SPRING BOOT

**AND**

MICROSERVICE

**BY**

**Kumar4Java**

**SPRING BOOT**

1. **Spring Boot :--**

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

=>Available versions of Spring Boot are a>Spring Boot 1.x.

b>Spring Boot 2.x.

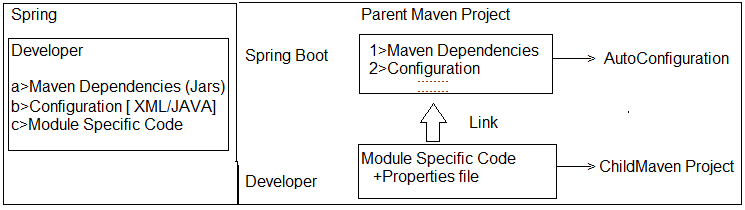
c>Spring Boot 3.x

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

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

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

a>Properties File (application.properties). b>YAMAL File (application.yml).



2>Spring Boot:-- a>application.yml:--

server.port: 8082

spring:

datasource:

url: jdbc:mysql://${MYSQL\_HOST:localhost}:3306/db

username: user

password: password

jpa:

hibernate.ddl-auto: update

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

## <dependency>

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

## <artifactId>spring-boot-starter-data-jpa</artifactId>

## </dependency>

## <dependency>

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

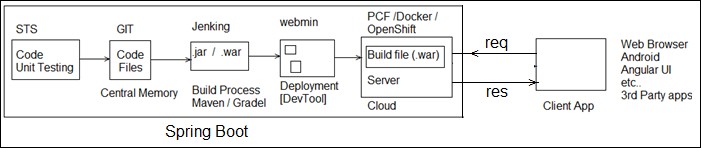
## <artifactId>spring-boot-starter-web</artifactId>

## </dependency>

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

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

1. >GIT (github.com) is used to store our code files. It is called as **Central Repository or version Control Tool.**
2. >.Java is converted to .class (Compile) .class + (other files .xml, .html…) converted to .jar/.war finally (build process).
3. >Place .jar/.war in server and start server is called as Deployment.
4. >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:--

a>Spring Boot supports two build tools **Maven** and **Gradle**.

b>Spring Boot supports 3 embedded servers and 3 embedded databases. These are not required to download and install.

**i>Embedded Servers:--** 1>Apache Tomcat (default) 2>JBoos Jetty

3>Undertow

## ii>Embedded DataBase:--

1>H2

2>HSQL DB

3>Apache Derby

## c>Spring Boot supports cloud apps with micro services pattern. [“Both coding and Deployment”].

=>Coding is done using Java and Netflix Tools

=>Deployment can be done on various clouds.

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

1>WebMVC and WebServices (Rest).

MVC🡪Model, View , Controller

Important Annotations:

@Controller +@ResponseBody = RestController

@RequestMapping + GET = @GetMapping

2>JDBC and ORM (Hibernate with JPA). 3>Email, Scheduling, JMS, Security.

4>Cache and Connection Pooling.

5>DevTools, Swagger UI, Actuator and Profiles. 6>UI Design using HTML, JSP, Thymeleaf …etc.

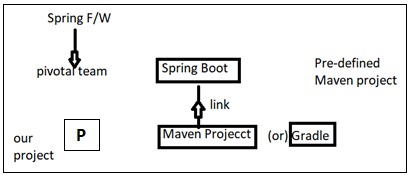
## e> Supports Input Data (Key = val) Using (for AutoConfiguration code):--

=>Properties file or YAML files.

* 1. **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

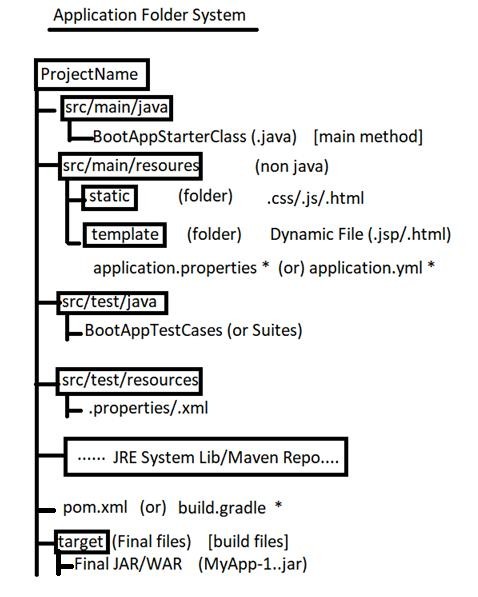
1. SpringBootStarter class
2. application.properties /application.yml
3. pom.xml/build.gradle
4. **SpringBootStarter class:--** It is a main method class used to start our app. It is entry point in execution. Even for both stand alone and web this file used.
5. **application.properties/application.yml:--** 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.

1. **pom.xml (or) build.gradle:--** This file holds all information about
   1. Parent boot project version
   2. App properties (JDK version/maven/cloud versions….)
   3. Dependencies (JARS Details)
   4. Plugins (Compiler/WAR…etc)



# CHAPTER#1 SPRING BOOT CORE

1. **Spring Boot Runners:--**

=>A Runner is an auto-executable component which is called by container on application startup only once.

=>In simple this concept is used to execute any logic (code) one time when application is started.

## Types of Runners(2):--

* 1. **CommandLineRunner :--** This is legacy runner (old one) which is provided in Spring boot 1.0 version.

=>It has only one abstract method “run(String… args) : void”.

=>It is a Functional Interface (having only one abstract method).

=>Add one **stereotype Annotation** over Implementation class level (Ex:- @Component). So that container can detect the class and create object to it.

## Code Setup:--

#Setup : JDK 1.8 and Eclipse / STS.

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

=>File=>new=>Maven Project (\*\*\*Click check box [ v ])=> Create Simple Project

=>Next =>Enter Details (example) Group Id : com.app

ArtifactId : SpringBootRunners Version : 1.0

=>Finish

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

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

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

## pom.xml:--

<project xmlns="<http://maven.apache.org/POM/4.0.0>" xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>" xsi:schemaLocation="<http://maven.apache.org/POM/4.0.0> <http://maven.apache.org/xsd/maven-4.0.0.xsd>">

<modelVersion>4.0.0</modelVersion>

<groupId>com.app</groupId>

<artifactId>SpringBootRunner</artifactId>

<version>1.0</version>

<!-- a. Parent Project details -->

<parent>

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

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

<version>2.1.2.RELEASE</version>

</parent>

<!--b. Versions/properties -->

<properties>

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

</properties>

<!-- c. dependencies/jars -->

<dependencies>

<dependency>

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

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

</dependency>

</dependencies>

<!-- d. build plugins -->

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

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

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

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

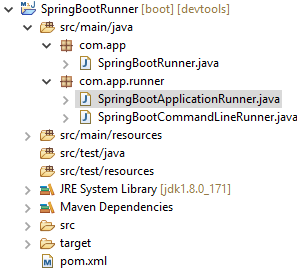
## #4. Write Spring Boot starter class under src/main/java folder:--

=>Right click on “src/main/java”> new => class => Enter details, like: Package Name : com.app

Name : MyAppStarter > Finish

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

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



Code:--

## 1>SpringBootRunner.java (Spring Boot Starter class):-- package com.app;

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

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

@SpringBootApplication

**public class** SpringBootRunner {

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

}

}

**#Runner #1: SpringBootCommandLineRunner.java package** com.app.runner;

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

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

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

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

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

@Override

**public void** run(String... args) **throws** Exception { System.***out***.println("Hii CommandLine Runner");

}

@Override

**public int** getOrder() {

**return** 50;

}

}

## #Runner #2: SpringBootApplicationRunner.java

package com.app.runner;

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

import org.springframework.context.annotation.Configuration; import org.springframework.core.annotation.Order;

/\*ApplicationRunner with Ordered implications with Annotations\*/ @Configuration

@Order(50)

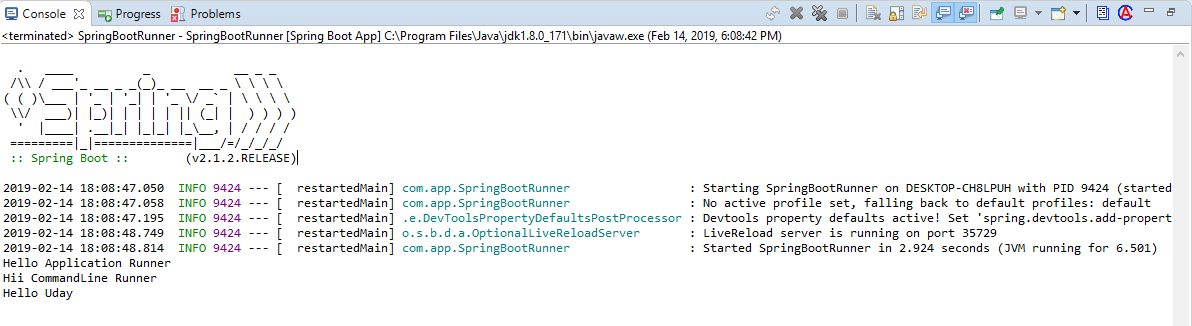
public class SpringBootApplicationRunner implements ApplicationRunner { @Override

public void run(ApplicationArguments args) throws Exception { System.out.println("Hello Application Runner");

}

}

## Output:--



**NOTE:--**

a>Boot Application can have multiple runners Ex:-- EmailRunner, JmsRunner, SecurityRunner, CloudEnvRunner, DevOpsRunner, DatabaseRunner etc… b>Boot provides default execution order.

=>To specify programmer defined order use i>Interface : Ordered or else ii>Annotation : @Order

=>If we are configures both Runner but not implements Ordered then by default

**Annotation based Configuration** will be executed first.

## Input Data to Runners using (CommandLineArguments):--

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

a>Option Arguments b>NonOption Arguments

Syntax : --key =val [Option Arguments] Ex:-- --db=MySQL --db=Oracle

--env=prod --server.port=9876 etc…

Syntax : data [NonOption Argument]

Test clean package execute rollnone etc…

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

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

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

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

package com.app;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication public class MyRunner {

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

}

}

**2>Runner #1 MyInputRunner.java:-- package** com.app.runner;

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

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

@Component

public class MyInputRunner implements CommandLineRunner { public void run (String… args) throws Exception {

System.***out***.println("Hello CommandLineRunner"); System.out.println(args[1]); System.out.println(Arrays.asList(args)); System.***out***.println("End of CommandLineRunner");

} }

## Execution:--

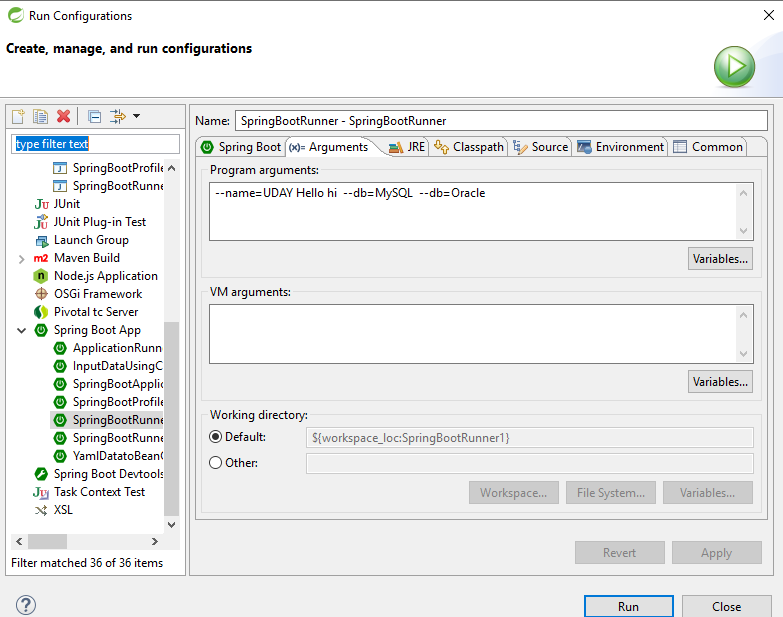
=>Right Click on starter class code (main)

=>Run As => Run Configuration

=>Choose “Arguments” tab

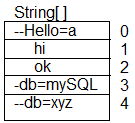
=>Enter data in Program arguments (with space)

--name=UDAY Hello hi --db=MySQL --db=Oracle



=>Click on Apply and Run

=>It is internally converted to String [] (String… args)



## Anonymous Inner class:--

=>A nameless class and object created for an interface having abstract methods.

=>In simple create one class without name and create object at same time without name. Used only one time.

Syntax:--

new InterfaceName() {

//Override all methods

}

**Example#1:--** interface Sample { void show ();

}

## -----Anonymous Inner class--

new Sample() { public void show() {

System.out.println(“Hello”);

}

}

## Example#2:--

interface CommandLineRunner {

void run(String… args) throws Exception;

}

## Anonymouse Inner class:--

new CommandLineRunner() {

public void run (String… args) throws Excception { System.out.println(“HI”);

} }

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

**Code:--**

package com.app;

//Ctrl+shift+O @Configuration

**public class** AppConfig {

//JDK 1.7 or before (Inner class style) @Bean

**public** CommandLineRunner cob () {

**return new** CommandLineRunner() {

**public void** run (String... args) **throws** Exception{ System.***out***.println(Arrays.*asList*(args));

} };

}

//JDK 1.8 or higher (lambda) @Bean

**public** CommandLineRunner cob2() {

**return** (args) -> {

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

};

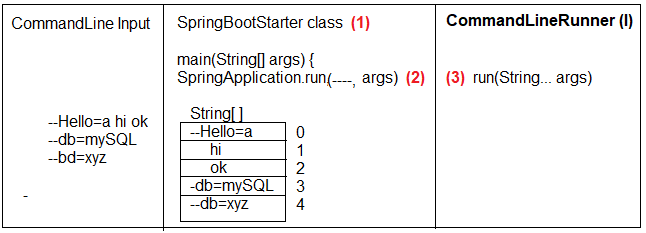
}

}

## Q>How CommandLineRunner works?

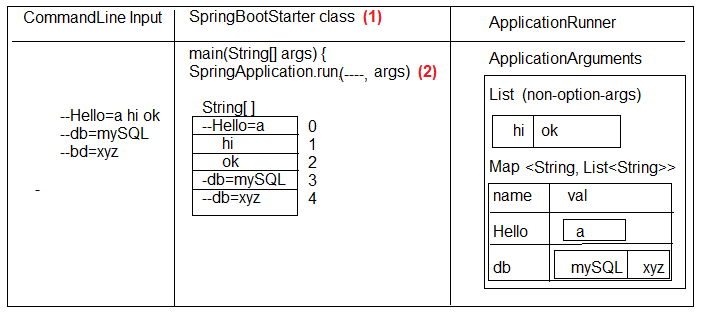
A>CommandLine arguments which are passed to application which will be given to Spring Boot starter main(..) method. Those are stored as “string Array” (String[]).

SpringApplication.run(…) reads this input and internally calls run(..) methods of RunnerImpl classes and pass same data.



* 1. **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.

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

package com.app.runner; import java.util.Arrays;

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

@Component

public class SpringBootApplicationRunner implements ApplicationRunner { public void run(ApplicationArguments args) throws Exception {

System.out.println("hello Application Runner"); System.out.println(Arrays.asList(args.getSourceArgs())); System.out.println(args.getNonOptionArgs()); System.out.println(args.getOptionNames()); System.out.println(args.getOptionValues("db")); System.out.println(args.containsOption("bye")); System.out.println("End of Application Runner");

}

}

## Output:--

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

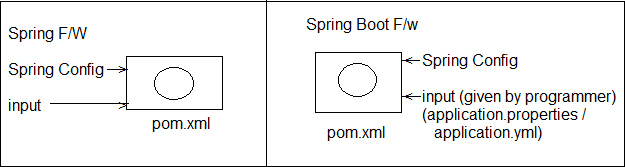
A> Working process of CommandLineRunner and ApplicationRunner are same, but CommandLineRunner (CLR) holds data in String[] format where as Application (AR) holds data as ApllicationArguments as Option/Non-Option format.

1. **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-*](https://docs.spring.io/spring-%20%20%20%20%20%20%20%20%20%20%20%20%20boot/docs/current/reference/html/common-application-properties.html)[*boot/docs/current/reference/html/common-application-properties.html*](https://docs.spring.io/spring-%20%20%20%20%20%20%20%20%20%20%20%20%20boot/docs/current/reference/html/common-application-properties.html)*)*

b>Programmer defined

## #3. 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=999A my.info.product.code=xyz my.info.product.model-version=44.44 my.info.product.release\_dtl\_enable=false my.info.product.start-key=N

## NOTE:--

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

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

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

**#3.** Starter class same as above.

## #4. Runner with key data class (SpringBootRunnerWithInputData.java):-- package com.app.runner;

**import** org.springframework.beans.factory.annotation.Value; **import** org.springframework.boot.CommandLineRunner; **import** org.springframework.stereotype.Component;

@Component

**public class** SpringBootRunnerWithInputData **implements** CommandLineRunner

{

@Value("${my.info.product.id}") **private int** prodId; @Value("${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

//Setters and Getters method

//toString method @Override

**public** String toString() {

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

}

//Overridden run method

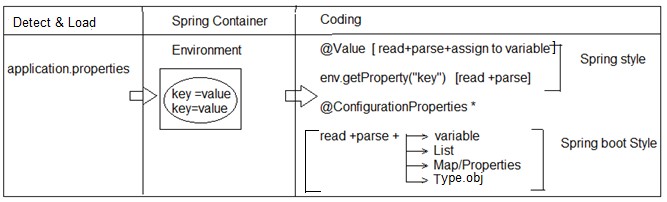
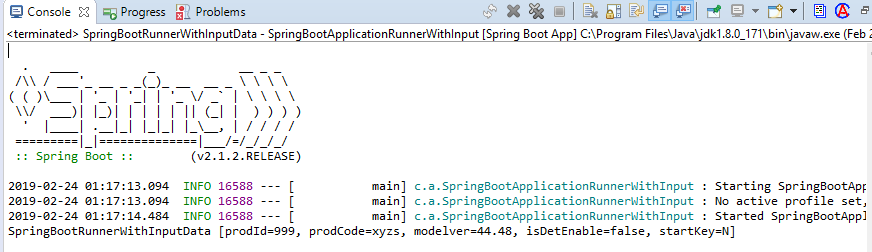
**public void** run(String... args) **throws** Exception { System.***out***.println(**this**);

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

}

}

## Output:--



**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)

=>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

**4>@ConfigurationProperties:--**

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

=>Possible conversions are. a>1key = 1 variable

b>Multiple keys with index = List/Set/Array

c>Multiple keys with key-value format = Map or Properties d>Multiple keys with common type = Class Object (Has-A)

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

Example:--

## Starter class (SpringBootApplicationEx.java):--

package com.app;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication public class SpringBootStarter {

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

}

}

1. **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 generater 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}

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

package com.app.input;

import org.springframework.beans.factory.annotation.Autowired; import org.springframework.boot.CommandLineRunner;

import org.springframework.core.env.Environment; import org.springframework.stereotype.Component;

@Component

public class UsingEnvironment implements CommandLineRunner{

@Autowired

private Environment env;

@Override

public void run(String... args) throws Exception { System.out.println(env.getProperty("my.prod.ID"));

System.out.println(env.getProperty("my.prod.code")); System.out.println(env.getProperty("my.prod.Ty\_pe")); System.out.println(env.getProperty("my.prod.MOD-E\_L"));

}

}

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

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

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

package com.app.input; import java.util.Arrays; import java.util.List; import java.util.Set;

import org.springframework.boot.CommandLineRunner;

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

@ConfigurationProperties("my.prod") @Component

public class UsingApplicationProperties implements CommandLineRunner {

private int id; private String code;

private boolean type; private char model;

private List<String> projname; private Set<String> projname1; private String[] projname2;

public UsingApplicationProperties() { super();

}

public int getId() {

return id;

}

public void setId(int id) { this.id = id;

}

public String getCode() { return code;

}

public void setCode(String code) { this.code = code;

}

public boolean isType() { return type;

}

public void setType(boolean type) { this.type = type;

}

public char getModel() { return model;

}

public void setModel(char model) { this.model = model;

}

public List<String> getProjname() { return projname;

}

public void setProjname(List<String> projname) { this.projname = projname;

}

public Set<String> getProjname1() { return projname1;

}

public void setProjname1(Set<String> projname1) { this.projname1 = projname1;

}

public String[] getProjname2() { return projname2;

}

public void setProjname2(String[] projname2) { this.projname2 = projname2;

}

@Override

public String toString() {

return "UsingApplicationProperties [id=" + id + ", code=" + code + ", type=" + type + ", model=" + model+ ", projname=" + projname + ", projname1=" + projname1 + ", projname2=" + Arrays.toString(projname2)+ "]";

}

@Override

public void run(String... args) throws Exception { System.out.println(this.toString());

}

}

**Example #3 Generating Random Values:--**We can use a direct expression @Value(“{random.---}”) in java code or in properties file.

=>Possible random data is a>Hexa Decimal Value b>int or long type

c>int or long with range

d>UUID (Universal Unique Identifier)

## Example #3:-- 1>application.properties:--

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

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

package com.app.input;

import org.springframework.beans.factory.annotation.Value; import org.springframework.boot.CommandLineRunner;

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

@Component

public class UsingValueAnnotation implements CommandLineRunner {

//@Value("${my.random.stringval}")

//@Value("${my.random.stringval}")

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

@Value("${my.random.num}")

//@Value("${my.random.num-rang}")

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

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

@Override

public void run(String... args) throws Exception { System.out.println(this);

}

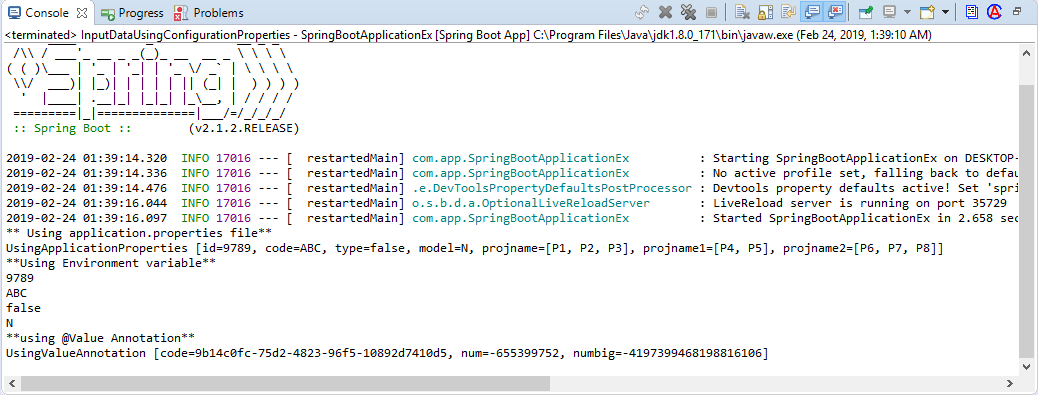
public UsingValueAnnotation() { super();

}

public UsingValueAnnotation(String code, int num, long numbig) { super();

this.code = code; this.num = num; this.numbig = numbig;

}



@Override

public String toString() {

return "UsingValueAnnotation [code=" + code + ", num=" + num + ", numbig=" + numbig + "]";

}

}

## Output:--

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

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

1> Under Project:-- Under Config folder:-- Project/config/application.properties (file:./config/application.properties)

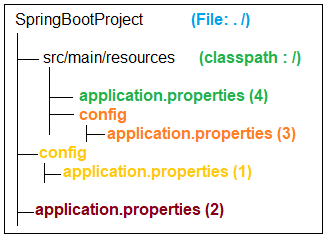
2> Under Project (Only):--

Project/application.properties (file: ./application.properties) 3> Under Project (Under resources/config):--

Project/src/main/resources/config/application.properties (classpath:/config/application.properties)

3> Under “Project” folder:-- Project/src/main/resource/application.properties (classpath:/application.properties)

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



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

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

**Step#1:-** Create your file under one location Ex:-- src/main/resources

|-mydata.properties

(or any other location also valid)

**Step#2:-** Use Run Configuration and provide option argument => apply and Run Ex:-- s--spring.config.name=mydata (user defined)

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

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

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

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

1. **YAML (YAMlian Language):--**

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

=>File extension is “.yml”.

=>It will hold data in below format key : <space> value

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

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

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

=>Snake YAML will

a>Check for space and prefix levels b.>Trace keys for data find.

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

=>Consider below example properties file

-=-=- “application.properties” -=-=- Ex#1:-

my.data.id-num=10 my.data.core\_val=AB my.data.enabled.costBit=3.6 my.data.enabled.valid=true

Ex#2:--

my.code.id=56 my.code.mdn.type=big my.code.str.service=ALL my.code.str.service.info=true my.code.mdn.obj=CRL my.code.cost=3.67 my.code.mdn.sale=YES

=>Its equal YAML file looks as

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

my:

code:

id: 56

cost: 3.67 mdn:

type: big obj: CRL sale: YES

str:

service: ALL

info: true

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

=>In properties file we can use from zero.

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

Ex:-- application.proeprties--- my.code.version[0]=V1 my.code.version[1]=V2 my.code.version[2]=V3

---application.yml:--- my:

code:

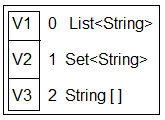
version:

-V1

-V2

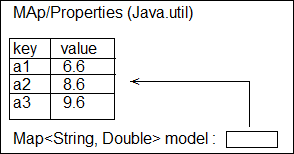
-V3

=>final meaning is:



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

=>Consider below example:--



=>Its equal properties file will be Ex:-- application.properties my.data.model.a1=6.6 my.data.model.a2=8.6 my.data.model.a3=9.6

=>Its equal : application.yml file my:

data:

model:

a1: 6.6

a2: 8.6

a3: 9.6

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

**1>pom.xml**

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

<dependency>

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

<artifactId>spring-boot-configuration-processor</artifactId>

<optional>true</optional>

</dependency>

## >Write starter class (same as before) [main method]

package com.app;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class ApplicationProperiesUsingYml { public static void main(String[] args) {

SpringApplication.run(ApplicationProperiesUsingYml.class, args);

}

}

## >application.yml:--

#Normal Data my:

prod:

id: 5 code: AB cost: 4.5

#List Data

version:

-V1

-V2

-V3

#Map Data

model: a1: 6.6

a2: 8.6

a3: 9.6

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

package com.app.model; import java.util.List; import java.util.Map;

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

@ConfigurationProperties("my.prod") @Component

public class Product {

private int id; private String code; private double cost;

private List<String> version;

private Map<String, Double> model;

public Product() {

super();

}

public Product(int id) { super();

this.id = id;

}

public Product(int id, String code, double cost, List<String> version, Map<String, Double> model) {

super(); this.id = id;

this.code = code; this.cost = cost; this.version = version; this.model = model;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getCode() { return code;

}

public void setCode(String code) { this.code = code;

}

public double getCost() { return cost;

}

public void setCost(double cost) { this.cost = cost;

}

public List<String> getVersion() { return version;

}

public void setVersion(List<String> version) { this.version = version;

}

public Map<String, Double> getModel() { return model;

}

public void setModel(Map<String, Double> model) { this.model = model;

}

@Override

public String toString() {

return "Product [id=" + id + ", code=" + code + ", cost=" + cost + ", version=" + version + ", model=" + model+ "]";

}

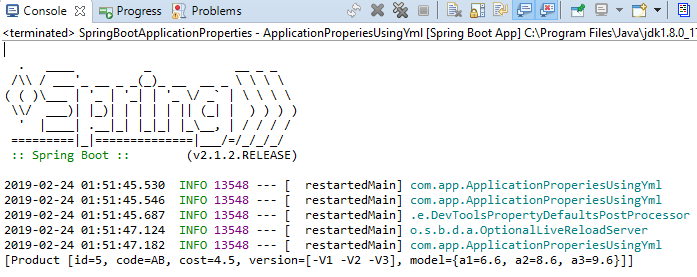
}

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

package com.app.runner; import java.util.Arrays;

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

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



@Component

public class ApplicationRunnerEx implements ApplicationRunner { @Autowired

private Product prod; @Override

public void run(ApplicationArguments args) throws Exception { System.out.println(Arrays.asList(prod));

}

}

## Output:--

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

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

## POJO Rules :--

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

2>Java Bean :-- POJO Rules + Inheritance + Special methods + Param constructor. 3>Spring Bean:-- OJO Rules + Inheritance (Spring API) + Annotations (Spring API) + Special methods (Object class and Spring API) [toString()…]

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

Example Code:--

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

**public class** SpringBootStarter {

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

}

}

## #2. application.yml:--

my:

dt:

pid: 55 mo:

mid: 67 mcode: ABC colors:

-RED

-GREEN

-YELLOW

**#3. Model class #1(Child ) Model-- package** com.app.bean;

**import** java.util.List;

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

**public class** Model {

**private int** mid;

**private** String mcode;

**private** List<String> colors;

**public** Model() {

**super**();

}

**public int** getMid() {

**return** mid;

}

**public void** setMid(**int** mid) {

**this**.mid = mid;

}

**public** String getMcode() {

**return** mcode;

}

**public void** setMcode(String mcode) {

**this**.mcode = mcode;

}

**public** List<String> getColors() {

**return** colors;

}

**public void** setColors(List<String> colors) {

**this**.colors = colors;

}

@Override

**public** String toString() {

**return** "Model [mid=" + mid + ", mcode=" + mcode + ", colors=" + colors + "]";

}

}

**#4. Model class(Parent) Product :-- package** com.app.bean;

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

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

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

@ConfigurationProperties("my.dt") @Component

**public class** Product

{

**private int** pid;

@Autowired

**private** Model mo; //HAs-A

**public** Product() {

**super**();

}

**public int** getPid() {

**return** pid;

}

**public void** setPid(**int** pid) {

**this**.pid = pid;

}

**public** Model getMo() {

**return** mo;

}

**public void** setMo(Model mo) {

**this**.mo = mo;

}

@Override

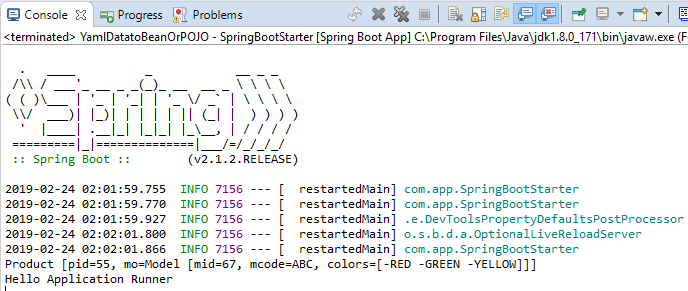
**public** String toString() {

**return** "Product [pid=" + pid + ", mo=" + mo + "]";

}

}

**#5. Runner class (CommandLineRunnerForYaml):-- package** com.app.runner;



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

**import** org.springframework.boot.CommandLineRunner; **import** org.springframework.stereotype.Component; **import** com.app.bean.Product;

@Component

**public class** CommandLineRunnerForYaml **implements** CommandLineRunner

{

@Autowired

**private** Product pob;

**public void** run(String... args) **throws** Exception { System.***out***.println(pob); System.***out***.println("Hello Application Runner");

}

}

## Output:--

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

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

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

**application.properties:--** my.dt.pid=68 my.dt.mid=${my.dt.pid}

## application.yml:--

my:

dt:

pid: 68

mid: ${my.dt.pid}

## NOTE:--

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

Ex:-- application.yml:-- #Hello data to Product my:

dt

---

my:

pid: 57

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

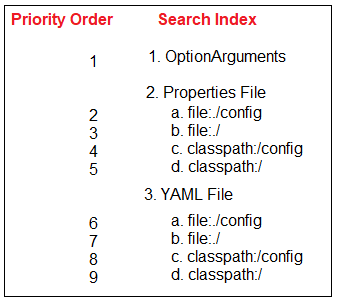
## Priority Order for key Search:--

=>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:--

1. **Spring Boot Profiles:--**

=>In RealTime, Application is

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

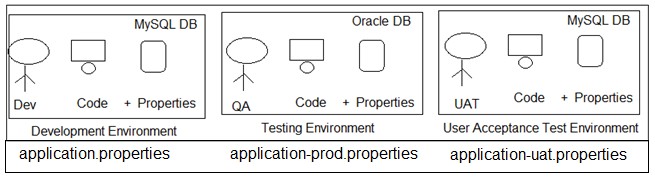
\*\*\*>Environment is place where our application is running or what is current state of application

Example:-- dev= development

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

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

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



**Profile Specific Tasks:--**

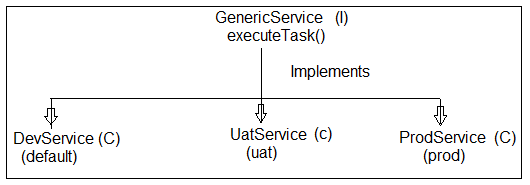
=>Profile supports Environment based Code (Task) selection, not only Properties (yaml).

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

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

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

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



## Example:--

=>File => new => Spring Starter Project => Enter Details Project Name : SpringBootProfiles and also

GroupId : org.sathyatech ArtifactId : SpringBootProfiles Version : 1.0

=> next => next => finish.

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

**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

Ex:-- my.profile.code=Hello from default b> application-prod.properties

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

c> application-uat.properties

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

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

## 3>service interface:--

**#1 Create an Interface (GenericService.java):-- package** com.app.profile.service;

**public interface** GenericService {

**public void** executeTask();

}

## 4>Create Multiple classes like and implements GenericService interface:--

1. **DevService.java:--**

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

**import** org.springframework.beans.factory.annotation.Value; **import** org.springframework.context.annotation.Profile; **import** org.springframework.stereotype.Component;

**import** com.app.profile.service.GenericService;

@Component @Profile("default")

**public class** DevService **implements** GenericService { @Value("${my.profile.code}")

**private** String code;

**public** DevService(String code) {

**super**(); **this**.code = code;

}

**public** String getCode() {

**return** code;

}

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

**this**.code = code;

}

@Override

**public** String toString() {

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

}

@Override

**public void** executeTask() { System.***out***.println("From Dev Profiles"); System.***out***.println("code is "+code);

}

}

## ProdService.java:--

package com.app.profile.service.impl;

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

import com.app.profile.service.GenericService;

@Component @Profile("prod")

public class ProdService implements GenericService {

@Value("${my.profile.code}") private String code;

public ProdService() { super();

}

public ProdService(String code) { super();

this.code = code;

}

public String getCode() { return code;

}

public void setCode(String code) { this.code = code;

}

@Override

public String toString() {

return "ProdService [code=" + code + "]";

}

public void executeTask() { System.out.println("From Prod Profile"); System.out.println("code is "+code);

}

}

## UatService class:--

package com.app.profile.service.impl;

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

import com.app.profile.service.GenericService;

@Component @Profile("uat")

public class UatService implements GenericService { @Value("${my.profile.code}")

private String code;

public String getCode() { return code;

}

public void setCode(String code) { this.code = code;

}

@Override

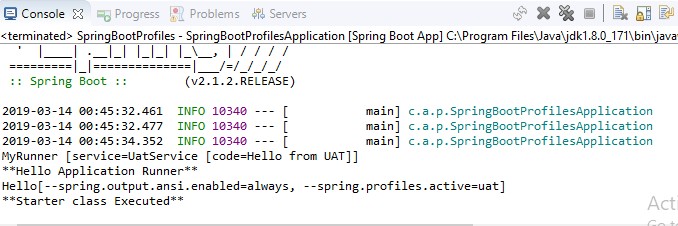
public String toString() {

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

}

public void executeTask() { System.out.println(this); System.out.println("From Uat Profiles");

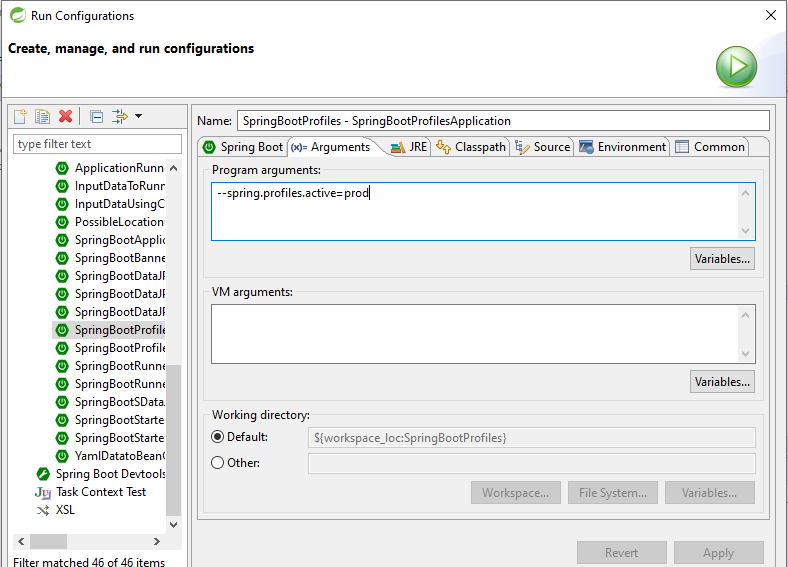
System.out.println("code is "+code);



}

}

## Execution of Program:--

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

## Output:--

**NOTE:--**

1>Use key spring.profiles.active=profile

=>We can provide using 3 ways. Those are

a> Command Line Arguments (Option Arguments)

Ex spring.profiles.active=prod

b>In application.proeprties

Ex:-- spring.profiles.active=prod b> VM (JVM/System) Argument:--

Ex Dspring.profiles.active=prod

=>Right click on Starter class => Run As=> Run Config =>Choose Arguments

=>Enter below format in VM Arguments

-Dspring.profiles.active=prod

=>apply and Run

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

=>If no where key is found then

a>@Value generate Exception (IllegalArgumnetException) b>@ConfigurationProperties :- Default value of DataType is chosen by container.

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

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

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

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

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

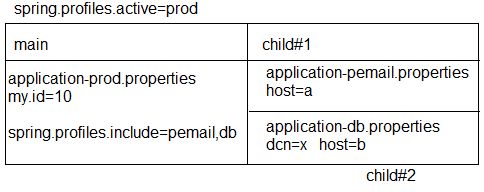
## Including Child Profiles:--

=>In spring Boot applications active profile can be specified using key Ex: --spring.profiles.active=[ ]

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

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

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



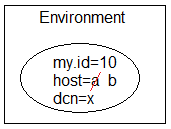
## Spring Container will load:--

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

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

=>For above example, priority for loading (loading order is) a>application-prod.properties

b>application-pemail.properties c>application-db.properties



## Profiles using YAML:--

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

=>File Naming Rule:- application-{profile}.yml **Example:--**

application.yml (default) application-prod.yml (prod) application-uat.yml (uat)

## Multiple Profiles using one YAML:--

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

## application.yml:--

my:

profile:

id: 666

---

my:

profile:

id: 999

spring:

profiles: prod

---

my:

profiles:

id: 888 spring:

profiles: uat

## NOTE:--

#1 To specify active and include profiles use

## a>Option Arguments:--

--spring.profiles.active=prod

--spring.profiles.include=prodemail

**b>use Properties file:--** spring.profiles.active=prod spring.profiles.include=prodemail

## c>use YAML file:--

spring:

profiles:

active: prod include:

-prodemail

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

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

package com.app;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringBootProfilesUsingYmlApplication { public static void main(String[] args) {

SpringApplication.run(SpringBootProfilesUsingYmlApplication.class, args);

}

}

## 2>application.yml:--

my:

profile:

id: 666

spring:

profiles:

active: prod include:

-prodemail #Production Profiles

---

my:

profile:

id: 999

spring:

profiles: prod #Uat profiles

---

my:

profile:

id: 888

spring:

profiles: uat #Prodemail profile

---

my:

profile:

email: [udaykumar0023@gmail.com](mailto:udaykumar0023@gmail.com)

spring:

profile: prodemail

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

package com.app.bean;

import org.springframework.boot.CommandLineRunner;

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

@Component @ConfigurationProperties("my.profile")

public class ProductRunner implements CommandLineRunner {

private int id; private String email;

public ProductRunner() { super();

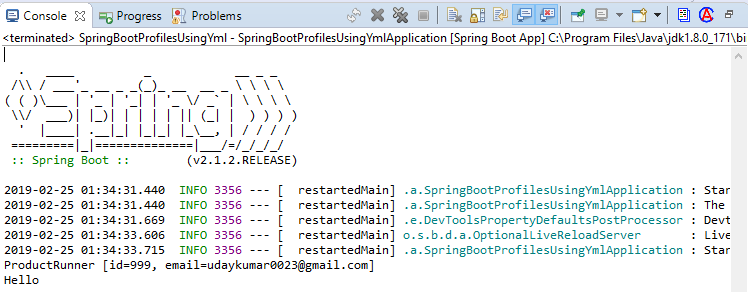
}

public int getId() {

return id;

}

public void setId(int id) { this.id = id;



}

public String getEmail() { return email;

}

public void setEmail(String email) { this.email = email;

}

@Override

public String toString() {

return "ProductRunner [id=" + id + ", email=" + email + "]";

}

public void run (String... args)throws Exception{ System.out.println(this);

}

}

## Output:--

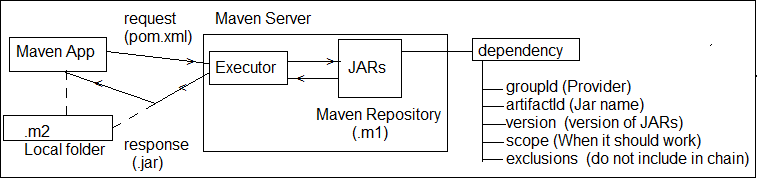
1. **pom.xml (Maven Process) : --**

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

## Dependency Management:--

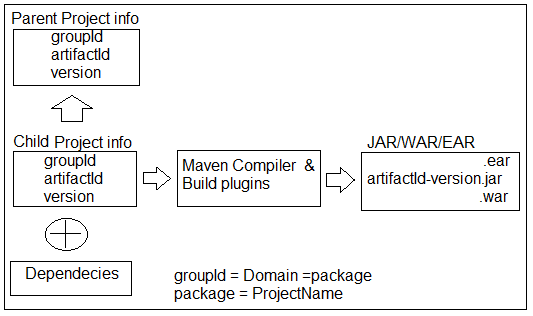
Getting Main Jars and its child jars with version support (without conflicts) into project workspace is called as Dependency Management.

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



## Major components of pom.xml:--

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



## pom.xml format:--

<project….>

<modelVersion>4.0.0</modelVersion>

<!-- CUrrent Project info -->

<groupId>a.l</groupId>

<artifactId>HelloApp</artifactId>

<version>1.0</version>

<!-- Parent Project info -->

<parent>

<groupId>a.a</groupId>

<artifactId>DBApp</artifactId>

<version>6.1</version>

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

</parent>

<!-- Current Project info -->

<properties>

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

</properties>

<!-- Project JARs 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>

Ex:--

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.1.5</version>

<exclusions>

<exclusion>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

</exclusion>

</exclusions>

</dependency>

</dependencies>

## Scope (<scope> </scope> in dependency:--

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

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

## POM format:--

<dependency>

<groupId>…</groupId>

<artifactId>…</artifactId>

<scope>….</scope>

</dependency>

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

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

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

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

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

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

=>In this case we should also give <SystemPath> with location of JAR. Ex:-- <systemPath>D:/asf/lib/</systemPath>

**NOTE:--** There is a dependency jar which not existing in the maven centre but locally. After mvn clean install, this dependency jar can't be found from the fat jar. is this an known-issue? the workaround is have to install it into local maven repo with command:

`mvn install:install-file -Dfile=lib/routines.jar -DgroupId=org.talend - DartifactId=routines -Dversion=1.0 -Dpackaging=jar`

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

<dependency>

<groupId>org.talend</groupId>

<artifactId>routines</artifactId>

<version>1.0</version>

</dependency>

## Format of Scope:--

<dependencies>

<!-- Compiler time (Default) Execution -->

<dependency>

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

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

<scope>compile</scope> <!-- Optional -->

</dependency>

<!-- Runtime Execution -->

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

<!-- Test time Execution -->

<dependency>

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

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

<scope>test</scope>

</dependency>

<!—framework or container time Execution -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>4.0.0</version>

<scope>provided</scope>

</dependency>

<!—From local system Execution -->

<dependency>

<groupId>routines</groupId>

<artifactId>routines</artifactId>

<version>1.0</version>

<scope>system</scope>

<systemPath>${basedir}/lib/routines.jar</systemPath>

</dependency>

</dependencies>

## Maven Goals Execution:--

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

**2>Maven install :--** It will downloaded all required plugins and also

=>compile the source files.

=>load required properties.

=>Execute JUnit Test cases.

=>Create final build (.jar/.war).

## #1. clean :--

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

## #2. install:--

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

## #3. Build:--

=>right click on project => Run As => maven build.. =>provide goals like clean install

=> Also choose skipTests =>Apply and Run.

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

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

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

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

## Version Management in Spring Boot Application:--

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

=>To see all jars provided with what version, a.>Goto pom.xml

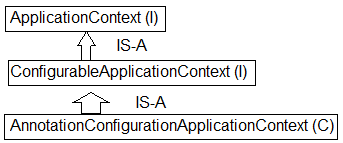
b.>Ctrl + Mouse over <artifactId> then click

c.>Search for tag properties

1. **Spring Boot Starter class Concepts:--**

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

=>Spring Boot Starter class uses run (..) method from class “**SpringApplication**” defined in package : **org.springframework.boot** which creates Spring container using “**AnnotationConfigApplicationContext (C)**”



=>In case of Web/WebServices, Container is created using classes “**AnnotationServletWebServerApplicationContext (C)**”.

## NOTE:--

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

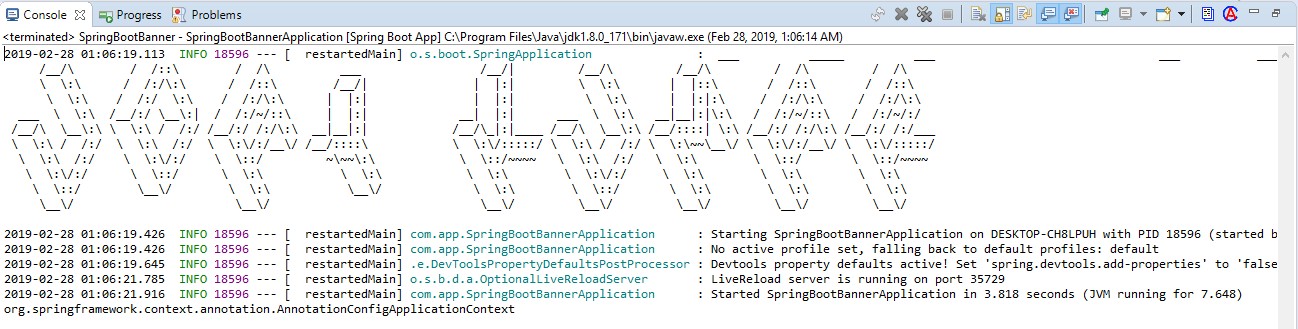
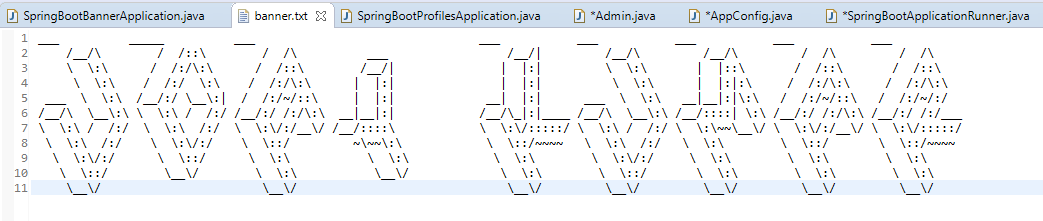
Ex:--

* 1. >For banner use Banner.Mode.Property, Banner.Mode.OFF (to turn off banner)
  2. >Mode is Inner enum defined in functional Interface “Banner”.
  3. >We can provide our own banner using file : banner.txt (Create under classpath)

i.e : src/main/resource/banner.txt (file) (<https://devops.datenkollektiv.de/banner.txt/index.html>)

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

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



package com.app;

import org.springframework.boot.Banner;

import org.springframework.boot.SpringApplication;

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

@SpringBootApplication

public class SpringBootBannerApplication

{

public static void main(String[] args)

{

SpringApplication sa = new SpringApplication (SpringBootBannerApplication.class);

//sa.setBannerMode(Banner.Mode.OFF); //to Disable the banner

//sa.setBannerMode(Banner.Mode.CONSOLE); //to Disable the banner on console sa.setBannerMode(Banner.Mode.LOG); //to Display the banner in Log file

//some other configuration ConfigurableApplicationContext c = sa.run(args); System.out.println(c.getClass().getName().toString());

}

}

## 2>Banner.txt file:--

**Output with Banner:--**

## Use of Starter class:--

1>Define Spring Container.

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

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

=>For server based Application, Impl class is :

## AnnotationConfigServletWebServerApplicationContext (C).

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

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

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

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

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

**Consider below starter class:--** package com.app; @SpringBootApplication

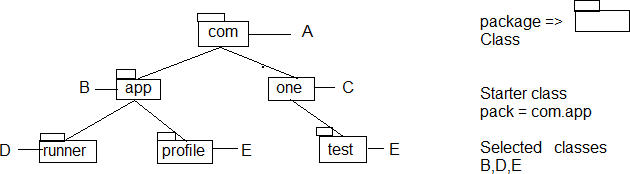
//@ComponentScan(“com.app”) -->Added by Spring Boot public class MyStarter {…….}

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

=>Programmer can provide externally basePackage using @ComponentScan

a>Avoid Starter class package and provide our own. Ex:-- @ComponentScan (“com.app”)

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



b>Provide our own starter package (even other packages) using array style {-,-,-,-,} Ex:-- @ComponentScan({“com.app”,”com.one”,”com”})

c>We can provide one common package name which covers all sub-levels. Ex:-- @ComponentScan(“com”)

**Example:-- package** com.app;

@SpringBootApplication

//@ComponentScan("com.one") @ComponentScan("com")

//@ComponentScan({"com.one", "com", "com.app"})

**public class** MyStarter {

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

{

StringApplication s = **new** SpringApplication(AppStarter.**class**); ConfigurableApplicationContext ac = s.run(args); System.out.println(ac.getClass().getName());

System.out.println(ac.getBean(“Product”)); System.out.println(ac.getBean(“Info”));

}

}

3.> Every Spring Boot Application Starter class itself Component. i.e @SpringBootApplication is having @Component annotation internally.

=>It is only highest Priority component by default, if app has multiple components. Ex:- We can convert starter even as Runner.

**package** com.app;

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

@SpringBootApplication

public class MyStarter implements CommandLineRunner { public void run (String… args) throws Exception {

System.out.println(“From Starter”);

}

**public static void** main(String[] args) { SpringApplication.*run*(MyStarter.**class**, args); System.***out***.println("\*\*Starter class Executed\*\*");

}

}

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

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

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

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

## @Configuration.

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

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

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

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

package com.app.model;

public class Admin

{

private int adminId; private String adminName;

public Admin() {

super();

}

public int getAdminId() { return adminId;

}

public void setAdminId(int adminId) { this.adminId = adminId;

}

public String getAdminName() { return adminName;

}

public void setAdminName(String adminName) { this.adminName = adminName;

}

@Override

public String toString() {

return "Admin [adminId=" + adminId + ", adminName=" + adminName + "]";

}

}

## b>Product.java:--

package com.app.model; public class Product {

private int 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.app.config;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration; import com.app.model.Admin;

@Configuration

public class AppConfig { @Bean

public Admin aobj() {

Admin a = new Admin(); a.setAdminId(100); a.setAdminName("Uday"); return a;

}

}

## 3>Starter class:--

package com.app;

import org.springframework.beans.factory.annotation.Autowired; import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication; import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Import; import com.app.config.AppConfig;

import com.app.model.Admin; import com.app.model.Product;

@SpringBootApplication

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

public class SpringBootStarterApplicationWithRunner implements CommandLineRunner

{

@Autowired private Product p; @Autowired private Admin a;

public void run(String... args)throws Exception { System.out.println("From starter class :"+p); System.out.println("From starter class :"+a);

}

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

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

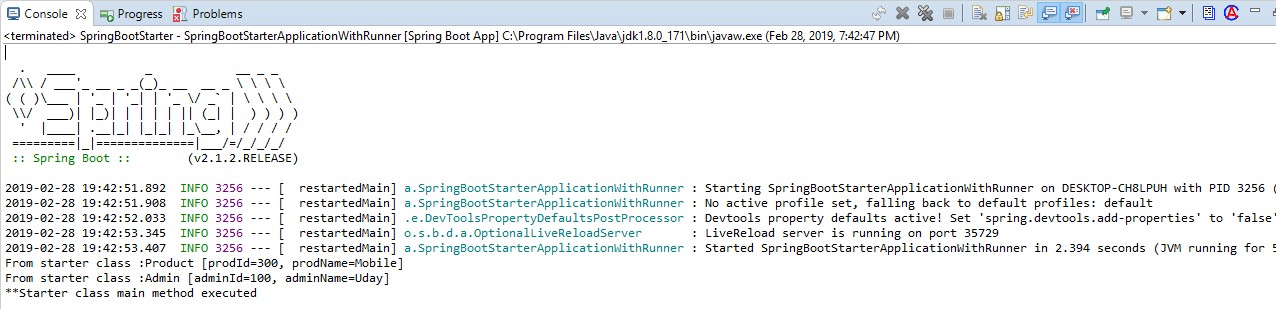
}

@Bean

public Product proj() {

Product p = new Product(); p.setProdId(300); p.setProdName("Mobile"); return p;

}



}

## Output:--

**8. Spring Initializer:--**

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

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

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

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

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

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

**Step#2:-** Provide all details and click on generate Project. **Step#3:-** It will be downloaded as .zip, Extract this to one Folder. **Step#4:-** Open Eclipse (or any IDE), then

>Right click on Project Explorer

>Choose Import => type maven

>select Existed Maven Project

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

>Click enter => choose next/finish

# CHAPTER#2: SPRING BOOT DATA JPA

**1. Introdution about Data-JPA:--**

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

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

=>When we add 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 App Stops.

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

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

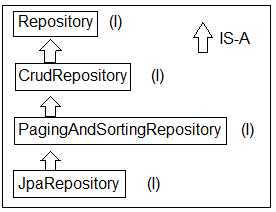
**#4:-** Data JPA Supports Special concept “Query Methods an easy way to code and fetch data from DB” (ex : findBy, @Query).

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

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

## Repository Interfaces:--

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

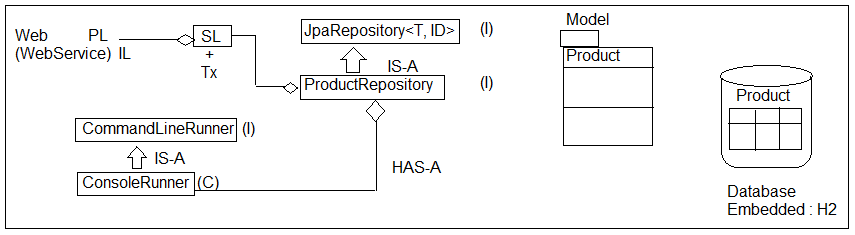


## Spring Boot Data JPA Module Design:--

Required:

1> Database (Using Embedded : H2) 2> Model class : Product (C)

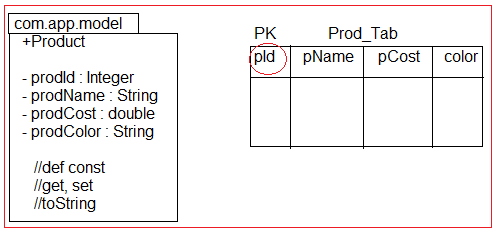
3> Repository : ProductRepository 4> Runner : ConsoleRunner



T = ? = Model class Name

ID = ? = Pk DataType = Integer

=>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 +DownArraw => Copy current line, paste

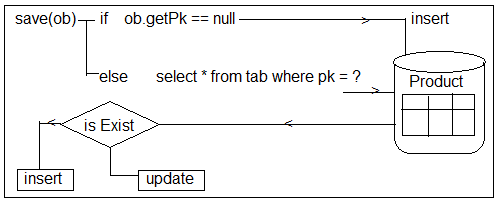
Select Lines

+ctrl + shift + / =>comment lines

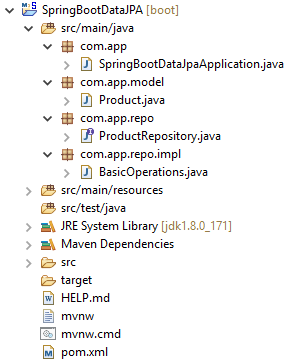
+ctrl +shift + \ =>Uncomment lines

**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;

}

@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())

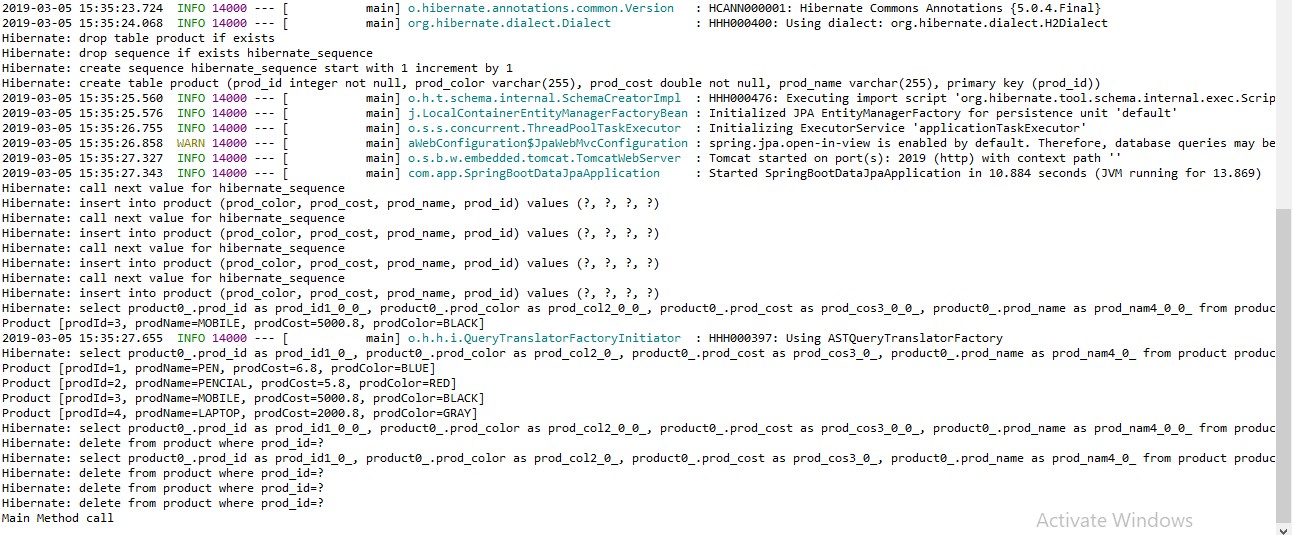
{

System.out.println(p.get());

} else {

System.out.println("No Data found");

}



}

## Output:--

//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();

}

## 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]

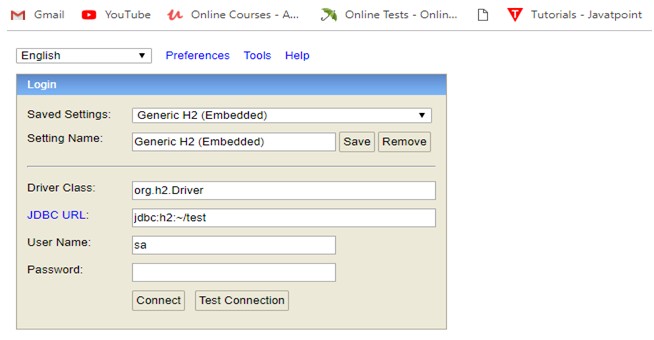
**6>deleteAllInBatch () :--** To delete All rows at a time ex: delete from <tableName>

=>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

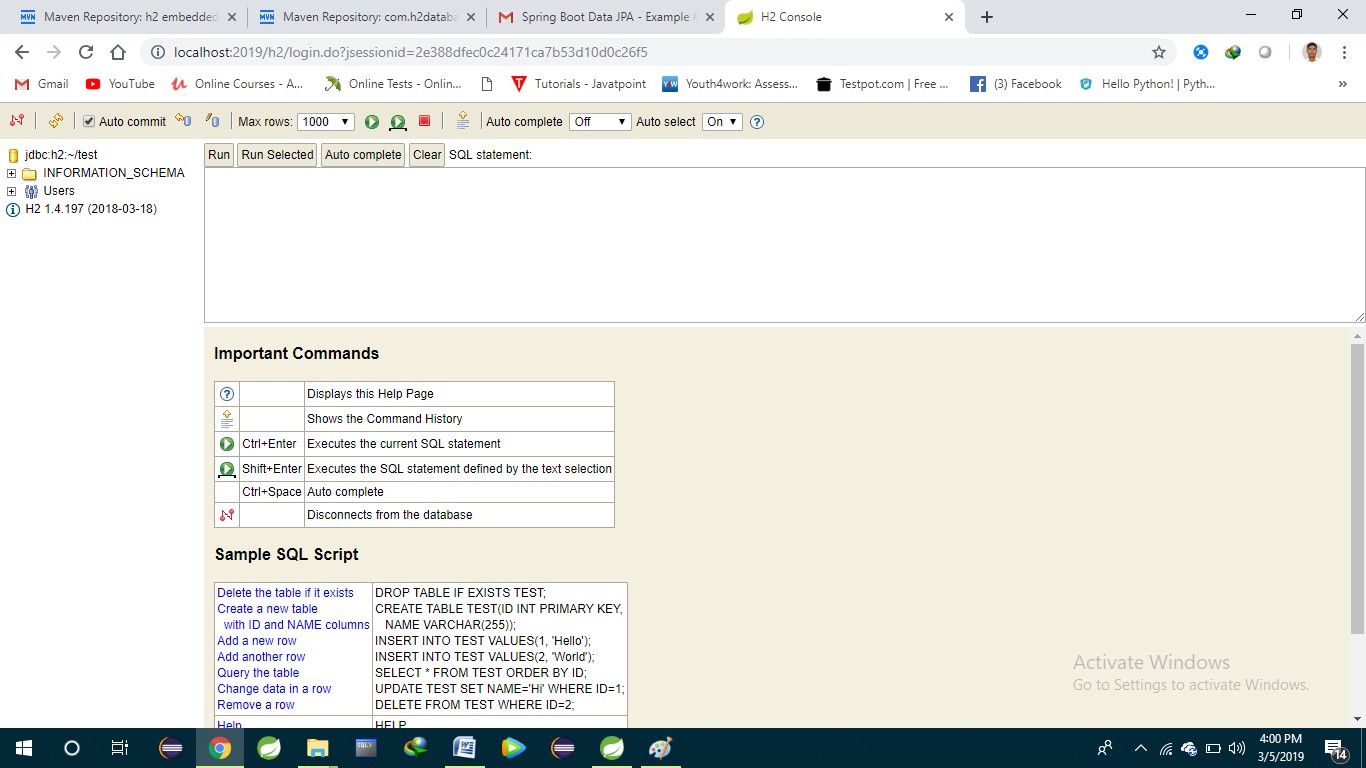
**How to Show Data in H2 DataBase:-- Step#1:-** Execute Program:--

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



=>Click on connect.

## Step #3:-



**2.2 Query Method in Spring Boot Data:--**

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

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

1>findBy

2>@Query (manual Query) 3>Special Parameters/ReturnTypes

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

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

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

Syntax:--

RT findBy (Parameters …);

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **[NareshIT, Hyd]**  **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  preended %) |  |
|  | **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**  **Naresh IT, Hyderaba** | findByActiveTrue()  **d P: 040-2374 6666,9000994007 /0** | … where x.active = true  **8] Page 80** |  |

**False** findByActiveFalse() … where x.active = false

… where UPPER(x.firstame) =

**IgnoreCase** findByFirstnameIgnoreCase UPPER(?1)

## LB Connect using Spring Data JPA

## Folder Structure:--

## 

**1>Starter class LBConnectApplication.java:--**

@SpringBootApplication

**public** **class** LbconnectApplication {

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

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

}

}

**2>Model class Borrower.java,Lender.java,etc.**

## 3>Repository Interface (BorrowerRepository.java):--

=>Add below methods in Repository package com.app.repo;

import java.util.Collection;

import java.util.List;

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

import com.app.model.Product;

@Repository

public interface BorrowerRepository extends CRUDRepository<Borrower, Integer> {

}

**Service classes:**

Borrower Service.java

Lender Service.java

**Controller classes:**

Borrower Cotroller.java

LenderController.java

**Typical Application flow:**

Borrower Controller🡪Borrower Service🡪Borrower Repository🡪(MySQL Data(Borrower->Borrower table)

Connection will be created automatically using pom.xml in any spring-boot application.

# CHAPTER#3 MONGODB

1. **Introduction:--**

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

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

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

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

=>JSON format is : {“key” : value,….}.

=>JACKSON is a converter API used to convert java object <=> JSON. Another example for converter is GSON.

|  |  |
| --- | --- |
| **Java** | **Java Script** |
| int a =10;  Spring b = “Hi”; | var a = 10;  var b = “Hi”; |
| Employee e = new Employee(); e.setEmpId(10);  e.setEmpName(“A”); e.setEmpSal(3.5);  //Java Notation | var = {  “empId” : 10,  “empName” : “A”, “empSal” : 3.2  } |

**Download and Install MongoDB (NoSQL):-- Step#1:-** Goto MongoDB Download Center.

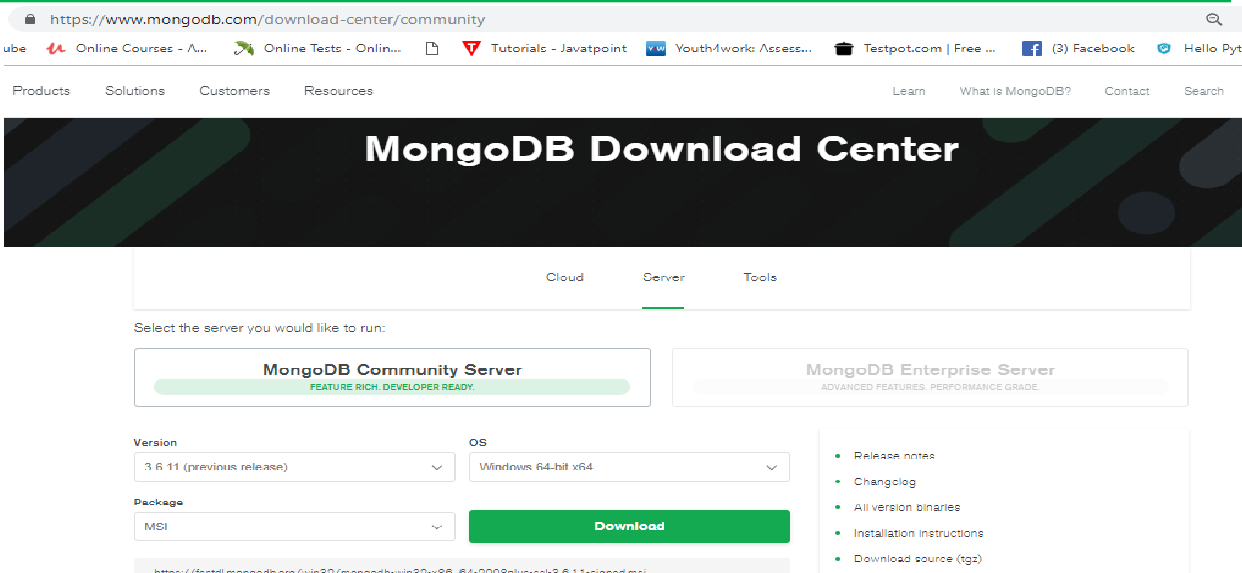
<https://www.mongodb.com/download-center/community>

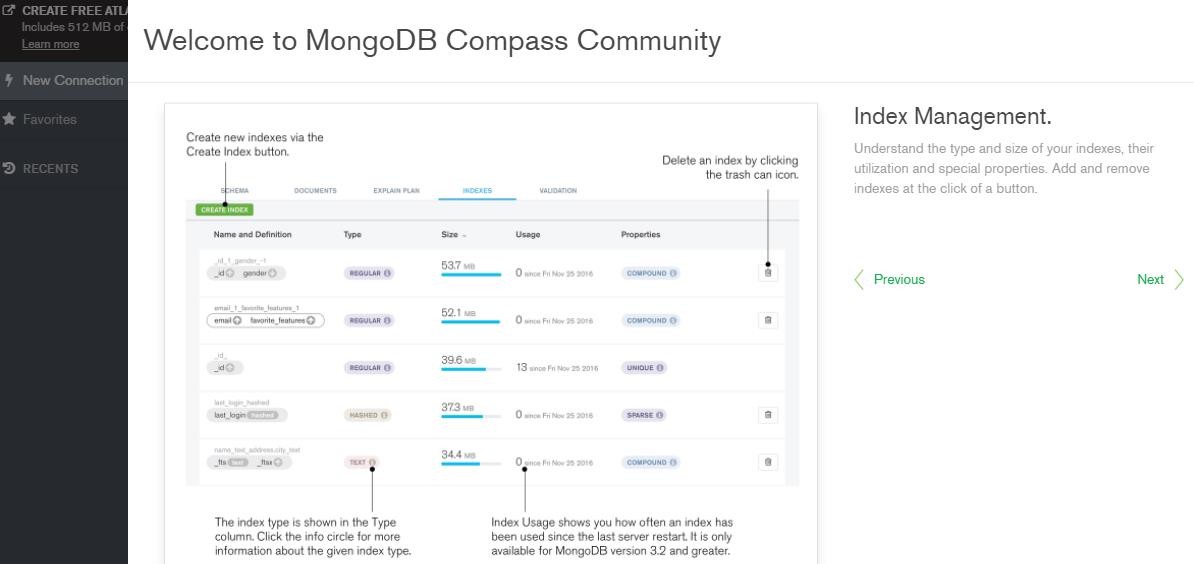
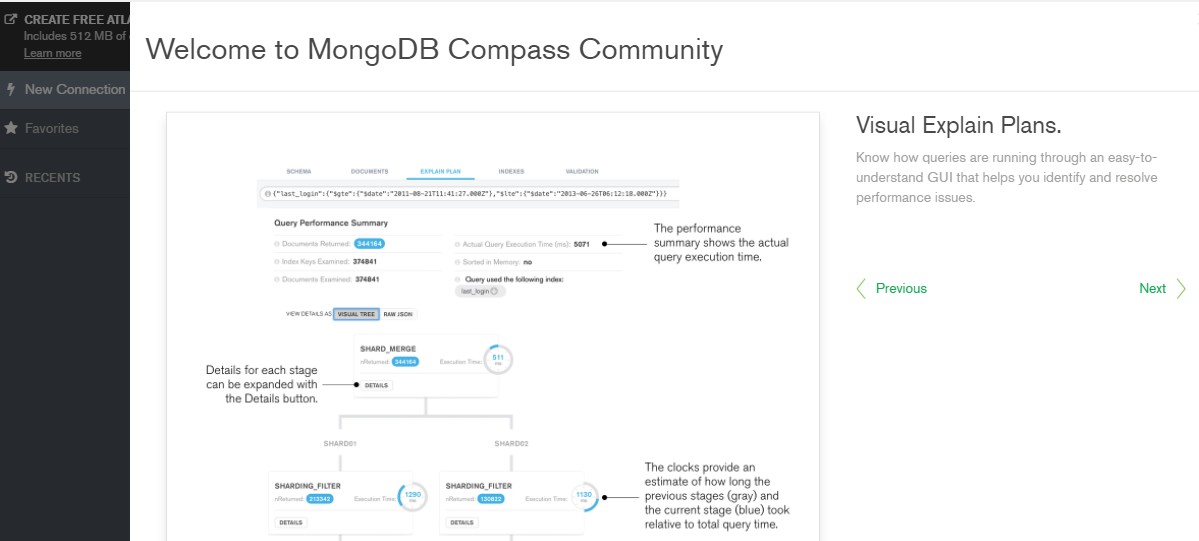
**Step#2:-** Choose “Server” option and select details OS, Version and package (MSI) then click on download.

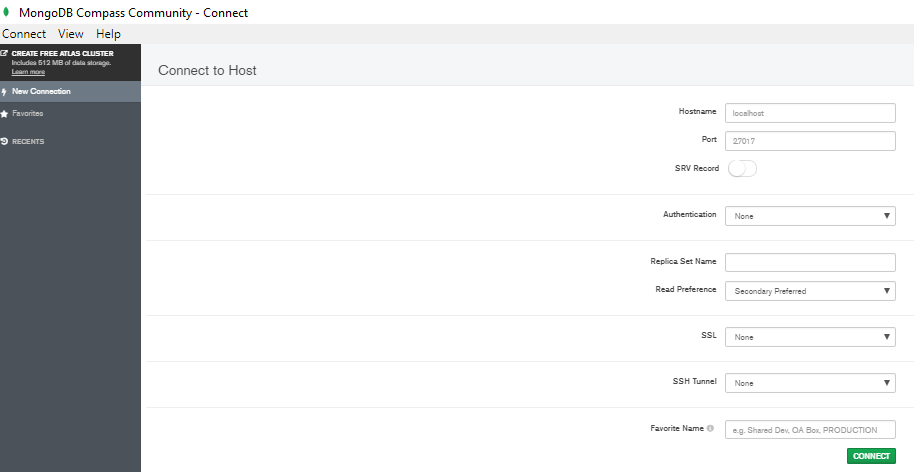
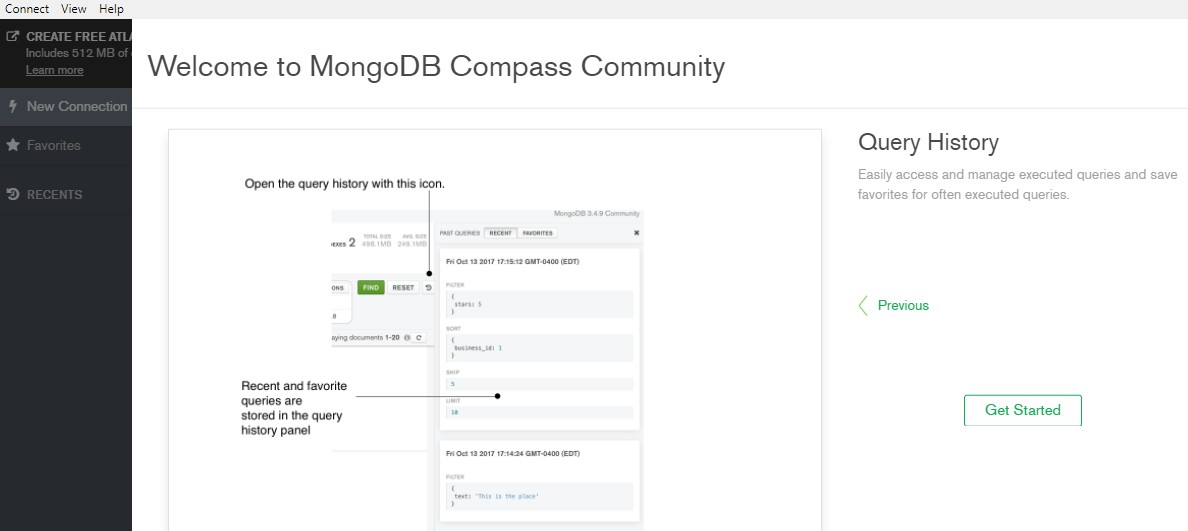
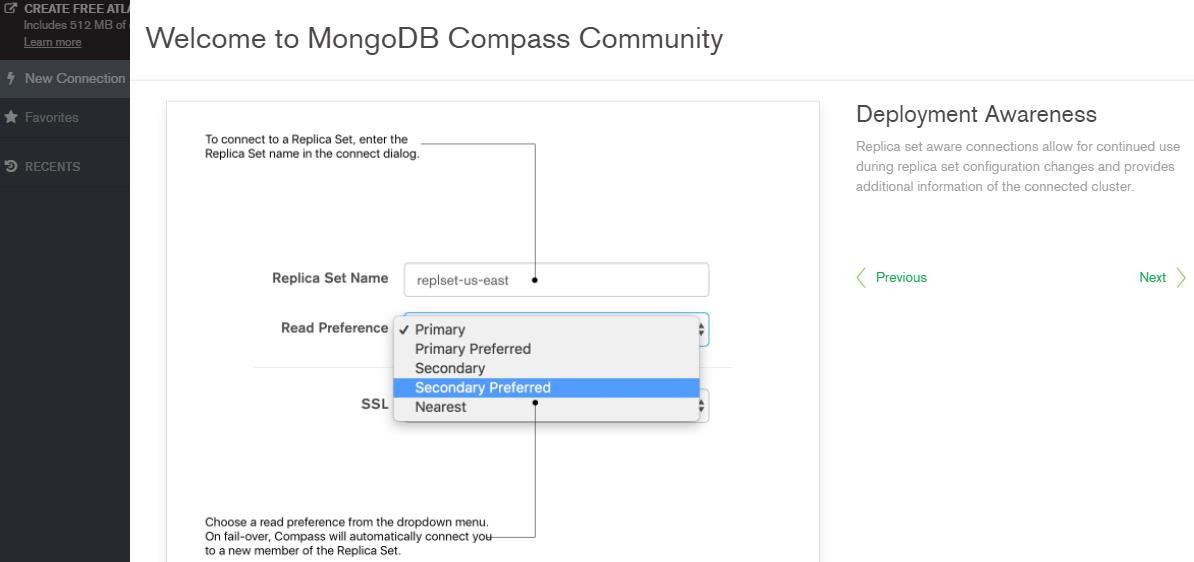
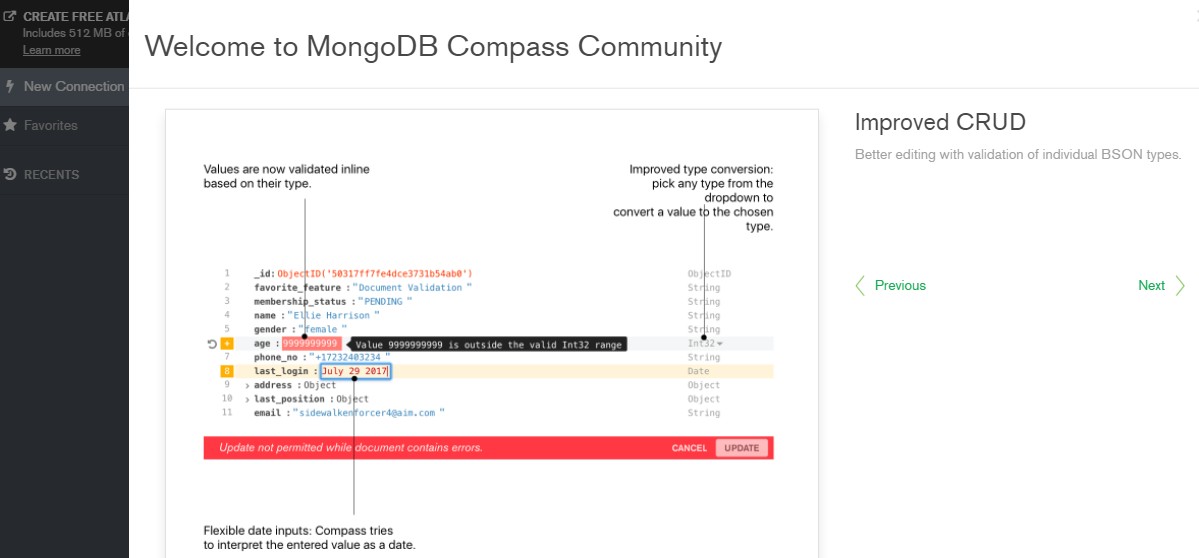
Details:

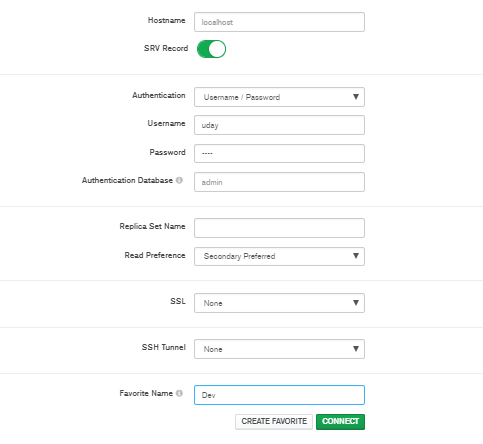
Version : 3.6 or 3.4

Type : MSI (Do not use ZIP)

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

=>While Installing Full (Complete) Type.



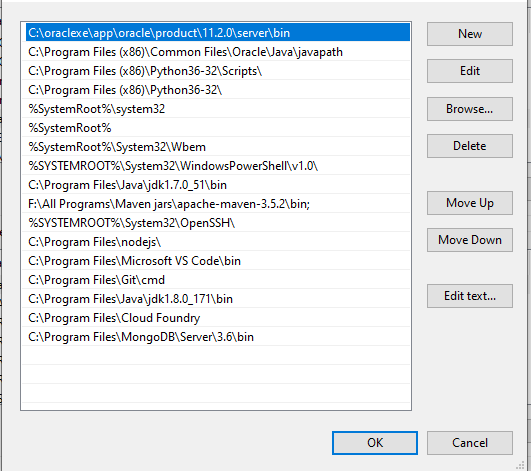


=>Open Notepad and Type below command

->mongod > save with .bat Ex:-- “mongo-server.bat”

->mongo > save with .bat Ex:--“mongo-client.bat”

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



=>Copy location of MongoDB installed address till bin folder Ex:- C:\Program Files\MongoDB\Server\3.6\bin

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

**Step#5:-** Create service (server) folder in C:/ drive looks like “C:/data/db”. NOTE:-- Here folder name can be any things.

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

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

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

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

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

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

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

\*\* It will start client.

## MongoDB commands:--

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

=>To insert one JSON row into Collection.

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

=>To get JSON rows from collection.

=>Delete one Collection object

* db.book.remove({"bookCode":"JAVA"})



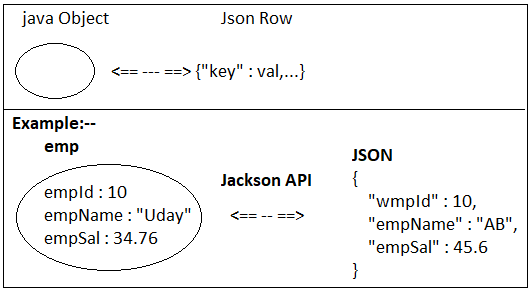
\*\*More Commands goto <https://docs.mongodb.com/manual/reference/command/>

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

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

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

=>ORM concepts says



* 1. One class = one table.
  2. One variable = one column.
  3. One Object = one row.

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

=>Collection holds data in JSON format.

1. One class = One collection.
2. One Object = one JSON Row.

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

## ORM (SQL):--

Class Table (Model) (DB)

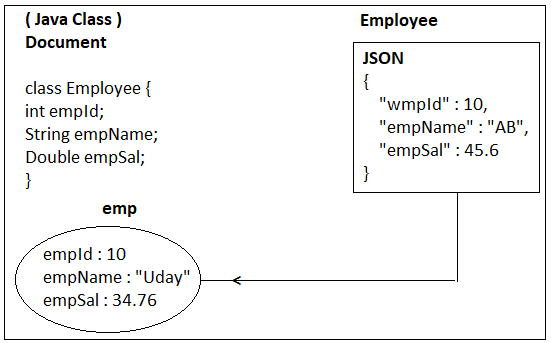
Object <== -- ==> Row

## (NoSQL):--

Class Collection (Document) (DB)

Object <== -- ==> JSON Rows

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



<dependency>

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

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

</dependency>

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

<dependency>

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

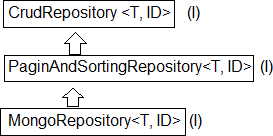
<artifactId>de.flapdoodle.embed.mongo</artifactId>

</dependency>

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

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

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



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



1. **Embedded MongoDB:--**

**Coding Steps:-- Embedded MongoDB Step#1:-** Create one starter project.

=>File => new => Spring Starter Project

=>Enter name : “SpringBootMangoDBEmbedded”

=>next => Search using “mongo”

=>Choose 2 dependencies

->Spring Data MongoDB

->Embedded MongoDB database

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

**Step#2:-** \*\*\* open pom.xml and remove

<scope>test</scope>

=>line for embedded mongo dependency.

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

(UUID = Universal Unique Identifier).

## Document class:--

**package** com.app.document;

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

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

@Document

**public class** Product { @Id

**private** String id;

**private** Integer prodId; **private** String prodName; **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** Product(String id, Integer prodId, String prodName, Double prodCost) {

**super**(); **this**.id = id;

**this**.prodId = prodId; **this**.prodName = prodName; **this**.prodCost = prodCost;

}

**public** String getId() {

**return** id;

}

**public void** setId(String id) {

**this**.id = id;

}

**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 [id=" + id + ", prodId=" + prodId + ", prodName=" + prodName + ", prodCost=" + prodCost + "]";

}

}

1. **Repository:--** Define one Repository interface that takes 2 generic params : Document and ID types.

**package** com.app.repo;

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

**import** com.app.document.Product;

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

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

**package** com.app.runner;

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

**import** org.springframework.boot.CommandLineRunner; **import** org.springframework.stereotype.Component; **import** com.app.document.Product;

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

@Component

**public class** ProductRunner **implements** CommandLineRunner{

@Autowired

**private** ProductRepository repo;

@Override

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

repo.save(**new** Product(10, "Mobile", 567.8)); repo.save(**new** Product(11, "Laptop", 867.8)); repo.save(**new** Product(12, "Computer", 467.8)); System.***out***.println(" ");

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

}

}

\*\*\* Run Starter class.

## Output:--

**Note:--**

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

Ex:-

@Id

private String id;

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

@Id

private Integer id;

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

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

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

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

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

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

## application.yml :--

spring:

data:

mongodb:

host: localhost port: 27017 database: sathya username: sa password: sa

**Step#1:-** Create one Spring Boot Starter project NAME : SpringBootMongoDBExternal Dependency : Spring Data MongoDB

## MongoDB Dependency:--

<dependency>

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

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

</dependency>

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

**pom.xml:--**

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<project xmlns=*"*[*http://maven.apache.org/POM/4.0.0*](http://maven.apache.org/POM/4.0.0)*"* xmlns:xsi=*"*[*http://www.w3.org/2001/XMLSchema-instance*](http://www.w3.org/2001/XMLSchema-instance)*"*

xsi:schemaLocation=*"*[*http://maven.apache.org/POM/4.0.0*](http://maven.apache.org/POM/4.0.0)[*http://maven.apache.org/xsd/maven-4.0.0.xsd*](http://maven.apache.org/xsd/maven-4.0.0.xsd)*"*>

<modelVersion>4.0.0</modelVersion>

<parent>

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

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

<version>2.1.1.RELEASE</version>

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

</parent>

<groupId>com.app</groupId>

<artifactId>MongoDBExternal</artifactId>

<version>1.0</version>

<name>MongoDBExternal</name>

<description>Spring Boot MongoDB Connectivity Application</description>

<properties>

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

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

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

**application.properties:--** spring.data.mongodb.host=localhost spring.data.mongodb.port=27017 spring.data.mongodb.database=abc

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

**package** com.app.repo;

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

**import** com.app.document.Employee;

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

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

package com.app.runner;

import org.springframework.beans.factory.annotation.Autowired; import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component; import com.app.document.Employee;

import com.app.repo.EmployeeRepo;

@Component

public class MongoDBExternalAllBasicOperation implements CommandLineRunner {

@Autowired

private EmployeeRepo repo;

@Override

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

}

}

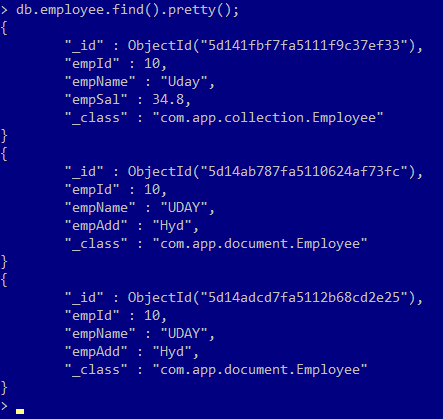
**Execution Process:--** Goto MongoDB install directory till bin folder and open two cmd prompt.

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

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

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

## Console Output:--

1. **Mongo client output:--**
2. **Working with Multiple values of JSON in MongoDB:--**

=>In general JSON key=value combination indicates primitive Data format. Ex:-- {“empId” : 25, “empName”: ”Uday”}

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

* 1. **List/Set/Array :--**

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

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

**application.properties:--** spring.data.mongodb.host=localhost spring.data.mongodb.port=27017 spring.data.mongodb.database=sathya

1. **Document class:-- package** com.app.document; **import** java.util.Arrays; **import** java.util.List;

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

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

@Document

**public class** Book {

@Id //UUID

**private** String id;

**private** String bookCode; **private** String bookAuth; **private** Double bookCost;

//Collection dataType **private** List<String> codes; **private** String[] grades;

**public** Book() {

**super**();

}

**public** Book(String bookCode, String bookAuth, Double bookCost, List<String> codes, String[] grades) {

**super**();

**this**.bookCode = bookCode; **this**.bookAuth = bookAuth; **this**.bookCost = bookCost; **this**.codes = codes; **this**.grades = grades;

}

**public** String getId() {

**return** id;

}

**public void** setId(String id) {

**this**.id = id;

}

**public** String getBookCode() {

**return** bookCode;

}

**public void** setBookCode(String bookCode) {

**this**.bookCode = bookCode;

}

**public** String getBookAuth() {

**return** bookAuth;

}

**public void** setBookAuth(String bookAuth) {

**this**.bookAuth = bookAuth;

}

**public** Double getBookCost() {

**return** bookCost;

}

**public void** setBookCost(Double bookCost) {

**this**.bookCost = bookCost;

}

**public** List<String> getCodes() {

**return** codes;

}

**public void** setCodes(List<String> codes) {

**this**.codes = codes;

}

**public** String[] getGrades() {

**return** grades;

}

**public void** setGrades(String[] grades) {

**this**.grades = grades;

}

@Override

**public** String toString() {

**return** "Book [id=" + id + ", bookCode=" + bookCode + ", bookAuth=" + bookAuth + ", bookCost=" + bookCost + ", codes=" + codes + ", grades=" + Arrays.*toString*(grades) + "]";

}

}

1. **Repository:-- package** com.app.repo;

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

**import** com.app.model.Book;

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