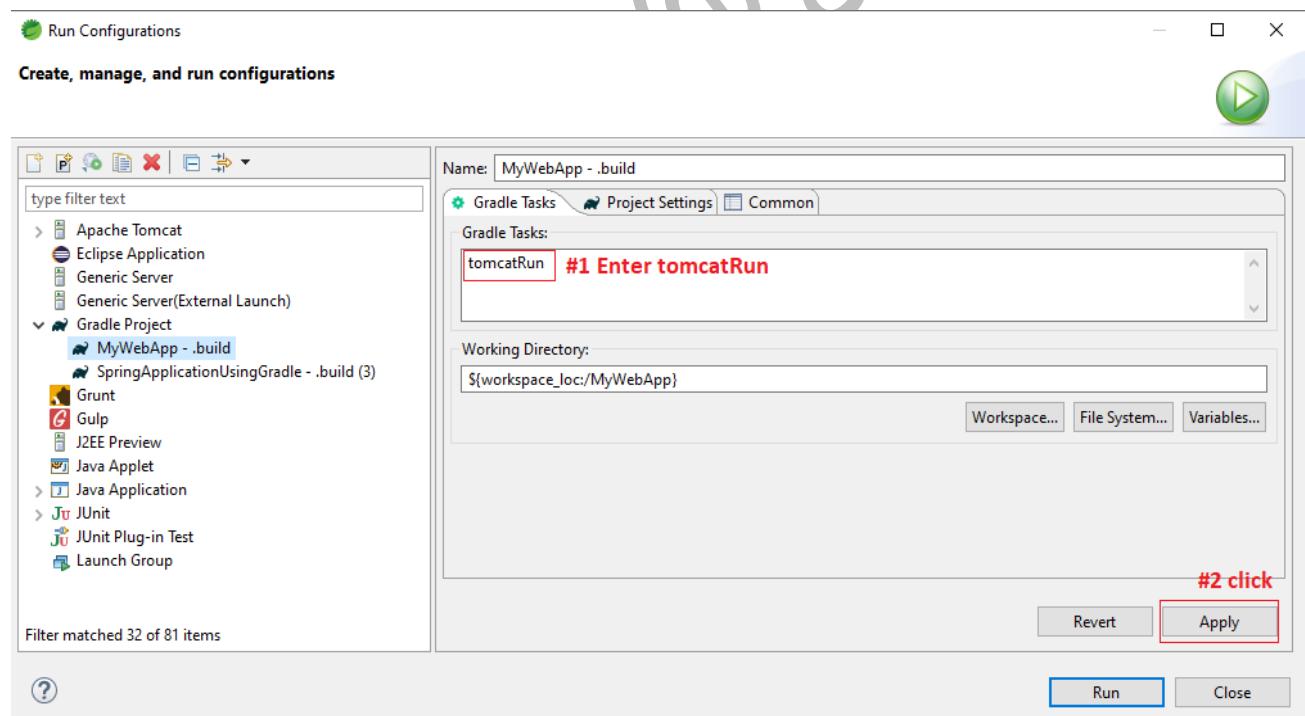


## 6. Configure To Run Application:-

**Step#1:-** Right click on Project > Run As > Run Configuration

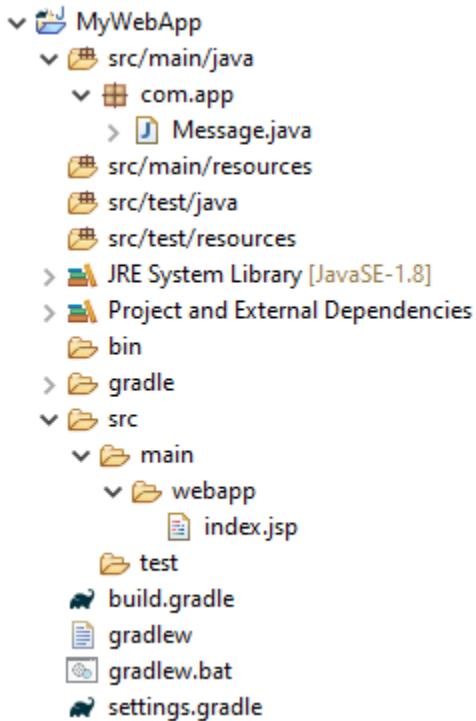


**Step#2:- Enter > Apply > Run**



## 1. SERVLETS/JSP WEB APPLICATION USING GRADLE:--

### Folder Structure of Servlet Web App:--



**Code:--**

**1. build.gradle:--**

```

apply plugin: 'java'
apply plugin: 'war'
apply plugin: 'com.bmuschko.tomcat'
apply plugin: 'eclipse-wtp'

sourceCompatibility = 1.8
targetCompatibility = 1.8

repositories {
    //mavenCentral()
    jcenter()
}

dependencies {
    providedCompile 'javax.servlet:javax.servlet-api:3.1.0'
    compile 'org.springframework:spring-webmvc:5.1.8.RELEASE'
    compile 'com.fasterxml.jackson.core:jackson-databind:2.9.5'
    runtime 'javax.servlet:jstl:1.2'

    def tomcatVersion = '8.0.53'

    tomcat "org.apache.tomcat.embed:tomcat-embed-core:${tomcatVersion}",
}

```

```

        "org.apache.tomcat.embed:tomcat-embed-logging-juli:${tomcatVersion}",
        "org.apache.tomcat.embed:tomcat-embed-jasper:${tomcatVersion}"
    }

buildscript {
repositories {
    //mavenCentral()
    jcenter()
}

dependencies {
    classpath 'com.bmuschko:gradle-tomcat-plugin:2.5'
}
}

```

## 2. Servlet class (Message.java):--

```

package com.app;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/home")
public class Message extends HttpServlet {

    private static final long serialVersionUID = -4633811587840173475L;

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {

        PrintWriter out=resp.getWriter();
        out.println("Hello App");
    }
}

```

## 3. index.jsp:--

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
```

```

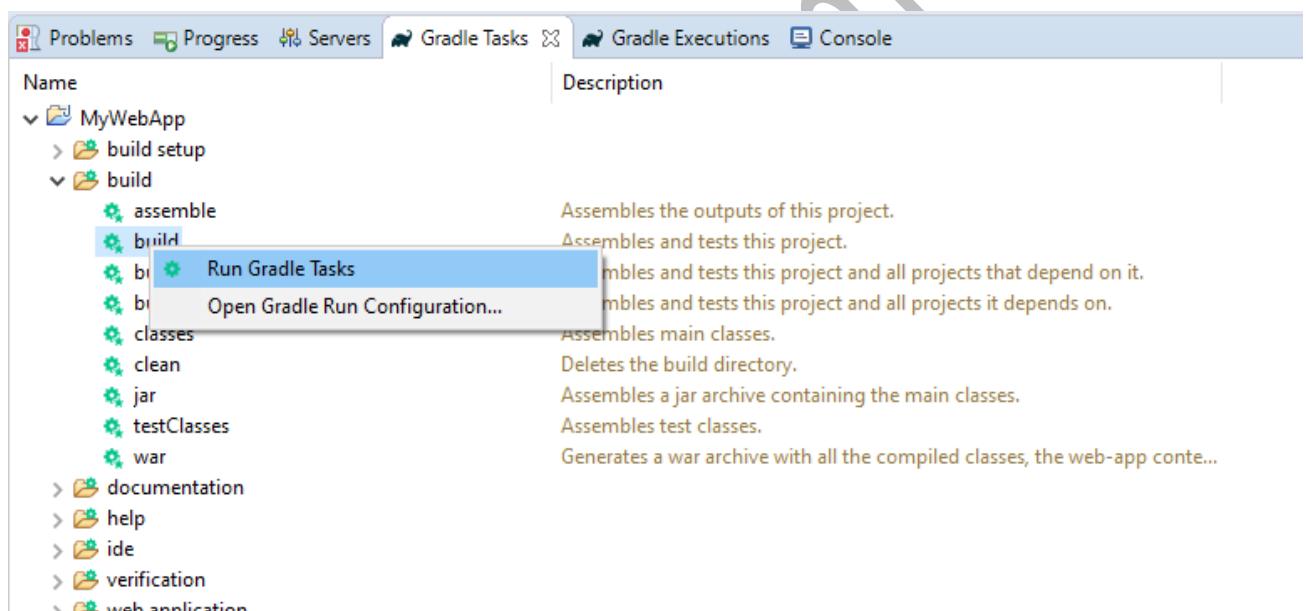
pageEncoding="ISO-8859-1">%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h1>Welcome to All</h1>
</body>
</html>

```

### Execution flow:--

#### Step#1:- Execute Gradle build task

=>Open "Gradle Task" view > Expand Project > choose build > Run Gradle Tasks



=>See Success message here

Operation	Duration
Run build	20.763 s
Load build	0.063 s
Configure build	0.623 s
Calculate task graph	1.335 s
Run tasks	18.703 s
Notify task graph whenReady listeners	0.005 s
:compileJava	16.621 s
:processResources	0.024 s
:classes	0.012 s
:war	1.755 s
:assemble	0.009 s
:compileTestJava	0.014 s
:processTestResources	0.008 s
:testClasses UP-TO-DATE	0.001 s
:test	0.015 s
:check UP-TO-DATE	0.001 s
:build	0.001 s
Build model 'java.lang.Void' for root project'	0.000 s

=>See Build success message on console.

```

Problems Progress Servers Gradle Tasks Gradle Executions Console
MyWebApp - .build (1) [Gradle Project] :build in F:\All Programs\Spring Boot by Raghu Sir\MyWebApp (Oct 8, 2019 3:20:39 PM)
Working Directory: F:\All Programs\Spring Boot by Raghu Sir\MyWebApp
Gradle user home: C:\Users\Uday\.gradle
Gradle Distribution: Gradle wrapper from target build
Gradle Version: 5.4
Java Home: C:\Program Files\Java\jdk1.8.0_171
JVM Arguments: None
Program Arguments: None
Build Scans Enabled: false
Offline Mode Enabled: false
Gradle Tasks: :build

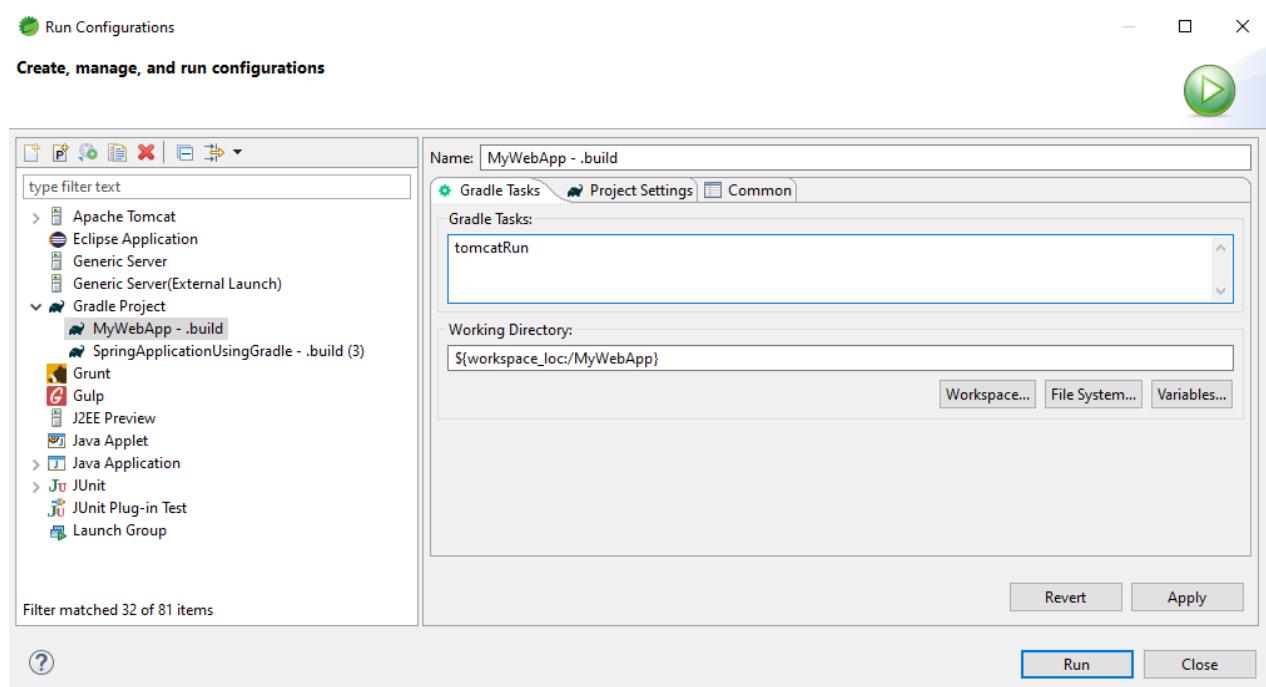
> Task :compileJava
> Task :processResources NO-SOURCE
> Task :classes
> Task :war
> Task :assemble
> Task :compileTestJava NO-SOURCE
> Task :processTestResources NO-SOURCE
> Task :testClasses UP-TO-DATE
> Task :test NO-SOURCE
> Task :check UP-TO-DATE
> Task :build

BUILD SUCCESSFUL in 21s
2 actionable tasks: 2 executed

```

## STEP#2:- RUN APPLICATION IN TOMCAT

=> Right click on Project > Run As > enter tomcatRun > Apply and Run



=>Goto Console to See the confirmation on which port no Application is running

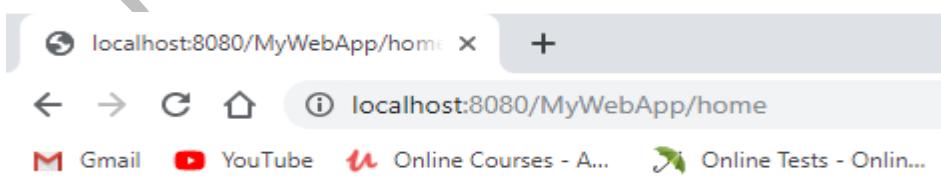
```

Problems Progress Servers Gradle Tasks Gradle Executions Console
Spring-Project-Using-Gradle - .build (1) [Gradle Project] tomcatRun in F:\All Programs\Spring Boot by Raghu Sir\Spring-Project-Using-Gradle (Oct 8, 2019 2:24:23 AM)
Working Directory: F:\All Programs\Spring Boot by Raghu Sir\Spring-Project-Using-Gradle
Gradle user home: C:\Users\Uday\.gradle
Gradle Distribution: Gradle wrapper from target build
Gradle Version: 5.4
Java Home: C:\Program Files\Java\jdk1.8.0_171
JVM Arguments: None
Program Arguments: None
Build Scans Enabled: false
Offline Mode Enabled: false
Gradle Tasks: tomcatRun

> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :tomcatRun
Started Tomcat Server
The Server is running at http://localhost:8080/Spring-Project-Using-Gradle

```

**STEP#3:- ENTER URL IN BROWSER ( <http://localhost:8080/MyWebApp/home> )**



Hello -App

## 2. Spring Application Using Gradle:--

**Step#1:-** Create one Gradle Project  
Ex:- Spring-Project-Using-Gradle

### Folder Structure of Spring Application using Gradle:-

```
└── Spring-Project-Using-Gradle
    ├── src/main/java
    │   ├── com.app.config
    │   │   └── AppConfig.java
    │   ├── com.app.init
    │   │   └── Applit.java
    │   └── com.app.rest
    │       └── EmployeeController.java
    ├── src/main/resources
    ├── src/test/java
    └── src/test/resources
    ├── JRE System Library [JavaSE-1.8]
    ├── Project and External Dependencies
    │   ├── bin
    │   ├── gradle
    └── src
        ├── main
        │   ├── webapp
        │   └── test
        ├── build.gradle
        ├── gradlew
        ├── gradlew.bat
        └── settings.gradle
```

**Step#2:-** open build.gradle file and type below content

```
apply plugin: 'java'
apply plugin: 'war'
apply plugin: 'com.bmuschko.tomcat'
apply plugin: 'eclipse-wtp'

sourceCompatibility = 1.8
targetCompatibility = 1.8

repositories {
    mavenCentral()
    //jcenter()
}
dependencies {
```

```

providedCompile 'javax.servlet:javax.servlet-api:3.0.1'
compile 'org.springframework:spring-webmvc:5.1.8.RELEASE'
compile 'com.fasterxml.jackson.core:jackson-databind:2.9.5'
runtime 'javax.servlet:jstl:1.2'

def tomcatVersion = '8.0.53'

tomcat "org.apache.tomcat.embed:tomcat-embed-core:${tomcatVersion}",
        "org.apache.tomcat.embed:tomcat-embed-logging-juli:${tomcatVersion}",
        "org.apache.tomcat.embed:tomcat-embed-jasper:${tomcatVersion}"
}

buildscript {
    repositories {
        mavenCentral()
        //jcenter()
    }
}

dependencies {
    classpath 'com.bmuschko:gradle-tomcat-plugin:2.5'
}
}

```

**Step#3:- AppConfig class**

```

package com.app.config;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;

@Configuration
@EnableWebMvc
@ComponentScan("com.app")
public class AppConfig {

}

```

**Step#4:- AppInit class**

```

package com.app.init;
import org.springframework.web.servlet.support.
AbstractAnnotationConfigDispatcherServletInitializer;
import com.app.config.AppConfig;

```

```
public class AppInit extends AbstractAnnotationConfigDispatcherServletInitializer {

    @Override
    protected Class<?>[] getRootConfigClasses() {
        return null;
    }
    @Override
    protected Class<?>[] getServletConfigClasses() {
        return new Class[] { AppConfig.class };
    }
    @Override
    protected String[] getServletMappings() {
        return new String[] { "/" * };
    }
}
```

### Step#5:- Controller class

```
package com.app.rest;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/emp")
public class EmployeeController {

    @GetMapping("/msg")
    public String show() {
        return "Hello R-APP";
    }
}
```

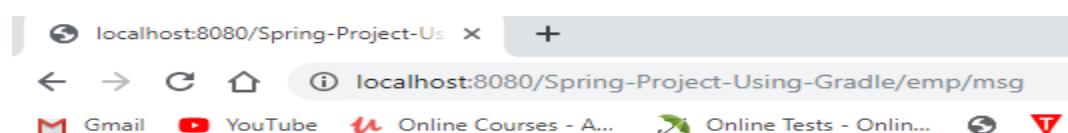
### Execution flow:-

Step#1:- Execute Gradle build task

Step#2:- Run application in tomcat

Step#3:- Enter URL in Browser

<http://localhost:8080/Spring-Project-Using-Gradle/emp/msg>



Hello R-APP

### 3. Spring Boot Application Using Gradle:--

=>In spring boot Gradle Application two plugins are provided those are

- a. Parent Project plugin
- b. BOM (Bill of Managements)

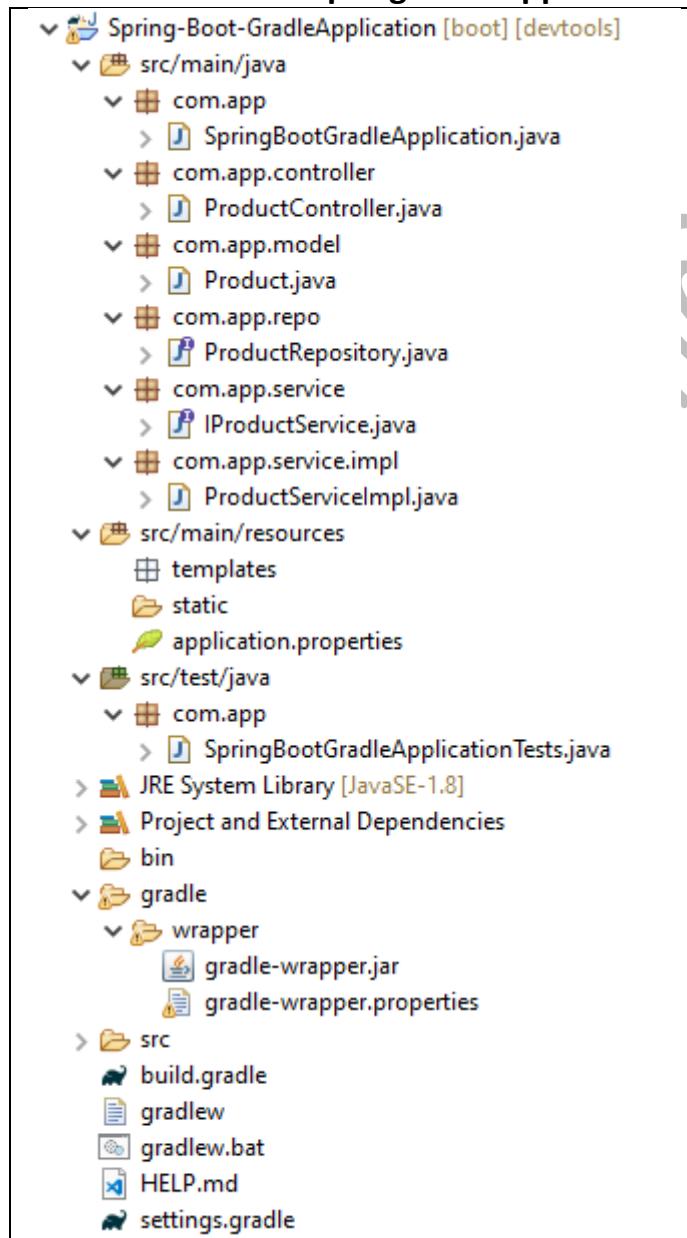
Parent : org.springframework.boot

BOM : io.spring.dependency-management

=>BOM means all child details and their versions as one project (.pom/.gradle file)

### Spring Boot Application using Gradle:--

#### Folder Structure of Spring Boot Application using Gradle:--



**Coding Steps of CRUD operation using Gradle:--**

**1. build.gradle:--**

```

plugins {
    id 'org.springframework.boot' version '2.1.8.RELEASE'
    id 'io.spring.dependency-management' version '1.0.8.RELEASE'
    id 'java'
}

group = 'com.app'
version = '1.0'
sourceCompatibility = '1.8'
targetCompatibility = '1.8'

configurations {
    developmentOnly
    runtimeClasspath {
        extendsFrom developmentOnly
    }
}

repositories {
    mavenCentral()
    mavenLocal()
}

dependencies {
    implementation('org.springframework.boot:spring-boot-starter-web',
        'org.springframework.boot:spring-boot-starter-data-jpa',
        'org.projectlombok:lombok')
    compile('org.springframework.boot:spring-boot-starter-actuator',
        'com.oracle:ojdbc6:11.2.0')
    developmentOnly 'org.springframework.boot:spring-boot-devtools'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
}

```

**2. application.properties:--**

```

server.port=2019
##DataSource##
spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe

```

```
spring.datasource.username=system
spring.datasource.password=system
##JPA##
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=create
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.Oracle10gDialect
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.open-in-view=true
```

**3. Starter class:--**

```
package com.app;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringBootGradleApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringBootGradleApplication.class, args);
        System.out.println("Spring Boot Gradle Application...!!!");
    }
}
```

**4. Model class:--**

```
package com.app.model;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;
import lombok.Data;

@Entity
@Table(name="PRODUCTTAB")
@Data
public class Product {

    @Id
    @Column(name="id")
    @GeneratedValue
    private Integer id;
```

```

    @Column(name="code")
    private String productCode;

    @Column(name="name")
    private String productName;
    @Column(name="cost")
    private Double productCost;
}

```

**5. Repository Interface:--**

```

package com.app.repo;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import com.app.model.Product;

@Repository
public interface ProductRepository extends JpaRepository<Product, Integer>{

    Product getProductById(Integer id);
}

```

**6. Service Interface:--**

```

package com.app.service;
import java.util.List;
import com.app.model.Product;

public interface IProductService {

    public Integer saveProduct(Product p);
    public List<Product> getAllProducts();
    public void deleteProduct(Integer id);
    public Product getProductById(Integer id);
    public boolean isProductExist(Integer id);
}

```

**7. ServiceImpl class:--**

```

package com.app.service.impl;
import java.util.List;
import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

```

```
import org.springframework.transaction.annotation.Transactional;
import com.app.model.Product;
import com.app.repo.ProductRepository;
import com.app.service.IProductService;
@Service
public class ProductServiceImpl implements IProductService {

    @Autowired
    private ProductRepository repo;

    //1. Save method
    @Transactional
    public Integer saveProduct(Product p) {
        p=repo.save(p);
        Integer proId=p.getId();
        return proId;
    }

    //2. Get all(List) Product details from Database
    @Transactional(readOnly= true)
    public List<Product> getAllProducts() {
        return repo.findAll();
    }

    //3. Delete Record based on ID
    //@Transactional
    public void deleteProduct(Integer id) {
        repo.deleteById(id);
    }

    //4. Get Record based on ID
    @Transactional
    public Product getProductById(Integer proId) {
        Optional<Product> p=repo.findById(proId);
        if(p.isPresent()) {
            return p.get();
        }else {
            return new Product();
        }
    }

    //5. Check product available or not
```

```

@Override
public boolean isProductExist(Integer id) {
    return repo.existsById(id);
}
}

```

**8. Controller class:--**

```

package com.app.controller;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.app.model.Product;
import com.app.service.IProductService;

@RestController
@RequestMapping("/product")
public class ProductController {

    @Autowired
    private IProductService service;

    //1. save student data
    @PostMapping("/save")
    public ResponseEntity<String> save(@RequestBody Product product){
        ResponseEntity<String> resp=null;

        try {
            Integer id=service.saveProduct(product);
            resp = new ResponseEntity<String>("Product "+id+" Created", HttpStatus.OK);
        } catch (Exception e) {
            resp = new ResponseEntity<String>(e.getMessage(),
                HttpStatus.INTERNAL_SERVER_ERROR);
            e.printStackTrace();
        }
    }
}

```

```

        return resp;
    }

//2. get All Records
@GetMapping("/all")
public ResponseEntity<?> getAll(){
    ResponseEntity<?> resp=null;

    List<Product> list=service.getAllProducts();

    if(list==null || list.isEmpty()) {
        String message="No Data Found";
        resp=new ResponseEntity<String>(message,HttpStatus.OK);
    } else {
        resp=new ResponseEntity<List<Product>>(list,HttpStatus.OK);
    }
    return resp;
}

//3. delete based on id , if exist
@DeleteMapping("/delete/{id}")
public ResponseEntity<String> deleteById(@PathVariable Integer id)
{
    ResponseEntity<String> resp=null;
    //check for exist
    boolean present=service.isProductExist(id);
    if(present) {
        //if exist
        service.deleteProduct(id);
        resp=new ResponseEntity<String>("Deleted "+id+" Successfully",HttpStatus.OK);

    } else { //not exist
        resp=new ResponseEntity<String>(""+id+" Not Exist",HttpStatus.BAD_REQUEST);
    }
    return resp;
}

```

**Execution:-**

- =>Right click on Project > Run As > Spring Boot App (Alt+Shift+X, B)
- =>Rest the application using postman

- 1.> <http://localhost:2019/product/save>
- 2.> <http://localhost:2019/product/delete/100>
- 3.> <http://localhost:2019/product/all>

## \*\*\*DOCKER\*\*\*

### Docker:--

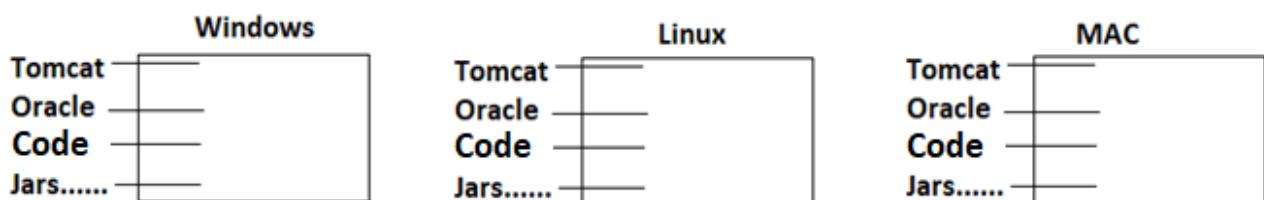
Docker is “CONTAINER SYSTEM” which includes all software’s a unit to run application, On any Platform (Windows, Linux, Mac...).

=>Docker supports running application on cloud servers also.

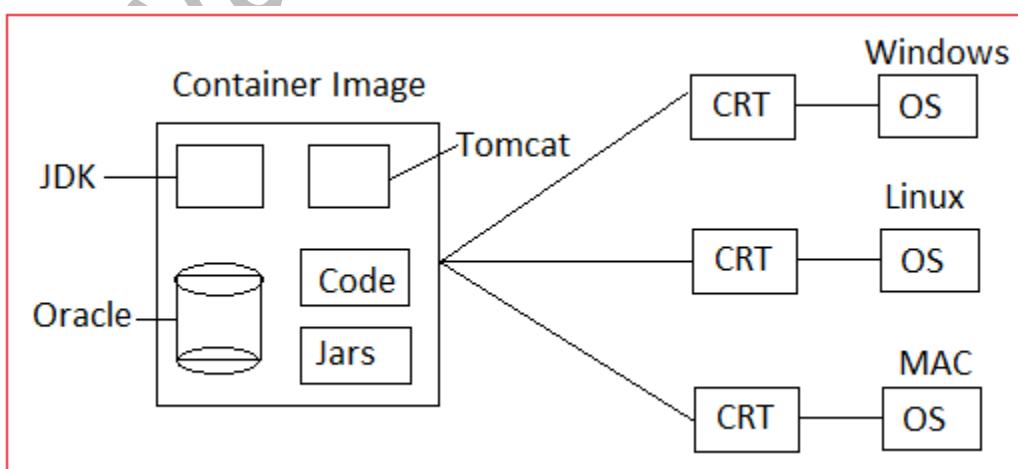
=>Docker supports follow of “CROSS-OS”. It means docker behaves a middleware between our runtime software and actual operating System.

=>Docker tool is used for Application Deployment (Running Application).

### Before Using Container System:--

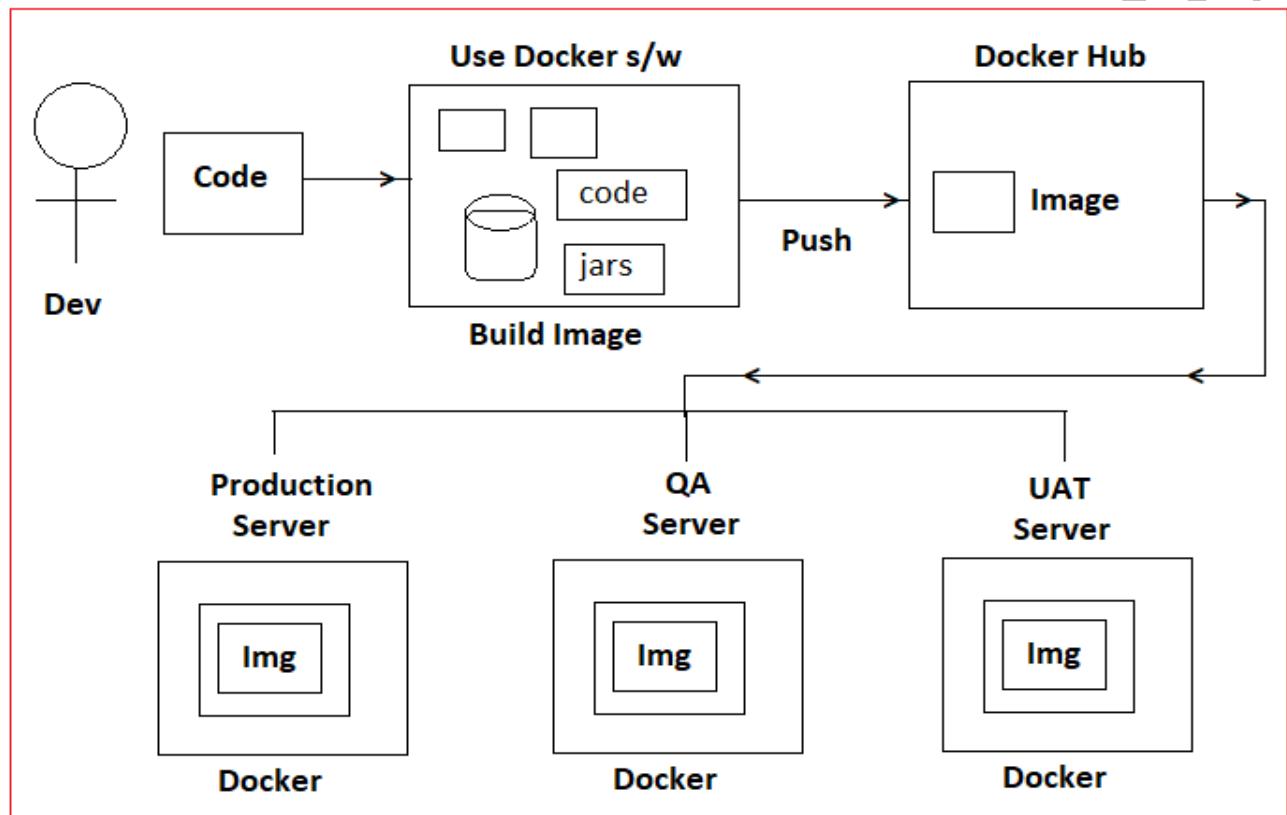


### After Container System (Docker):--



=>CRT = Container Run time.

## Docker Workflow:-



## Working with Docker:--

## #1:- DOWNLOAD DOCKER TOOLBOX.

- a. Download DockerToolbox-- <https://github.com/docker/toolbox/releases>

The screenshot shows a GitHub repository page for 'docker / toolbox'. At the top, there's a banner encouraging users to stay up-to-date on releases. Below it, the 'Releases' tab is selected, showing a list of releases. The first release, 'v19.03.1', is highlighted with a red box and labeled '#1 Click on'. The release notes mention 'guillaumerose released this on Jul 31 · 2 commits to master since this release'.

=>Goto down and Click on DockerToolbox-19.03.1.exe

Please ensure that your system has all of the latest updates before attempting the installation. In some cases, this will require a reboot. If you run into issues creating VMs, you may need to uninstall VirtualBox before re-installing the Docker Toolbox.

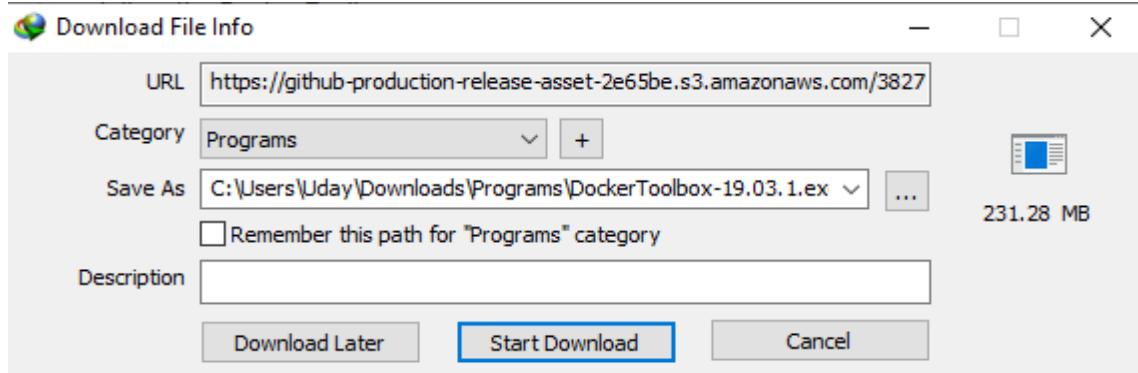
The following list of components is included with this Toolbox release. If you have a previously installed version of Toolbox, these installers will update the components to these versions.

#### Included Components

- docker 19.03.1
- docker-machine 0.16.1
- docker-compose 1.24.1
- Kitematic 0.17.7
- Boot2Docker ISO 19.03.1
- VirtualBox 5.2.20

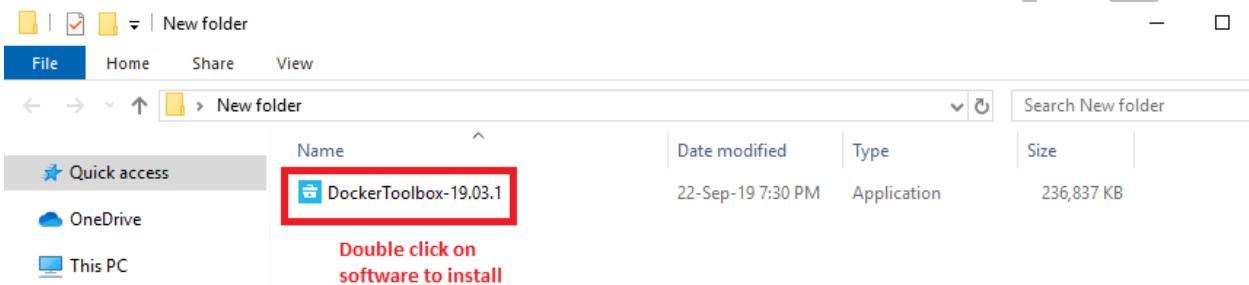
#### ▼ Assets 6

	#2 Download DockerToolbox-19-03-1.exe	
	DockerToolbox-19.03.1.exe	231 MB
	DockerToolbox-19.03.1.pkg	235 MB
	md5sum.txt	102 Bytes
	sha256sum.txt	168 Bytes
	Source code (zip)	
	Source code (tar.gz)	



## #2. INSTALL DOCKER TOOLBOX:-

=>Double click on software

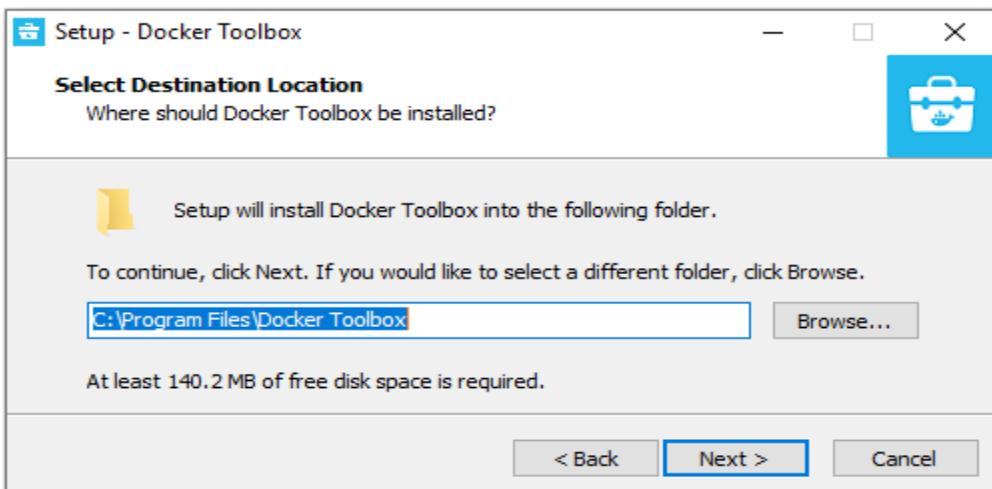


=>Click on Yes

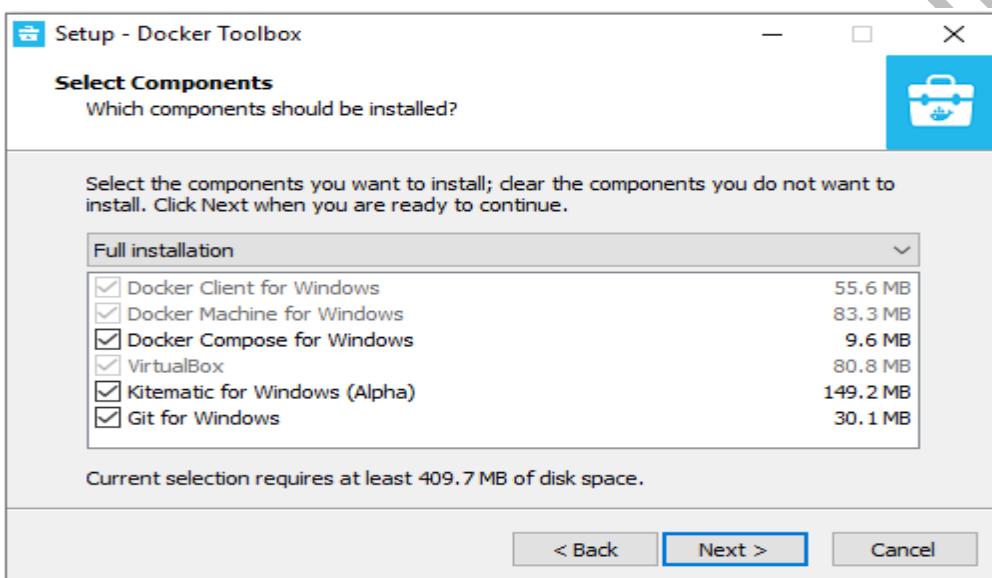
=>Click on Next



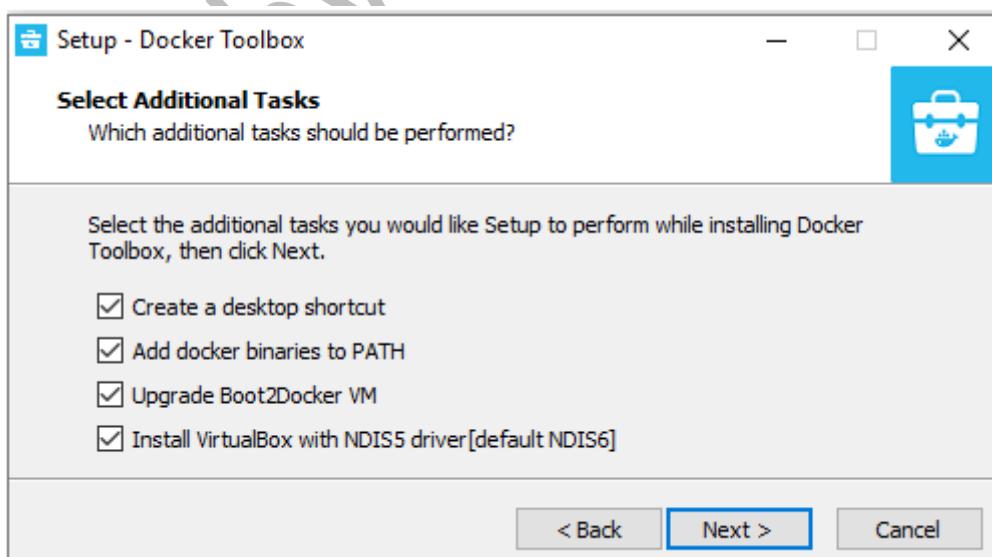
=>Click on Next



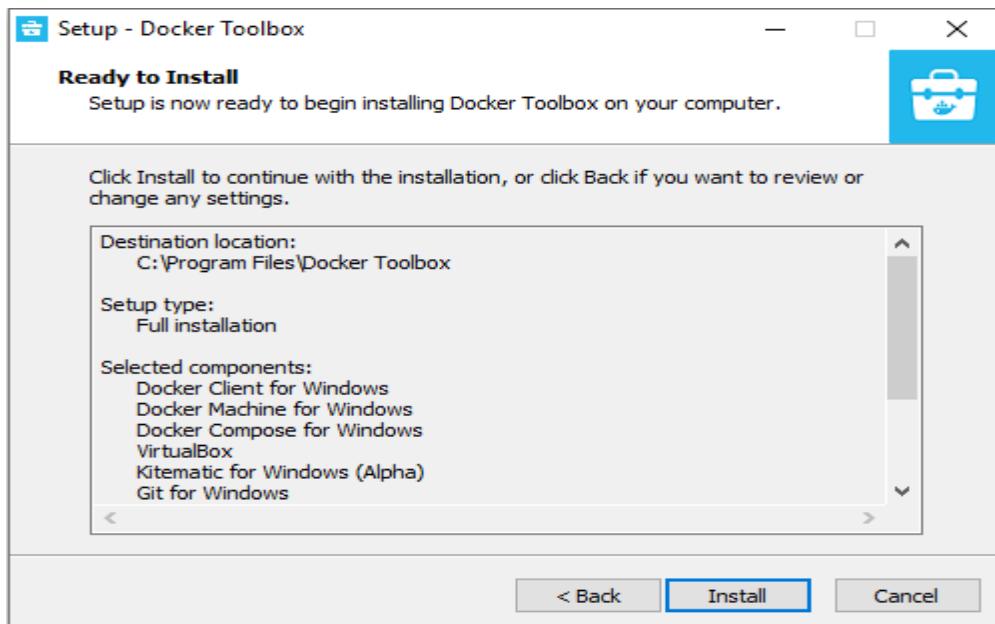
=>Click on Next



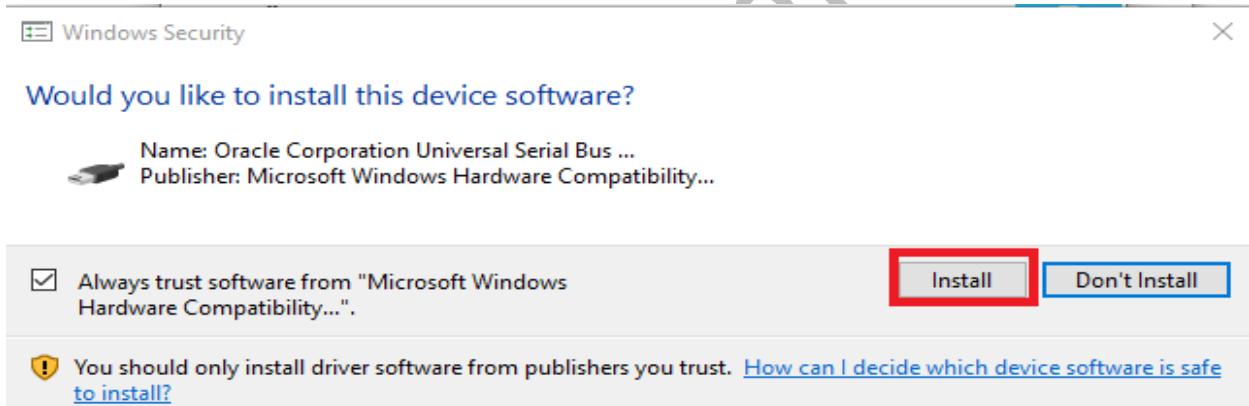
=>Click on Next



=>Click on Install



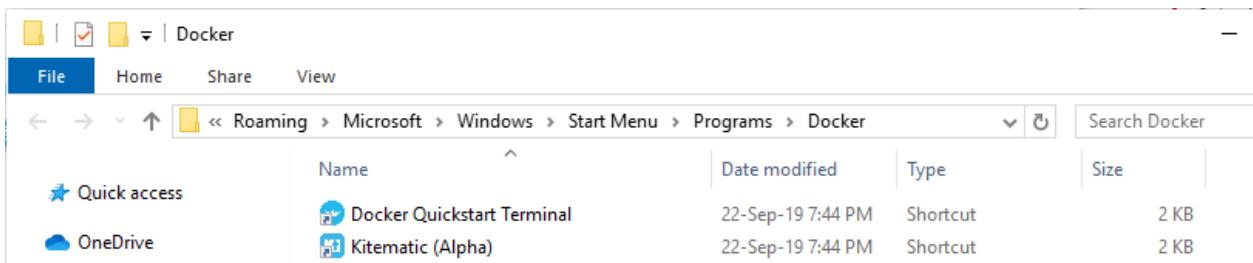
=>Click on Install



=>Click on Finish

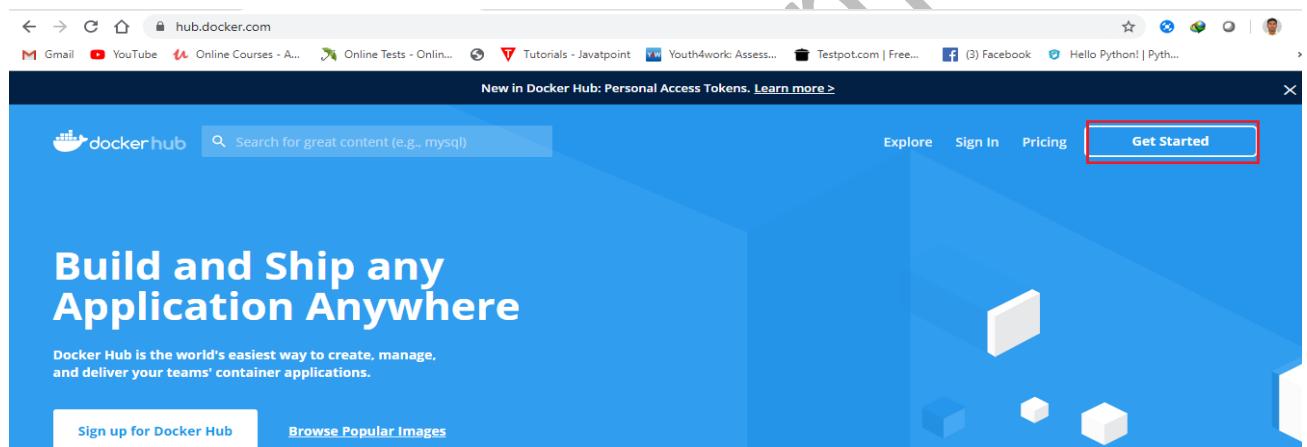


=>After installation two Icons are created



### #3:- CREATE ACCOUNT IN DOCKER HUB

- >Goto <https://hub.docker.com>
- >click on signup up for Docker Hub / Get Started option and register once
- >enter details
- >verify email
- >Login here



=>After click on Get Started/Sign up for Docker Hub.



## Docker Identification

In order to get you started, let us get you a Docker ID.  
Already have an account? [Sign In](#)

udaykumar0023

.....



udaykumar0023@gmail.com

- I agree to Docker's [Terms of Service](#).  
 I agree to Docker's [Privacy Policy](#) and [Data Processing Terms](#).

(Optional) I would like to receive email updates from Docker, including its various services and products.

I'm not a robot



[Continue](#)

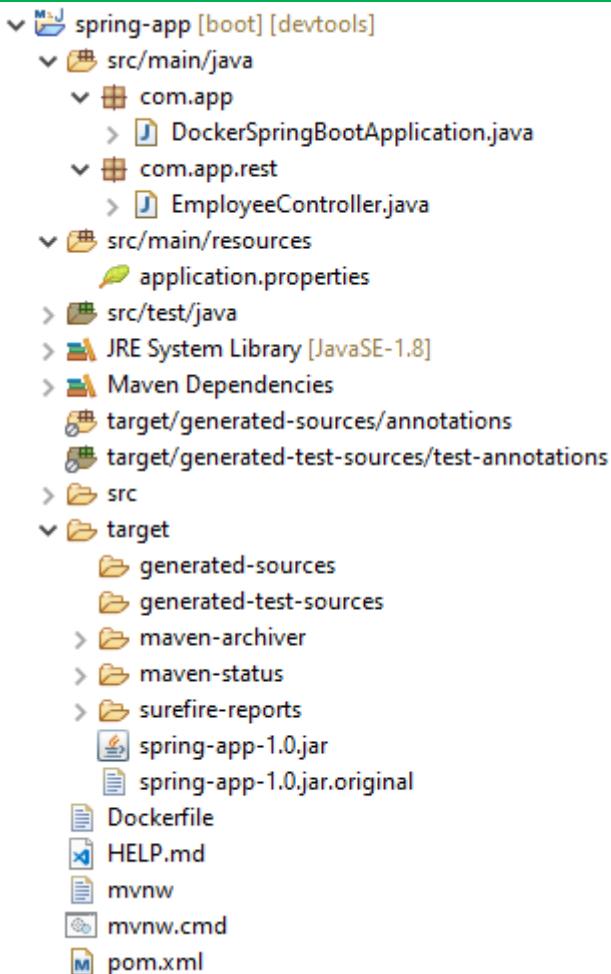
=>Complete the Registration process after Gmail verification.

=>Login to Docker hub.

## #4:- CREATE ONE SPRING BOOT APPLICATION:--

=>Spring Boot Application with RestController

## FOLDER STRUCTURE OF DOCKER-SPRING-BOOT APPLICATION:--



### 1. Starter class:--

```
package com.app;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class DockerSpringBootApplication {
    public static void main(String[] args) {
        SpringApplication.run(DockerSpringBootApplication.class, args);
        System.out.println("Hello Docker Application");
    }
}
```

### 2. Controller class:--

```
package com.app.rest;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
@RestController
```

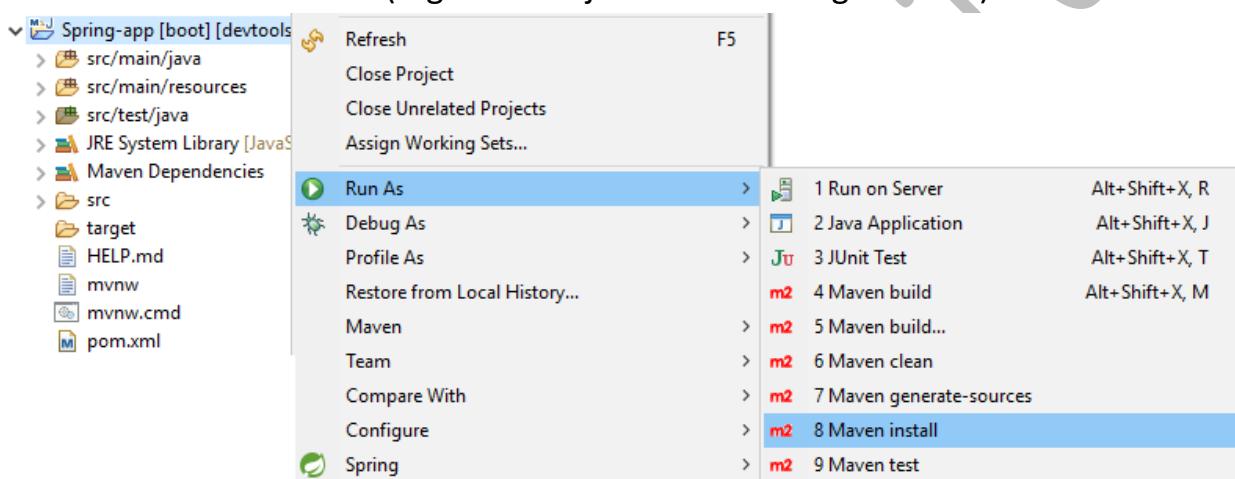
```
@RequestMapping("/employee")
public class EmployeeController {

    @GetMapping("/show")
    public String show() {
        return "Hello Docker";
    }
}
```

### 3:- Create Jar file

=>Right click on Project

=>Run As > Maven Install (It generates jar file under target folder)



### Q>How to rename a jar file?

=>Add a tag <finalName>...</finalName> in side build properties in pom.xml

<build>

```
<plugins>
    <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
</plugins>
    <finalName>docker-spring-boot</finalName>
```

</build>

=>Whatever name you are providing inside <finalName> tag by that file name a new jar file is created inside target folder. If we are not provided this tag by default jar name will be artifactId name which is mentioning pom.xml.

**5. DOCKERFILE:**-- Create/Add one Dockerfile under project

=>Right click on Project => New => file =>Enter file name as “Dockerfile” => finish.

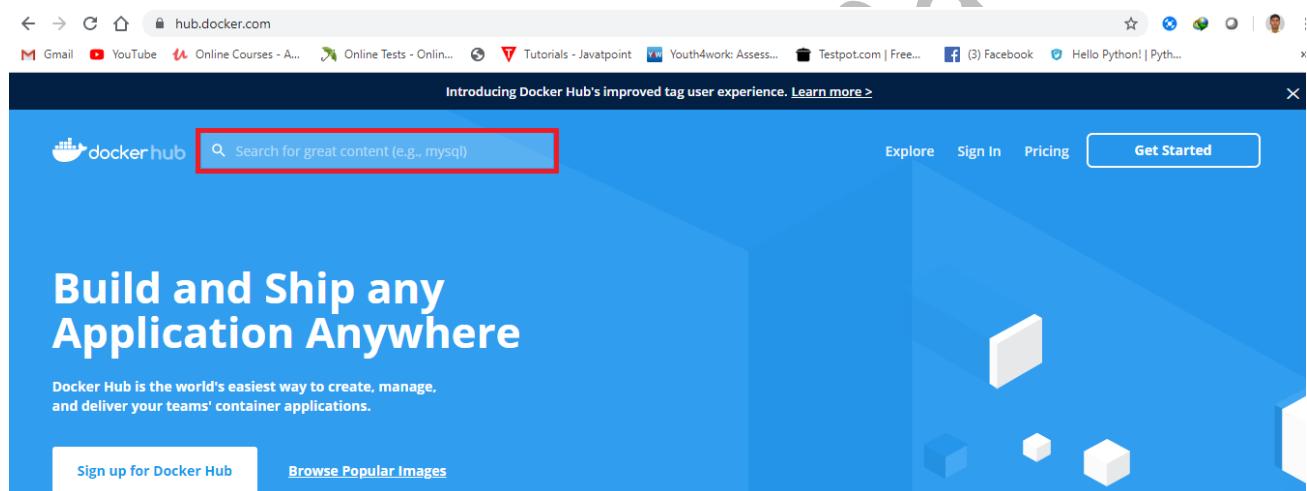
->Naming convention wise Dockerfile name starts with D capital.

=> Write bellow details inside Dockerfile.

```
FROM openjdk:8 //Java software image
ADD target/spring-app.jar spring-app.jar
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "spring-app"]
```

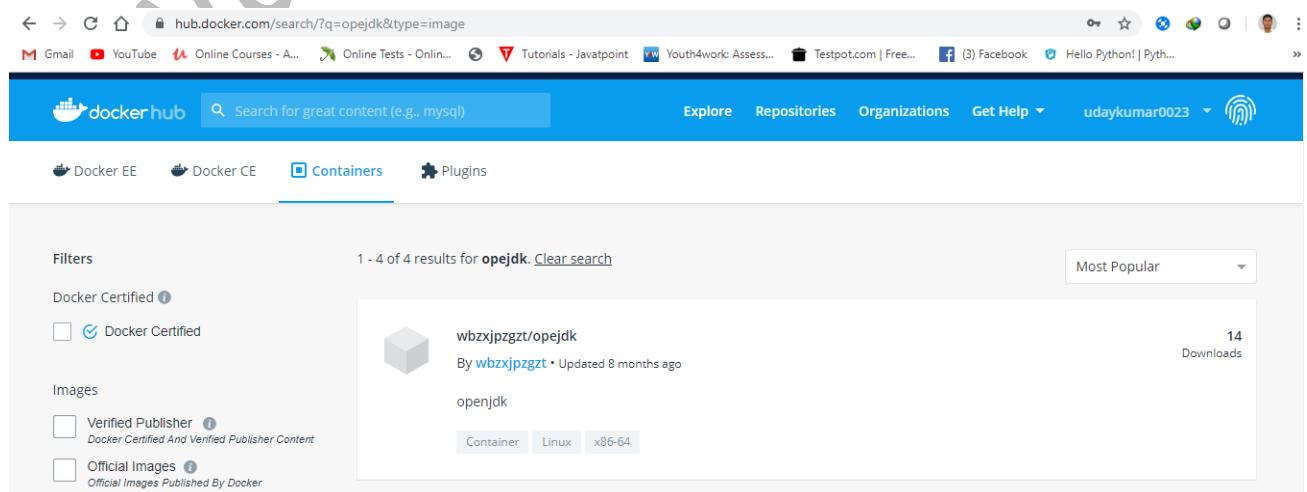
**Q>Where to get Software Images.**

=>Go to hub.docker.com website and Sign In



=>Search with openjdk, Tomcat, Oracle, MySQL ...etc according to your requirement.

->and click on searched result to see the details of software images.



## #6:- OPEN DOCKER TERMINAL AND EXECUTE COMMANDS:--

=> Double click on “DockerQuickStart” Installed Icon.

=>Wait for few minutes (First time takes time).

### a. Navigate to project folder:--

Ex:- If project is created in STS under

**E:\STS\_WSpace\MicroservicesPracticeExample\spring-app**

=>Then goto this specified location by using cd command step by step.

```
MINGW64:/e/STS_WSpace/MicroservicePracticeExample/spring-app
Amit@AMIT-PC MINGW64 /c/Program Files/Docker Toolbox
$ cd E:

Amit@AMIT-PC MINGW64 /e
$ cd STS_WSpace

Amit@AMIT-PC MINGW64 /e/STS_WSpace
$ cd MicroservicePracticeExample

Amit@AMIT-PC MINGW64 /e/STS_WSpace/MicroservicePracticeExample
$ cd spring-app
```

### b. build image:--

docker build -f Dockerfile -t <AnyImageName> .

=> \$ docker build -f Dockerfile -t spring-app .

=>Here dot(.) indicate current directory, must give one space.

```
Amit@AMIT-PC MINGW64 /e/STS_WSpace/MicroservicePracticeExample/spring-app
$ docker build -f Dockerfile -t spring-app .
Sending build context to Docker daemon 16.97MB
Step 1/4 : FROM openjdk:8
8: Pulling from library/openjdk
092586df9206: Pull complete
ef59477fae0: Pull complete
4530c6472b5d: Pull complete
d34d61487075: Pull complete
272f46008219: Pull complete
12ff6ccfe7a6: Pull complete
f26b9e1adb1: Pull complete
Digest: sha256:350761f9310c2b6665ac5e62b15d03dce859bc20dff59d6dbc63b114d9c39001
Status: Downloaded newer image for openjdk:8
--> e8d00769cb88
Step 2/4 : EXPOSE 8080
--> Running in b6ec3e4ec8f3
Removing intermediate container b6ec3e4ec8f3
--> e5593b863b71
Step 3/4 : ADD target/spring-app.jar spring-app.jar
--> 63b522914abd
Step 4/4 : ENTRYPOINT ["java", "-jar", "/spring-app.jar"]
--> Running in 40e5c18254d5
Removing intermediate container 40e5c18254d5
--> 643d8a059df7
Successfully built 643d8a059df7
Successfully tagged spring-app:latest
SECURITY WARNING: You are building a Docker image from Windows against a non-Windows Docker host. All files and directories added to build context will have ' permissions. It is recommended to double check and reset permissions for sensitive files and directories.

Amit@AMIT-PC MINGW64 /e/STS_WSpace/MicroservicePracticeExample/spring-app
$
```

### c>check all images:--

\$ docker images

\$ docker image ls

**d. run image:--**

```
docker run -p 9090:8080 spring-app
```

```
mit@AMIT-PC MINGW64 /e/STS_WSpace/MicroservicePracticeExample/spring-app
$ docker run -p 9090:8080 spring-app

      .___._____._____._____._____._____
     / \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \
    (( ) ( ) ( ) ( ) ( ) ( ) ( ) ( )
   \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \
  ===== [=====] /=====/
 :: Spring Boot ::      (v2.1.8.RELEASE)

2019-10-05 08:05:06.141  INFO 1 --- [           main] com.app.SpringAppApplication      : Starting SpringAppApplication v1.0 on 4e92b75710e4 with PID 1 (/spring-app.jar started by root in /)
2019-10-05 08:05:06.199  INFO 1 --- [           main] com.app.SpringAppApplication      : No active profile set, falling back to default profiles: default
2019-10-05 08:05:22.336  INFO 1 --- [           main] o.s.w.e.m.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8075 (http)
2019-10-05 08:05:22.808  INFO 1 --- [           main] o.apache.catalina.core.StandardService: Starting service [Tomcat]
2019-10-05 08:05:22.816  INFO 1 --- [           main] org.apache.catalina.core.StandardEngine: Starting Servlet engine: [Apache Tomcat/9.0.24]
2019-10-05 08:05:24.141  INFO 1 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[]: Initializing Spring embedded WebApplicationContext
2019-10-05 08:05:24.158  INFO 1 --- [           main] o.s.web.context.ContextLoader      : Root WebApplicationContext: initialization completed in 16976 ms
2019-10-05 08:05:27.220  INFO 1 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor: Initializing ExecutorService 'applicationTaskExecutor'
2019-10-05 08:05:29.978  INFO 1 --- [           main] o.s.w.e.m.tomcat.TomcatWebServer  : Tomcat started on port(s): 8075 (http) with context path ''
2019-10-05 08:05:30.016  INFO 1 --- [           main] com.app.SpringAppApplication      : Started SpringAppApplication in 30.411 seconds (JVM running for 35.081)
Spring Boot Application.....!!
```

=>Here 9090 is docker container port no and 8080 is application port no.

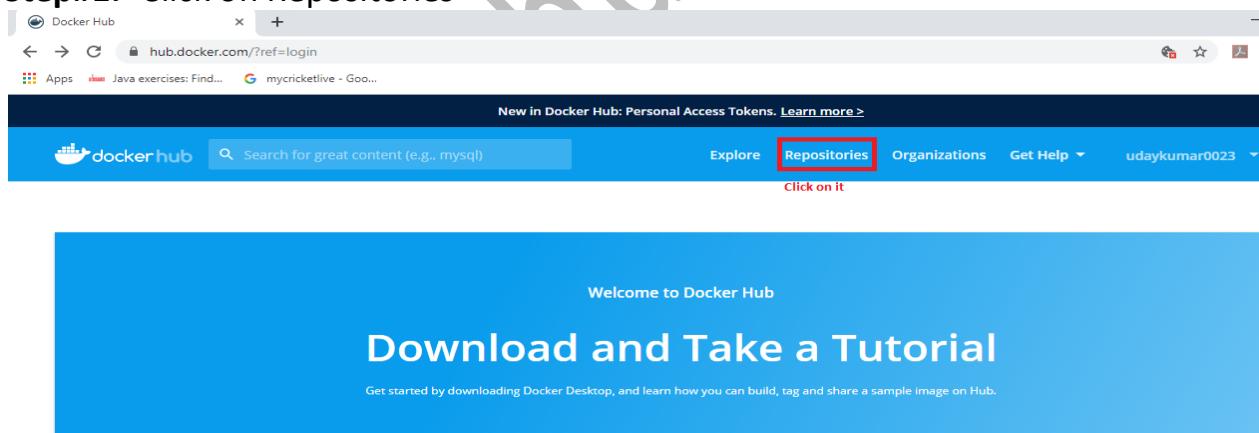
=> Goto browser and enter URL:

Example URL: <http://192.168.99.100:9090/show>

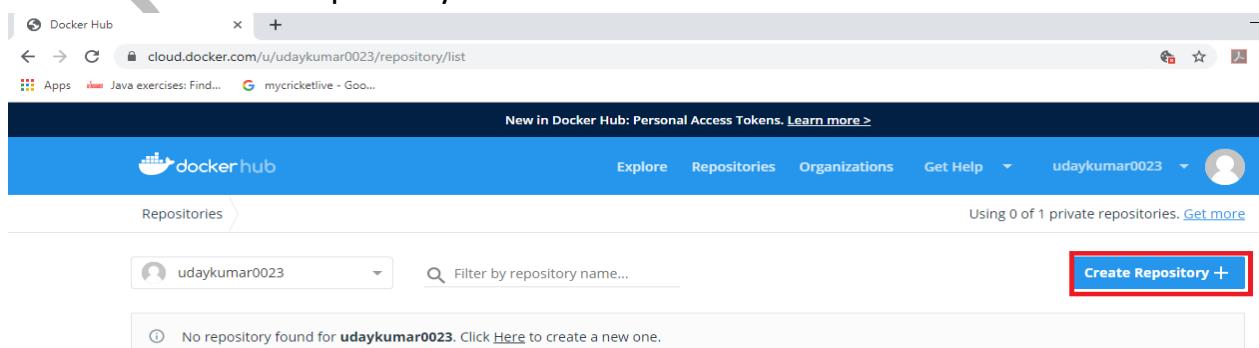
**#7. CREATE REPOSITORY IN DOCKER HUB:--**

=>Login Docker hub and create one repository [Click on button “Create repository +”]

Ex:--myrepo > Create

**Step#1:- Click on Repositories**

=>Click on “Create Repository +”.



=>Enter the repository name and choose any one visibility like (Public/Private).

Docker Hub Create Repository Page:

- #1 Repository name:** myrepo
- #2 Visibility:** Private (radio button selected)
- #3 Choose any one Public/Private:** Public (radio button unselected)
- #4 Create Button:** The 'Create' button is highlighted.

=>Final Screen of Repository.

Docker Hub Repository Details Page (myrepo):

- #1 Repository is created:** myrepo
- #2 Command to push the Image to docker repository:** docker push udaykumar0023/myrepo:tagname

**#8. LOGIN TO DOCKER HUB:--**

```
docker login
```

UserName : docker account username (udaykumar0023)  
 password: docker password (Uday123)

```
Amit@AMIT-PC MINGW64 /e/STS_WSpace/MicroservicePracticeExample/spring-app
$ docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one.
Username: udaykumar0023
Password:
WARNING! Your password will be stored unencrypted in C:\Users\Amit\.docker\config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

**#9:- CREATE NEW TAG:--**

=>Come to docker terminal and create one tag between docker and hub-repository  
 =>Format to create tag name is

```
=>docker tag local-image:tagname reponame:tagname
=>docker tag <imageName> <username>/<repoName>:<tagname>
```

```
docker tag spring-app udaykumar0023/myrepo:latest
```

```
Amit@AMIT-PC MINGW64 /e/STS_WSpace/MicroservicePracticeExample/spring-app
$ docker tag spring-app udaykumar0023/myrepo:latest

Amit@AMIT-PC MINGW64 /e/STS_WSpace/MicroservicePracticeExample/spring-app
$ docker tag spring-app udaykumar0023/myrepo:latest8

Amit@AMIT-PC MINGW64 /e/STS_WSpace/MicroservicePracticeExample/spring-app
$ docker images
REPOSITORY          TAG        IMAGE ID      CREATED       SIZE
8421087771/myrepo  latest     643d8a059df7  27 hours ago  505MB
myrepo              latest     643d8a059df7  27 hours ago  505MB
myrepo              latest5    643d8a059df7  27 hours ago  505MB
spring-app          latest     643d8a059df7  27 hours ago  505MB
udaykumar0023/myrepo latest    643d8a059df7  27 hours ago  505MB
udaykumar0023/myrepo latest8   643d8a059df7  27 hours ago  505MB
openjdk              8         e8d00769c8a8   3 weeks ago   488MB
```

**#10:- PUSH IMAGE INTO DOCKER HUB:--**

Syntax:- docker push <username>/<repoName>:tagname

```
=>docker push udaykumar0023/myrepo:latest
```

```
Amit@AMIT-PC MINGW64 /e/STS_WSpace/MicroservicePracticeExample/spring-app
$ docker push udaykumar0023/myrepo:latest
The push refers to repository [docker.io/udaykumar0023/myrepo]
eccdd29c4296: Pushed
57e6a3d20ce9: Mounted from library/openjdk
1690af51cb08: Mounted from library/openjdk
5a30999619d7: Mounted from library/openjdk
2e669e0134f5: Mounted from library/openjdk
8bacec4e3446: Mounted from library/openjdk
26b1991f37bd: Mounted from library/openjdk
55e6b89812f3: Mounted from library/openjdk
latest: digest: sha256:3a9c8311bb4da94f0cf8c099d3c353c9bfc6a37ebb16a398375ae5402c9c47d9 size: 2007
```

=>Goto docker hub and refresh to see the latest update:--

The screenshot shows the Docker Hub interface for the repository 'udaykumar0023 / myrepo'. The 'General' tab is active. The repository is described as a Spring Boot Application. The last push was 2 minutes ago. A 'Docker commands' section contains a button labeled 'docker push udaykumar0023/myrepo:tagname'.

The screenshot shows the Docker Hub interface for the repository 'udaykumar0023 / myrepo' with the 'Tags' tab selected. It lists two tags: 'new' and 'latest'. Both tags were last updated 3 and 4 minutes ago respectively.

## #13:- LINK DOCKER HUB WITH GITHUB/BIGBUKET ETC...

The screenshot shows the Docker Hub interface for the repository 'udaykumar0023/myrepo'. The 'Builds' tab is selected. The 'Build Activity' section shows three recent builds with times: 1.05 min, 0.7 min, and 0.35 min. The 'Automated Builds' section shows an active Autobuild for the 'latest' tag from the 'master' source, which is currently 'BUILDING'. The 'Recent Builds' section lists three builds: one successful build in 'master' (8419bf95) and two failed builds in 'master' (8419bf95), along with a 'Github Ping' entry.

## #14:- PULL THE IMAGE FROM DOCKER HUB:--

Syntax:-- docker pull <username>/myrepo  
 docker pull udaykumar0023/myrepo

```
Amit@AMIT-PC MINGW64 /e/STS_WSpace/MicroservicePracticeExample/spring-app
$ docker pull udaykumar0023/myrepo
Using default tag: latest
latest: Pulling from udaykumar0023/myrepo
Digest: sha256:3a9c8311bb4da94f0cf8c099d3c353c9bfc6a37ebb16a398375ae5402c9c47d9
Status: Image is up to date for udaykumar0023/myrepo:latest
docker.io/udaykumar0023/myrepo:latest
```

**NOTE:-** Press **ctrl+c** to shutdown the docker container.

**MICROSERVICES END**

**email : [javabyragh@gmail.com](mailto:javabyragh@gmail.com)**

**FB Group : <https://www.facebook.com/groups/thejavatemple/>**