***Spring boot (2.x)***

🡺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.

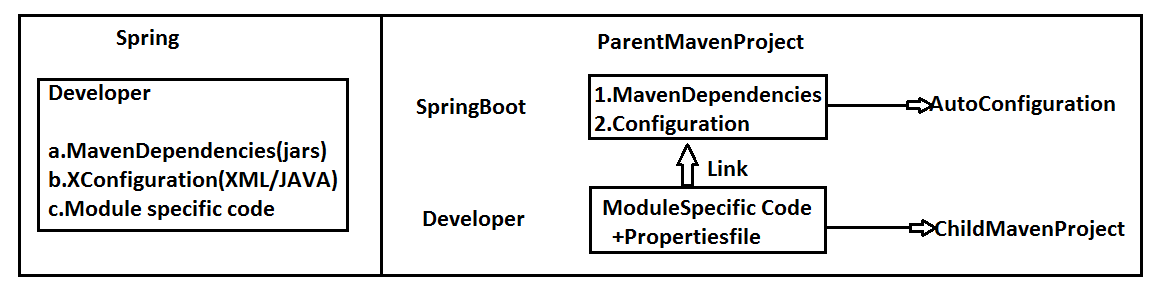
b. Spring Boot 2.x.

🡺Spring Boot provides Auto Configuration 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

1. Properties File (application.properties).
2. YAMAL File (application.yml).



**Example: --**

**Spring JDBC and Spring Boot JDBC [Sample code Comparison] :-**

1. **Spring JDBC: -**
2. **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>

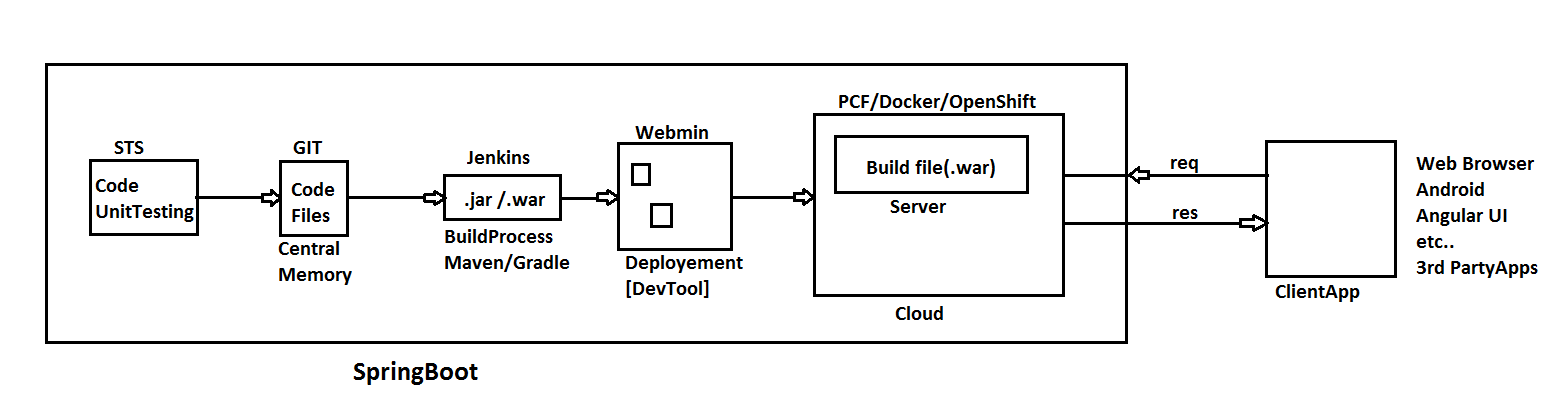
</dependency>

🡺Spring Boot supports end to end process that is called

🡺Coding 🡺Unit testing 🡺Version control 🡺Build

🡺 Deployment 🡺Client Integration.

* GIT (github.com) is used to store our code files. It is called as Central Repository or version Control Tool.
* Java is converted to .class (Compile) .class + (other files .xml, .html…) converted to .jar/.war finally (build process).
* Place .jar/.war in server and start server is called as Deployment.
* Spring Boot Application is a service provider app which can be integrated with any UI client like Android, Angular UI, RFID (Swiping Machine), Any 3rd party Apps, Web Apps using Rest and JMS.



**NOTE:-**

* Spring Boot supports two build tools **Maven** and **Gradel**.
* 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: -**

* WebMVC and WebServices (Rest).
* JDBC and ORM (Hibernate with JPA).
* Email, Scheduling, JMS, Security.
* Cache and Connection Pooling.
* DevTools, Swagger UI, Actuator and Profiles.
* UI Design using HTML, JSP, Thymeleaf…etc.

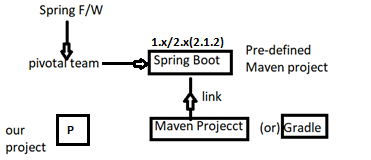
**e. Supports Input Data (Key = val) Using (for AutoConfiguration code):-**

* Properties file or YAML files

**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.

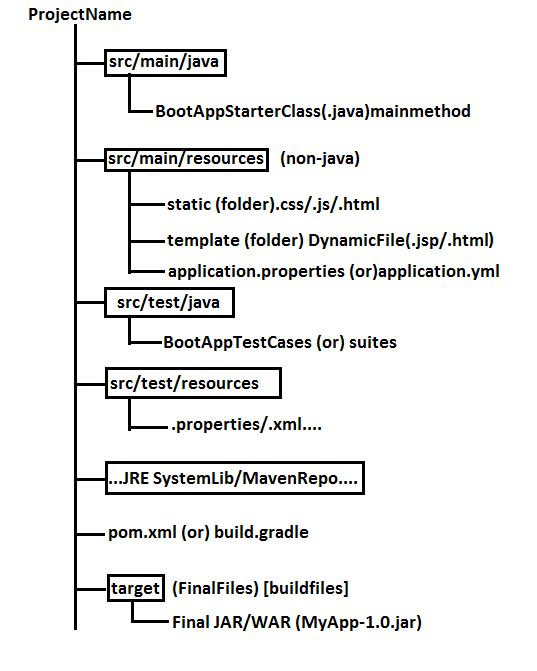
**3. pom.xml (or) build.gradle:-** This file holds all information about

a. Parent boot project version

b. App properties (JDK version/maven/cloud versions….)

c. Dependencies (JARS Details)

d. Plugins (Compiler/WAR…etc.)



**Chapter#1 Spring Boot Core**

**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.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: -**

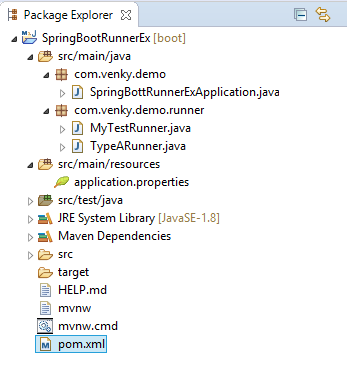
🡺Right click on “src/main/java”🡺new 🡺 class 🡺 Enter details, like:

PackageName: com.app

Name: MyAppStarter 🡺 Finish

#5 Create one or more Runner classes under src/main/java folder with package “com.app”.

**Folder Structure of CommandLineRunner & ApplicationRunner with Ordered interface implementations:-**



**Code:-**

**SpringBootRunner.java (Spring Boot Starter class)**

**package** com.venky.demo;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

**public** **class** SpringBottRunnerExApplication {

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

SpringApplication.*run*(SpringBottRunnerExApplication.**class**, args);

System.***out***.println("Hello Venkat");

}

}

**#Runner #1: MyTestRunner1.java**

**package** com.venky.demo.runner;

**import** org.springframework.boot.CommandLineRunner;

**import** org.springframework.core.Ordered;

**import** org.springframework.stereotype.Component;

@Component

/\*CommandLineRunner with Ordered implementations Manual Approach\*/

**public** **class** MyTestRunner **implements** CommandLineRunner,Ordered{

@Override

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

System.***out***.println("-----From MyTestRunner----");

}

@Override

**public** **int** getOrder() {

**return** 55;

}

}

**#Runner #2: TypeARunner.java**

**package** com.venky.demo.runner;

**import** org.springframework.boot.CommandLineRunner;

**import** org.springframework.core.annotation.Order;

**import** org.springframework.stereotype.Component;

@Component

@Order(53)

**public** **class** TypeARunner **implements** CommandLineRunner {

@Override

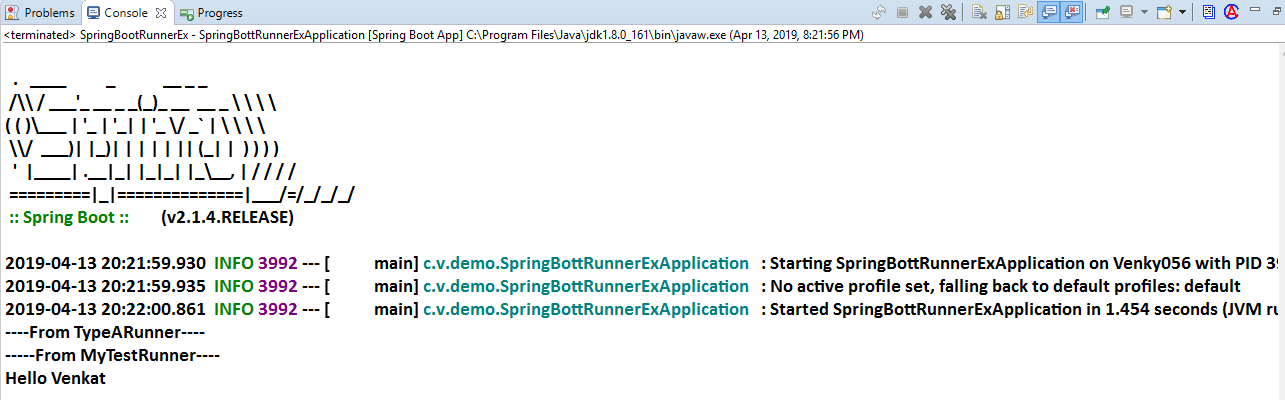
**public** **void** run(String... args) **throws** Exception {

System.***out***.println("----From TypeARunner----");

}

}

**Output :-**



**NOTE :-**

1. Boot Application can have multiple runners

Ex:- Email Runner, JmsRunner, SecurityRunner, CloudEnvRunner, DevOpsRunner, DatabaseRunner etc…

2. Boot provides default execution order.

🡺To specify programmer defined order use

**Interface: Ordered (or) Annotation: @Order**

🡺If we are configures both Runners but not implements Ordered then by default Annotation based Configuration will be executed first.

**Input Data to Runners using (CommandLineArguments)**

Programmer can pass onetime 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…

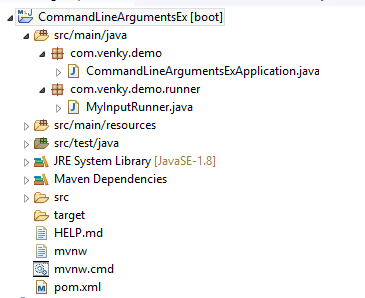
**Syntax :** data [NonOption Argument]

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.

**Folder Structure of Reading Input Data Using CommandLine Arguments**

****

**Code:-**

**1. CommandLineArgumentsExApplication.java:-**

**package** com.venky.demo;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

**public** **class** CommandLineArgumentsExApplication {

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

SpringApplication.*run*(CommandLineArgumentsExApplication.**class**, args);

}

}

**2. MyInputRunner.java :-**

**package** com.venky.demo.runner;

**import** java.util.Arrays;

**import** org.springframework.boot.CommandLineRunner;

**import** org.springframework.stereotype.Component;

@Component

**public** **class** MyInputRunner **implements** CommandLineRunner {

@Override

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

System.***out***.println("=-=-=-=-Hello CommandLineRunner=-=-=-=");

System.***out***.println(args[0]);

System.***out***.println(args[1]);

System.***out***.println(args[2]);

System.***out***.println(args[3]);

System.***out***.println(Arrays.*asList*(args));

System.***out***.println("=-=-=-=-End of CommandLineRunner-=-=-=");

}

}

**Execution:-**

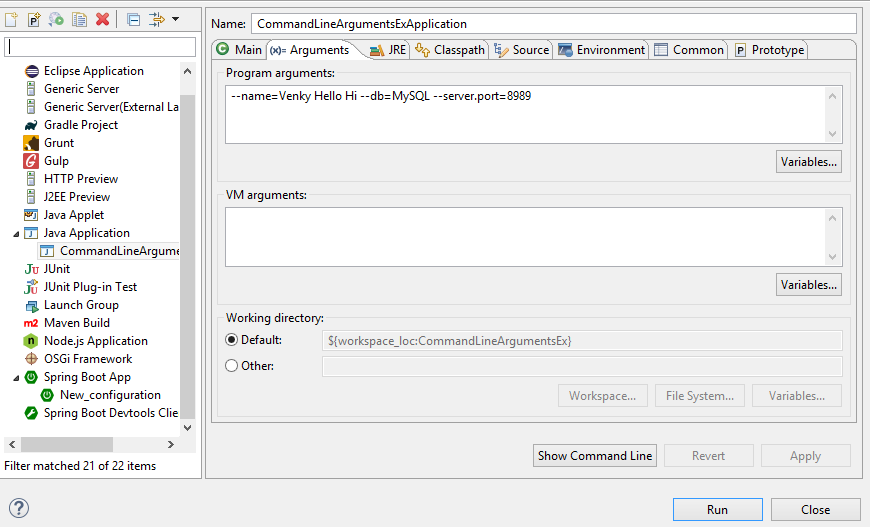
🡺Right Click on starter class code (main)

🡺Run As 🡺Run Configuration

🡺Choose “Arguments” tab

🡺Enter data in Program arguments (with space)

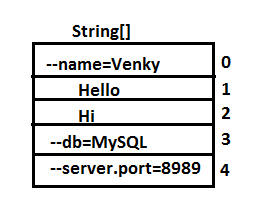
**--name=Venky Hello Hi --db=MySQL --server.port=8989**



🡺Click on Apply and Run

🡺It is internally converted to

String [] (String… args)



**Anonymouse 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 ();

}

**-----Anonymouse Inner class----**

new Sample() {

public void show() {

System.out.println(“Hello”);

}

}

**Example#2 :-**

interface CommandLineRunner {

void run(String… args) throws Exception;

}

**Anonymouse Inner class :-**

new CommandLineRunner() {

public void run (String… args) throws Excception {

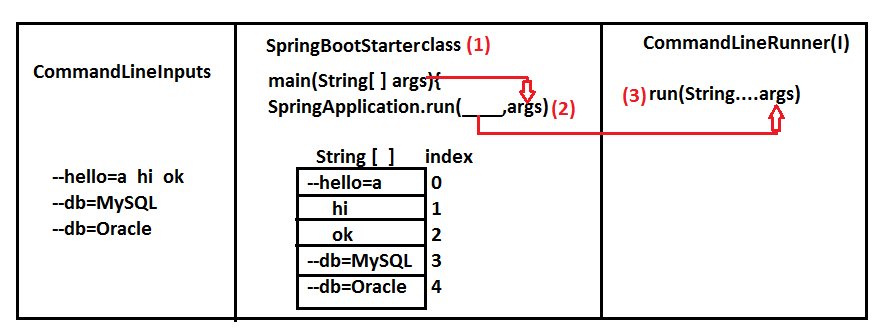
System.out.println(“HI”);

}

}

**Q)How CommandLineRunner works?**

Ans:- 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.

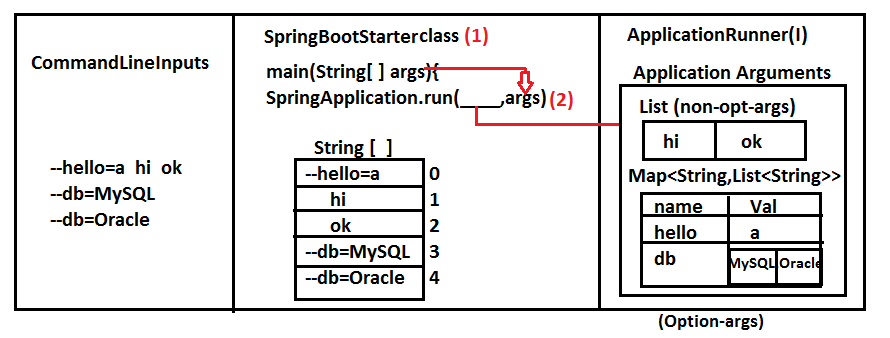


**#2.ApplicationRunner(I):-** It is new type runner added in Spring boot 1.3 which makes easy to access arguments.

🡺This will separate Option Arguments (as Map<String, List<String>>) and

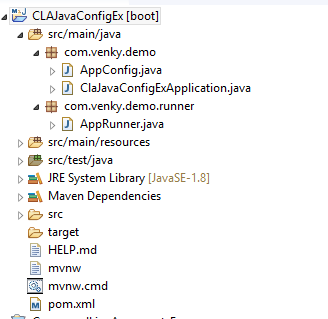
Non-Option Arguments (<List<String>)

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



**EXAMPLE:-**

**FOLDER STRUCTURE:**

****

**1. ClaJavaConfigExApplication.java:-**

**package** com.venky.demo;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

**public** **class** ClaJavaConfigExApplication {

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

SpringApplication.*run*(ClaJavaConfigExApplication.**class**, args);

}

}

**2.AppCOnfig.java:-**

**package** com.venky.demo;

**import** java.util.Arrays;

**import** org.springframework.boot.CommandLineRunner;

**import** org.springframework.context.annotation.Bean;

**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));

};

} }

**3.AppRunner.java:-**

**package** com.venky.demo.runner;

**import** java.util.Arrays;

**import** org.springframework.boot.ApplicationArguments;

**import** org.springframework.boot.ApplicationRunner;

**import** org.springframework.stereotype.Component;

@Component

**public** **class** AppRunner **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"));

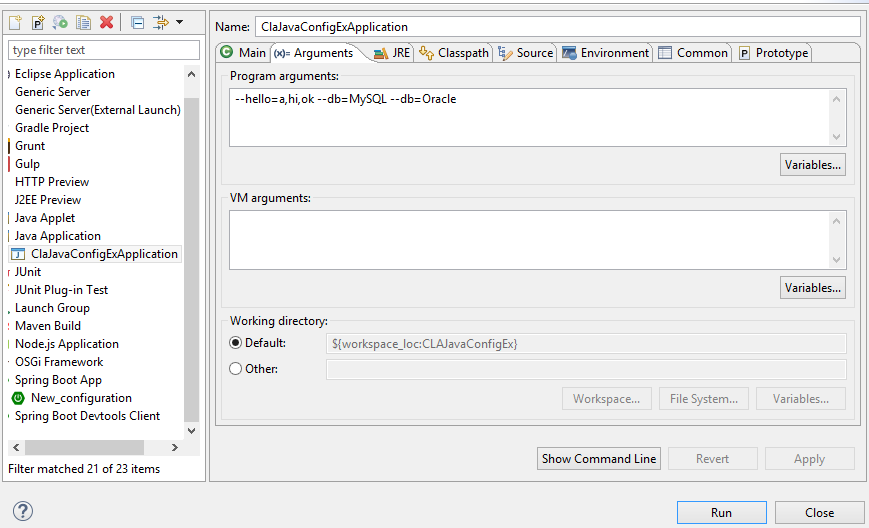
System.***out***.println(args.containsOption("bye"));

System.***out***.println("End of Application Runner");

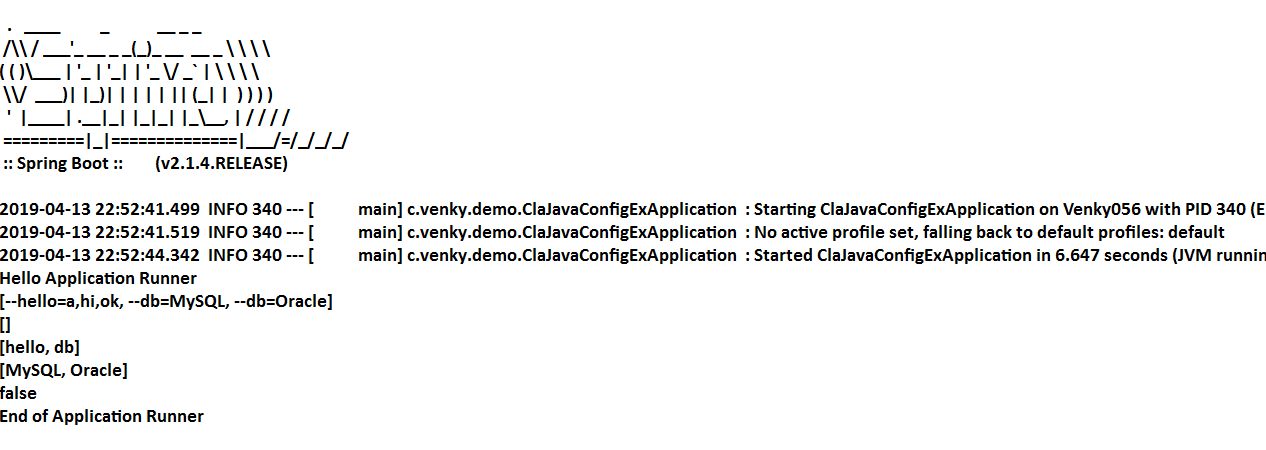
}

}

**🡺**Run as 🡺Run Configurations🡺Arguments

****

**Output:-**

****

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

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

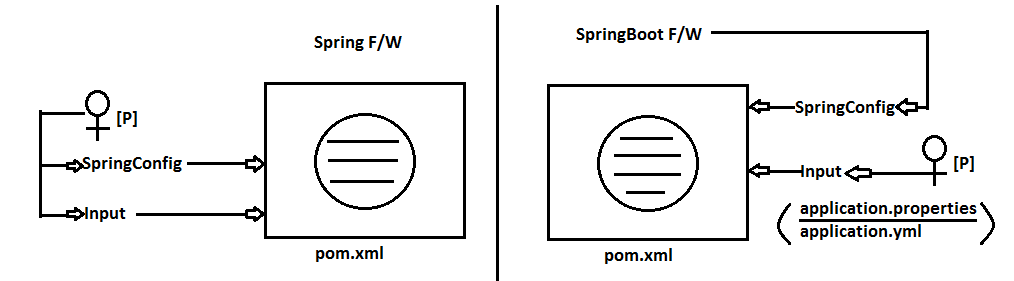
**Spring Boot Input Data (using application.properties)**

🡺 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)

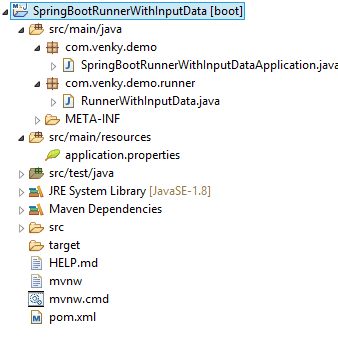
Reference Link:

[https://docs.spring.io/spring-boot/docs/ current/ reference /html](https://docs.spring.io/spring-boot/docs/ current/ reference /html      /common-application-properties.html)

[/common-application-properties.html](https://docs.spring.io/spring-boot/docs/ current/ reference /html      /common-application-properties.html)

B.Programmer defined

**Folder Structure of Spring Boot Input Data using application.aproperties :**

****

**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=999

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**

**package** com.venky.demo;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

**public** **class** SpringBootRunnerWithInputDataApplication {

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

SpringApplication.*run*(SpringBootRunnerWithInputDataApplication.**class**, args);

}

}

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

**package** com.venky.demo.runner;

**import** org.springframework.beans.factory.annotation.Value;

**import** org.springframework.boot.CommandLineRunner;

**import** org.springframework.stereotype.Component;

@Component

**public** **class** RunnerWithInputData **implements** CommandLineRunner {

@Value("${my.info.product.id}")

**private** **int** prodId;

@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

**public** RunnerWithInputData() {

**super**();

}

**public** RunnerWithInputData(**int** prodId, String prodCode, **double** modelver, **boolean** isDetEnable, **char** startKey) {

**super**();

**this**.prodId = prodId;

**this**.prodCode = prodCode;

**this**.modelver = modelver;

**this**.isDetEnable = isDetEnable;

**this**.startKey = startKey;

}

//Setters and Getters method

**public** **int** getProdId() {

**return** prodId;

}

**public** **void** setProdId(**int** prodId) {

**this**.prodId = prodId;

}

**public** String getProdCode() {

**return** prodCode;

}

**public** **void** setProdCode(String prodCode) {

**this**.prodCode = prodCode;

}

**public** **double** getModelver() {

**return** modelver;

}

**public** **void** setModelver(**double** modelver) {

**this**.modelver = modelver;

}

**public** **boolean** isDetEnable() {

**return** isDetEnable;

}

**public** **void** setDetEnable(**boolean** isDetEnable) {

**this**.isDetEnable = isDetEnable;

}

**public** **char** getStartKey() {

**return** startKey;

}

**public** **void** setStartKey(**char** startKey) {

**this**.startKey = startKey;

}

//toString method

@Override

**public** String toString() {

**return** "SpringBootRunnerWithInputData [prodId=" + prodId + ", prodCode=" + prodCode + ", modelver=" + modelver+ ", isDetEnable=" + isDetEnable + ", startKey=" + startKey + "]";

}

@Override

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

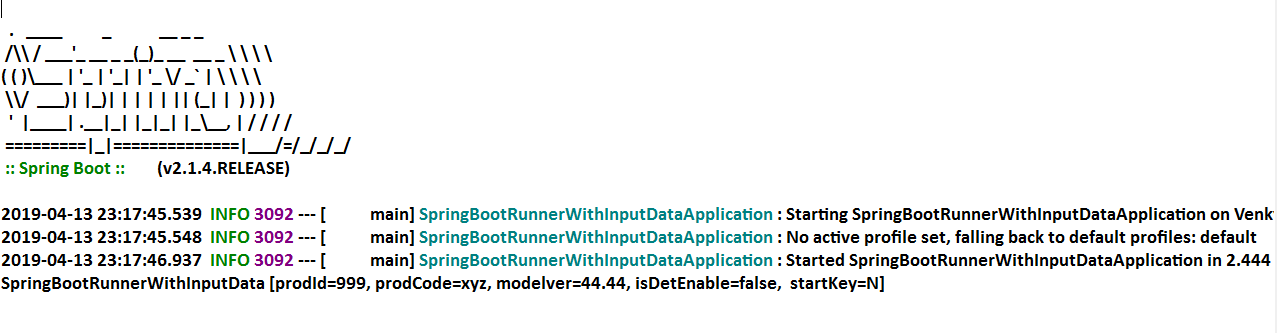
System.***out***.println(**this**);

//System.out.println(this.toString());

}

}

**5# Output :-**

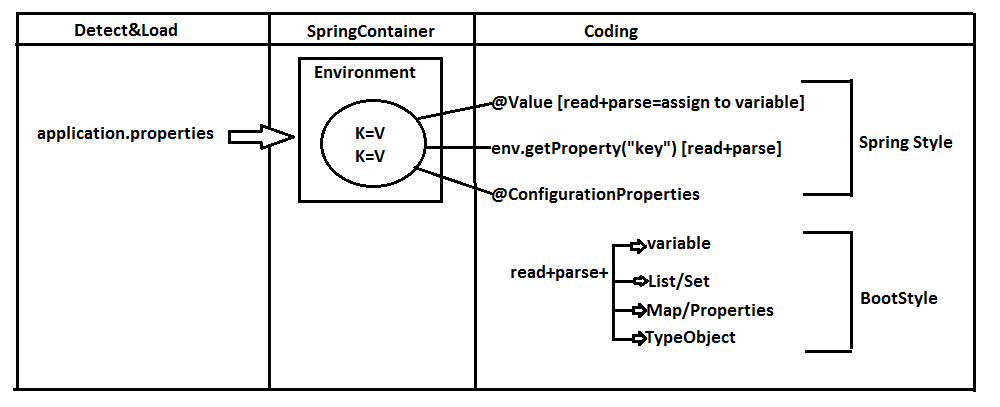


**NOTE:-** If key data is mismatched with variable data type, then Spring Container throws **Exception :** TypeMistchException : Failed to convert value.

**Internal flow :-**

Spring Boot will search for file “application.properties” in project (4 different locations)

* 1.Once found (detected) then load into Container and store as “Environment” obj.
* 2.We can read data in legacy style @Value or 2>env.getProeprty(..).
* 3.Boot Style (Bulk Loading) can be done using Annotation. \*\*\*\*
* 4.@ConfigurationProperties

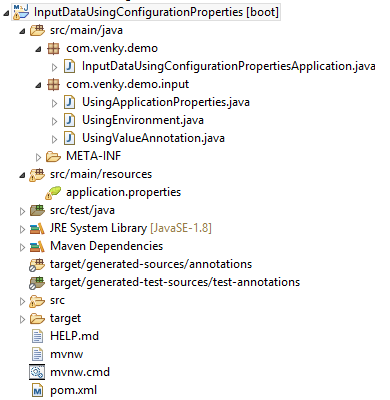


**@ConfigurationProperties**

* 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.

1. 1key = 1 variable
2. Multiple keys with index = List/Set/Array
3. Multiple keys with key-value format = Map or Properties
4. Multiple keys with common type = Class Object (Has-A)

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



**Example :-**

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

**package** com.venky.demo;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

**public** **class** InputDataUsingConfigurationPropertiesApplication {

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

SpringApplication.*run*(InputDataUsingConfigurationPropertiesApplication.**class**, args);

}

}

**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<DT>/Set<DT>/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.venky.demo.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.venky.demo.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

1. Hexa Decimal Value
2. int or long type
3. int or long with range
4. 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.venky.demo.input;

**import** org.springframework.beans.factory.annotation.Value;

**import** org.springframework.boot.CommandLineRunner;

**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;

**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 + "]"; }

@Override

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

System.***out***.println(**this**);

} }

**Output :-**

****

**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)

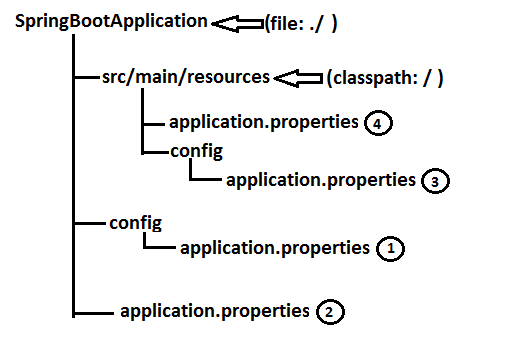
**3. Under Project (Under resources/config) :-**

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

(classpath:/config/application.properties)

**4.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 Properrties (YAML) file name.

🡺Which can be placed in any of $ 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

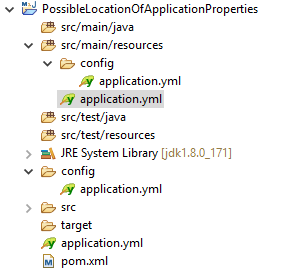
Ex:- --spring.config.name=mydata (user defined)

**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 :-**



**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 :-**

-=-=- “application.properties” -=-=-

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:

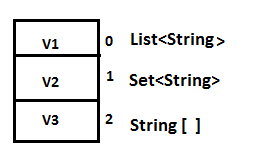
version:

-V1

-V2

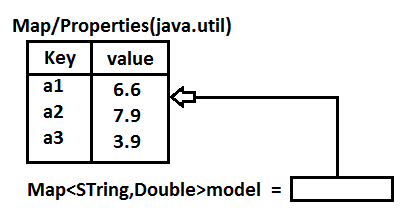
-V3

🡺final meaning is:



**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:

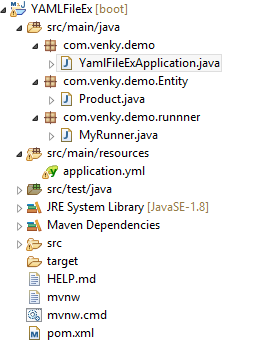
a1: 6.6

a2: 8.6

a3: 9.6

6. Example Application for application.yml properties :-

Folder Structure :-



**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.venky.demo;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

**public** **class** YamlFileExApplication {

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

SpringApplication.*run*(YamlFileExApplication.**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.venky.demo.Entity;

**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, 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.venky.demo.runnner;

**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.venky.demo.Entity.Product;

@Component

**public** **class** MyRunner **implements** ApplicationRunner{

@Autowired

**private** Product prod;

@Override

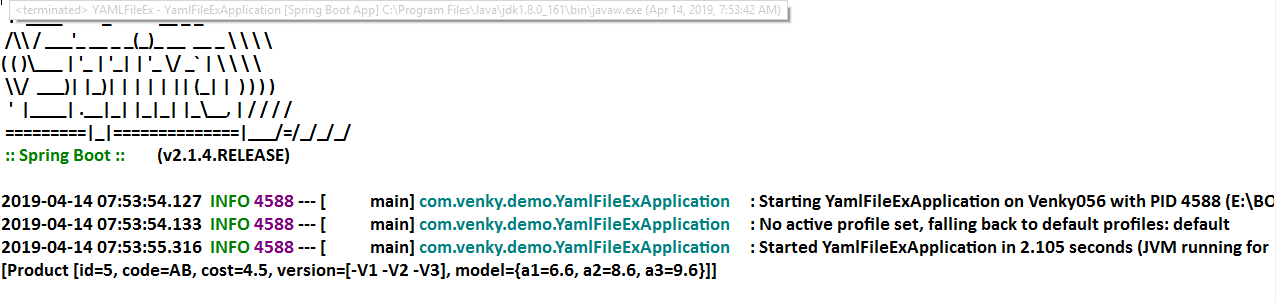
**public** **void** run(ApplicationArguments args) **throws** Exception {

System.***out***.println(Arrays.*asList*(prod));

}

}

**Output :-**



**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.

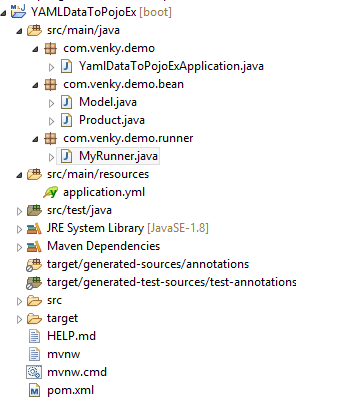
**POJO Rules :-**

1.Class, variable \*\*default constructor with set/get methods.

2.Java Bean :- POJO Rules + Inheritance + Special methods + Param constructor.

3.Spring Bean :- POJO Rules + Inheritance (Spring API) + Annotations (Spring API) + Special methods (Object class and Spring API) [toString()…]

**Folder Structure for providing Yaml data to Bean / POJO :-**



**Example Code :-**

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

**package** com.venky.demo;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

**public** **class** YamlDataToPojoExApplication {

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

SpringApplication.*run*(YamlDataToPojoExApplication.**class**, args);

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

}

}

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

my:

dt:

pid: 55

mo:

mid: 67

mcode: ABC

colors:

-RED

-GREEN

-BLUE

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

**package** com.venky.demo.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.venky.demo.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** Product(**int** pid, Model mo) {

**super**();

**this**.pid = pid;

**this**.mo = mo;

}

**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.venky.demo.runner;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.boot.CommandLineRunner;

**import** org.springframework.stereotype.Component;

**import** com.venky.demo.bean.Product;

@Component

**public** **class** MyRunner **implements** CommandLineRunner

{

@Autowired

**private** Product pob;

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

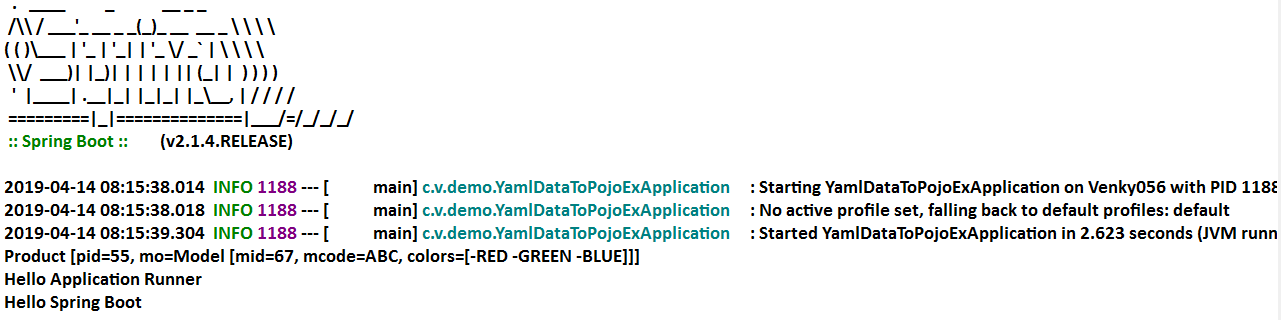
System.***out***.println(pob);

System.***out***.println("Hello Application Runner");

}

}

**Output :-**

****

**Place Holder process in yml (or) properties File**

🡺Internal placeholders 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 :-**

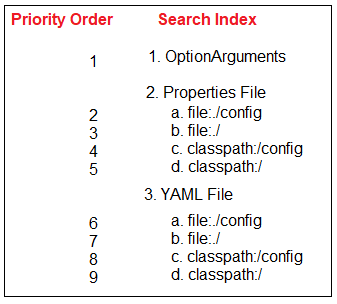
🡺Spring boot has provided default priority to “Option Arguments” (Command Line Args)

1. With format **–key=value**

2. If not found, next level is **.properties.**

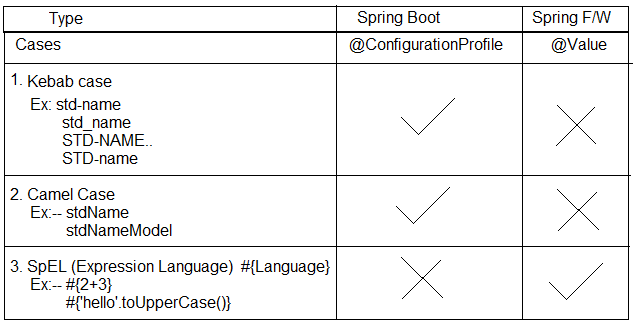
3. else finally chosen **.yml**

4. No-where found default value



**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 :-**



**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.

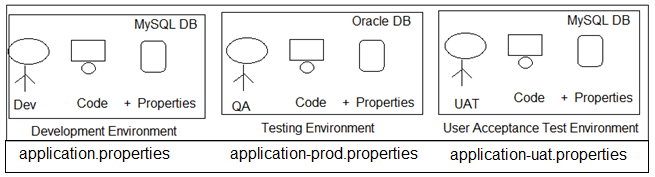
**Example :- dev= development**

* 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).

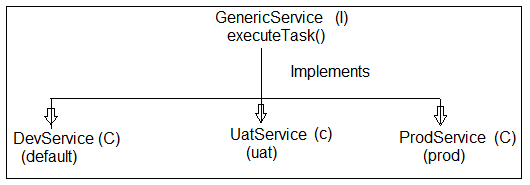
🡺But in this case class should have @Profile(“---“) with @Configuration or @Component (or its equal StereoType).

🡺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

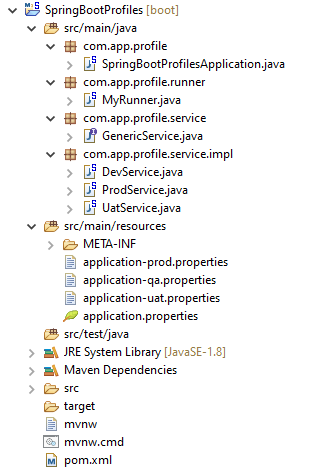
GroupId : org.sathyatech

ArtifactId : SpringBootProfiles

Version : 1.0

🡺 next 🡺 next 🡺 finish.

**Folder structure of Spring Boot Profiles using application.properies :-**



**Code :-**

**1.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.Properties files :-**

**a.application.properties**

my.profile.code=Hello from default

**b. application-prod.properties**

my.profile.code=Hello from PROD

**c. application-uat.properties**

my.profile.code=Hello from UAT

**d. application-qa.properties**

my.profile.code=Hello from QA

**3.service interface :-**

**# Create an Interface (GenericService.java) :-**

**package** com.app.profile.service;

**public** **interface** GenericService {

**public** **void** executeTask();

}

**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.prf.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 + "]";

}

@Override

**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.prf.code}")

**private** String code;

**public** UatService() {

**super**();

}

**public** UatService(String code) {

**super**();

**this**.code = code;

}

**public** String getCode() {

**return** code;

}

**public** **void** setCode(String code) {

**this**.code = code;

}

@Override

**public** String toString() {

**return** "UatService [code=" + code + "]";

}

@Override

**public** **void** executeTask() {

System.***out***.println("From uat profiles");

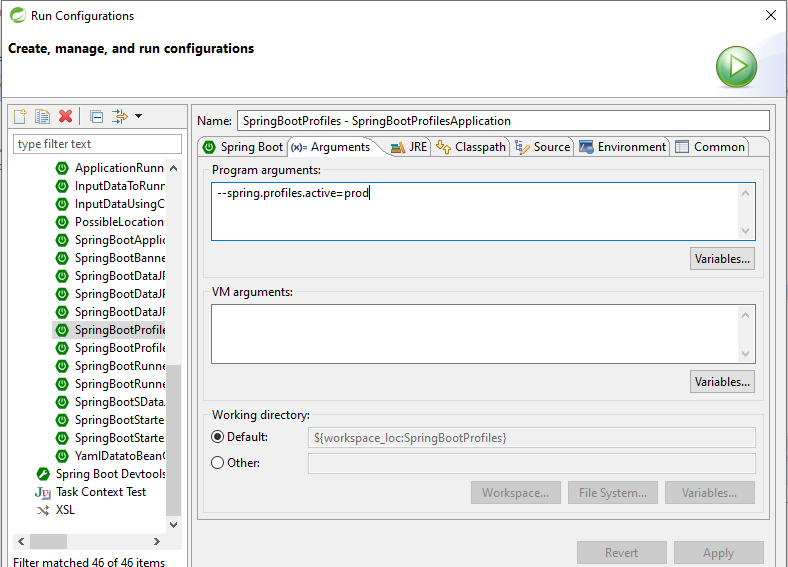
System.***out***.println("code is "+code);

}

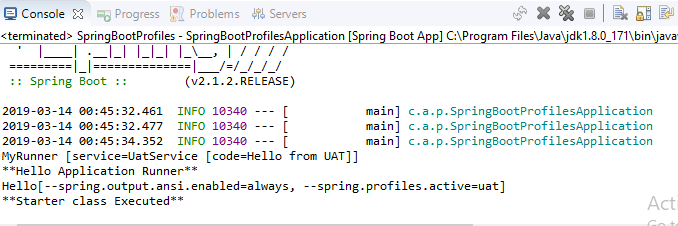
}

**Execution of Program :-**

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



**Output :-**

****

**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

1. In application.proeprties

Ex :- spring.profiles.active=prod

1. 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

1. @Value generate Exception (IllegalArgumnetException)
2. @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

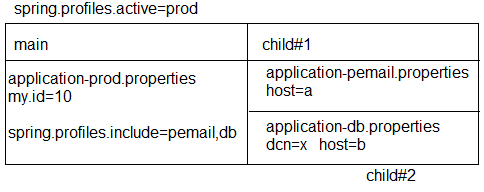
Ex: --spring.profiles.active=[----]

🡺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=-,-,-,-

**Example :-**



**Spring Container will load :-**

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

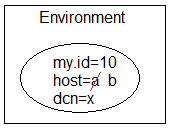
🡺Then child profiles in given order will be loaded.

🡺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:

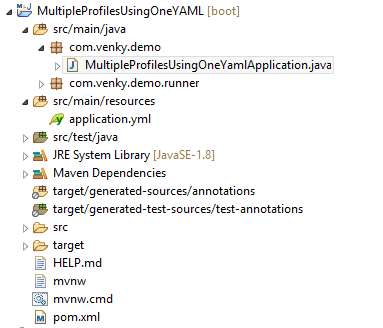
profiles:

active: prod

include:

-prodemail

**Folder Structure of include & active properties in Profiles using Yml :-**

****

**Code :-**

**1.pom.xml same as before one**

**2.Starter class(MultipleProfilesUsingOneYamlApplication.java)**

**package** com.venky.demo;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

**public** **class** MultipleProfilesUsingOneYamlApplication {

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

SpringApplication.*run*(MultipleProfilesUsingOneYamlApplication.**class**, args);

}

}

**2. application.yml :-**

my:

prf:

id: 666

spring:

profiles:

active: prod

include:

-prodemail

#Production Profiles

---

my:

prf:

id: 999

spring:

profiles: prod

#Uat profiles

---

my:

prf:

id: 888

spring:

profiles: uat

#Prodemail profile

---

my:

prf:

email: venkatadri053@gmail.com

spring:

profiles: prodemail

**3.Bean and Runner class (MyRunner.java) :-**

**package** com.venky.demo.runner;

**import** org.springframework.boot.CommandLineRunner;

**import** org.springframework.boot.context.properties.ConfigurationProperties;

**import** org.springframework.stereotype.Component;

@Component

@ConfigurationProperties("my.prf")

**public** **class** MyRunner **implements** CommandLineRunner{

**private** **int** id;

**private** String email;

**public** MyRunner() {

**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**);

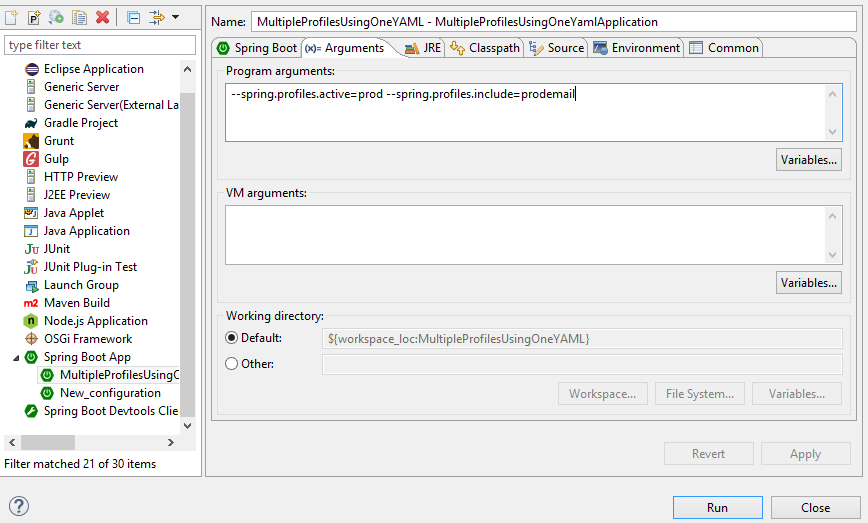
}

}

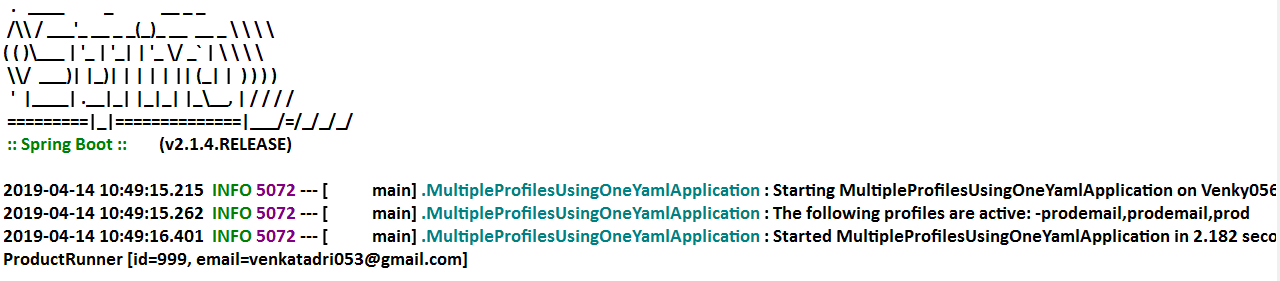
**Execution Process:-**

Run As🡺Run Configurations🡺select Arguments🡺Pass the values in Program Arguments

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

****

**Output :-**



**pom.xml (Maven Process)**

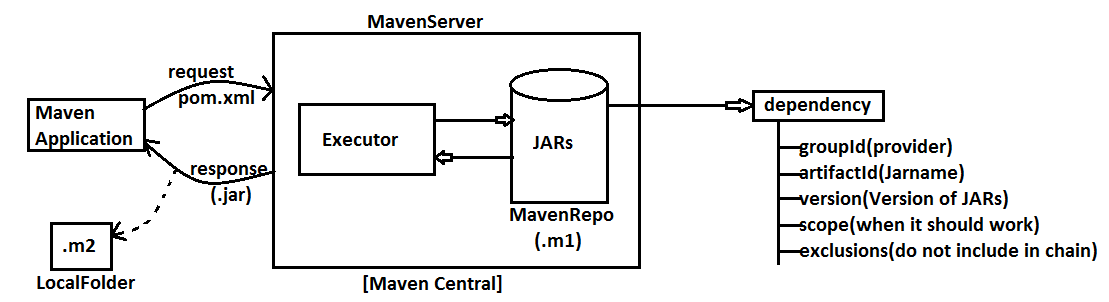
🡺Maven is Dependency management and build tool used to handle both stand alone and Archetype (web, restful…) applications.

**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.

**Diagram :-**



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

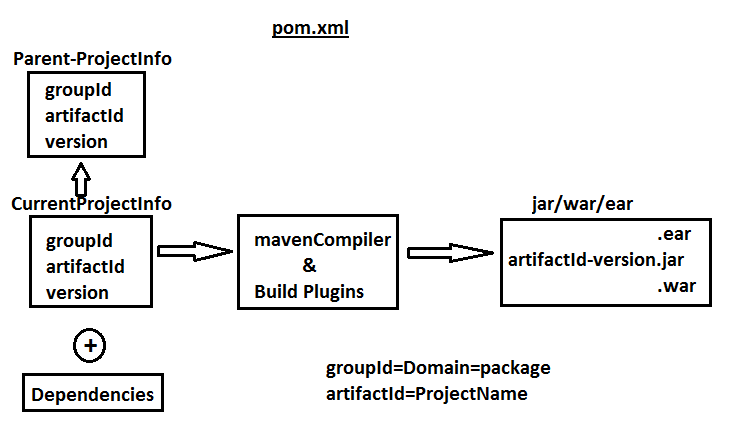
1.Current Project

2.Parent Project

3.Dependencies

4.Build plugins

**Diagram :-**



**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 Dtails -->

<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>

**Example:-**

<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) :-**

* **compile :-** A jar Required from compilation time onwards. It is only default scope.
* **runtime :-** A jar required when we are running an Application, not before that.
* **test :-** A Jar required only for UnitTesting time.
* **Provided :-** A jar provided by servers or Frameworks (Container….).
* **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 downloaded all required plugins and also

🡺compile the source files.

🡺load required properties.

🡺Execute JUnit Test cases.

🡺Create final build (.jar/.war).

**# clean :-**

🡺Right click on project 🡺 Run As 🡺Maven clean

**# install:-**

🡺Right click on project 🡺 Run As 🡺Maven install.

**# 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,

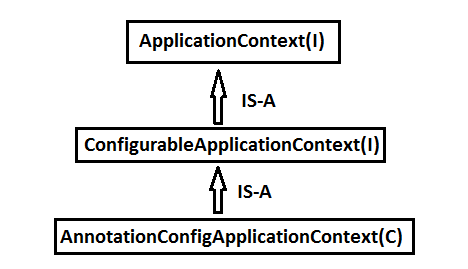
a. Goto pom.xml

b. Ctrl + Mouse over <artifactId> then click

c. Search for tag properties

**Spring Boot Starter class Concept**

**#1#** 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 it supportive methods and API Types.

**Ex:-**

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

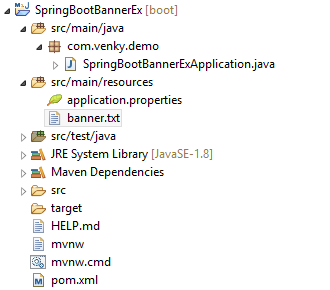
b. Mode is Inner enum defined in functional Interface “Banner”.

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>)

**Folder Structure of Banner in Spring Boot :-**

****

**Code :--**

**1.SpringBootBannerExApplication.java**

**package** com.venky.demo;

**import** org.springframework.boot.Banner;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

**import** org.springframework.context.ConfigurableApplicationContext;

@SpringBootApplication

**public** **class** SpringBootBannerExApplication {

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

SpringApplication sa = **new** SpringApplication (SpringBootBannerExApplication.**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

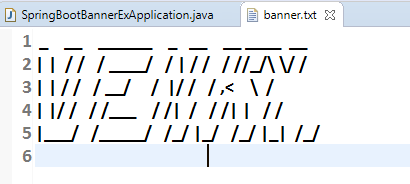
ConfigurableApplicationContext c = sa.run(args);

System.***out***.println(c.getClass().getName().toString());

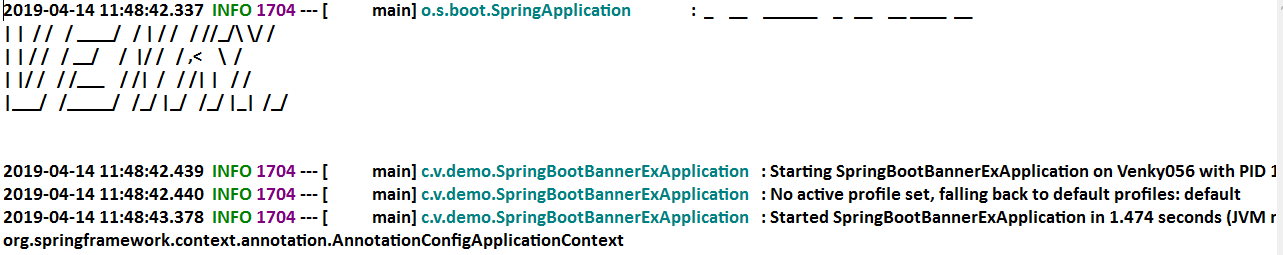
}

}

**2.Banner.txt file :-**



**Output with Banner :-**



**#2#** Spring Boot starter class package behaves as base package for component scan of all classes having annotated with @Component [or its equal].

🡺Annotations are: (class should have any one)

1. @Component
2. @Repository
3. @Service
4. @Controller
5. @RestController
6. @Configuration

**Example:-**

**Consider below starter class :-**

package com.app;

@SpringBootApplication

public class MyStarter {…….}

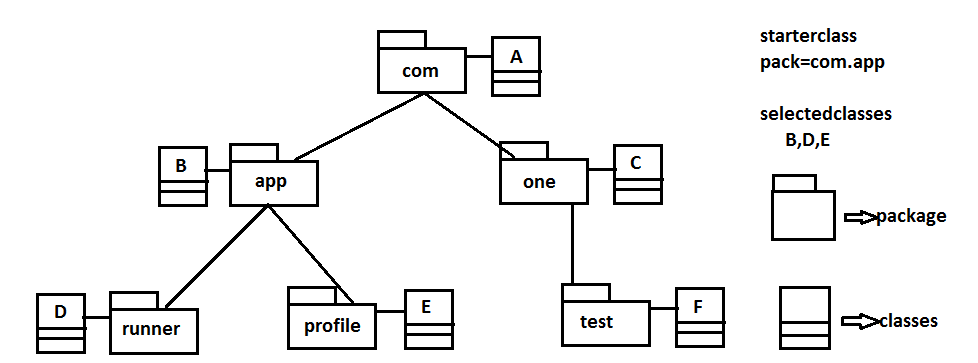
🡺In this case only classes under package “app” and its sub package classes are detected by Spring (Boot) Container by default.

🡺Programmer can provide externally basepackage using @ComponentScan

1. Avoid Starter class package and provide our own.

Ex :- @ComponentScan (“com.app”)

🡺In this case Starter package classes are not included.



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

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

1. 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 {…..}

**#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.

@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\*\*");

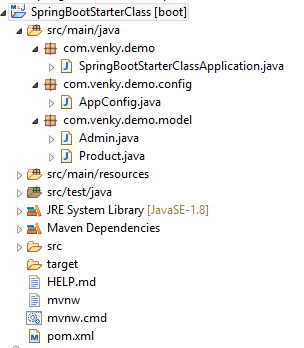
}

}

**#4#** 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).

**Folder Structure of Starter class with AutoConfiguration&import :-**



**1.Model classes**

**a.Admin.java :-**

**package** com.venky.demo.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.venky.demo.model;

**public** **class** Product {

**private** **int** prodId;

**private** String prodName;

**public** Product() {

**super**();

}

**public** **int** getProdId() {

**return** prodId;

}

**public** **void** setProdId(**int** prodId) {

**this**.prodId = prodId;

}

**public** String getProdName() {

**return** prodName;

}

**public** **void** setProdName(String prodName) {

**this**.prodName = prodName;

}

@Override

**public** String toString() {

**return** "Product [prodId=" + prodId + ", prodName=" + prodName + "]";

}

}

**2.AppConfig.java :-**

**package** com.venky.demo.config;

**import** org.springframework.context.annotation.Bean;

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

**import** com.venky.demo.model.Admin;

@Configuration

**public** **class** AppConfig {

@Bean

**public** Admin aobj() {

Admin a = **new** Admin();

a.setAdminId(100);

a.setAdminName("Venkatadri");

**return** a;

}

}

**3.Starter class :-**

**package** com.venky.demo;

**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** com.venky.demo.model.Admin;

**import** com.venky.demo.model.Product;

@SpringBootApplication

//@Import (AppConfig.class) //Its not required

**public** **class** SpringBootStarterClassApplication **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*(SpringBootStarterClassApplication.**class**, args);

System.***out***.println("\*\*Starter class main method executed");

}

@Bean

**public** Product proj() {

Product p = **new** Product();

p.setProdId(300);

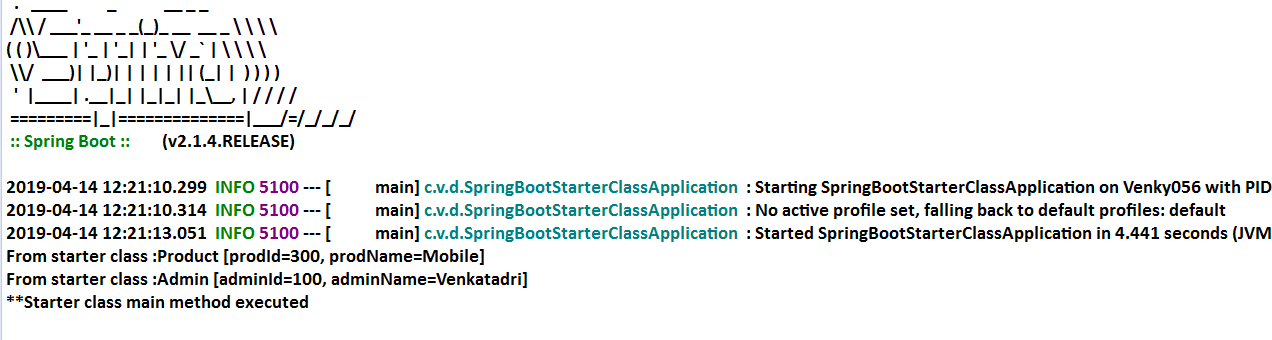
p.setProdName("Mobile");

**return** p;

}

}

**Output :-**



**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 EclipseIDE (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** 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 bellow 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 appstops.

🡺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).

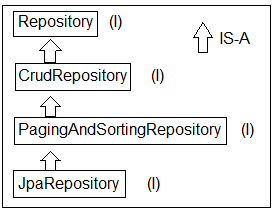
**#5** Data JPA supports Easy Connection Pooling (Auto Config) concept.

**#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 :-**

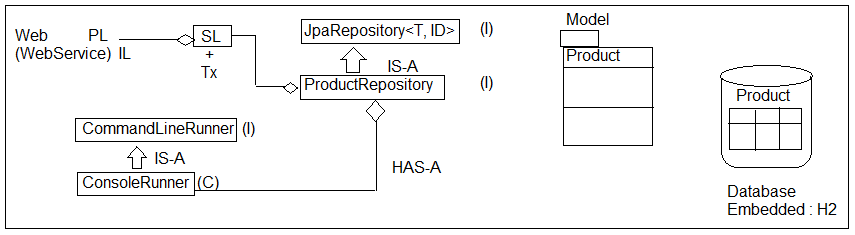
**Required:**

1.Database (Using Embedded : H2)

2.Model class : Product (C)

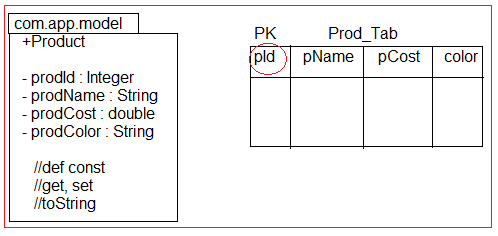
3.Repository : ProductRepository

4.Runner : ConsoleRunner



**T = ? = Model class Name**

**ID = ? = Pk DataType = Integer**



🡺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)

Crtl +alt +DownArrow 🡺 Copy current line, paste

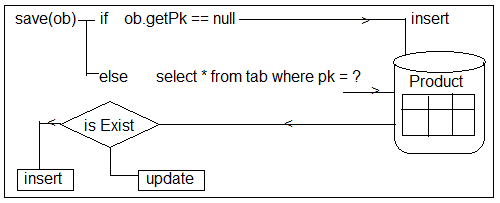
Select Lines

+ctrl + shift + / 🡺comment lines

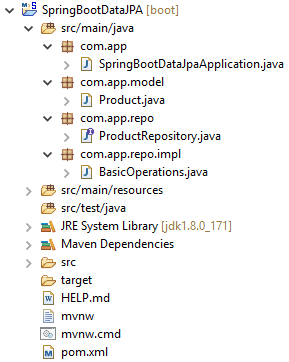
+ctrl +shift + \ 🡺Uncomment lines

**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 prodId;

private String prodName;

private double prodCost;

private String prodColor;

//super constructor

public Product() {

super();

}

//Id (PK) based constructor

public Product(Integer prodId) {

super();

this.prodId = prodId;

}

//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 prodId, String prodName, double prodCost, String prodColor)

{

super();

this.prodId = prodId;

this.prodName = prodName;

this.prodCost = prodCost;

this.prodColor = prodColor;

}

//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 getProdColor() {

return prodColor;

}

public void setProdColor(String prodColor) {

this.prodColor = prodColor;

}

//toString method

@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

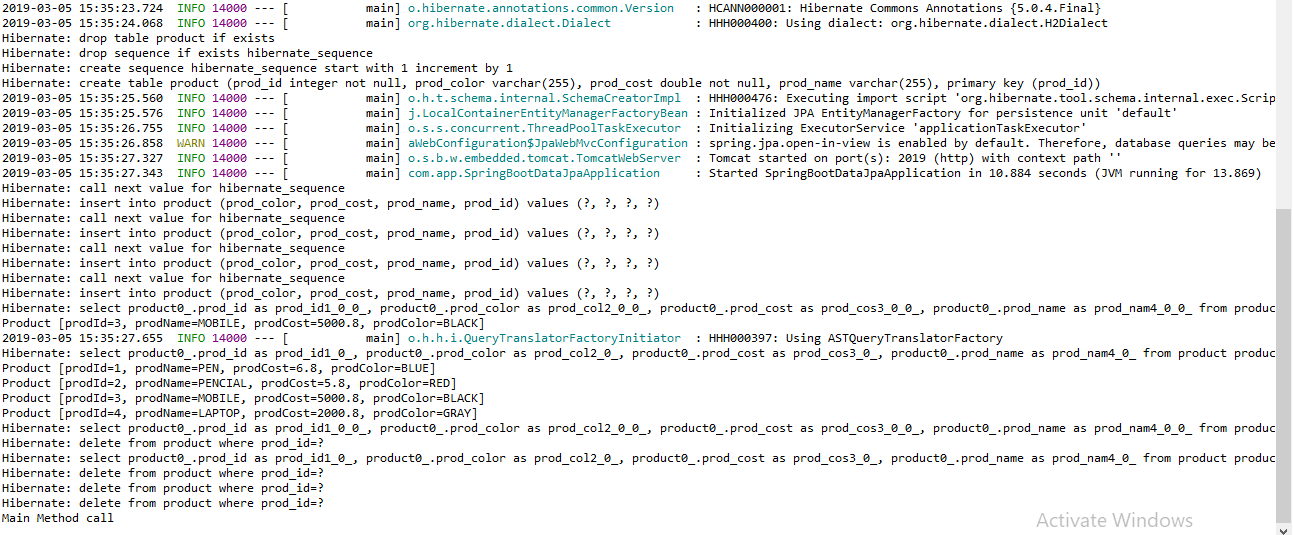
//3.3 Delete all rows in Batch (Single Query fired)

repo.deleteAllInBatch();

}

}

**Output :-**



**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)finadAll () :-** 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.

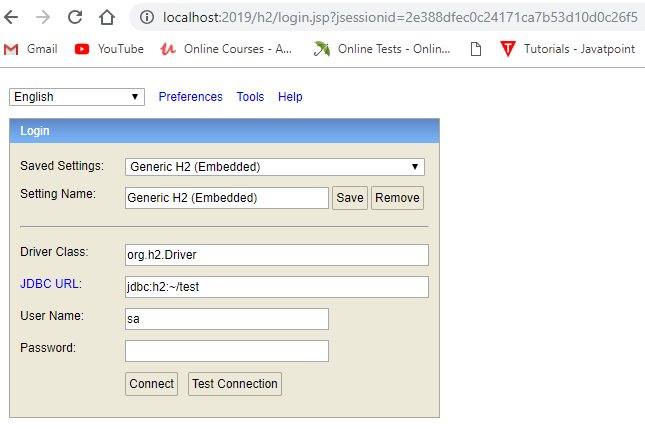
🡺H2 console works only if use WebApp and default path is : /h2-console,

default port : 8080

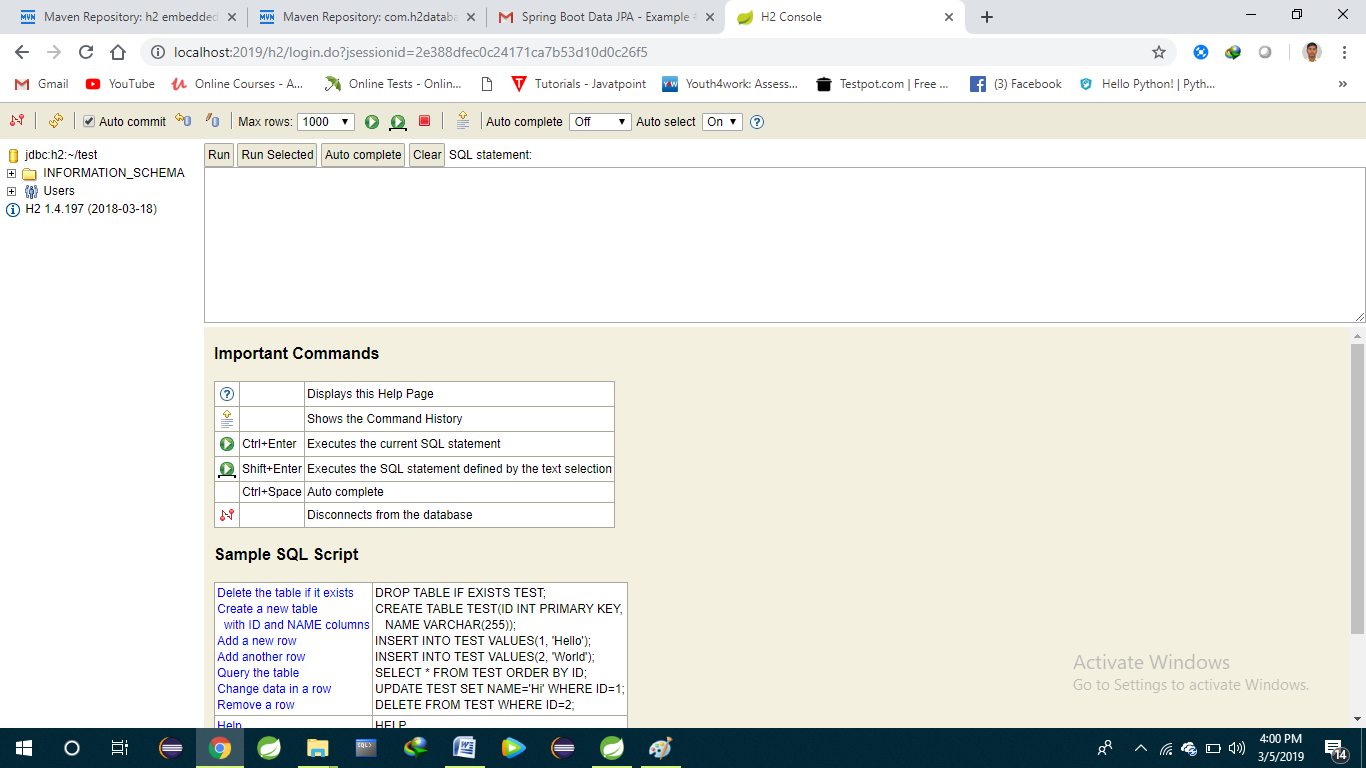
**How to Show Data in H2 DataBase :-**

**Step #1:-** Execute Program

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



🡺Click on connect.

**Step #3 :**

**Query Method in Spring Boot DataJPA**

🡺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 …);**

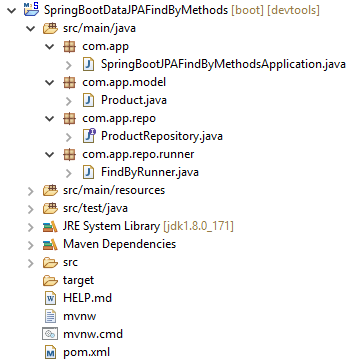
Here, RT = ReturnType,

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 | … where x.lastname = ?1 or  x.firstname = ?2 |
|  | **Is,Equals** | findByFirstname,findByFirstnameIs,  findByFirstnameEquals | … where x.firstname = ?1 |
|  | **Between** | findByStartDateBetween | … where x.startDate between ?1  and ?2 |
|  | **LessThan** | findByAgeLessThan | … where x.age < ?1 |
|  | **LessThanEqual** | findByAgeLessThanEqual | … where x.age <= ?1 |
|  | **GreaterThan** | findByAgeGreaterThan | … where x.age > ?1 |
|  | **GreaterThanEqual** | findByAgeGreaterThanEqual | … where x.age >= ?1 |
|  | **After** | findByStartDateAfter | … where x.startDate > ?1 |
|  | **Before** | findByStartDateBefore | … where x.startDate < ?1 |
|  | **IsNull** | findByAgeIsNull | … where x.age is null |
|  | **IsNotNull,NotNull** | findByAge(Is)NotNull | … where x.age not null |
|  | **Like** | findByFirstnameLike | … where x.firstname like ?1 |
|  | **NotLike** | findByFirstnameNotLike | … where x.firstname not like ?1 |
|  | **StartingWith** | findByFirstnameStartingWith | … where x.firstname like ?1  (parameter bound with appended %) |
|  | **EndingWith** | findByFirstnameEndingWith | … where x.firstname like ?1  (parameter bound with prepended %) |
|  | **Containing** | findByFirstnameContaining | … where x.firstname like ?1  (parameter bound wrapped in %) |
|  | **OrderBy** | findByAgeOrderByLastnameDesc | … where x.age = ?1 order by  x.lastname desc |
|  | **Not** | findByLastnameNot | … where x.lastname <> ?1 |
|  | **In** | findByAgeIn(Collection<Age> ages) | … where x.age in ?1 |
|  | **NotIn** | findByAgeNotIn(Collection<Age> ages) | … where x.age not in ?1 |
|  | **True** | findByActiveTrue() | … where x.active = true |
|  | **False** | findByActiveFalse() | … where x.active = false |
|  | **IgnoreCase** | findByFirstnameIgnoreCase | … where UPPER(x.firstame) =  UPPER(?1) |

**14. 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 prodId;

private String prodCode;

private String prodName;

private double prodCost;

//super constructor

public Product() {

super();

}

//Id (PK) based constructor

public Product(Integer prodId) {

super();

this.prodId = prodId;

}

//Parameterized constructor

public Product(String prodName, double prodCost, String prodCode) {

super();

this.prodName = prodName;

this.prodCost = prodCost;

this.prodCode = prodCode;

}

public Product(Integer prodId, String prodName, double prodCost, String prodCode)

{

super();

this.prodId = prodId;

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

//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 findByProdCostGreaterThan(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> findByProdCostLessThan OrderByProdCode(Double cost);

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

List<Product> findByProdIdLessThanAndProdCostGreaterThanAndVendorNotNullOrderByPcode(Integer prodId, Double prodCost);

**4.Runner class (FindByRunner.java) :-**

findByIdOrProdCost(10, 25.5).forEach(System.out : : prodCost);

**2)@Query(“hql”)**

This annotation 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)

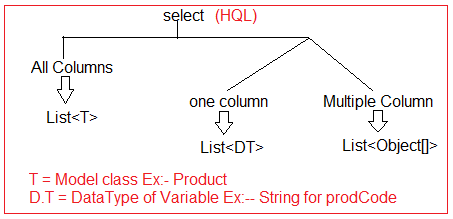
Named parameters in style [:name]:a, :b, :c, :hello, :mydata

🡺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 (both are same)



**Folder Structure of @Query param in Spring Boot :-**

**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 prodId;

private String vendorCode;

private String prodName;

private Double prodCost;

public Product() {

super();

}

public Product(Integer prodId) {

super();

this.prodId = prodId;

}

public Product(Integer prodId, 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 :-

//select \* from producttab 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 prodtab where vendor vendorCode=? or prodCost=?

@Query(“select p.prodCode from Product p where p.vendorCode=?1 or p.prodCost=?2”)

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

//select p.prodCode, p.prodCost from product

@Query(“select p.prodCode, p.prodCost from Product p”)

List<Object[]> getAllProductData();

🡺CommandLineRunner code for getAllProductData method

//JDK 1.5

List<Object[]> obs=repo.getAllProductData();

for(Object[] ob:obs)

{

System.out.println(ob[0]+”, “ob[1]);

}

//JDK 1.8 (Streams + Lambda+Method Reference)

repo.getAllProductData().stream().map((ob)-> ob[0]+”,”+ob[1]).forEach(System.out::println);

**@Query(“……..”) non-select operations**

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

🡺@Transaction 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);

**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

1. Static Projections
2. 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\_\_\_(…);

}

**Example:**

**1. Repository code:**

package com.app.repo;

public interface ProductRepository extends JpaRepository<Product, Integer>{

interface MyView{

//Copy from getter method (RT and methodName)

String getProdCode();

Double getProdCost();

}

//select code,cost from prodtab where ven\_code=?

List<MyView> findByVendorCode(String vc);

}

**2. CommandLineRunner code:**

//@Component

repo.findByVendorCode(“V2”)

.stream()

.map((p))->p.getProdCode()+”,”+p.getProdCost()

.forEach(System.out::println);

**b. Dynamic Projections:**

In this case findBy(\_\_\_) method return type is decided at runtime (ie It is Generics)

**Format&Syntax:**

**<T> List<T> findBy\_\_\_(...,Class<T> clz);** //fixed Line

**Note:** Here “T” is child interface type, provide at runtime.

**1. Repository code:**

package com.app.repo;

public interface ProductRepository extends JpaRepository<Product, Integer>{

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:**

repo.findByVendorCode(“V2”,MyViewTwo.class)

.stream()

.map((ob)->ob.getProdId()+”,”+ob.getProdCost())

.forEach(System.out::println);

//or

repo.findByVendorCode(“V2”,MyView.class)

.stream()

.map((a)->a.getProdId()+”,”+a.getProdCost())

.forEach(System.out::println);

**Note:**

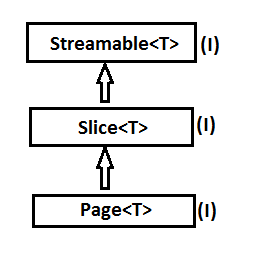
1. Dynamic Projections are slow to static Projections.
2. For dynamic Projections “Type select, validate and execution” done at Runtime, where as for static select and validate done at compile time, only execution done at runtime.
3. Static Projections also called as compile time (selected) Projections and Dynamic Projections also called as Runtime (selected) Projections.

**Special Types in Query methods**

1. Page<T>

2. Slice<T>

3.Stream<T>



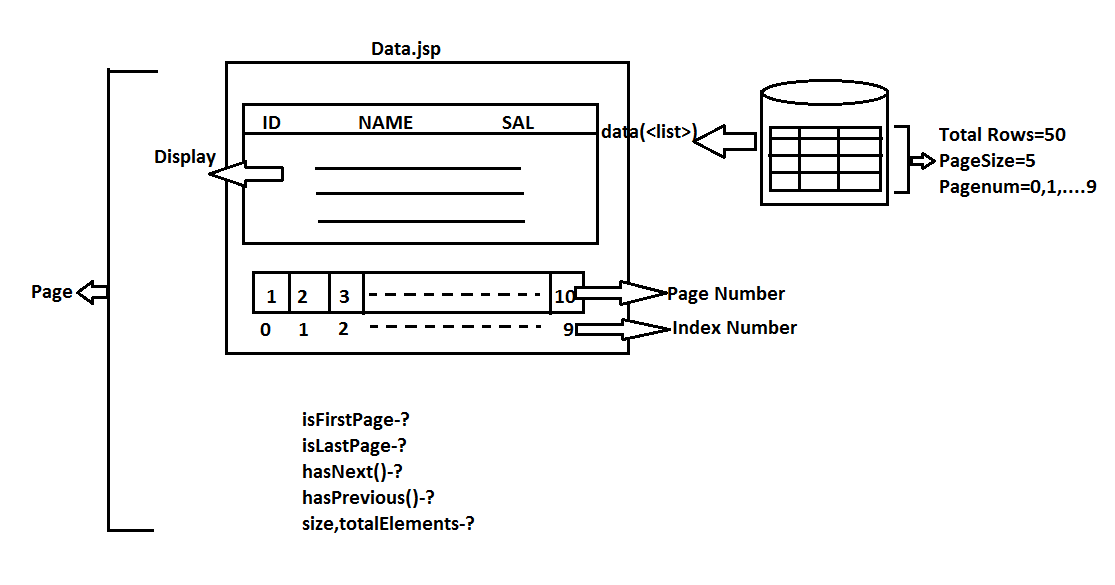
These are Special type outputs foer Query methods to get **“Pagination data”**

**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

**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.

**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 givwen 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

**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 springboot configures using class input **“HikariConfig”** with all default vaiues (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 #3Spring Boot NoSQL Database [MongoDb]**

🡺MongoDb is a simple and open-source document database and leading NOSQL databse.

🡺MongoDB can be integrated with Spring Boot using Dependency

<dependency>

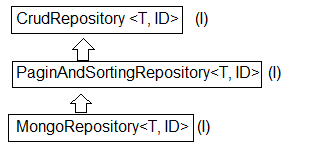
<groupId>ord.springframework.boot</gropId>

<artifactId>spring-boot-starter-data-mangodb</artifactId>

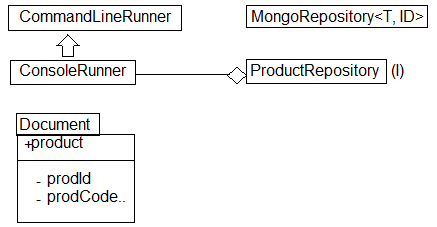
</dependency>

🡺Boot also supports Embedded MongoDB for every coding and Testing Process

**Design#1 Spring Boot MongoDB Repository Levels**



**Design#2 Spring Boot MongoDB Coding files**



**DOWNLOAD AND INSTALLATION OF MONGODB**

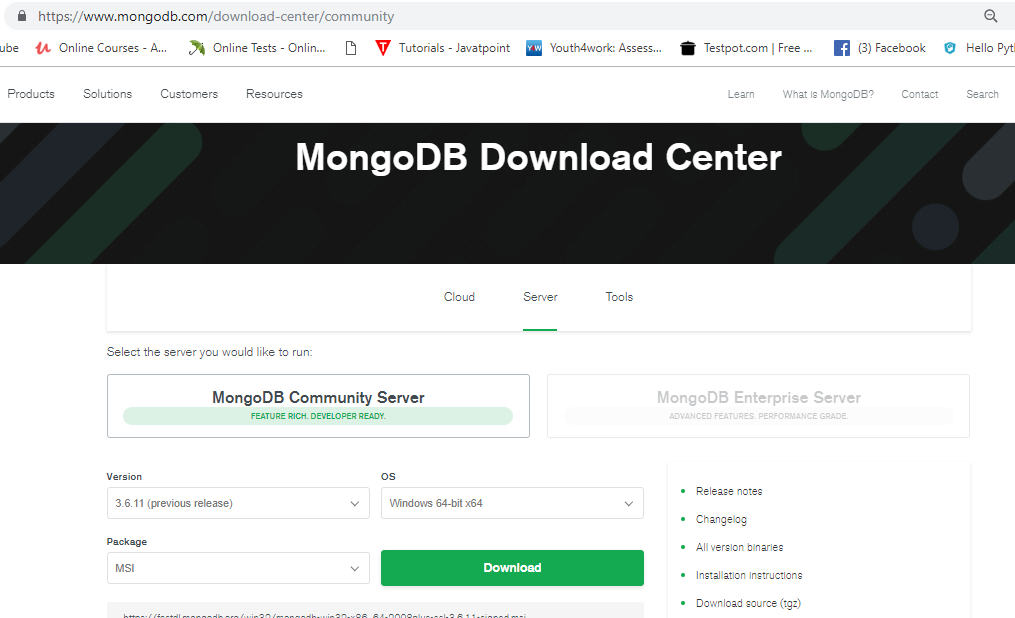
Download and Install MongoDB (NoSQL) URL :

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

Details:

Version : 3.6 or 3.4

Type : MSI (Do not use ZIP)



🡺While Installing Full (Complete) Type.

🡺After install create one folder in “C” drive.

(Create folder data>> create folder db\_

**Like : c:/data/db**

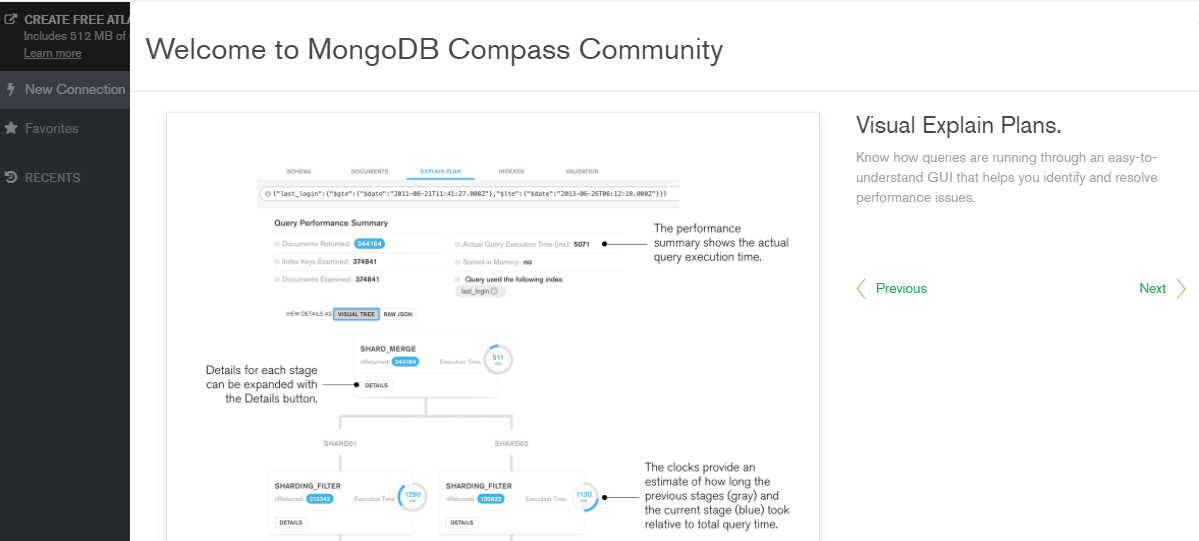
🡺Set Path to MongoDB

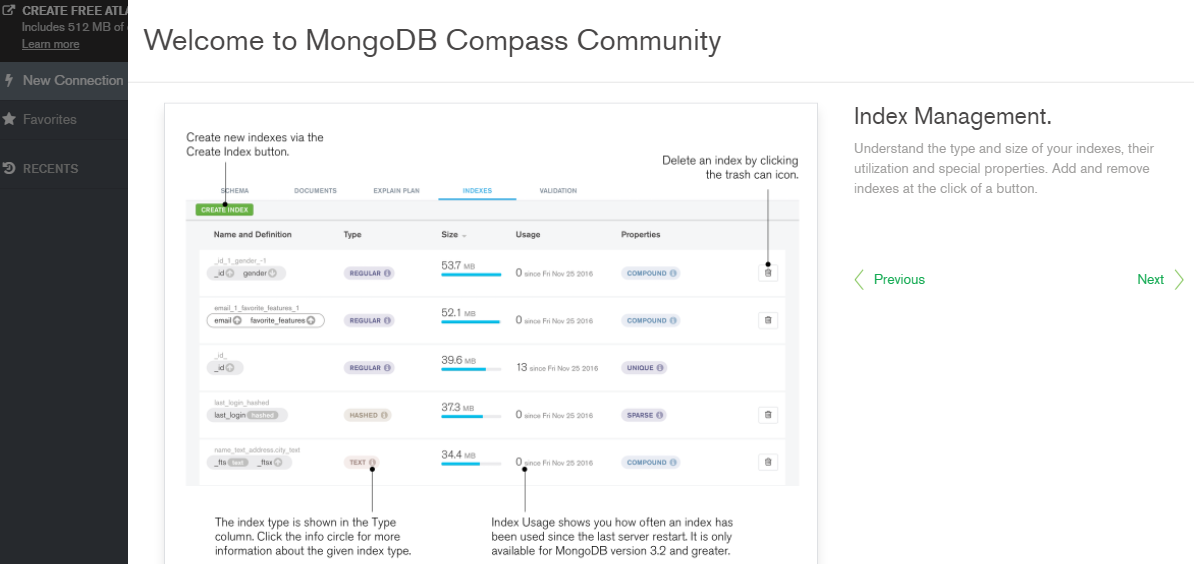
Location:

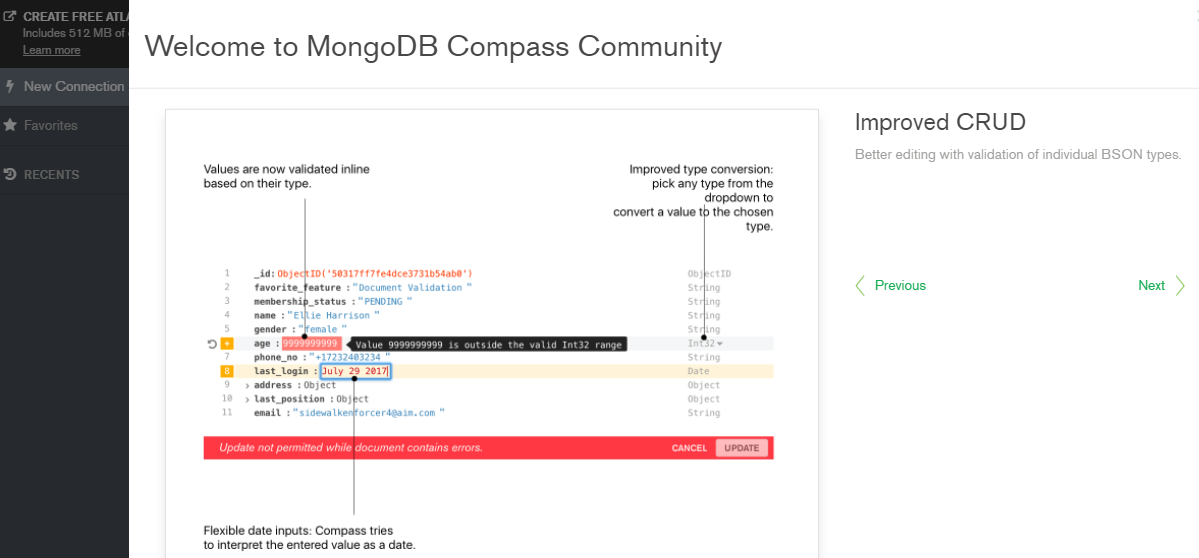
C:\program Files\MongoDB\Server\3.4\bin

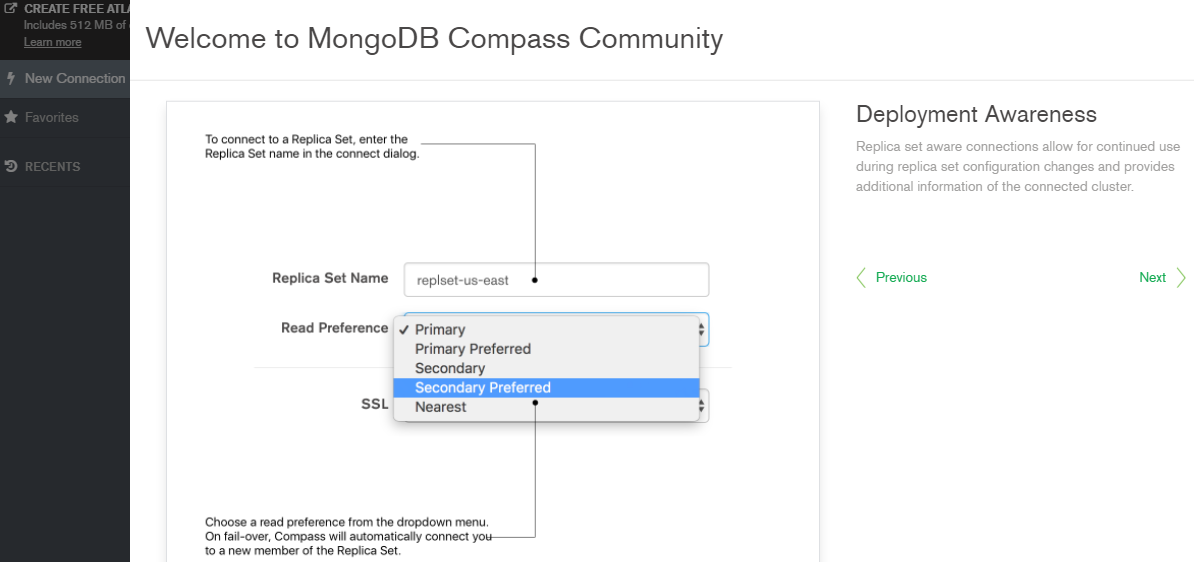
🡺Open cmd-1 and type > mongodd (to start server)

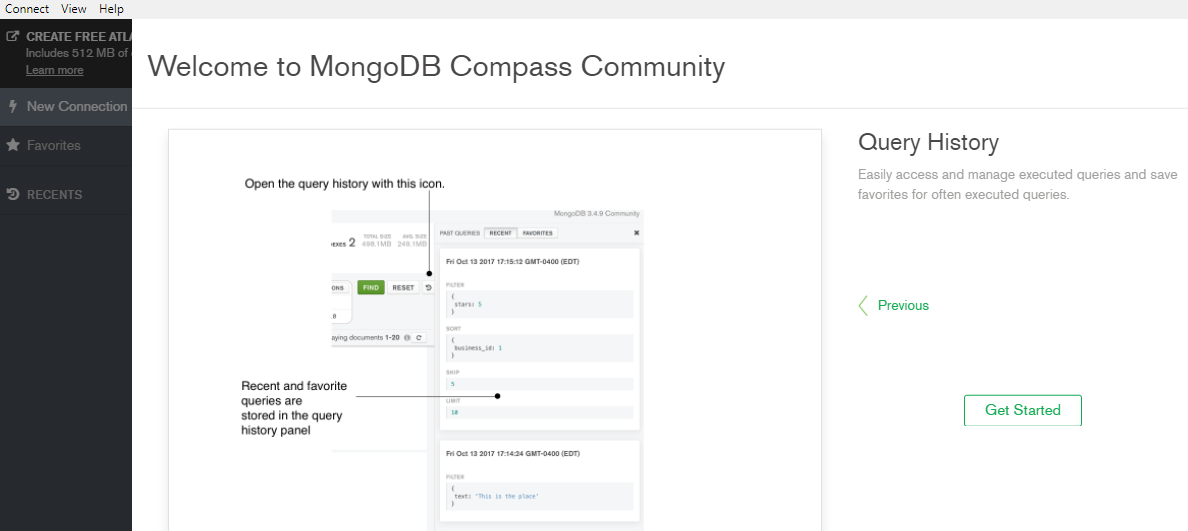
🡺Open cmd-2 and type > mongo (to open and work on DB)

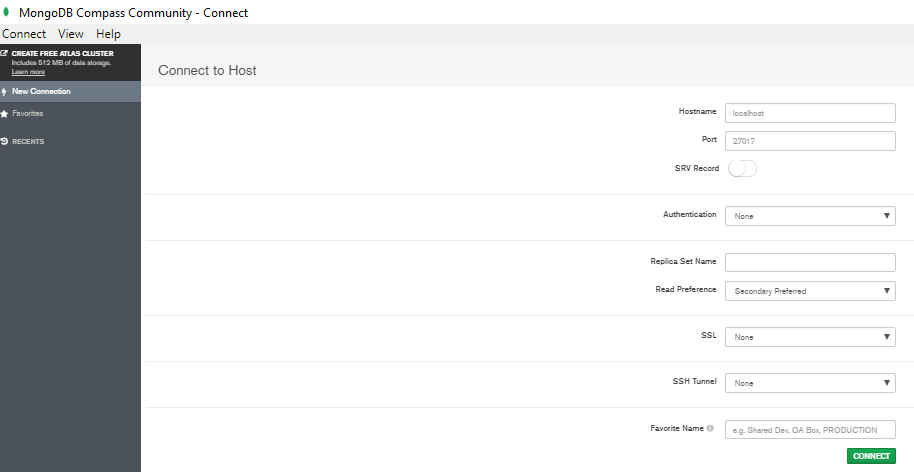


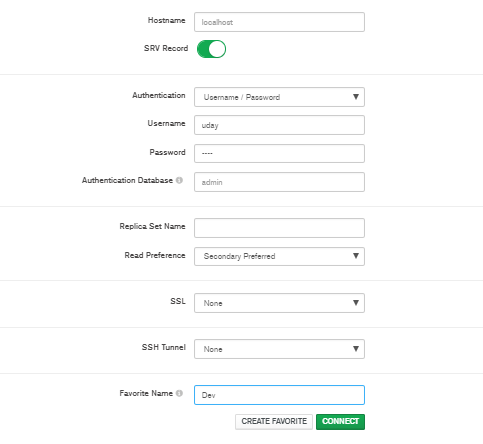












🡺Open Notepad and Type below command

🡪mongod🡪save with .bat

**Ex :- “mongo-server.bat”**

🡪mongo 🡪 save with .bat

**Ex :-“mongo-client.bat”**

**MongoDB:-** It is a NOSQL Database which holds data in Collection Format.

🡺Every Collection (Similar to table, but not table) holds data in JSON Objects (Similar to rows, but not rows).

🡺JSON (Java Objet Notation) format looks as : {“key” : value,…}

🡺Every Collection also called as **Document** [Word used in Programming Language]

**Collection Format:-**

**Example Collection:-**

Student

{“stdId”:25,”stdName”: “Uday Kumar”, “stdFee”: 2.3},

{“stdId”:35,”stdName”: “Neha Kumari”, “stdFee”: 5.6},

{“stdId”:35,”stdName”: “Venkat”, “stdFee”: 8.2}

**Step#1:-** After Installing MongoDb, set Path

Location : c:\ProgramFiles\MongoDb\Servr\3.4\bin

**Step#2:-** Start server using command “mongod”

Ex:- d:/user/>mongod

**Step#3:-** Start Client using cmd “mongo”

Ex:- d:/user/>mogo

**Step#4:-** Work on Database

**#View All Database**

🡺use sathya

**#Create one Collection/insert one row**

🡺db.student.insert({stdId”:5, “stdName”:”Uday”, stdFee” : 2.3})

**#View All Collections in current DB**

🡺show collection

**#View data in one collection**

🡺db.student.find() (or)

🡺db.student.find().pretty()

🡺db.student.find()({“stdId”:5}).pretty()

**#Delete one Collection object**

🡺db.student.remove({“stdId”:5})

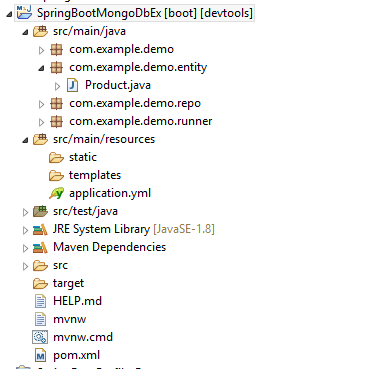
**Coding:-**

**Step#1:-** Create one Spring Project

Ex :-SpringBootMongoDb

**Dependencies :** MongoDb, Embadded MongoDb and Web

**Folder Structure :-**

****

**1.Starterclass(SpringBootMongoDbExApplication.java)**

**package** com.example.demo;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

**public** **class** SpringBootMongoDbExApplication {

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

SpringApplication.*run*(SpringBootMongoDbExApplication.**class**, args);

}

}

**Step#2:- Define one Document (Collection)**

**2. Product.java**

**package** com.example.demo.entity;

**import** org.springframework.data.annotation.Id;

**import** org.springframework.data.mongodb.core.mapping.Document;

@Document

**public** **class** Product {

@Id

**private** String prodId;

**private** String prodName;

**private** Double prodCost;

**private** String vendor;

**public** Product() {

**super**();

}

**public** Product(String prodName, Double prodCost, String vendor) {

**super**();

**this**.prodName = prodName;

**this**.prodCost = prodCost;

**this**.vendor = vendor;

}

**public** Product(String prodId, String prodName, Double prodCost, String vendor) {

**super**();

**this**.prodId = prodId;

**this**.prodName = prodName;

**this**.prodCost = prodCost;

**this**.vendor = vendor;

}

**public** String getProdId() {

**return** prodId;

}

**public** **void** setProdId(String 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 getVendor() {

**return** vendor;

}

**public** **void** setVendor(String vendor) {

**this**.vendor = vendor;

}

@Override

**public** String toString() {

**return** "Product [prodId=" + prodId + ", prodName=" + prodName + ", prodCost=" + prodCost + ", vendor=" + vendor

+ "]";

}

}

**Step#3 Define one Repository interface**

**3.ProductRepository.java**

**package** com.example.demo.repo;

**import** org.springframework.data.mongodb.repository.MongoRepository;

**import** com.example.demo.entity.Product;

**public** **interface** ProductRepository **extends** MongoRepository<Product, String> {

}

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

**4.MyRunner.java**

**package** com.example.demo.runner;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.boot.CommandLineRunner;

**import** org.springframework.stereotype.Component;

**import** com.example.demo.entity.Product;

**import** com.example.demo.repo.ProductRepository;

@Component

**public** **class** MyRunner **implements** CommandLineRunner{

@Autowired

**private** ProductRepository repo;

@Override

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

repo.deleteAll();

repo.save(**new** Product("pen",3.3,"p-ven"));

repo.save(**new** Product("book",233.3,"b-ven"));

repo.save(**new** Product("car",4653.3,"c-ven"));

repo.save(**new** Product("phone",683.3,"p-ven"));

repo.findAll().forEach(System.***out***::println);

}

}

**Step#5:-** remove <scope>test</test> for Embedded Mongo dependency in pom.xml (or)

🡺To Work on external (Install) MongoDB

🡺Delete/comment Embedded Mongo Dependency in pom.xml

<!-- <dependency>

<groupId>de.flapdoodle.embed</groupId>

<artifactId>de.flapdoodle.embed.mongo</artifactId>

<scope>test</scope>

</dependency> -->

🡺Add keys in application.properties

**##MongoDB External Details**

spring.data.mongodb.host=llocalhost

spring.data.mongodb.port=27017

spring.data.mongodb.database=uday

**Equal application.yml file is :-**

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

server:

port : 6780

spring:

data:

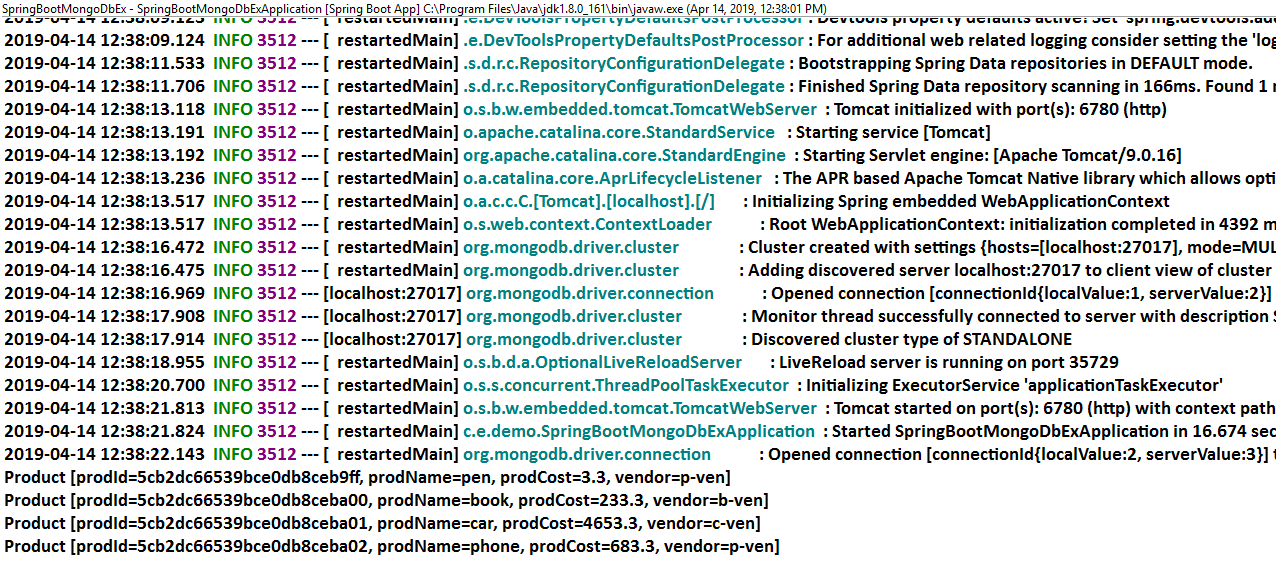
mongodb:

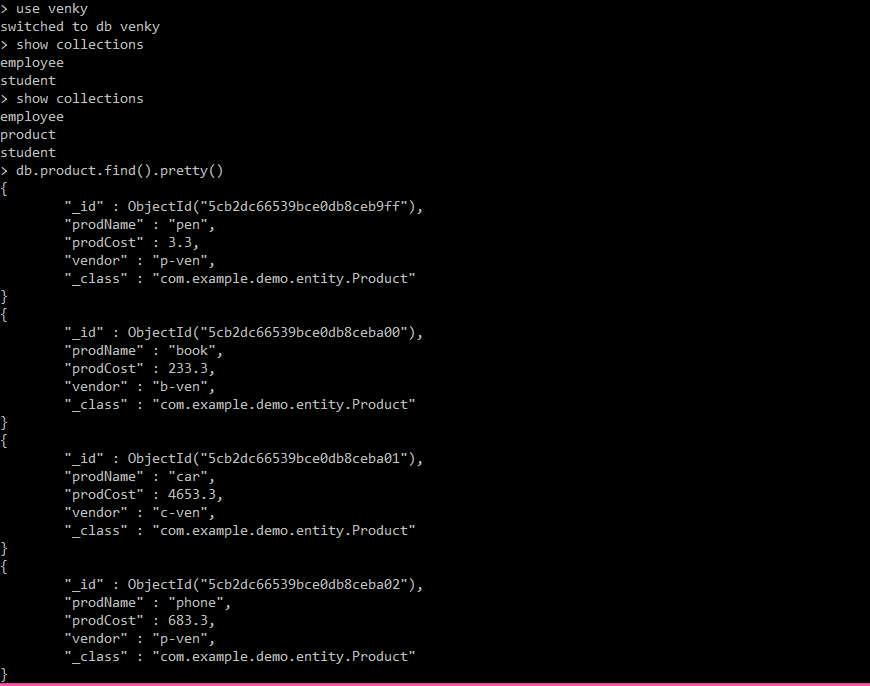
host: localhost

port: 27017

database: venky

**Output in SpringConsole:**

**output in MongoDb Console:-**



**CHAPTER#4 SPRINGBOOT-SPRING WEB MVC**

**#1** Spring Boot provides 3 embedded servers Tomcat, Jetty and Undertow. Default is Tomcat.

**#2**Default Port number is 8080 (server.port).

**#3** FrontController named as “DispatcherServlet” is configured by Spring Boot with URL-Pattern=/.

**#4** Context-Path (ProjectName) also mapped to /.

**#5** Handler Mapper (DefaultAnnotationHandlerMapper) also configured by Spring Boot

**#6** ComponentScan (basePackage) is set to starter class package name by Spring boot.

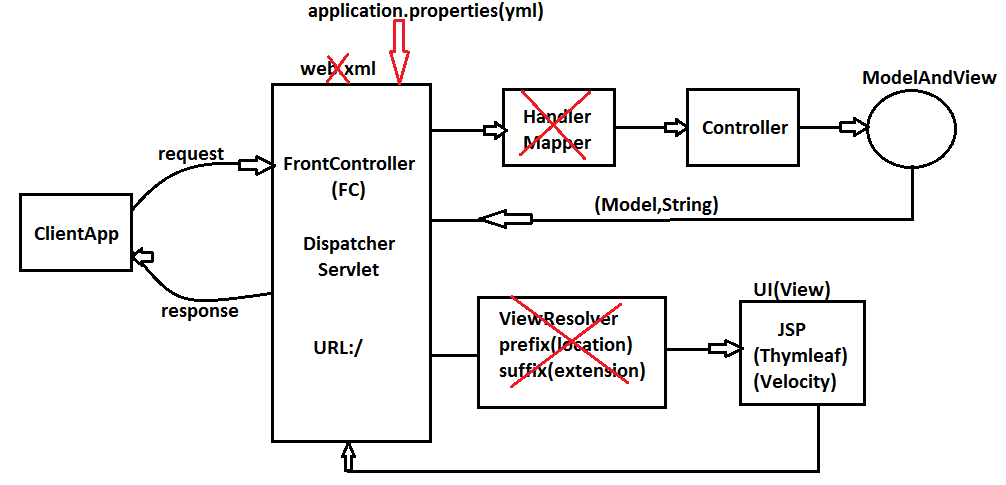
**#7** ViewResolver is autoConfigured by Spring boot, but programmer has to provide input using Properties/YML file.

**#8** All WEB MVC Required JARS are set to Project by Spring Boot.

**#9** Boot provide maven plugin for “JAR/WAR” packing for cloud deployment.

**#10** SpringBoot-springWEBalso supports ReSTFul webservices also.

**Diagram :-**

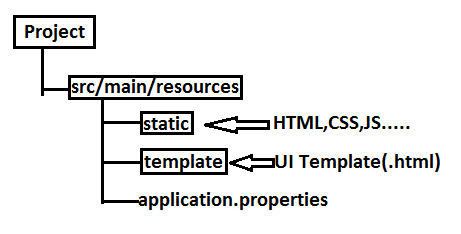


**NOTE :-**

a) Spring Boot creates Controller using New Container Style **i.e. ApplicationContext (I)**, For non Web/WebServices Applications impl class is

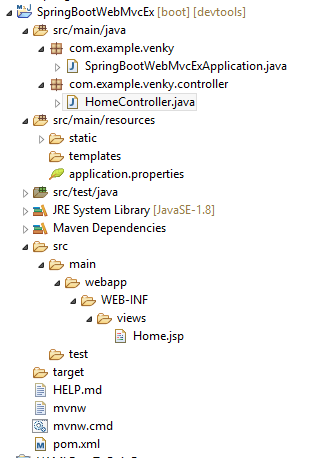
“**AnnotationConfigServletWebServerApplictaionContext”**.

b) Default Static Resource Handler is added to folder static and template which are provided under src/main/resources folder.



**Example :-**

**FolderStructure:-**



**Step#1** create new spring starterAppwith Web dependency and also add below one for JASPER ENGINE.

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.apache.tomcat.embed</groupId>

<artifactId>tomcat-embed-jasper</artifactId>

</dependency>

🡺Embedded Tomcat comes with only lightweight Servlet Engine. So, JSP Translate will not work hence show Error.

**SpringBootWebMvcExApplication.java:-**

**package** com.example.venky;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

**public** **class** SpringBootWebMvcExApplication {

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

SpringApplication.*run*(SpringBootWebMvcExApplication.**class**, args);

}

}

**Step#2** Add details in application.properties

server.port=6789

spring.mvc.view.prefix=/WEB-INF/views/

spring.mvc.view.suffix=.jsp

**Step#3** Create folder(S) /webapp/WEB-INF/views under src🡺main folder

**Step #4** Create one jsp (Home.jsp) under views

**Home.jsp:-**

<%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*

pageEncoding=*"ISO-8859-1"*%>

<%@page isELIgnored=*"false"* %>

<!DOCTYPE html>

<html>

<head>

<meta charset=*"ISO-8859-1"*>

<title>Insert title here</title>

</head>

<body>

<h2>Welcome to HomePage!!!</h2>

${msg}

</body>

</html>

**Step#5** Define Controller class

**package** com.example.venky.controller;

**import** java.util.Date;

**import** org.springframework.stereotype.Controller;

**import** org.springframework.ui.Model;

**import** org.springframework.web.bind.annotation.RequestMapping;

@Controller

@RequestMapping("/home")

**public** **class** HomeController {

@RequestMapping("/show")

**public** String showHome(Model model) {

model.addAttribute("msg",**new** Date());

**return** "Home";

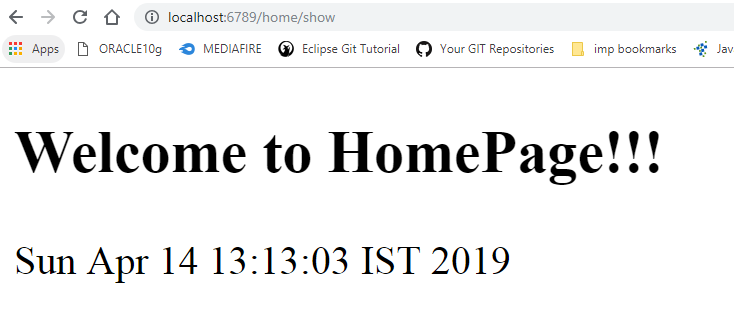
}

}

**Step#6** Run starter class and enter URL

<http://localhost:2019/home/show>

**Output:-**



**SpringBoot Web Application Working on Embedded Servers:**

By default when we add dependency **spring-boot-starter-web** then boot also provides EmbeddedDefaultServer “Tomcat” (Apache product).

🡺It works on port number 8080 by default.

🡺Embedded Tomcat is a light weight engine which contains only catalina (ServletEngine),

no JASPER(no JSP ENGINE).

🡺Two more Embedded Servers are Jetty(Eclipse[GlassFish]) and Undertow (JBOSS Product).

**## Working with Jetty and Undertow Server**

🡺create one starter Application with web dependency.

🡺create one index.html file under static folder.

🡺\*\*Exclude Tomcat first before adding any other server.

<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>

🡺Now include (add dependency for )Jetty Server

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-jetty</artifactId>

<dependency>

🡺To use Undertow server include(add dependency for)

<dependency>

<groupId>org.springframework.boot</groupId>

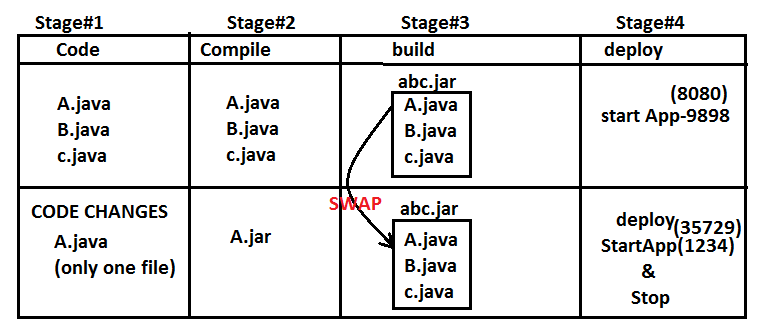
<artifactId>spring-boot-starter-undertow</artifactId>

<dependency>

**DevTools:**

🡺This is also called as DeveloperTools used at DEV environment mainly to avoid multiple re-starts of application for even small changes.

🡺Used mainly for **HOT-DEPLOYMENT** process.It means only copy files which are changed into current JAR/WAR without stopping server.



🡺This swap process is known as Hot deployment which is done using DevTools.

🡺For this process we need to add dependency

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

</dependency>

🡺DevTools will not support HotDeployment for few locations. Those are:-

**META-INF/maven/\*\* , META-INF/resources/\*\* , public/\*\* ,**

**resources/\*\*, META-INF/build-info.properties ,static/\*\* ,**

**\*\*/\*Test.class ,\*\*/\*Tests.class,git.properties, templates/\*\*.**

🡺By default all these places are under exclude locations.

(spring.devtools.restart.exclude)

🡺To enable any specific location to be Hot deployed,then add that location using **Key:**

**spring.devtools.restart.additional-paths**

Ex: spring.devtools.restart.additional-paths=static/\*\*,templates/\*\*

**Thymeleaf UI Template**

🡺Thymeleaf UI is light weight Java UI Engine used to design Dynamic web pages in web application.

🡺JSP internally servlet (heavy weight) but Thymeleaf is simple (core) java code so, compared to JSP, Thymeleaf is lightweight (less memory).

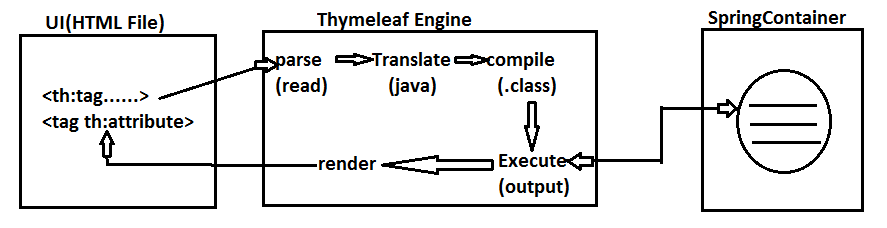
🡺Use Thymeleaf namespace in HTML file <html xmlns:th=”https://www.thymeleaf related to Thymeleaf must start with prefix ‘th’.

🡺File must be saved with .html only under src/main/resources and inside ‘template’ folder.

**Step#1:-** Write Thymeleaf tag/attribute in UI file.

**Step#2:-** Thyeleaf Engine converts it into Simple Java code.

**Step#3:-** This code will be executed by Engine only and renders output.



**Execution Flow of Thymeleaf Engine**

**#1.** Parse data from HTML (UI) file which has prefix ‘th’

(No other tags like HTML or any).

**#2.** Translate tag/attribute into equal java code.

**#3.** Compile Generated Java code (.class)

**#4.** Execute Code and communicate with Container to read or write data.

**#5.** Generated output will be placed back to UI file in place of tag/attribute known as

“**Data-Rendring”.**

**Thymeleaf Sample code**

**#a:- Link CSS file with UI file**

<link rel=”stylesheet” th:href=”@{/\_\_\_\_ /\_\_\_\_.class}”>

**#b:- Link Javascript files with UI file**

<scpript type=”text/javascript” th:src=”@{/ / .js}”</script>

**#c:- Define one form.**

<form action = “#” **th : action** = “@{/url}” **th : object** = ”${modelClassObject}”>

**#d:- Input fieId (variable) linking**

<input type=”\_\_\_\_\_“ **th : fieId** = “\*{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>

**NOTE:-**

1. Here Symbol **@** indicates URL upto contextpath. Ex: **@{/product/save}** means

http://localhost:8080/AppName/product/save

2) Symbol **${ }** indicates expression (may read data from memory)

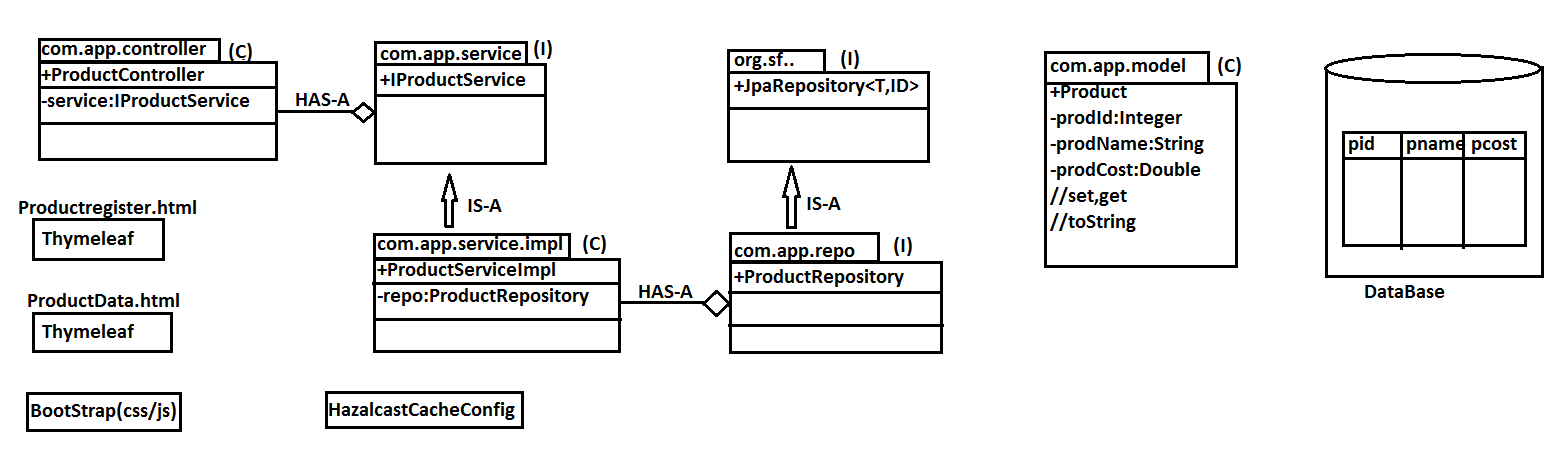
3) Symbol **\* { }** indicate pointing to variable (or any Property)

**SpringBootApplication(WebMVC+DataJPA+UI Thymeleaf+Bootstrap+HazelCastCacheConfig+MySQL DB)**

**Coding Order:-**

1. application.properties
2. Model class
3. Repository
4. IService and ServiceImpl
5. Controller
6. UI File (.html)

**Diagram:**



**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>2.1.3.RELEASE</version>

<relativePath /> <!-- lookup parent from repository -->

</parent>

<groupId>com.app</groupId>

<artifactId>SpringBoot2WebMavenCurd</artifactId>

<version>1.0</version>

<name>SpringBootWebMavenCurd</name>

<description>Demo project for Spring Boot</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-thymeleaf</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>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

</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>

**2.SpringBootWebMavenCurdApplication.java**

**package** com.app;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

**public** **class** SpringBootWebMavenCurdApplication {

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

SpringApplication.*run*(SpringBootWebMavenCurdApplication.**class**, args);

}

}

**3.ProductController**

**package** com.app.controller;

**import** **static** org.springframework.web.bind.annotation.RequestMethod.***POST***;

**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** com.app.model.Product;

**import** com.app.service.IProductService;

@Controller

@RequestMapping("/product")

**public** **class** ProductController {

@Autowired

**private** IProductService service;

//1. show Register Page

@RequestMapping("/register")

**public** String showPage(Model model) {

//set form backing object-[Form<=>Object]

model.addAttribute("product", **new** Product());

**return** "ProductRegister";

}

//2. save/update data

@RequestMapping(value="/save",method=***POST***)

**public** String saveProduct(@ModelAttribute Product product,Model model) {

//save operation

Integer prodId=service.saveProduct(product);

//clean form

model.addAttribute("product", **new** Product());

//return message to UI

model.addAttribute("message", "Product '"+prodId+"' saved");

**return** "ProductRegister";

}

//3. display all values

@RequestMapping("/all")

**public** String showAllProducts(Model model) {

List<Product> prods=service.getAllProducts();

model.addAttribute("list", prods);

**return** "ProductData";

}

//4. Delete Product based on PathVariable-id

@RequestMapping("/delete/{id}")

**public** String deleteProduct(@PathVariable Integer id,Model model) {

//delete record

service.deleteProduct(id);

//get new data and goto UI

List<Product> prods=service.getAllProducts();

model.addAttribute("message", "Product '"+id+"' Deleted");

model.addAttribute("list", prods);

**return** "ProductData";

}

//5. show Edit Page

@RequestMapping("/edit/{id}")

**public** String showEditPage(@PathVariable Integer id,Model model) {

//load Product from DB

Product p=service.getProductById(id);

//send object to UI => Form data

model.addAttribute("product", p);

**return** "ProductRegister";

}

}

**4.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;

@Entity

@Table(name="prodtab")

**public** **class** Product {

@Id

@GeneratedValue

@Column(name="pid")

**private** Integer prodId;

@Column(name="pname")

**private** String prodName;

@Column(name="pcost")

**private** Double prodCost;

**public** Product() {

**super**();

}

**public** Product(Integer prodId, String prodName, Double prodCost) {

**super**();

**this**.prodId = prodId;

**this**.prodName = prodName;

**this**.prodCost = prodCost;

}

**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;

}

@Override

**public** String toString() {

**return** "Product [prodId=" + prodId + ", prodName=" + prodName + ", prodCost=" + prodCost + "]";

}

}

**5. ProductRepository.java**

**package** com.app.repo;

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

**import** com.app.model.Product;

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

}

**6. IProductService.java**

**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 prodId);

**public** Product getProductById(Integer prodId);

**public** List<Product> getAllProducts();

}

**7. ProductServiceImpl.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** com.app.model.Product;

**import** com.app.repo.ProductRepository;

**import** com.app.service.IProductService;

@Service

**public** **class** ProductServiceImpl **implements** IProductService{

@Autowired

**private** ProductRepository repo;

@Override

**public** Integer saveProduct(Product p) {

//save or update -> same object with PK

p=repo.save(p);

Integer prodId=p.getProdId();

**return** prodId;

}

@Override

**public** **void** deleteProduct(Integer prodId) {

repo.deleteById(prodId);

}

@Override

**public** Product getProductById(Integer prodId) {

Optional<Product> p=repo.findById(prodId);

**if**(p.isPresent()) {

**return** p.get();

}**else** {

**return** **new** Product();

}

}

@Override

**public** List<Product> getAllProducts() {

List<Product> prods=repo.findAll();

**return** prods;

}

}

**8.application.properties**

##Server Config##

server.port=9988

## DataSource##

spring.datasource.driver-class-name=com.mysql.jdbc.Driver

spring.datasource.url=jdbc:mysql://localhost:3306/springboot

spring.datasource.username=root

spring.datasource.password=root

## ORM Properties ##

spring.jpa.show-sql=true

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

spring.jpa.hibernate.ddl-auto=create

**9.ProductRegister.html**

<!DOCTYPE html>

<html xmlns:th=*"http://www.thymeleaf.org/"*>

<head>

<meta charset=*"ISO-8859-1"*>

<title>Product Register</title>

<link rel=*"stylesheet"* th:href=*"@{/bootstrap/css/bootstrap.min.css}"*>

<script type=*"text/javascript"*

th:src=*"@{/bootstrap/js/bootstrap.min.js}"*></script>

<script type=*"text/javascript"* th:src=*"@{/bootstrap/js/popper.min.js}"*></script>

<script type=*"text/javascript"* th:src=*"@{/bootstrap/js/jquery.min.js}"*></script>

</head>

<body>

<!-- Menu bar starts Here -->

<nav class=*"navbar navbar-expand-sm bg-dark navbar-dark"*>

<!-- Brand/logo -->

<a class=*"navbar-brand"* href=*"#"*>

<img th:src=*"@{/image/logo.png}"* width=*"160"* height=*"60"*>

</a>

<!-- Links -->

<ul class=*"navbar-nav"*>

<li class=*"nav-item"*>

<a class=*"nav-link text-white"* th:href=*"@{/product/register}"*><h4>REGISTER</h4></a>

</li>

<li class=*"nav-item"*>

<a class=*"nav-link text-white"* th:href=*"@{/product/all}"*><h4>VIEW ALL</h4></a>

</li>

</ul>

</nav>

<!-- Menu bar ends Here -->

<div class=*"container"*>

<div class=*"card"*>

<!-- Card Head Starts Here -->

<div class=*"card-header bg-primary text-white"*> <h3>Welcome to Product Register Page</h3>

</div>

<!-- Card Head ends Here -->

<!-- Card Body Starts Here -->

<div class=*"card-body"*>

<form action=*"#"* th:action=*"@{/product/save}"* th:object=*"${product}"*

method=*"post"* class=*"form"*>

<input type=*"hidden"* th:field=*"\*{prodId}"* />

<!-- one row starts here -->

<div class=*"row"*>

<div class=*"col-4"*>

<label for=*"prodName"* class=*"control-label"*>Product Name</label>

</div>

<div class=*"col-4"*>

<input type=*"text"* th:field=*"\*{prodName}"* class=*"form-control"* />

</div>

</div>

<div class=*"row"*>

<!-- two row starts here -->

<div class=*"col-4"*>

<label for=*"prodCost"* class=*"control-label"*>Product Cost</label>

</div>

<div class=*"col-4"*>

<input type=*"text"* th:field=*"\*{prodCost}"* class=*"form-control"* />

</div>

</div>

<div class=*"row"*>

<div class=*"col-2"*></div>

<div class=*"col-4"*>

<input type=*"submit"* value=*"Register"* class=*"btn btn-success"* />

</div>

</div>

</form>

</div>

<!-- Card Body ends Here -->

<!-- Card Footer starts Here -->

<div class=*"card-footer"*>

<span th:text=*"${message}"*></span>

</div>

<!-- Card Footer ends Here -->

</div>

<!-- Card ends Here -->

</div>

<!-- container ends Here -->

</body>

</html>

**10. ProductData.html**

<!DOCTYPE html>

<html xmlns:th=*"http://www.thymeleaf.org/"*>

<head>

<meta charset=*"ISO-8859-1"*>

<title>Product Data</title>

<link rel=*"stylesheet"* th:href=*"@{/bootstrap/css/bootstrap.min.css}"*>

<script type=*"text/javascript"*

th:src=*"@{/bootstrap/js/bootstrap.min.js}"*></script>

<script type=*"text/javascript"* th:src=*"@{/bootstrap/js/popper.min.js}"*></script>

<script type=*"text/javascript"* th:src=*"@{/bootstrap/js/jquery.min.js}"*></script>

</head>

<body>

<!-- Menu bar starts Here -->

<nav class=*"navbar navbar-expand-sm bg-dark navbar-dark"*>

<!-- Brand/logo -->

<a class=*"navbar-brand"* href=*"#"*>

<img th:src=*"@{/image/logo.png}"* width=*"160"* height=*"60"*>

</a>

<!-- Links -->

<ul class=*"navbar-nav"*>

<li class=*"nav-item"*>

<a class=*"nav-link text-white"* th:href=*"@{/product/register}"*>

<h4>REGISTER</h4>

</a>

</li>

<li class=*"nav-item"*>

<a class=*"nav-link text-white"* th:href=*"@{/product/all}"*>

<h4>VIEW ALL</h4>

</a>

</li>

</ul>

</nav>

<!-- Menu bar ends Here -->

<div class=*"container"*>

<div class=*"card"*>

<div class=*"card-header bg-primary text-white"*>

<h3>Welcome to Product Data Page!!</h3>

</div>

<div class=*"card-body"*>

<table class=*"table table-bordered table-hover"*>

<tr class=*"table-info"*>

<th>ID</th>

<th>NAME</th>

<th>COST</th>

<th colspan=*"2"*>OPERATIONS</th>

</tr>

<tr th:each=*"ob:${list}"*>

<td th:text=*"${ob.prodId}"*></td>

<td th:text=*"${ob.prodName}"*></td>

<td th:text=*"${ob.prodCost}"*></td>

<td><a th:href=*"@{/product/delete/{id}(id=${ob.prodId})}"*>

<span class=*"btn btn-danger"*>DELETE</span></a></td>

<td><a th:href=*"@{/product/edit/{id}(id=${ob.prodId})}"*>

<span class=*"btn btn-success"*>EDIT</span></a></td>

</tr>

</table>

</div>

<!-- Card Body End Here -->

<div class=*"card-footer"*>

<span th:text=*"${message}"*></span>

</div>

</div>

<!-- card end here -->

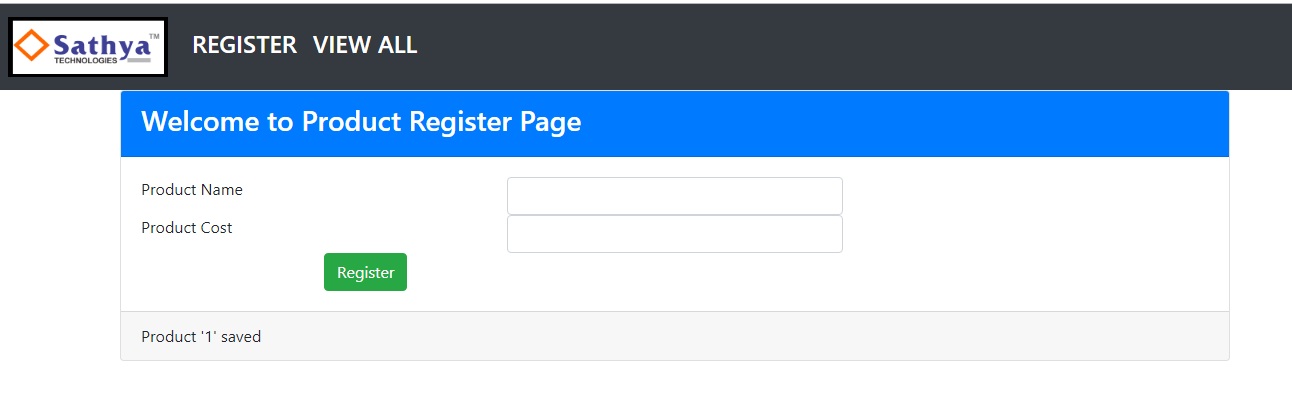
</div>

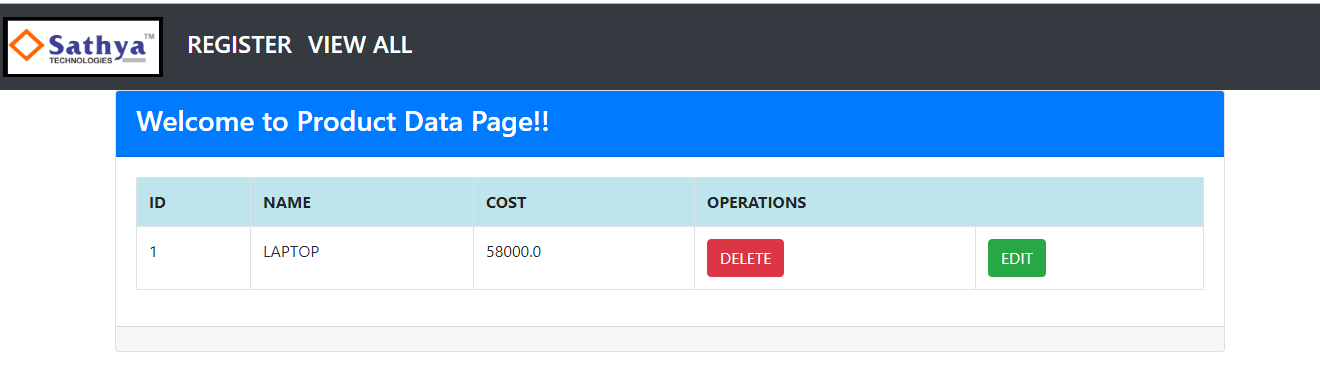
<!-- container End here -->

</body>

</html>

**11.Output:**



****

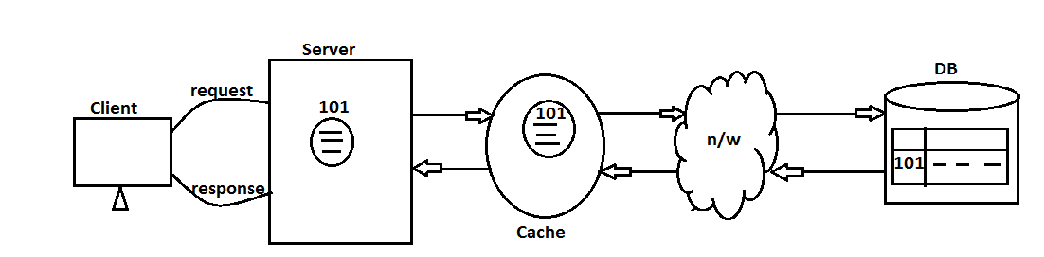
**Cache Management in Spring Boot**

🡺Cache is a temporary memory which exists at server side to hold commonly accessed data by Client.

🡺Cache concept is used to reduce network calls between Application (Server) and Database.

🡺Cache should never hold large data, apply only for few modules which are accepted more times by client.

🡺\*\*Don’t apply cache for findAll() type method.

**Design #1**

**Hazelcast-cache Config :-** It is a 3rd party Cache configuration process, supported for any java application caching.

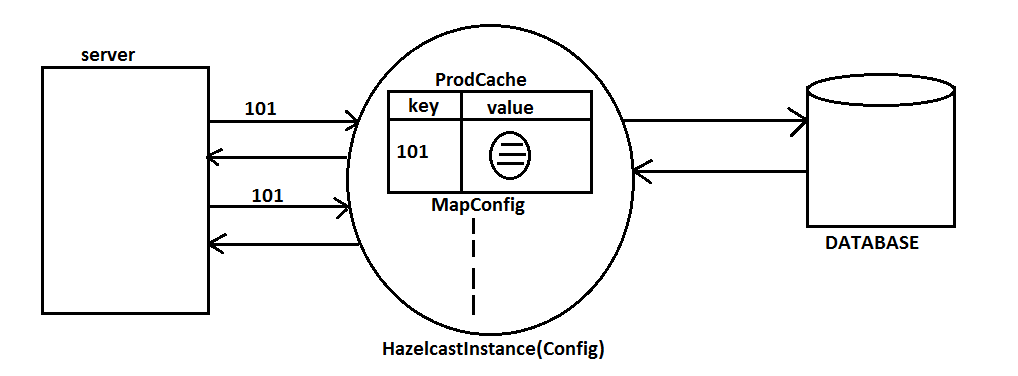
**Step#1:-** We should create Cache Memory also called as Hazelcast Instance using Class

**“Config”** (com.hazelcast).

**Step#2:-** For one module, provide one cache area known as **MapConfig**, which holds data in

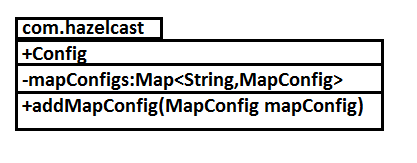
key (String-ID) and value (Object-data)

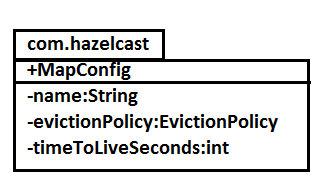
**Design#2:- Overview for product**

****

🡺In application write Java configuration code for Config (C) (com.hazelcast) and MapConfig(c)(com.hazelcast) classes

**Design#3 UML Designs:**

****

****

**Steps #1 Add Spring boot**

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-cache</artifactId>

</dependency>

<dependency>

<groupId>com.hazelcast<groupId>

<artifactId>hazelcast</artifactId>

</dependency>

<dependency>

<groupId></groupId>

<artifactId>hasalcast-spring</artifactId>

</dependency>

**Step#2** Add below annotation at starter class level, to enable /disable cache management.

**@EnableCaching**

**Step#3** Add below annotation at Service Methods Level

a) Over getOne (findOne) method

**@Cacheable(value=”cache-name”, key=”#PKId”)**

b) Over delete method (delete byId)

**@CacheEvict(vale=”cache-name”, key=”#PKId”)**

**Step#4:-** Define class for Cache Configuration using classes **Config** and **MapCOnfig**.

package com.app.config;

@Configuration

public class HazelcastCacheConfig () {

@Bean

public Config cacheConfig(){

return new Config().setInstanceName(“hazelcast-instance”)

.addMapConfig(new MapConfig()

.setNmae(“products-cache”)

.setMaxSizeCofig(new MaxSizeConfig(200,MaxSizePolicy.FREE\_HEAP\_SIZE))

.setEvictionPolicy(EvictionPolicy.LRU)

.setTimeToLiveSeconds(2000)

);

}

}

**Step#5:-** Make your class “implements Serializable” interface.

**Note:-**

**Step #1:** Create Cache memory and provide one name for that.

@Bean

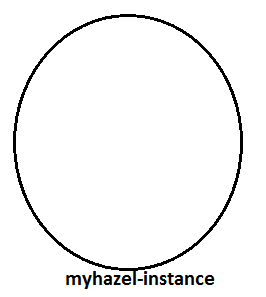
public Config cacheConfig() {

return new Config()

.setInstanceName(“myhazelcast-instance”);

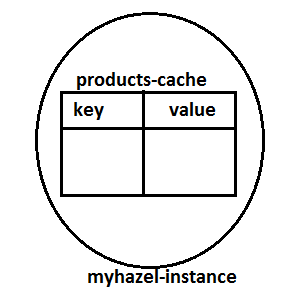
…

}



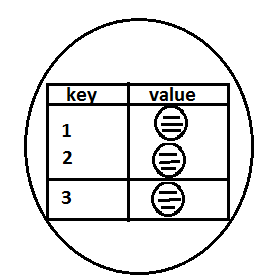
**Step #2:-** Define one MapConfig memory for one module cache.

new MapConfig().setName(“products-cache”);



**Step #3:-** Provide MAX size of cache (maximum no. of objects to be hold by cache)

setMaxSizeConfig(new MaxSizeConfig(100, MaxSizePolicy.FREE\_HEAP\_SIZE))



**Step #4:-** 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.

* **NONE:-** No Evict (No remove from cache)
* **LRU:-** Least Recently Used
* **LFU:-** Least Frequently Used
* **RANDOM:-** Any random object to be removed

**Step #5:-** Provide Life Time of object to be in cache (in second)

**setTimeToLiveSeconds (3000);**

**@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.

**@Cacheable:-** On calling select… SQL (load one row as object) same object will be placed in cache before give to Service (App).

**###Coding##**

1. Add below method in ProductController

@RequestMapping(“/one/{id}”)

public String showOnePage (@PathVariable Integer id, Model model){

System.out.println(“In Controller”);

Product p = service.getProductById(id);

model.addAttribute(“product”, p);

return “ProductOne”;

}

2. Define HTML UI (ProductOne.html)

<html xmlns:th =<http://www.thymelef.org/>>

<body>

<h3>Welcome to Product View One</h3>

<table border=”1”>

<tr>

<td>Name</td>

<td th:text=”${product.prodName}”></td>

</tr>

<tr>

<td>COST</td>

<td th:text=”${product.prodName}”</td>

</tr>

<tr>

</table></body></html>

3. Add below line in ProductData.html

<td>

<a th:href = “@{/product/one/{id}(id=${ob.prodId})}”>

<span class= “btn btn-success”>VIEW</span>

</a>

</td>

**Chapter #5 Spring Boot Batch Processing**

🡺Batch processing is used to execute length (or heavy) tasks in steps by step.

🡺Every task is called as JOB

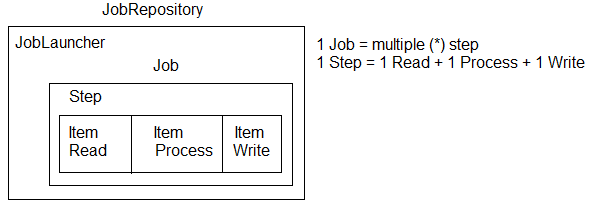
🡺One JOB can contain one or more Steps (Step can also called as SubTask)

🡺One Step contains

* 1. Item Reader (Read data from Source)
  2. Item Process (Do calculations and logics/operations etc..)
  3. Item writer (Provide output to next Step or final output)

🡺JOBs are invoked by JobLauncher is 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

* 1. ItemReader<T>
  2. ItemProcessor<I, O>
  3. ItemWriter<T>.

**a. ItemReader<T> :-** It is used to read data from source as input to current Step.

**b. ItemProcessor<T>:-** It is used to process input data given by Reader and returns in

Modified(or same) format of Data.

**c. ItemWriter<T> :-** It is used to write data to any Destination (ex: DB, Text File, CSV,

EXCEL, XML…etc).

**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 do some process (calculate, convert, check conditions, convert to any class Object etc..)

5. ItemProcessor will provide output (may be same as Input type) which is called as Transformed Type.

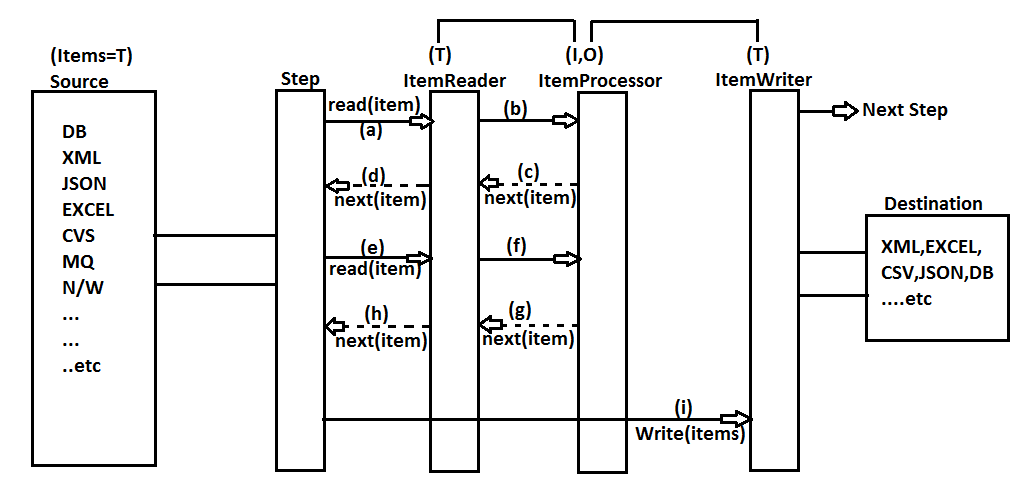
6. ItemWriter will collect all output Items into one List type from Processor at a time (Only one time).

7. ItemWriter writes data to any destination.

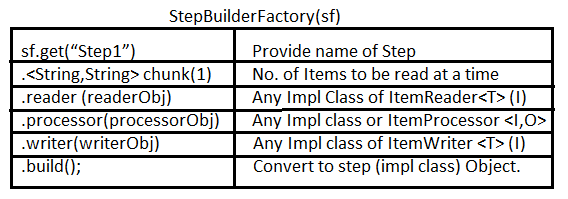
8. Source/Destinatation can be Text, DB, Network, MessageQueue, Excel, CSV, JSON data, XML etc…

**Execution Sequence of Step**

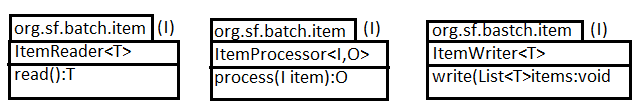
**Design:-**

****

**Step:-** One **Step** can be constructed using StepBuilderFactory (sf) class by providing name, chunk size, reader, processor and writer.



**UML Notation :-**



**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.

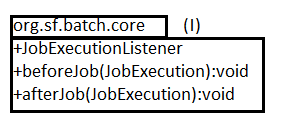
**a. beforeJob (JobExecution): void**

**b. afterJob (JobExecution): void**

**🡺**JobExecution is a class which is used to find current job details like jobParameters, BatchStatus, stepExecutions etc…

🡺\*\*\*BatchStatus is an **enum** having possible values:

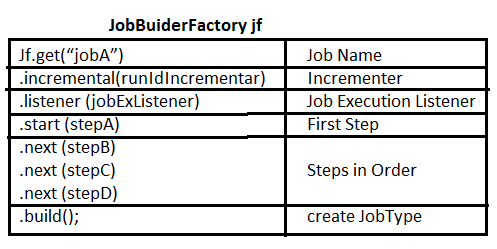
COMPLETED, STARTING, STARTED, STOPPING, STOPPED, FAILED, ABANDONED, UNKNOWN.



**JobBuilderFactory (C)**

🡺This class is used to create one or more Jobs using **Steps, listener, Incrementer**….

🡺Job (I) Construction flow.



**JobLauncher (I)**

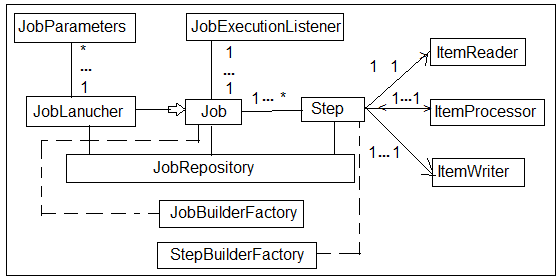
This interface is used to invoke our Job with Parameters (input) like creationTime, uniqueId(name), programmerData etc.

🡺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 Processing Implementation Diagram :-**



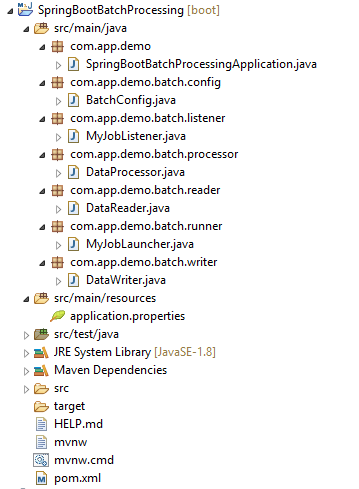
**Coding order:-**

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)

**FolderStructure:**



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>2.1.4.RELEASE</version>

<relativePath /> <!-- lookup parent from repository -->

</parent>

<groupId>com.venky</groupId>

<artifactId>SpringBootBatchProcessing</artifactId>

<version>1.0</version>

<name>SpringBootBatchProcessing</name>

<description>Demo project for Spring Boot</description>

<properties>

<java.version>1.8</java.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-batch</artifactId>

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.springframework.batch</groupId>

<artifactId>spring-batch-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>

1. **SpringBootBatchProcessingApplication.java**

**package** com.app.demo;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

**public** **class** SpringBootBatchProcessingApplication {

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

SpringApplication.*run*(SpringBootBatchProcessingApplication.**class**, args);

}

}

1. **BatchConfig.java**

**package** com.app.demo.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.demo.batch.listener.MyJobListener;

**import** com.app.demo.batch.processor.DataProcessor;

**import** com.app.demo.batch.reader.DataReader;

**import** com.app.demo.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();

}

}

1. **MyJobListener.java**

**package** com.app.demo.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());

}

}

1. **DataProcessor.java**

**package** com.app.demo.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();

}

}

1. **DataReader.java**

**package** com.app.demo.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","how r u"};

**int** index;

@Override

**public** String read()

**throws** Exception, UnexpectedInputException,

ParseException, NonTransientResourceException {

**if**(index<message.length) {

**return** message[index++];

}**else** {

index=0;

}

**return** **null**;

}

}

1. **MyJobLauncher.java**

**package** com.app.demo.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);

}

}

1. **DataWriter.java**

**package** com.app.demo.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);

}

}

}

1. **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

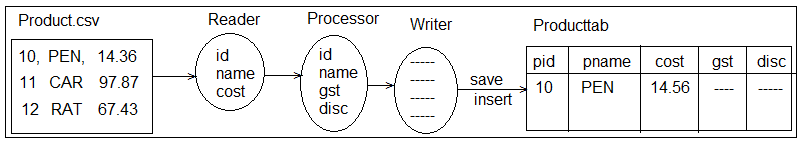
**Spring Boot Batch Processing : Converting .csv file data to DataBase Table**

🡺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.



**\*\*\*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 Test {

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();

}

};

**FlatFileItemReader<T> :**

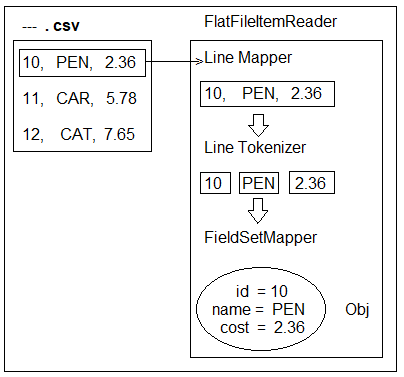
This class is provided by Spring Batch to read data from any text Related file to read data from any Text Related file (.txt,.csv,….).

**DefaultLineMapper:-** It will load one row(\n) at a time as “**one Line Object**”.

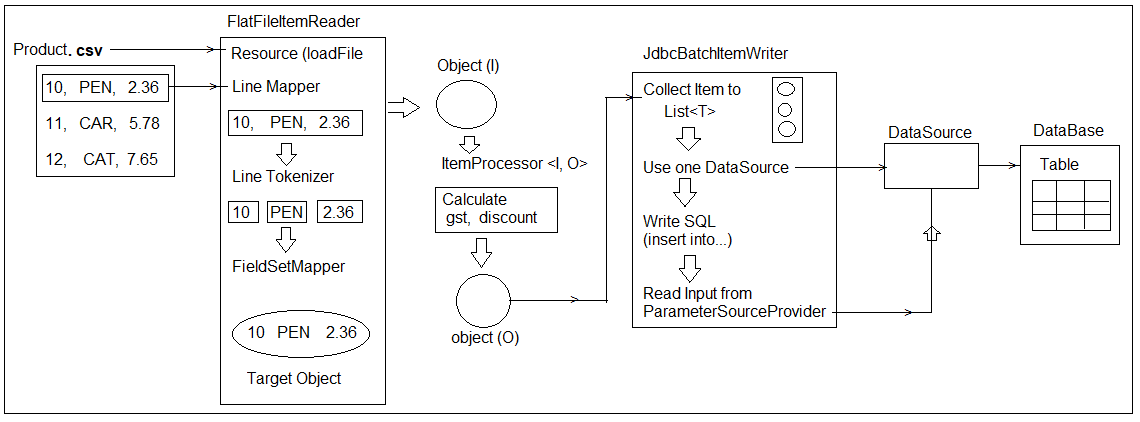
**DelimitedLineTokenizer:-** It will divided one line values into multiple values using any Delimiter (like comma, space, tab, new, line, dash.. etc).

🡺Default Delimiter is “,” [Comma].

**BeanWrapperFieldSetMapper:-** It maps data to variables, finally converted to one class type (TargetType).



**Diagram:-**

****

**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 (Delimitar = ,).

🡺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.

**Task:-**

**#1** Write Spring boot Batch Application To read data from database (Oracle DB) using “JdbcCursorItemReader” and write data to csv file using “FlatFIleItemWriter”.

**#2** Write data from MongoDB using “MongoItemReader” and write data to JSON file using “JsonFIleItemWriter”.

**\*\* Code snippet for CSV to ORACLE DB :-**

@Bean

**public** ItemReader<Product> reader() {

FlatFileItemReader <Product> reader = **new** FlatFileItemReader<Product>();

Reader.setResource(**new** ClassPathResource("myprods.csv"));

Reader.setLineMapper(**new** DefaultLineMapper<Product>(){{

setLineTokenizer (**new** DelimitedLineTokenizer() { {

setNames ("prodId", "prodName", "prodCost");

}});

setFieldSetMapper(**new** BeanWrapperFieldSetMapper<Product>(){{

setTargetType (Product.**class**);

}});

}});

**return** reader;

}

@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;

}

**🡪DB Table:-**

CREATE TABLE prodstab(PID **int**(10), PNAME varchar2(50), PCOST number, PGST number, PDISC number)

**🡪 product.csv : under src/main/Resources.**

10, PEN, 2.36

11, CAR, 8.8

12, BUS, 99.56

13,TRAIN,232.67

🡺To read file from outside Application from File System (D:/driver or C:/driver) then use FileSystemResource.

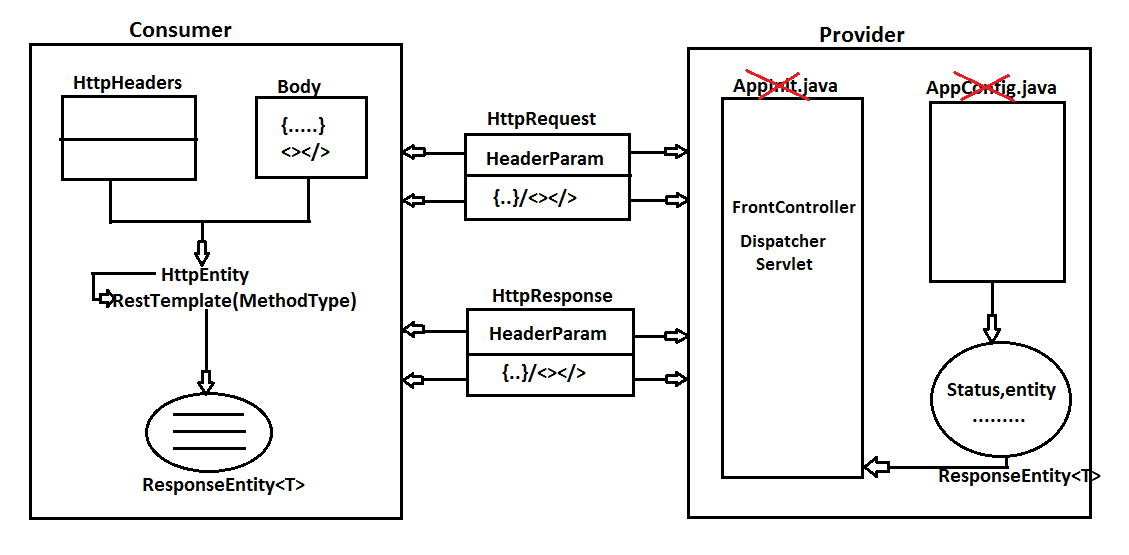
🡺It same way to read file from network (internet URL based) then use UrlResource in place of ClassPathResource.

**Chapter #6 Spring Boot Rest (Provider & Consumer)**

🡺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.

**Diagram**



**NOTE:-**

a) Consumer and Provider inter changes 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 | @GetMapping |
| POST | @PostMapping |
| PUT | @PutMapping |
| DELETE | @DeleteMapping |
| PATCH | @PatchMapping |

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.properties”)

k) Embedded server: Tomcat, default port: 8080

**#1. Spring Boot ReST Provider Example**

**Step#1:-** Create Starter Project using web and devtools Dependencies.

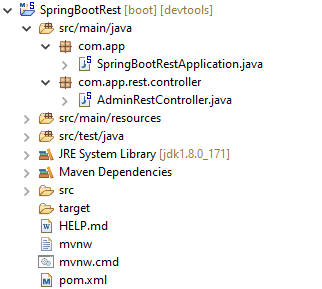
**Details:-**

GroupId : com.app

ArtifactId : SpringBootRestProvider

Version : 1.0

**Folder Structure of Spring Boot RestController with possible HttpRequest :-**



**Step #2:-** Writer one Controller class with class level URL different method with HTTP MethodTypes.

**AdminRestController.java**

**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:-**

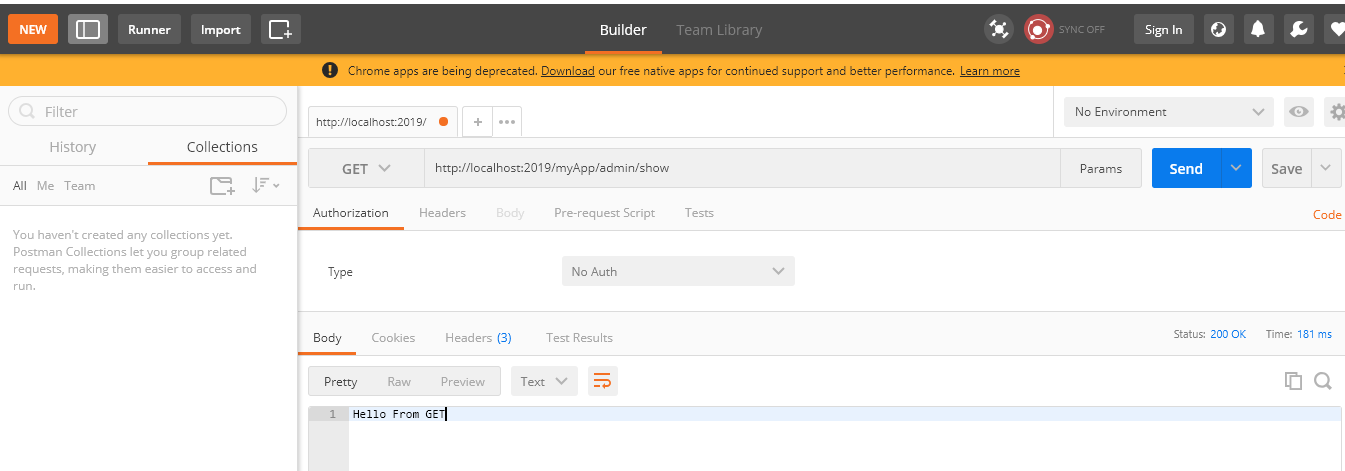
server.port=2019

server.servlet.context-path=/myApp

\*\*\* Run Starter class and Test Application using POSTMAN

<http://localhost>: 2019/ myapp /admin/show

**Postman Screen :-**

****

**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 #3 Read Data from Http Request (Head and Body)**

🡺Http Request sends data using Header and body Section (both).

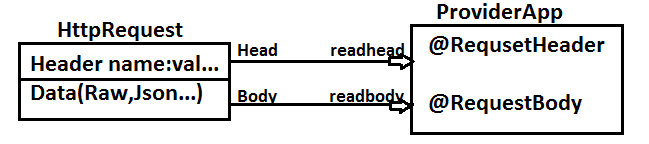
🡺To Read any Header Parameter use code

**a. @RequestHeader DataType localVariable**

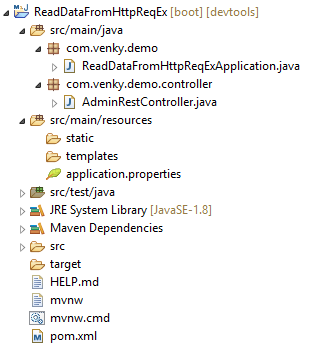
**b. @RequestHeader (required=false) DataType localVariable**

**c. @RequestHeader (“key”) DataType localVariable**

**🡺**To read data from Body (RAW Data) use code @RequestBody



**FolderStructure:**

****

**ReadDataFromHttpReqExApplication.java**

**package** com.venky.demo;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

**public** **class** ReadDataFromHttpReqExApplication {

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

SpringApplication.*run*(ReadDataFromHttpReqExApplication.**class**, args);

}

}

**AdminRestController.java**

**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)

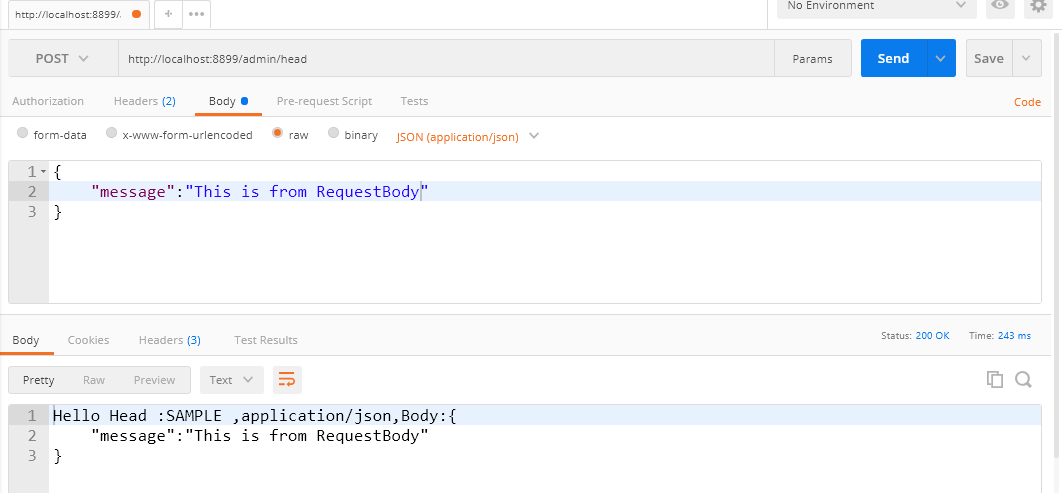
{

**return** "Hello Head :"+dept+" ," +type+ ",Body:" +mydata;

}

}

**Output:**

****

**NOTE:-**

1. Request Header is required (true) by default to make it optional, add code required=false.
2. If key name localVariable Name same then providing key name is optional.
3. Request Body Raw data (Characters or any…).
4. Can be Stored in String with any variableName.

**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:-**

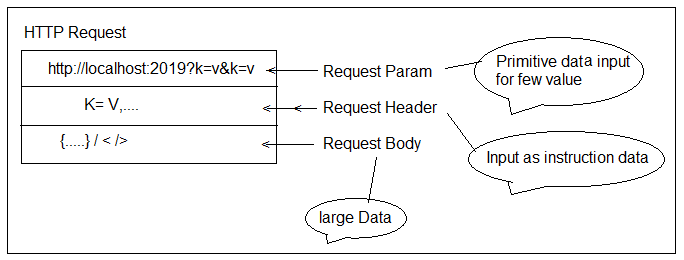
1. Request Header

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…

1. **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).

1. **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: @RequestParam("sname") String sn

**Syntax #2**

**@RequestParam DataType localVar**

Ex: @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

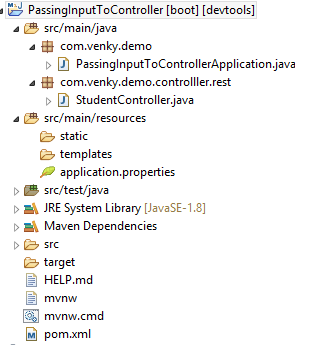
**Syntax #4**

**@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**

**FolderStructure**

****

**PassingInputToControllerApplication.java**

**package** com.venky.demo;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

**public** **class** PassingInputToControllerApplication {

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

SpringApplication.*run*(PassingInputToControllerApplication.**class**, args);

}

}

**StudentController.java**

**package** com.venky.demo.controlller.rest;

**import** org.springframework.web.bind.annotation.GetMapping;

**import** org.springframework.web.bind.annotation.RequestParam;

**import** org.springframework.web.bind.annotation.RestController;

@RestController

**public** **class** StudentController {

@GetMapping("/show")

**public** String showMsg(@RequestParam (value="sname", required=**false**,

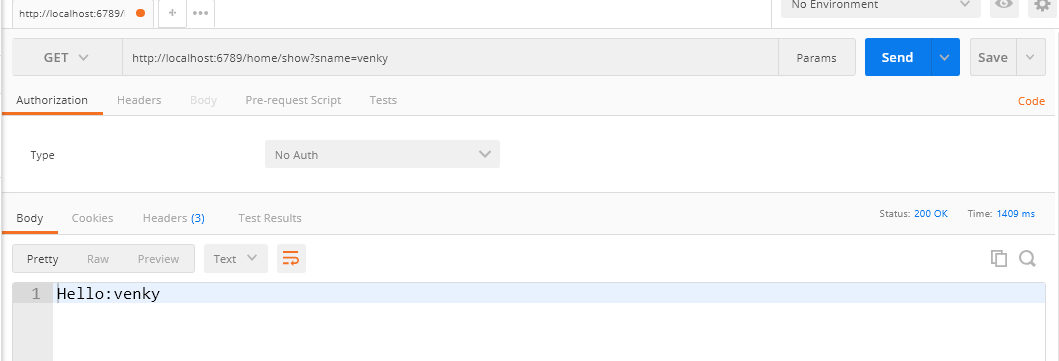
defaultValue="NO DATA") String sid){

**return** "Hello:" +sid;

}

}

**Output:**

****

**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

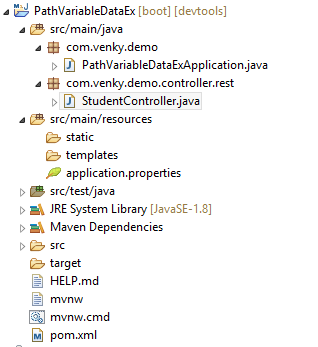
🡺Order must be followed in case of sending multiple Parameters.

🡺To read data at controller method, use annotation : **@PathVariable.**

**Syntax is:**

**@PathVariable datatype keyName**

**FolderStructure**

****

**PathVariableDataExApplication.java**

**package** com.venky.demo;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

**public** **class** PathVariableDataExApplication {

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

SpringApplication.*run*(PathVariableDataExApplication.**class**, args);

}

}

**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){

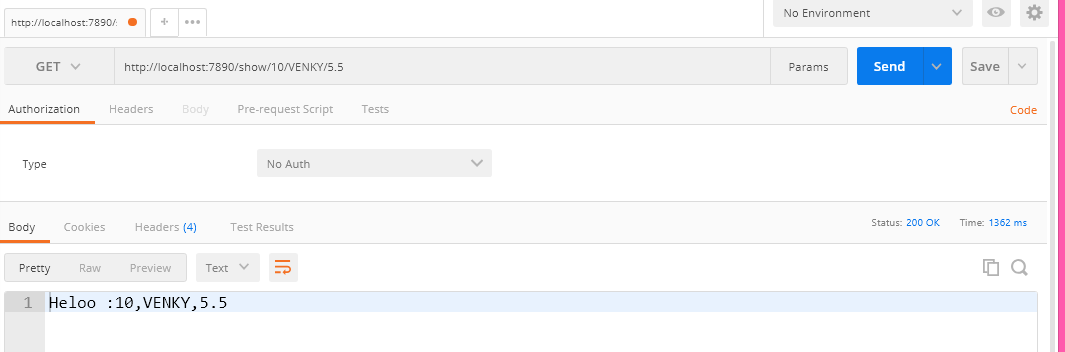
**return** "Heloo :" +sid+","+sname+ ","+sfee;

}

}

Example URL :- <http://localhost:2019/show/10/Venky/56.8>

**Output:**

****

**Rest Controller : 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.

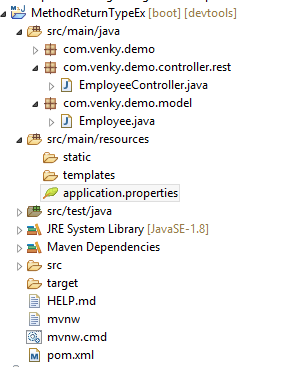
<dependency>

<groupId>com.fasterxml.jackson.dataformat</groupId>

<artifactId>jackson-dataformat-xml</artifactId>

</dependency>

**# 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 empId;

**private** String empName;

**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 + "]";

}

}

**Step #4 Controller class :-**

**package** com.venky.demo.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.venky.demo.model.Employee;

@RestController

**public** **class** EmployeeController

{

@GetMapping("/showA")

**public** String showA() {

**return** "Hello-String";

}

@GetMapping("/showB")

**public** Employee showB() {

**return** **new** Employee(22, "VENKY", 3.8);

}

@GetMapping("/showC")

**public** List<Employee> showC() {

**return** Arrays.*asList*(**new** Employee(22, "venky", 63.8),

**new** Employee(23, "srinath", 65.8),

**new** Employee(24, "Raju", 69.8)

);

}

@GetMapping("/showD")

**public** Map<String, Employee> showD() {

Map<String, Employee> map= **new** HashMap<>();

map.put("e1", **new** Employee(22, " venky ", 4.6));

map.put("e2", **new** Employee(23, " srinath ", 8.2));

map.put("e3", **new** Employee(24, "Raju", 9.5));

**return** map;

}

@GetMapping("/showE")

**public** ResponseEntity<String> showE() {

ResponseEntity<String> resp = **new** ResponseEntity<String> ("Hello RESPONSEENTITY ", HttpStatus.***OK***);

**return** resp;

}

}

🡺Run the application and enter below urls one by one and show output.

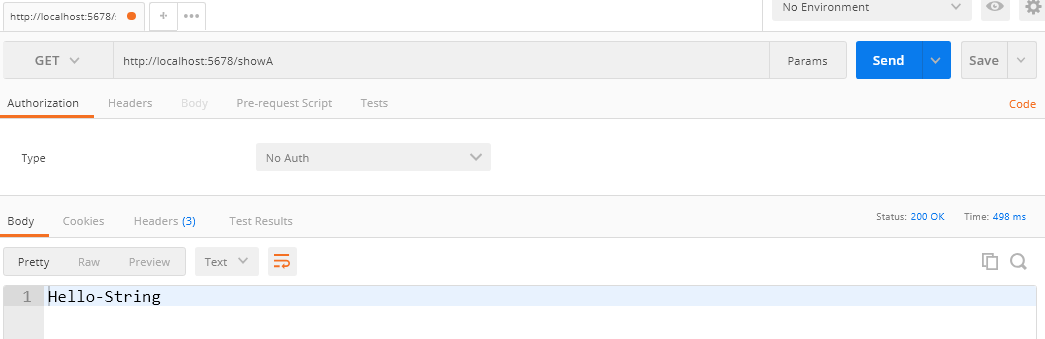
1) <http://localhost:2019/showA>

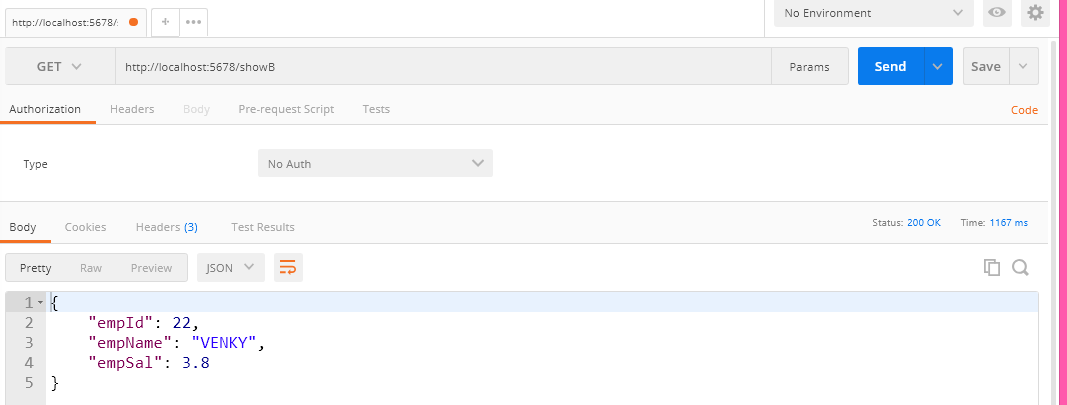
2) <http://localhost:2019/showB>

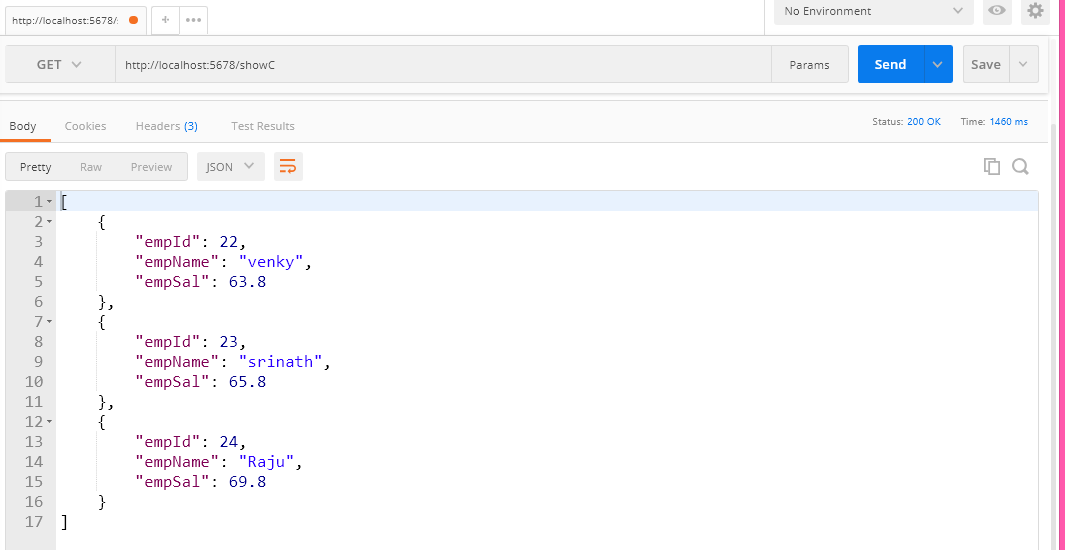
3) http://localhost:2019/showC

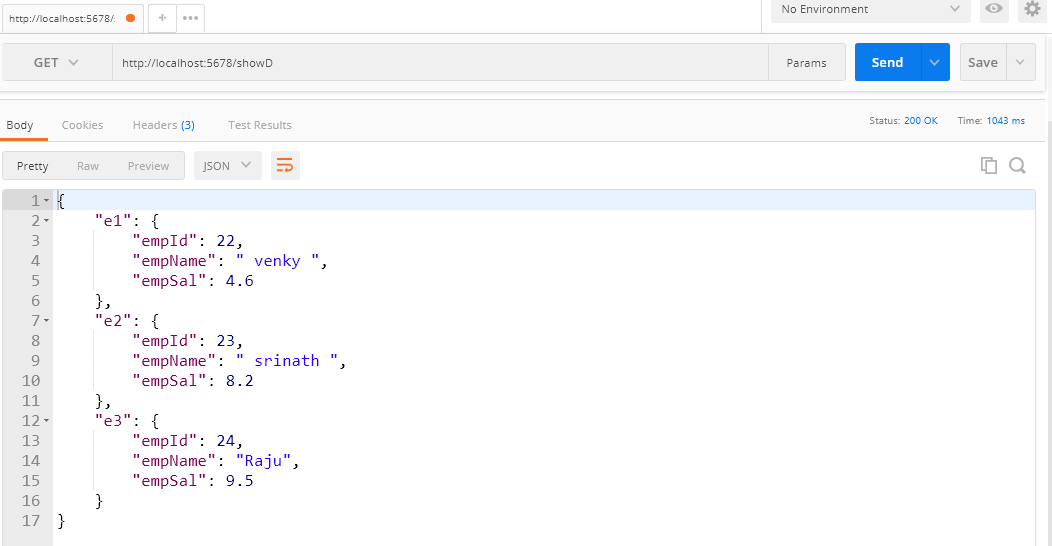
4) http://localhost:2019/showD

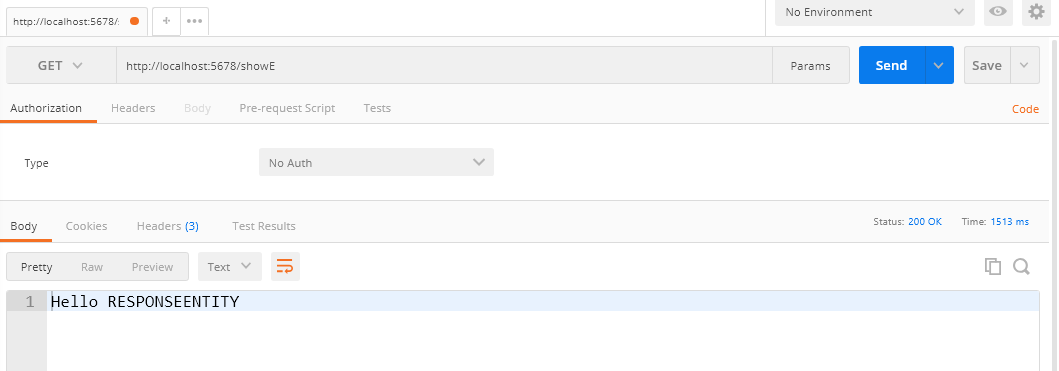
**Output:**











**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 |

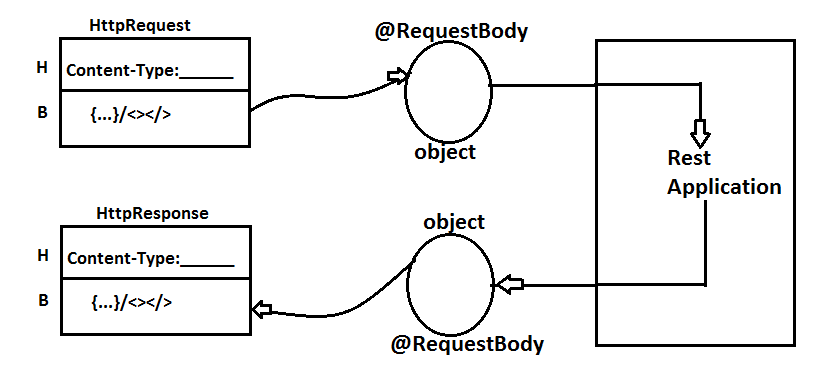
**MediaType Conversion annotations are:-**

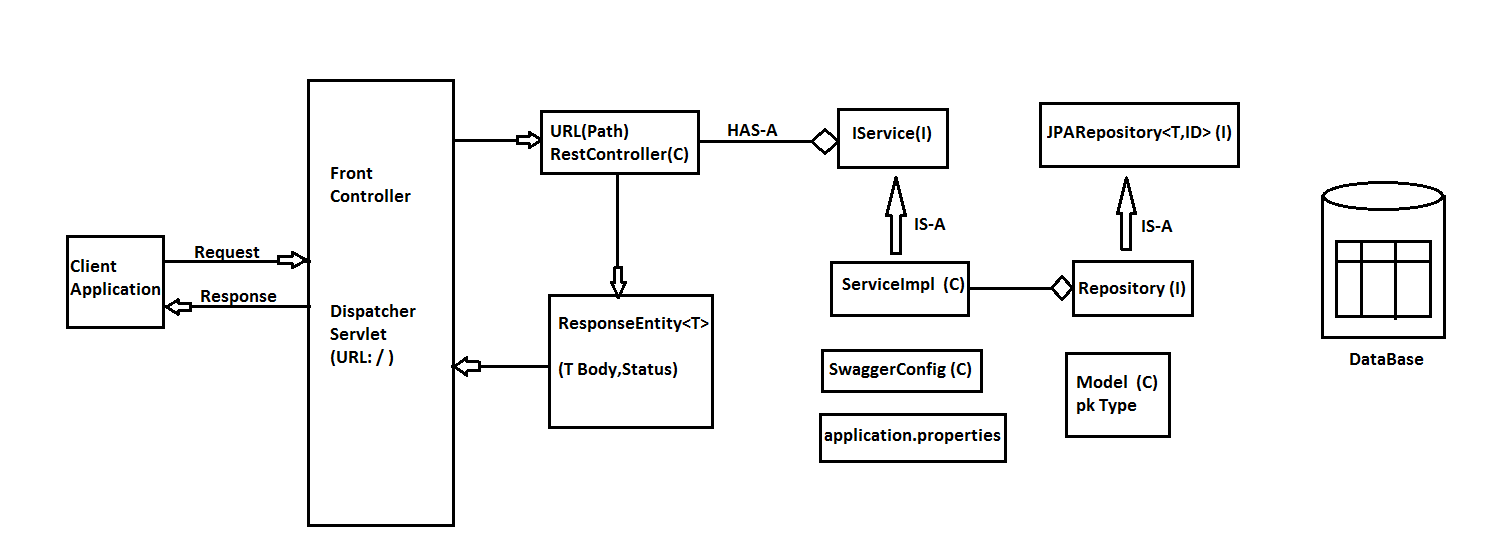
🡺@RequestBody and @ResponseBody the MediaType annotations

🡺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 :-**





\*\*\* Click on show whitespace character symbol (looks like PI symbol :

**Coding Order :-**

#1 Create Project with web, Jpa, devtools, mysql dependencies

#2 add jackon-dataformat-xml in pom.xml

#3 Define Model class, Repository, IService and ServiceImpl in order.

🡺Student.java = Model

🡺StudentRepository.java = Repository

🡺IStudentService.java = IService

🡺StudentServiceImpl.java = Impl

#4 In application.properties provide keys details server port, datasource and jpa (dialect, show-sql…)

#5 Define RestController

**EnableSwaggerUI in SpringBootRestApplication**

Compared to all other API tools Swagger is a RichSetAPI provides dynamic UI based on code written for RestController 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>

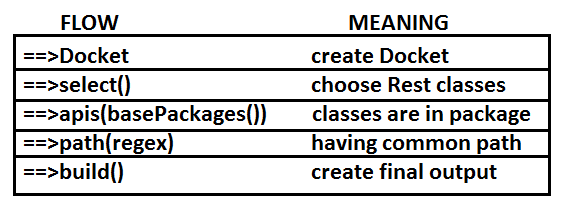
<dependency>

<groupId>io.springfox</groupId>

<artifactId>springfox-swagger-ui</artifactId>

<version>2.7.0</version>

</dependency>

****

**Step#2** Define Swagger Configuration class in application

**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.spi.DocumentationType;

**import** springfox.documentation.spring.web.plugins.Docket;

**import** springfox.documentation.swagger2.annotations.EnableSwagger2;

@Configuration

@EnableSwagger2

**public** **class** SwaggerConfig {

@Bean

**public** Docket myApi() {

**return** **new** Docket(DocumentationType.***SWAGGER\_2***)

.select()

.apis(*basePackage*("com.app.controller.rest"))

.paths(*regex*("/rest.\*"))

.build()

;

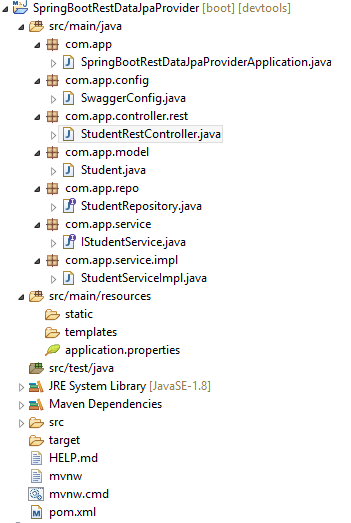
}

}

\*\*\* basePackages() is a static method defined in **“RequestHandlerSelector(C)”** and in same way regex() is a static method defined in **”PathSelectors(C)”**

**Step#3** Run Starter class and enter URL: http://localhost:8899/swagger-ui.html.

**FolderStructure:**



**SpringBootRestDataJpaProviderApplication.java**

**package** com.app;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

**public** **class** SpringBootRestDataJpaProviderApplication {

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

SpringApplication.*run*(SpringBootRestDataJpaProviderApplication.**class**, args);

}

}

**SwaggerConfig.java**

**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.spi.DocumentationType;

**import** springfox.documentation.spring.web.plugins.Docket;

**import** springfox.documentation.swagger2.annotations.EnableSwagger2;

@Configuration

@EnableSwagger2

**public** **class** SwaggerConfig {

@Bean

**public** Docket myApi() {

**return** **new** Docket(DocumentationType.***SWAGGER\_2***)

.select()

.apis(*basePackage*("com.app.controller.rest"))

.paths(*regex*("/rest.\*"))

.build();

}

}

**StudentRestController.java**

**package** com.app.controller.rest;

**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.Student;

**import** com.app.service.IStudentService;

@RestController

@RequestMapping("/rest/student")

**public** **class** StudentRestController {

@Autowired

**private** IStudentService service;

//1. save student data

@PostMapping("/save")

**public** ResponseEntity<String> save(@RequestBody Student student){

ResponseEntity<String> resp=**null**;

**try** {

Integer id=service.saveStudent(student);

resp=**new** ResponseEntity<String>("Student '"+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<Student> list=service.getAllStudents();

**if**(list==**null** || list.isEmpty()) {

String message="No Data Found";

resp=**new** ResponseEntity<String>(message,HttpStatus.***OK***);

}**else** {

resp=**new** ResponseEntity<List<Student>>(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.deleteStudent(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 Student student){

ResponseEntity<String> resp=**null**;

//check for id exist

**boolean** present=service.isPresent(student.getStdId());

**if**(present) { //if exist

service.updateStudent(student);

resp=**new** ResponseEntity<String>("Updated Successfully",HttpStatus.***OK***);

}**else** { //not exist

resp=**new** ResponseEntity<String>("Record '"+student.getStdId()+"' not found",HttpStatus.***BAD\_REQUEST***);

}

**return** resp;

}

}

**Student.java**

**package** com.app.model;

**import** javax.persistence.Entity;

**import** javax.persistence.GeneratedValue;

**import** javax.persistence.Id;

**import** javax.xml.bind.annotation.XmlRootElement;

@Entity

@XmlRootElement

**public** **class** Student {

@Id

@GeneratedValue

**private** Integer stdId;

**private** String stdName;

**private** Double stdFee;

**public** Student() {

**super**();

}

**public** Student(Integer stdId, String stdName, Double stdFee) {

**super**();

**this**.stdId = stdId;

**this**.stdName = stdName;

**this**.stdFee = stdFee;

}

**public** Integer getStdId() {

**return** stdId;

}

**public** **void** setStdId(Integer stdId) {

**this**.stdId = stdId;

}

**public** String getStdName() {

**return** stdName;

}

**public** **void** setStdName(String stdName) {

**this**.stdName = stdName;

}

**public** Double getStdFee() {

**return** stdFee;

}

**public** **void** setStdFee(Double stdFee) {

**this**.stdFee = stdFee;

}

@Override

**public** String toString() {

**return** "Student [stdId=" + stdId + ", stdName=" + stdName + ", stdFee=" + stdFee + "]";

}

}

**StudentRepository.java**

**package** com.app.repo;

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

**import** com.app.model.Student;

**public** **interface** StudentRepository **extends** JpaRepository<Student, Integer>{

}

**IStudentService.java**

**package** com.app.service;

**import** java.util.List;

**import** java.util.Optional;

**import** com.app.model.Student;

**public** **interface** IStudentService {

**public** Integer saveStudent(Student s);

**public** **void** updateStudent(Student s);

**public** **void** deleteStudent(Integer id);

**public** Optional<Student> getOneStudent(Integer id);

**public** List<Student> getAllStudents();

**public** **boolean** isPresent(Integer id);

}

**StudentServiceImpl.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.Student;

**import** com.app.repo.StudentRepository;

**import** com.app.service.IStudentService;

@Service

**public** **class** StudentServiceImpl **implements** IStudentService {

@Autowired

**private** StudentRepository repo;

@Transactional

**public** Integer saveStudent(Student s) {

**return** repo.save(s).getStdId();

}

@Transactional

**public** **void** updateStudent(Student s) {

repo.save(s);

}

@Transactional

**public** **void** deleteStudent(Integer id) {

repo.deleteById(id);

}

@Transactional(readOnly = **true**)

**public** Optional<Student> getOneStudent(Integer id) {

**return** repo.findById(id);

}

@Transactional(readOnly = **true**)

**public** List<Student> getAllStudents() {

**return** repo.findAll();

}

@Transactional(readOnly = **true**)

**public** **boolean** isPresent(Integer id) {

**return** repo.existsById(id);

}

}

**application.properties**

##Server Config ##

server.port=9898

## DataSource Config ##

spring.datasource.driver-class-name=com.mysql.jdbc.Driver

spring.datasource.url=jdbc:mysql://localhost:3306/springboot

spring.datasource.username=root

spring.datasource.password=root

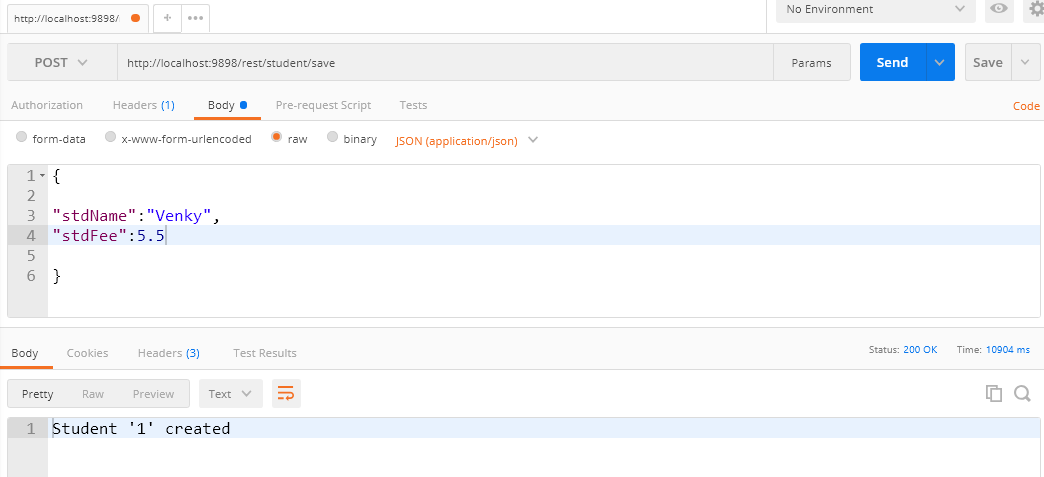
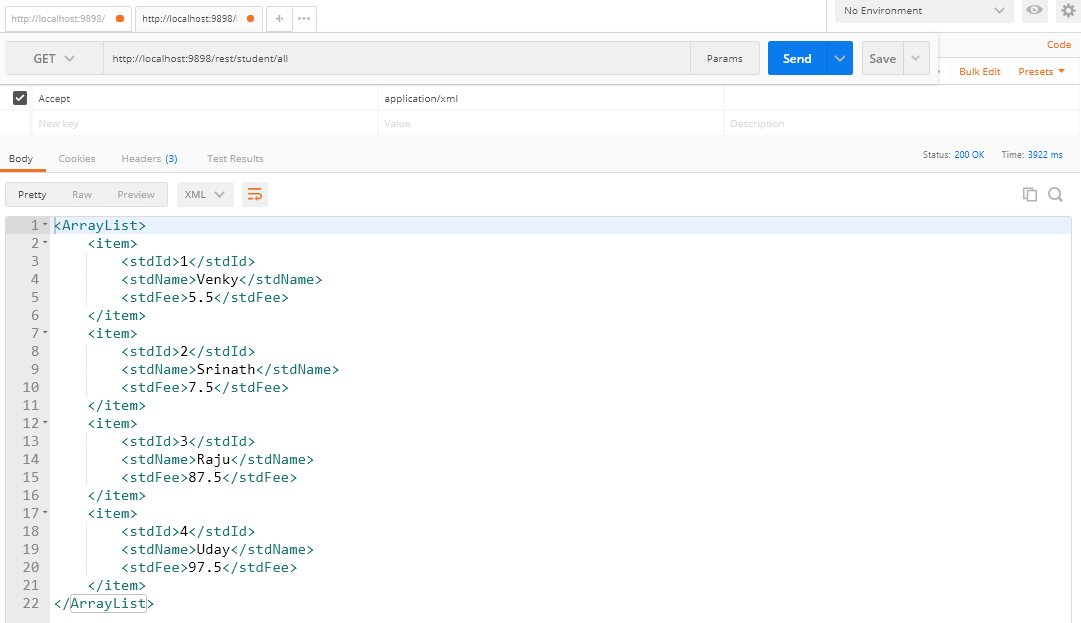
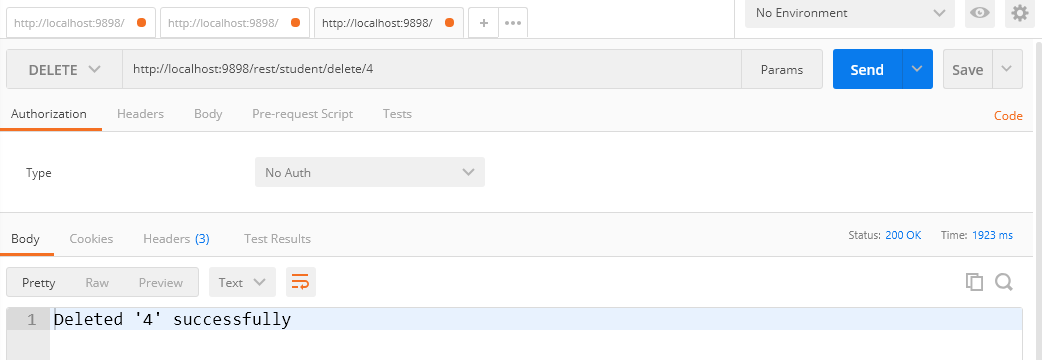
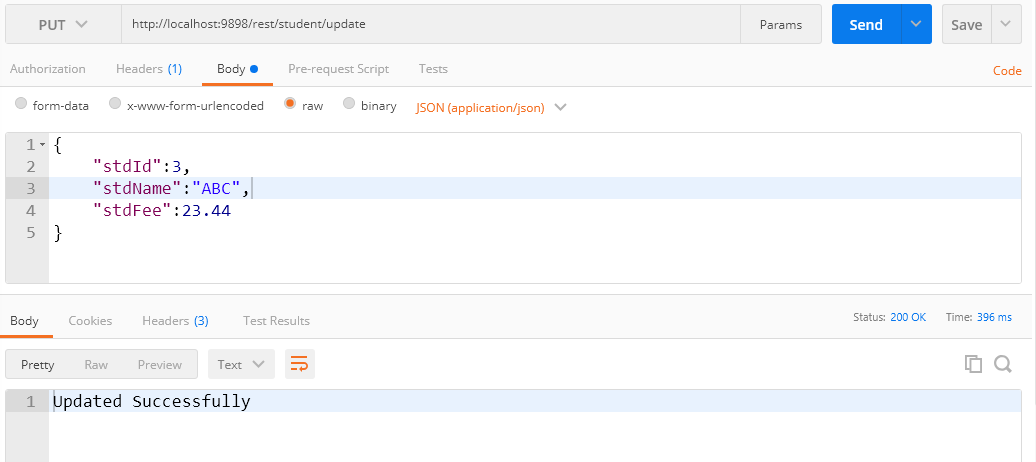
## Hibernate COnfig ##

spring.jpa.show-sql=true

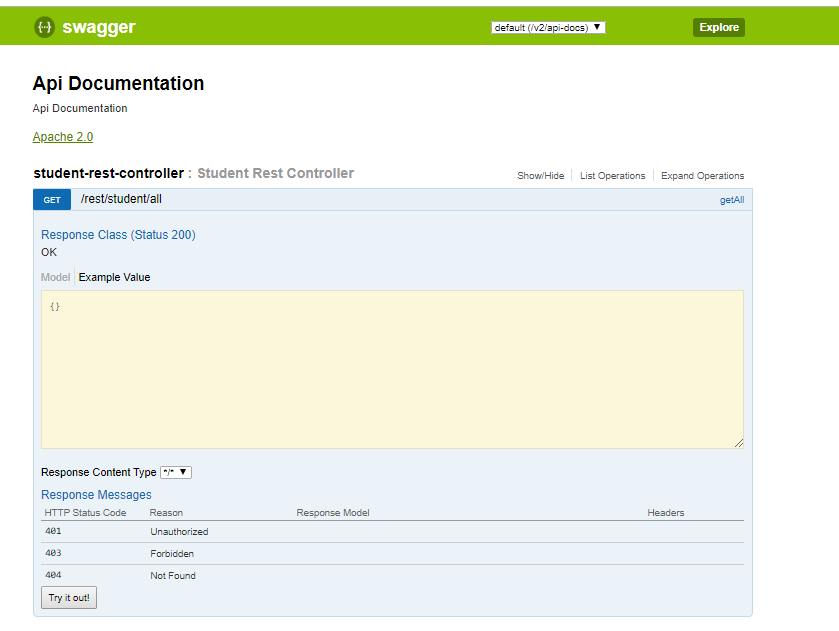
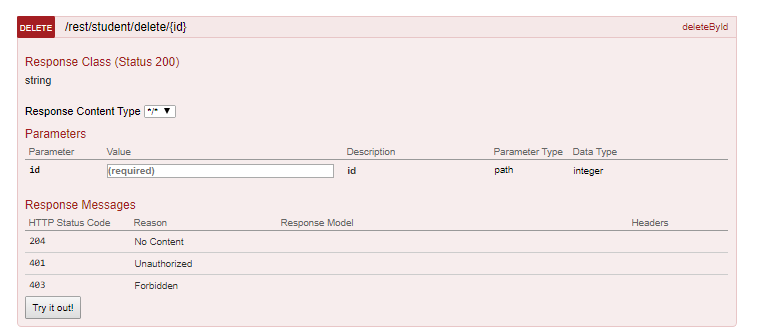
spring.jpa.hibernate.ddl-auto=create

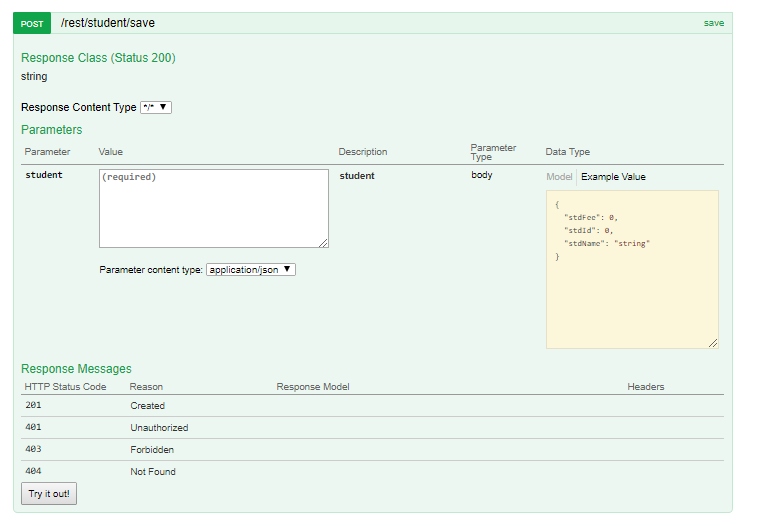
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

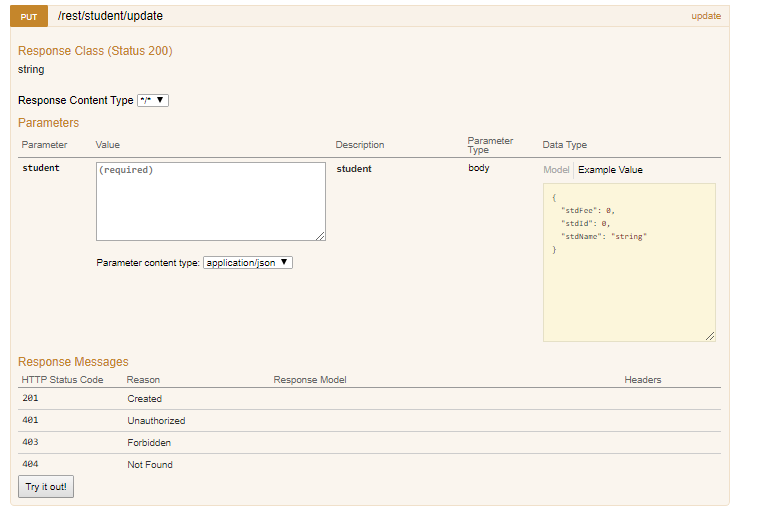
**Output**

**Swagger Output**





|  |  |
| --- | --- |
| **Chapter #1 SPRING CORE** |  |
| **1. Spring Boot Introduction**  **2. Spring Boot Runner**  **3. application.properties**  **4. yml**  **5. Profile**  **6. pom.xml**  **7. Starter class**  **8. Spring Initializer** |  |

|  |  |
| --- | --- |
| **Chapter #2 SPRING BOOT DATA JPA** |  |
| 1. **Query Method** |  |

**Chapter #3 Mango DB**

**Chapter #4 Spring Boot MVC**

1. **Thmeleaf**

Create GIT Account :-

Follow (me): <https://github.com/javabyraghu>

**Chapter #5 Spring Boot Batch**

**Chapter #6 Spring Boot Rest**

**6.1> Spring Boot ReST Provider**

**Path Variable**