
CHAPTER#3: SPRING BOOT EMAIL

1. Introduction:--

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

=>Spring Boot mail API is defined on top of JAVA Mail API which reduces common lines of code for Email Application.

=>To implement Email Service, we need to provide below keys in application.properties file.

=>Example given with Gmail Server Details.

Application.properties:--

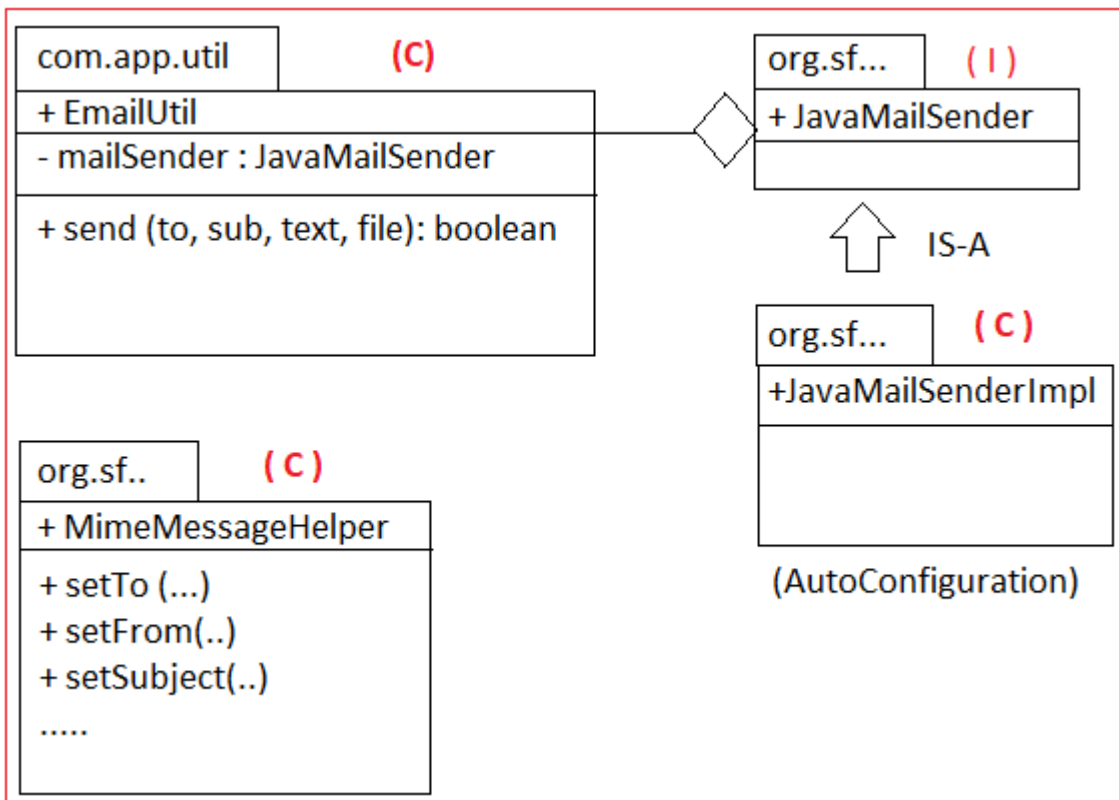
```
spring.mail.host=smtp.gmail.com  
spring.mail.port=587  
spring.mail.username=udaykumar461992@gmail.com  
spring.mail.password=Uday1234  
spring.mail.properties.mail.smtp.auth=true  
spring.mail.properties.mail.smtp.starttls.enable=true
```

application.yml:--

```
spring:  
  mail:  
    host: smtp.gmail.com  
    port: 587  
    username: udaykumar461992@gmail.com  
    password: Uday1234  
    properties:  
      mail:  
        smtp:  
          auth: true  
          starttls:
```

enable: true

UML Diagram:--



Spring Boot AutoConfiguration does,

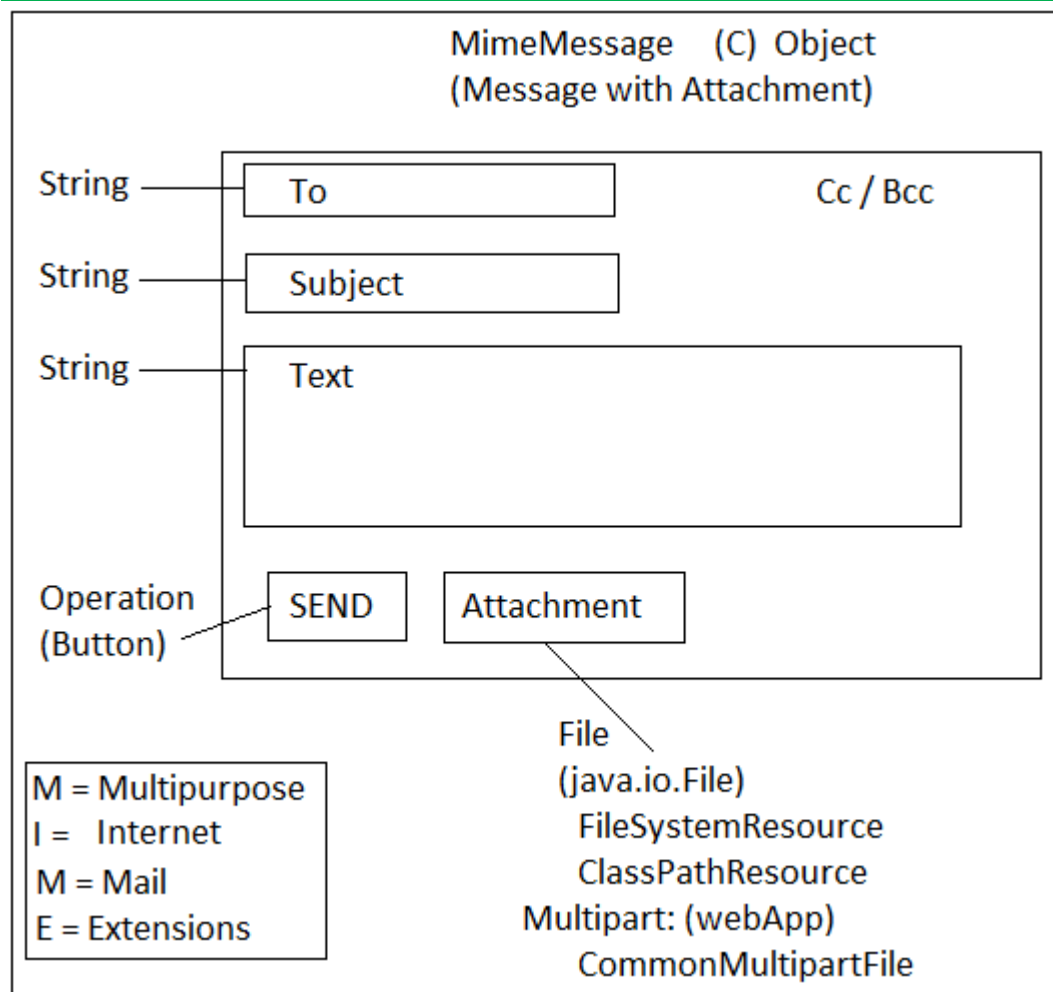
- 1> Get all Required Jars to Project.
- 2> Find for keys properties/yml
(auto-load based on prefix =spring.mail)
- 3> ***Writes Configuration for JavaMailSenderImpl Bean (i.e @Bean)
- 4> Creates Email session with Mail sender.

=Programmer not required defining bean for JavaMailSenderImpl in AppConfig (like Spring f/w).

=>But need to provide all inputs like host, port.. using Properties file.

=>We need to construct one object i.e **"MimeMessage"** which is equals to "New Message in Mail Application".

=>Spring f/w has provided one helper class **"MimeMessageHelper"**, to construct object for **"MimeMessage"**.



=>To Load Files (Attachments) in case of stand-alone mode application in spring or Boot use “**Resource**” impl classes. For Example

a. FileSystemResource :-- This is used to load one file from System drives.
(ex:- d:/images/mydata).

b. ClassPathResource :-- If file is available in src/main/resources folder then use this concept.

Step#1 :-- Create Spring Boot starter Application using dependency : Java Mail Sender

Java Mail Sender Dependency :--

<dependency>

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

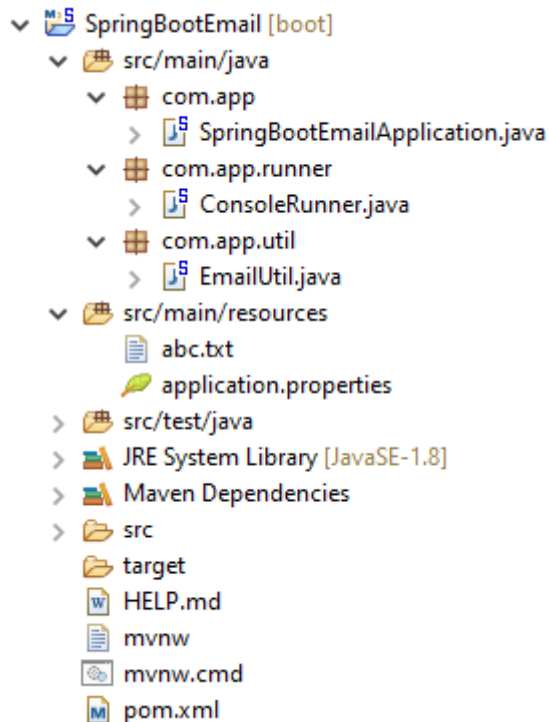
<artifactId>spring-boot-starter-mail</artifactId>

</dependency>

Step#2:-- Provide details in application.properties/yml
host, port = To identify Mail service Provider.
user, pwd = To do login from boot Application.
mail properties = Extra information to server.

Step#3:-- Define Service Utility class

#28. Folder Structure of Spring Boot Mail Service:--



1. application.properties:--

```
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=udaykumar461992@gmail.com
spring.mail.password=Uday1234
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
```

2. EmailUtil.java:--

```
package com.app.util;
import javax.mail.internet.MimeMessage;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.core.io.FileSystemResource;
```

```
import org.springframework.mail.javamail.JavaMailSender;
import org.springframework.mail.javamail.MimeMessageHelper;
import org.springframework.stereotype.Component;

@Component
public class EmailUtil {

    @Autowired
    private JavaMailSender mailSender;

    public boolean send(String to, String subject, String text,
        //String[] cc,
        //String[] bcc,
        FileSystemResource file)
        //ClassPathResource file)
    {
        boolean flag=false;

        try {
            //1. Create MimeMessage object
            MimeMessage message=mailSender.createMimeMessage();

            //2. Helper class object
            MimeMessageHelper helper=new MimeMessageHelper(message,
                file!=null?true:false);

            //3. set details
            helper.setTo(to);
            helper.setFrom("udaykumar461992@gmail.com");
            helper.setSubject(subject);
            helper.setText(text);
            //helper.setCc(cc); //array Inputs
            //helper.setBcc(bcc);

            if(file!=null)
                helper.addAttachment(file.getFilename(),file);
```

```
        //4. send button
        mailSender.send(message);
        flag=true;
    } catch (Exception e) {
        flag=false;
        e.printStackTrace();
    }
    return flag;
}
}
```

3. ConsoleRunner:--

```
package com.app.runner;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.core.io.FileSystemResource;
import org.springframework.stereotype.Component;
import com.app.util.EmailUtil;
```

```
@Component
```

```
public class ConsoleRunner implements CommandLineRunner {
```

```
    @Autowired
```

```
    private EmailUtil util;
```

```
    @Override
```

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

```
{
```

```
        //ClassPathResource file=new ClassPathResource("abc.txt");
```

```
        FileSystemResource file=new FileSystemResource("F:\\Uday Kumar\\Photo.jpg");
```

```
        boolean flag=util.send("udaykumar0023@gmail.com", "AA", "Hello", file);
```

```
        if(flag) System.out.println("SENT");
```

```
        else System.out.println("CHECK PROBLEMS");
```

```
    }
```

```
}
```

Modification in project for Email setup:--

=>To indicate any attachment (file) use concept '**Spring Boot multipart Programming**'.

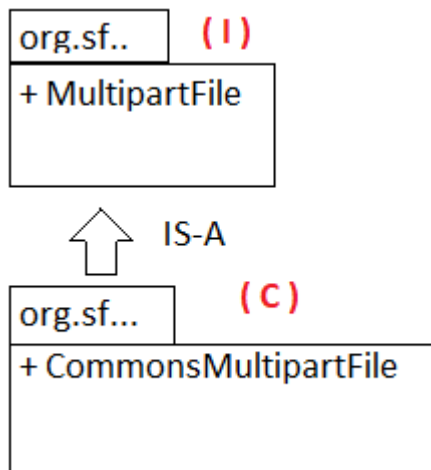
=>Multipart is a concept used to indicate 'java.io.File' with Input/Output Stream operations given.

=>Multipart concept works only for Web Applications.

=>Multipart is **enabled** by default in spring boot.

(spring.servlet.multipart.enabled=true)

"spring.servlet.multipart.max-file-size=10MB".



Code changes in project:--

Step#1:- In model class define Transient variable for email (which never gets stored in DB, only at UI).

Product.java

@Transient

private String email;

Step#2:- Indicate Form has attachment using encoding type

<form: form.... enctype="multipart/form-data">

Ex:-- <form:form action="save" method="POST" modelAttribute="product"
enctype="multipart/form-data">

Step#3:- add two inputs in Register.JSP page

EMAIL: <form:input path="email"/>

DOCUM: <input type="file" name="fileOb">

Step#4:- Copy EmailUtil.java, email properties, dependency (Starter-mail) into Project.

Step#5:- Use EmailUtil in Controller class

ProductController -----< > EmailUtil

Step#6:- Read file (attachment) in controller method (on save) **@RequestParam
MultipartFile fileOb**

Step#7:- call send(..) method from EmailUtil

mailUtil.send(product.getEmail(), "Product added", "Hello User", fileOb);

Step#8:- Change Parameter Type from FileSystemResource to multipartFile in EmailUtil.

package com.app.util;

import javax.mail.internet.MimeMessage;

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

import org.springframework.mail.javamail.JavaMailSender;

import org.springframework.mail.javamail.MimeMessageHelper;

import org.springframework.stereotype.Component;

import org.springframework.web.multipart.MultipartFile;

@Component

public class EmailUtil {

@Autowired

private JavaMailSender mailSender;

public boolean send(String to, String subject, String text,

 //String[] cc,

 //String[] bcc,

 MultipartFile file) //change from FileSystemResource to MultipartFile

 //ClassPathResource file)

{

boolean flag=false;

try {

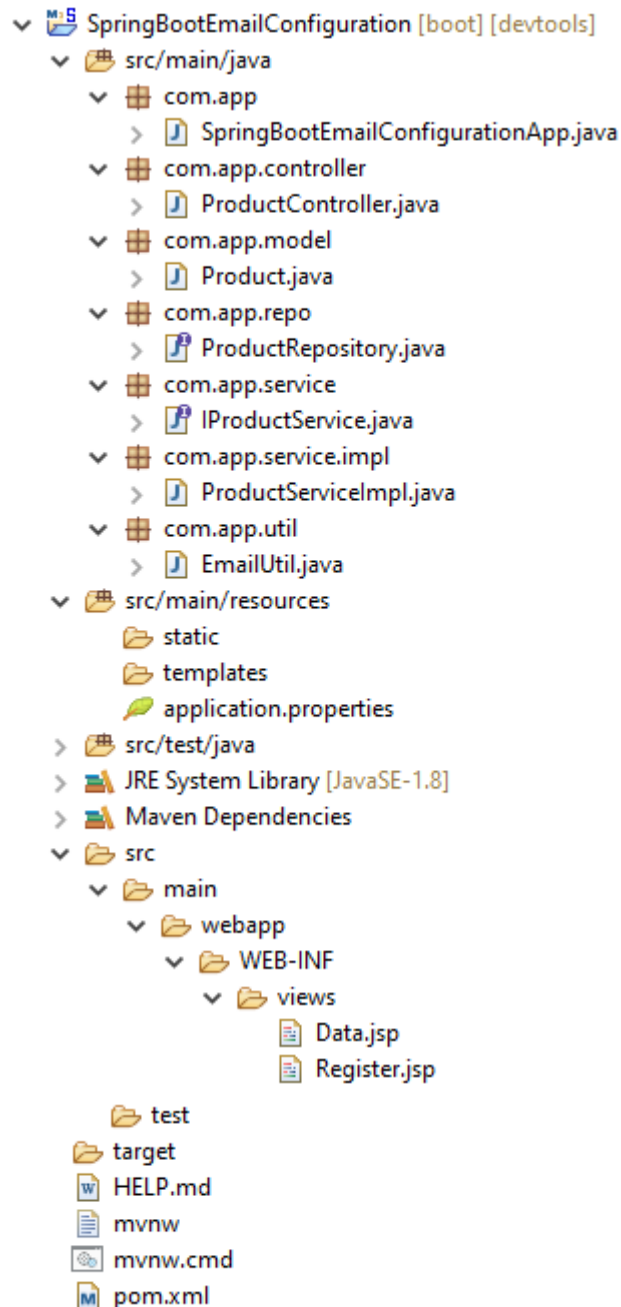
```
//1. Create MimeMessage object
MimeMessage message=mailSender.createMimeMessage();

//2. Helper class object
MimeMessageHelper helper=new MimeMessageHelper(message,
file!=null?true:false);
//3. set details
helper.setTo(to);
helper.setFrom("udaykumar461992@gmail.com");
helper.setSubject(subject);
helper.setText(text);
//helper.setCc(cc); //array Inputs
//helper.setBcc(bcc);
if(file!=null)
helper.addAttachment(file.getOriginalFilename(),file); //changes

//4. send button
mailSender.send(message);

flag=true;
} catch (Exception e) {
flag=false;
e.printStackTrace();
}
return flag;
}
}
```

#29. Folder Structure of EmailConfiguration with Web MVC application:-



application.properties:--

SERVER

server.port=9090

server.servlet.context-path=/myapp

WEB MVC

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

spring.mvc.view.suffix=.jsp

DataSource

spring.datasource.driver-class-name=[oracle.jdbc.driver.OracleDriver](#)

spring.datasource.url=[jdbc:oracle:thin:@localhost:1521:XE](#)

spring.datasource.username=[system](#)

spring.datasource.password=[system](#)

JPA

spring.jpa.show-sql=[true](#)

spring.jpa.hibernate.ddl-auto=[update](#)

spring.jpa.properties.hibernate.dialect=[org.hibernate.dialect.Oracle10gDialect](#)

spring.jpa.properties.hibernate.format-sql=[true](#)

Email

spring.mail.host=[smtp.gmail.com](#)

spring.mail.port=[587](#)

spring.mail.username=[udaykumar461992@gmail.com](#)

spring.mail.password=[Uday1234](#)

spring.mail.properties.mail.smtp.auth=[true](#)

spring.mail.properties.mail.smtp.starttls.enable=[true](#)

1>Model class:--

package com.app.model;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.Table;

import org.springframework.data.annotation.Transient;

import lombok.Data;

@Entity

@Data

@Table(name="producttab")

public class Product {

 @Id

```
@Column(name="pid")
@GeneratedValue
private Integer id;

@Column(name="pcode")
private String code;
@Column(name="pname")
private String name;

@Column(name="pcost")
private Double cost;
@Column(name="pgst")
private Integer gst;

@Column(name="pnote")
private String note;

@Transient
private String email;
}
```

2>ProductRepository:--

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

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

3>IProductService:--

```
package com.app.service;
import java.util.List;
import com.app.model.Product;

public interface IProductService {
```

```
        public Integer saveProduct(Product p);
        public List<Product> getAllProducts();
        public void deleteProduct(Integer id);
        public Product getProductById(Integer id);
    }
```

4>ProductServiceImpl:--

```
package com.app.service.impl;
import java.util.List;
import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import com.app.model.Product;
import com.app.repo.ProductRepository;
import com.app.service.IProductService;
```

@Service

```
public class ProductServiceImpl implements IProductService {
```

```
    @Autowired
```

```
    private ProductRepository repo;
```

```
    @Transactional
```

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

```
        //calculations here..
```

```
        //gstAmt= cost*gst/100
```

```
        //totalAmt=cost+ gstAmt - disc
```

```
        p=repo.save(p);
```

```
        return p.getId();
```

```
    }
```

```
    @Transactional(readOnly= true)
```

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

```
        return repo.findAll();
```

```
    }
```

```
@Transactional
public void deleteProduct(Integer id) {
    repo.deleteById(id);
}

@Transactional(readOnly=true)
public Product getProductById(Integer id) {
    Optional<Product> p=repo.findById(id);
    return p.get();
}
```

```
}
```

5>EmailUtil.java:--

```
package com.app.util;
import javax.mail.internet.MimeMessage;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.mail.javamail.JavaMailSender;
import org.springframework.mail.javamail.MimeMessageHelper;
import org.springframework.stereotype.Component;
import org.springframework.web.multipart.MultipartFile;
```

```
@Component
public class EmailUtil {

    @Autowired
    private JavaMailSender mailSender;

    public boolean send(
        String to,
        String subject,
        String text,
        //String[] cc,
        //String[] bcc,
        MultipartFile file
    )
    {
        boolean flag=false
```

```
try {
    //1. Create MimeMessage object
    MimeMessage message=mailSender.createMimeMessage();

    //2. Helper class object
    MimeMessageHelper helper=new MimeMessageHelper(message,
file!=null?true:false);

    //3. set details
    helper.setTo(to);
    helper.setFrom("udaykumar461992@gmail.com");
    helper.setSubject(subject);
    helper.setText(text);
    //helper.setCc(cc); //array Inputs
    //helper.setBcc(bcc);
    if(file!=null)
        helper.addAttachment(file.getOriginalFilename(),file);

    //4. send button
    mailSender.send(message);

    flag=true;
} catch (Exception e) {
    flag=false;
    e.printStackTrace();
}
return flag;
}
```

6>ProductController:--

```
package com.app.controller;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ModelAttribute;
```

```
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.multipart.MultipartFile;
import com.app.model.Product;
import com.app.service.IProductService;
import com.app.util.EmailUtil;
```

```
@Controller
```

```
@RequestMapping("/product")
```

```
public class ProductController {
```

```
    @Autowired
```

```
    private IProductService service;
```

```
    @Autowired
```

```
    private EmailUtil mailUtil;
```

```
    //1. Show Product Form with Backing Object
```

```
    @RequestMapping("/reg")
```

```
    public String showReg(Model map) {
```

```
        //Form Backing Object
```

```
        Product p=new Product();
```

```
        map.addAttribute("product",p);
```

```
        return "Register";
```

```
    }
```

```
    //2. Read Form Data on click submit
```

```
    @RequestMapping(value="/save",method=RequestMethod.POST)
```

```
    public String saveData(@ModelAttribute Product product,
```

```
    @RequestParam MultipartFile fileOb, Model map)
```

```
    {
```

```
        //call service layer
```

```
        Integer id=service.saveProduct(product);
```

```
        //send email
```

```
        mailUtil.send(product.getEmail(), "Product added", "Hello User", fileOb);
```

```
        map.addAttribute("message", "Product '"+id+"' created!!");
```

```
        //clean Form Backing Object
        map.addAttribute("product", new Product());
        return "Register";
    }

    //3. Fetch all Rows from DB to UI
    @RequestMapping("/all")
    public String showAll(Model map) {
        //fetch all rows from DB
        List<Product> obs=service.getAllProducts();
        //send to UI
        map.addAttribute("list", obs);
        return "Data";
    }

    //4. Delete row based on ID
    @RequestMapping("/delete")
    public String remove(@RequestParam Integer id) {
        //delete row based on ID
        service.deleteProduct(id);
        //response.sendRedirect
        return "redirect:all";
    }

    //5. Show Edit Page
    @RequestMapping("/edit")
    public String showEdit(
        @RequestParam Integer id
        ,Model map) {
        //load object from DB
        Product p=service.getProductById(id);
        //FORM BACKING OBJECT
        map.addAttribute("product",p);
        map.addAttribute("Mode", "EDIT");
        return "Register";
    }
}
```

7>UI Pages:--

a. Register.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1" isELIgnored="false" %>
<%@taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<!DOCTYPE html>
<html><head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h3>WELCOME TO PRODUCT REGISTER</h3>
<form:form action="save" method="POST" modelAttribute="product"
    enctype="multipart/form-data">
<pre>
<c:if test="${'EDIT' eq Mode }">
    ID : <form:input path="id" readonly="true"/><br>
</c:if>
CODE : <form:input path="code"/><br>
NAME : <form:input path="name"/><br>
COST : <form:input path="cost"/><br>
GST : <form:select path="gst">
        <form:option value="5">5%-SLAB</form:option>
        <form:option value="12">12%-SLAB</form:option>
        <form:option value="18">18%-SLAB</form:option>
        <form:option value="22">22%-SLAB</form:option>
        <form:option value="30">30%-SLAB</form:option>
    </form:select>

EMAIL: <form:input path="email"/><br>
DOCUM: <input type="file" name="fileOb"/><br>






NOTE : <form:textarea path="note"/><br>
<c:choose>
```





```
<c:when test="${'EDIT' eq Mode}">
    <input type="submit" value="UPDATE PRODUCT"/>
</c:when>
<c:otherwise>
    <input type="submit" value="CREATE PRODUCT"/>
</c:otherwise>
</c:choose>
</pre>
</form:form>
${message}
</body></html>
```

Execution:-- Run the application and type the below URL in browser.

<http://localhost:9090/myapp/product/reg>

Output:--

 localhost:9090/myapp/product/save

 Gmail  YouTube  Online Courses - A...  Online Tests - Onlin...

WELCOME TO PRODUCT REGISTER

CODE :

NAME :

COST :

GST :

5%-SLAB ▾

EMAIL :

DOCUM:

Choose File

 No file chosen

NOTE :

It is a Good Product.

CREATE PRODUCT

Q>Define on Spring Boot web application using Email service.

#1:- Design one Spring Boot web application using email service.

#2:- On click submit, form data should be converted to model class Object (com.app.model.message).

#3:- Use EmailUtil and send message to do Addr.

#4:- Clean Form and send source message back to UI.

*** Use Spring Validation API to avoid null or empty message (even Mail formats).

Ex:- To address cannot be null Email is invalid format subject is required

Text min 10 Characters. (Validate messages)

| | |
|-------------|--------------------|
| To | Cc / Bcc |
| Subject | |
| Text | |
| File | Choose file |
| SEND | Attachment |

CHAPTER#4 SPRING BOOT BATCH PROCESSING

1. Introduction:--

Batch:-- Multiple Operations are executed as one Task or one large/Big Task executed step by Step.

=>Every task is called as “**JOB**” and sub task is called as “**STEP**”.

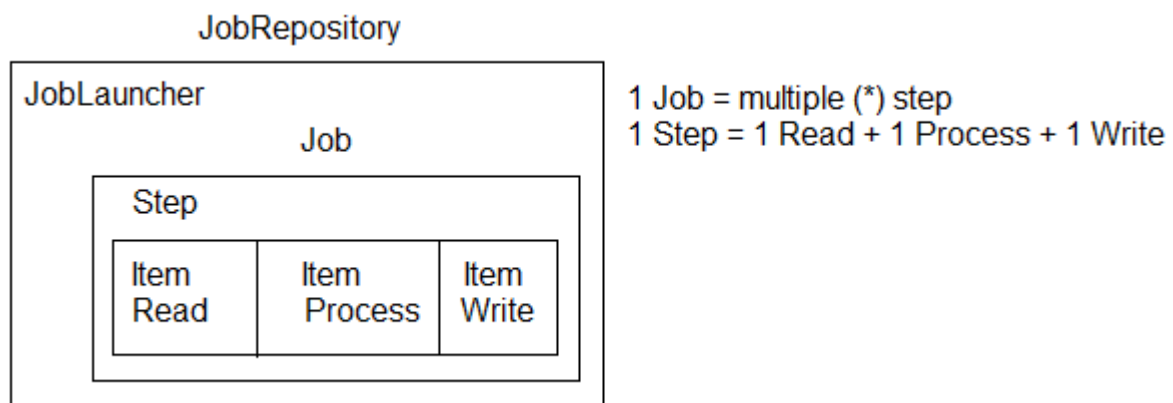
=>One JOB can contain one or more Steps (Step can also called as Sub Task).

=>One Step contains.

- a. Item Reader (Read data from Source).
- b. Item Process (Do calculations and logics/operations etc...).
- c. Item writer (Provide output to next Step or final output).

=>JOBS are invoked by JobLauncher also known as JobStarter.

=>JOB, Steps and Lanuncher details must be stored in JobRepository (Config file).



Step Implementation:--

=>In a Job (work) we can define one or multiple steps which are executed in order (step by step).

=>Job may contain 1 step, job may contain 2 step..... job may contains many Steps so, finally 1-job = * (many) Step.

=>Every step having 3 execution stages

- a. ItemReader<T>
 - b. ItemProcessor<I, O>
 - c. ItemWriter<T>.
-

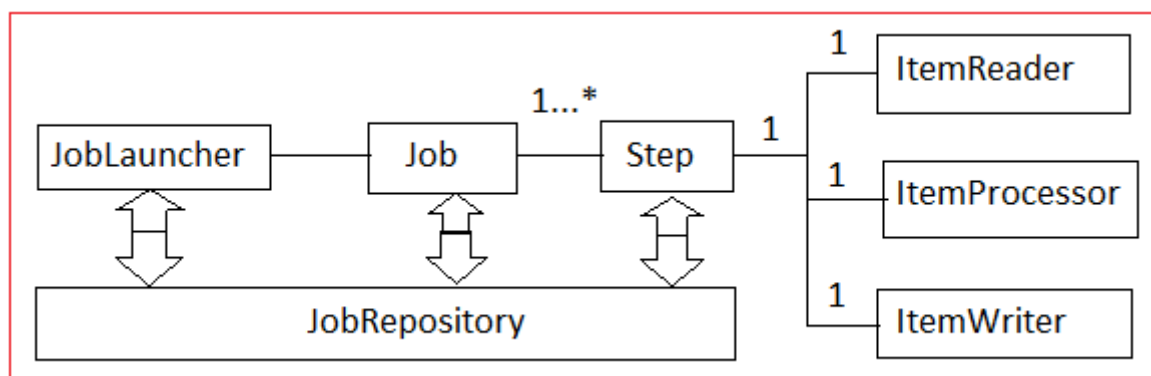
a>ItemReader<T> :-- It is used to read data (**Item**) from source (**Input location**) as input to current Step. A source can be File, DB, MQ, (Message Queues), Networks, XML, JSON... etc.

b>ItemProcessor<I, O> :-- It is used to process input data given by Reader and returns in Modifier (or same) format of data.

c>ItemWriter<T> :-- It will read bulk of Items from Processor at a time and writes to one destination. Even Destination can be a File (ex: DB, Text File, CSV, EXCEL, XML...etc).

=>To start Job, we need one **JobLauncher**... which works in general way or Scheduling based.

=>Here details like: Job, Steps, Launcher... are store in one memory which is called as **JobRepository** [Ex: H2DB or MySQL, DB ... any one DB].



NOTE:--

1. An Item can be String, Array, Object of any class, Collection (List/Set....).
 2. ItemReader will read one by one Item from Source. For example Source has 10 items then 10 times ItemReader will be executed.
 3. ItemReader GenericType must match with ItemProcessor Input GenericType.
 4. ItemProcess will read item by item from Reader and does some process (calculate, convert, check conditions, and convert to any class Object etc...).
 5. ItemProcessor will provide output (may be same as Input type) which is called as Transformed Type.
 6. ItemWriter will collect all output Items into one List type from Processor at a time (Only one time).
 7. ItemWriter writes data to any destination.
-

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

Step Execution Flow :--

=>Step contains 3 interfaces (impls classes) given with generic types.

- ItemReader<T> : (T= Input Data Type)
- ItemProcessor<I, O> : (I must match with Reader T type and O must match with writer T type).
- ItemWriter<T> : (T = Type after processing)

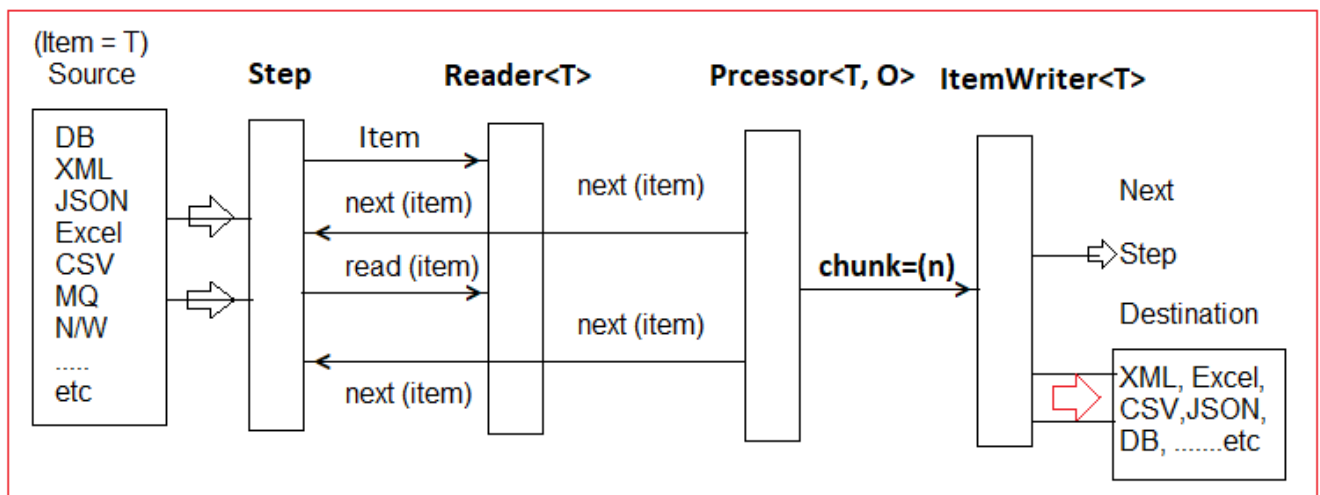
=>Here T/I/O can be String Object of any class or even collection (List, Set...).

=>Here ItemReader reads data from source with the helps of steps.

=>Reader and Processor are executed for every item, but writer is called based on chunk size.

Ex:-- No. of Items = 200, chunk=50 then Reader and Processor are executed 200 times but writer is called one time after 50 items are processed (Here 4 times executed).

=>ItemWriter writes data to destination with the help of step.



Spring Boot Batch coding Files and Steps:-

Batch programming is implemented in 3 stages.

1. Step Creation
2. Job Creation
3. Job Executions

1. Step Creation:- This process is used to construct one (or more) Step(s).

Ex:-- Step1, Step2, Step3..

=>here, Step is interface, It is constructed using **"StepBuilderFactory (C)"** with 3 inputs taken as (interfaces Impl class object) :

- a>ItemReader<T>
- b>ItemProcessor<I, O>
- c>ItemWriter<T>

=>Programmer can define above interfaces Impl classes or can be also use exist (pre-defined) classes.

=>Reader, Writer, Processor is functional interfaces. We can define them using Lambda Expression and methods references even.

2>Job Creation:-- One job is collection of Steps executed in order (one by one).

=>Even job may contain one Step also. Here, job is interface which is constructed using **"JobBuilderFactory <C>"** and Step (I) instances.

=>To execute any logic before or after job, define Impl class for **"JobExecutionListener (I)"** having method

Like: beforeJob(...) : void and
afterJob(...) : void

=>Here Listener is optional, It may be used to find current status of Batch (Ex: COMPLETED, STOPPED, FAILED...) start date and time, end date and time etc.

3>Job Execution:-- Once Steps and job are configured, then we need to start them using **"JobLauncher (I)"** run(...) method.

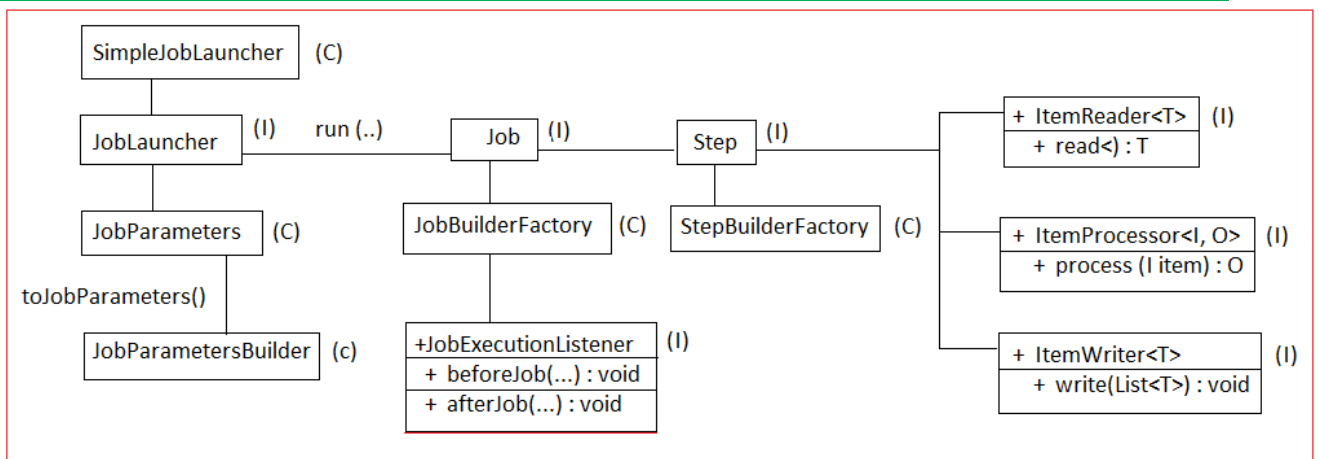
=>This run(...) method takes 2 parameters

- a. Job (I) and
- b. JobParameter (C)

=>Here, JobParameters (C) are inputs given to Job While starting this one.

Ex:- Parameters are : Server Data and Time, Customer Name flags (true/false), Task Name... etc.

=>JobParameters (C) object is created using **"JobParametersBuilder"** and its method toJobParameters().

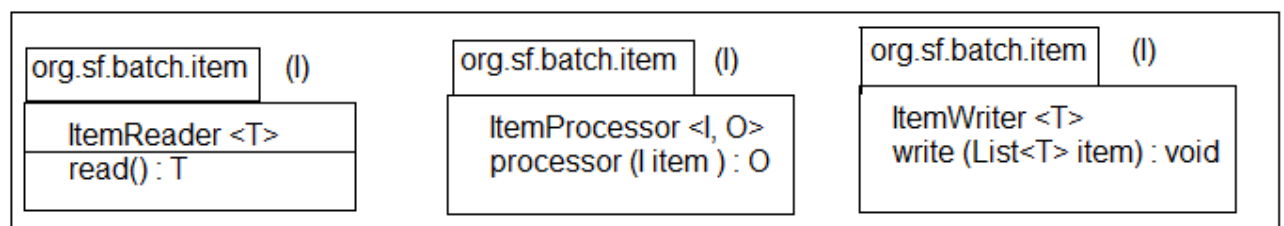


Step :-- One **Step** can be constructed using **StepBuilderFactory (sf)** class by providing name, chunk size, reader, processor and writer.

StepBuilderFactory (sf) :--

| | |
|----------------------------|--|
| sf.get("step1") | =>Provide name of Step. |
| .<String, String> chunk(1) | => No. of Items to be read at a time. |
| .reader (readerObj) | =>Any Impl class of ItemReader<T> (I) |
| .processor(processorObj) | =>Any Impl class of itemProcessor <I, O> |
| .writer(writerObj) | =>Any Impl class of ItemWriter <T> (I) |
| .build(); | =>Convert to step (impl class) Object. |

UML Notation :--



JobBuilderFactory (C) :--

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

=>Job (I) Construction flow

JobBuilderFactory jf :--

| | |
|--------------------------------|--------------------------|
| Jf.get("jobA") | =>Job Name |
| .incremental(runIdIncrementar) | =>Incrementer |
| .listener (jobExListener) | =>Job Execution Listener |
| .start (stepA | =>First Step |

| | |
|---|-------------------|
| .next (stepB) .next (stepC) .next (stepD) | =>Steps in Order |
| .build(); | =>Create job Type |

JobExecutionListener (I):--

=>This Interface is provided Spring Batch f/w, which gets called automatically for our Job.

=>For one job – one Listener can be configured.

=>It is an Interface which has provided two abstract methods.

a. beforeJob (JobExecution) : void

b. afterJob (JobExecution) : void

=>If we write any impl class for above Listener (I) we need to implement two abstract method in our class.

=>Some times we need only one method in our class file afterJob() method is required. Then go for JobListenerAdapter(C) which has provided default impl logic for both **beforeJob; and afterJob();** methods.

=>JobExecution is a class which is used to find current job details like jobParameters, BatchStatus, stepExecutions etc...

JobExecutionListener (I)



IS-A

JobListenerAdapter (C)



IS-A

MyJobListener (C)

=>***BatchStatus is an enum having possible values : COMPLETED, STARTING, STARTED, STOPPING, STOPPED, FAILED, ABANDONED, UNKNOWN.

Q>What are possible Job Status/Batch Status in Batch Programming?

A> All possible status in Batch Execution are given as enum “**BatchStatus**”. Those are

| | | |
|---|-----------|---------------------------|
| 1 | COMPLETED | =>Successfully done. |
| 2 | STARTING | =>About to call run(...). |
| 3 | STARTED | =>Entered into run(...). |

| | | |
|---|-----------|---------------------------------|
| 4 | STOPPED | =>Abort & come out of run(...). |
| 5 | STOPING | =>Run(...) method finished. |
| 6 | FAILED | =>Exception in run(...). |
| 7 | ABANDONED | =>Stopped by External service. |
| 8 | UNKNOWN | =>Unable to provide. |

```

org.springframework.batch.core (I)
+ jobExecutionListener
+ beforeJob (JobException) : void
+ afterJob (JobExecution) : void

```

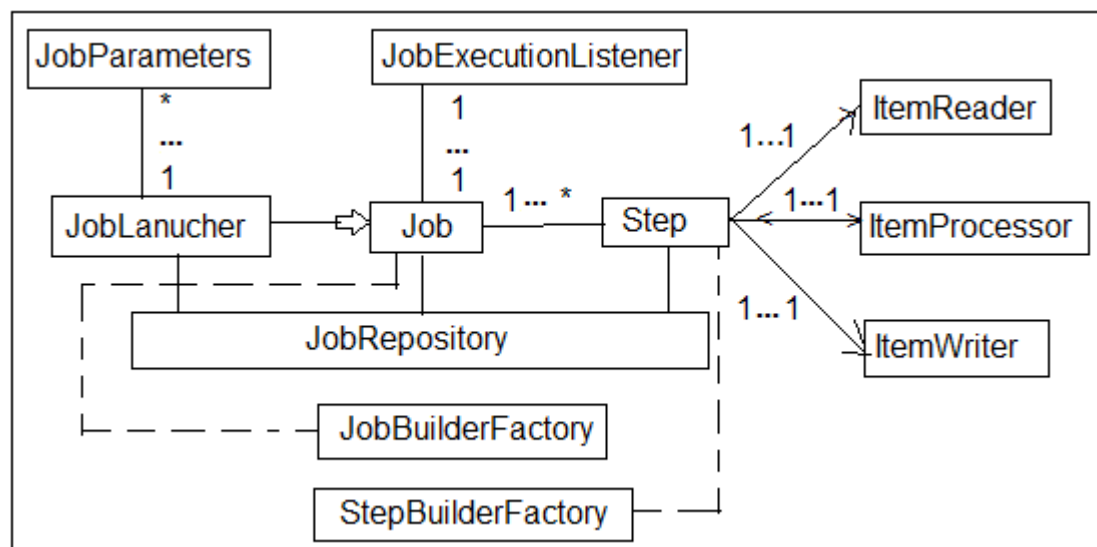
****JobLauncher (I) :-**

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

=>This Interface is having method run (Job, JobParameters).

=>This Interface object is created by JobParametersBuilder which holds data in Map <key, Value> style.

Final Spring Boot Batch Implementation Diagram:--



Step#1:- Define one Spring Starter Project and select **Spring Batch** or else add below dependencies.

```
<dependency>
```

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

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

```
</dependency>
```

Step#2:- Define Impl classes for ItemReader, ItemProcessor and ItemWriter
Ex: Reader (C), Processor (C), Writer (C).

Step#3:- Define one class for Batch Configuration.

a>Create Beans for Reader, Processor, Writer.

b>Create Bean for Step using StepBuilderFactory

(BatchConfig ----- <>StepBuilderFactory)

c>Create Bean for job using JobBuilderFactory

(BatchConfig ----- <>JobBuilderFactory)

d>Define JobExecutionListener (I) Impl class (It is optional)

Step#4:- Create ConsoleRunner to make job Execution using JobLauncher [run(...) method].

ConsoleRunner----- <>JobLauncher

ConsoleRunner----- <>Job

=>Provider JobParameters using its builder obj.

Step#5:- At Starter class level add annotation: @EnableBatchProcessing

Step#6:- add below key in application.properties

spring.batch.job.enabled=false

=>By default Starter class executes Job one time on startup application and even JobLauncher is executing another time. To avoid starter execution by starter class add above key as false value.

Coding order:--

1. Reader
2. Processor
3. Writer
4. Step Configuration using StepBuilderFactory
5. JobExecutionListener
6. Job Config –job BuilderFactory
7. JobParameters using JobParametersBuilder
8. JobLauncher using CommandLineRunner
9. ** Add key in application.properties

Spring.batch.job.enabled=false

=>To avoid execution of job multiple times (by Starter class)

#30. Folder Structure of SpringBoot batch Programing:--

- ▼ SpringBootBatch [boot] [devtools]
 - ▼ src/main/java
 - ▼ com.app
 - > SpringBootBatchApp.java
 - ▼ com.app.batch.config
 - > BatchConfig.java
 - ▼ com.app.batch.listener
 - > MyJobListener.java
 - ▼ com.app.batch.processor
 - > DataProcessor.java
 - ▼ com.app.batch.reader
 - > DataReader.java
 - ▼ com.app.batch.runner
 - > MyJobLauncher.java
 - ▼ com.app.batch.writer
 - > DataWriter.java
 - ▼ src/main/resources
 - application.properties
 - > src/test/java
 - > JRE System Library [JavaSE-1.8]
 - > Maven Dependencies
 - > src
 - target
 - HELP.md
 - mvnw
 - mvnw.cmd
 - pom.xml

application.properties:--

#Disable this otherwise job executed one time by SpringBoot on startup

And also one more time by our launcher

spring.batch.job.enabled=false

spring.batch.initialize-schema=always

spring.datasource.driverClassName=oracle.jdbc.driver.OracleDriver

spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe

spring.datasource.username=system

spring.datasource.password=system

1>DataReader:--

```
package com.app.batch.reader;
```

```
import org.springframework.batch.item.ItemReader;
```

```
import org.springframework.batch.item.NonTransientResourceException;
```

```
import org.springframework.batch.item.ParseException;
```

```
import org.springframework.batch.item.UnexpectedInputException;
```

```
import org.springframework.stereotype.Component;
```

@Component

public class DataReader implements ItemReader<String> {

String message[] = {"hi","hello","hoe"};

int index;

@Override

public String read() throws Exception, UnexpectedInputException,
ParseException, NonTransientResourceException {

if(index < message.length)

{

return message[index++];

} else {

index=0;

}

return null;

}

}

2. DataProcessor:--

package com.app.batch.processor;

import org.springframework.batch.item.ItemProcessor;

import org.springframework.stereotype.Component;

@Component

public class DataProcessor implements ItemProcessor<String, String> {

@Override

public String process(String item) throws Exception {

return item.toUpperCase();

}

}

3. DataWriter:--

package com.app.batch.writer;

import java.util.List;

import org.springframework.batch.item.ItemWriter;

import org.springframework.stereotype.Component;

@Component

public class DataWriter implements ItemWriter<String> {

```
        @Override
        public void write(List<? extends String> items) throws Exception {
            for(String item:items) {
                System.out.println(item);
            }
        }
    }
}
```

4. BatchConfig.java:--

```
package com.app.batch.config;
import org.springframework.batch.core.Job;
import org.springframework.batch.core.JobExecutionListener;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.configuration.annotation.
    EnableBatchProcessing;
import org.springframework.batch.core.configuration.annotation.JobBuilderFactory;
import org.springframework.batch.core.configuration.annotation.StepBuilderFactory;
import org.springframework.batch.core.launch.support.RunIdIncrementer;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import com.app.batch.listener.MyJobListener;
import com.app.batch.processor.DataProcessor;
import com.app.batch.reader.DataReader;
import com.app.batch.writer.DataWriter;
```

```
@Configuration
```

```
@EnableBatchProcessing
```

```
public class BatchConfig {
```

```
    @Autowired
```

```
    private JobBuilderFactory jobBuilderFactory;
```

```
    @Autowired
```

```
    private StepBuilderFactory stepBuilderFactory;
```

```
    @Bean
```

```
    public Job jobA() {
```

```
        return jobBuilderFactory.get("jobA")
```

```
            .incrementer(new RunIdIncrementer())
```

```
            .listener(listener())
```

```
            .start(stepA())
```

```
            //.next(step)
```

```
        .build();
    }
    @Bean
    public Step stepA() {
        return stepBuilderFactory.get("stepA")
            .<String, String>chunk(50)
            .reader(new DataReader())
            .processor(new DataProcessor())
            .writer(new DataWriter())
            .build();
    }
    @Bean
    public JobExecutionListener listener() {
        return new MyJobListener();
    }
}
```

5. MyJobListener.java:--

```
package com.app.batch.listener;
import org.springframework.batch.core.JobExecution;
import org.springframework.batch.core.JobExecutionListener;
import org.springframework.stereotype.Component;

@Component
public class MyJobListener implements JobExecutionListener {

    @Override
    public void beforeJob(JobExecution jobExecution) {
        System.out.println(jobExecution.getStartTime());
        System.out.println(jobExecution.getStatus());
    }
    @Override
    public void afterJob(JobExecution jobExecution) {
        System.out.println(jobExecution.getEndTime());
        System.out.println(jobExecution.getStatus());
    }
}
```

6. MyJobLauncher.java:--

```
package com.app.batch.runner;
import org.springframework.batch.core.Job;
import org.springframework.batch.core.JobParameters;
import org.springframework.batch.core.JobParametersBuilder;
```

```
import org.springframework.batch.core.launch.JobLauncher;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;
```

@Component

```
public class MyJobLauncher implements CommandLineRunner {
```

```
    @Autowired
```

```
    private JobLauncher jobLauncher;
```

```
    @Autowired
```

```
    private Job job;
```

```
    @Override
```

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

```
        JobParameters jobParameters = new JobParametersBuilder()
```

```
            .addLong("time", System.currentTimeMillis())
```

```
            .toJobParameters();
```

```
        jobLauncher.run(job, jobParameters);
```

```
    }
```

```
}
```

Output:--

```
2019-07-03 00:55:17.461 INFO 8488 --- [ restartedMain] o.s.b.c.l.support.SimpleJobLauncher : Job: [SimpleJob: [name=jobA]] launched v
Wed Jul 03 00:55:17 IST 2019
STARTED
2019-07-03 00:55:17.590 INFO 8488 --- [ restartedMain] o.s.batch.core.job.SimpleStepHandler : Executing step: [stepA]
HI
HELLO
HOE
Wed Jul 03 00:55:17 IST 2019
COMPLETED
2019-07-03 00:55:17.666 INFO 8488 --- [ restartedMain] o.s.b.c.l.support.SimpleJobLauncher : Job: [SimpleJob: [name=jobA]] completed
Main Class Executed
2019-07-03 00:55:17.705 INFO 8488 --- [ Thread-9] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown initiated...
2019-07-03 00:55:17.753 INFO 8488 --- [ Thread-9] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown completed.
```

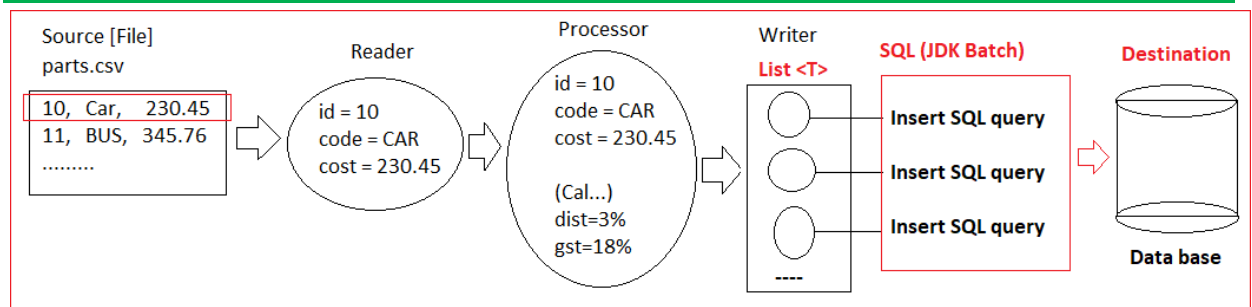
2. Spring Boot Batch Processing : Converting .csv file data to DataBase table:--

=>Consider input data given by csv file is related to products having product id, name, cost, by using one item reader convert .csv file data to Product class object.

=>Define one Processor class to calculate gst (Goods and Service Tax) and discount of product.

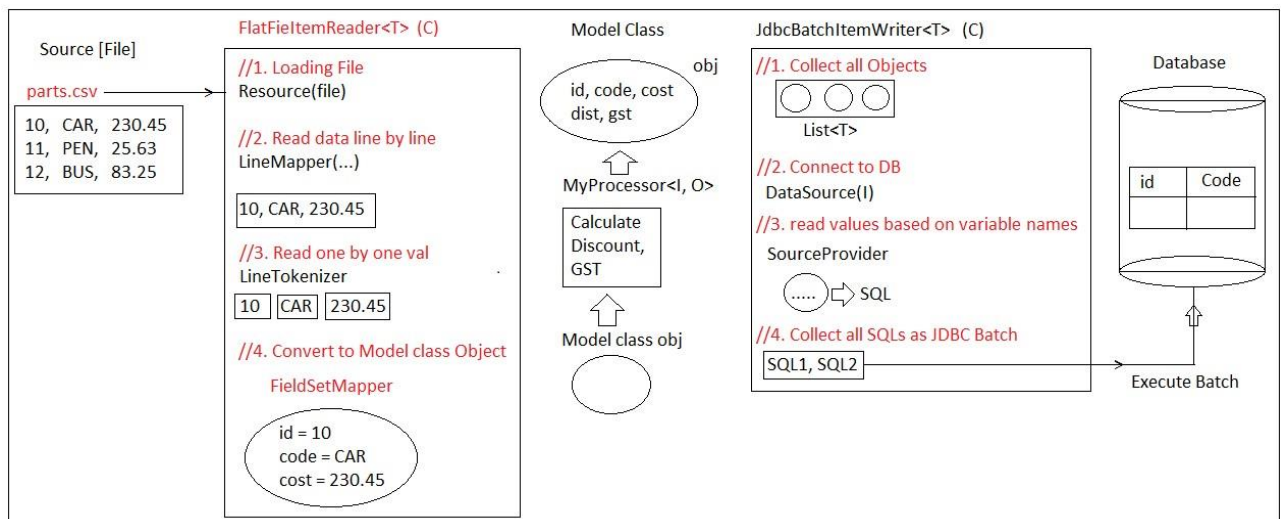
=>Finally Product should have id, name, cost, gst, discount.

=>Use one ItemWriter class to convert one object to one Row in DB table.



- =>In realtime CSV (Comma Separated Values) Files are used to hold large amount of data using symbol / Tokenizer ',' or (It will be , . - , \ , /).
- =>This data will be converted to one Model class(T) object format.
- =>It means one line data(id, code, cost...) converted to one java class object by Reader.
- =>Calculate Discount and GST... etc using Processor class. Processor may return same class (or) different Object (i.e l==O)
- =>Based on chunk(int) size all objects returned by Processor will be collected into one List by Writer.
- =>Every Object in List will be converted into its equal "INSERT SQL..."
- =>Multiple SQLs are converted to one JDBC Batch and sent to DB at a time.
- =>No of calls between Writer and Database depends on chunk size (and no of Items).

Technical Flow for csv to DB using Spring Boot Batch:--



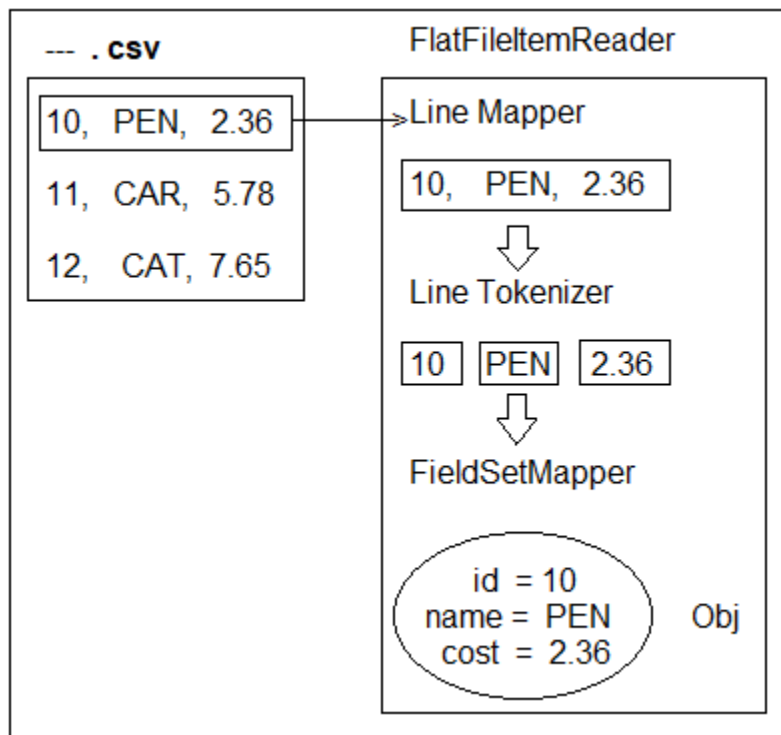
FlatFileItemReader:---

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

DefaultLineMapper:-- It will load one row(/n) at a time as one Line Object.

DelimitedLineTokenizer :-- It will divided one line values into multiple values using any Delimiter (like comma, space, tab, new, line, dash.. etc).
=>Default Delimiter is “,” [Comma].

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



Execution Flow:--

=>Spring Boot Batch f/w has provided pre-defined ItemReaders and ItemWriters.

=>`FlatFileItemReader <T>` is used to load any file (source) as input to read data example .txt, .csv,... etc.

=>It will read one line data based on `LineMapper (\n)`.

=>One Line data is divided into multiple values based on Tokenizer (Delimitar = ,).

=>These values are mapped to one class (T) type object also known as Target.

=>This Target object is input to `ItemProcessor`. After processing object (calculations, Logical checks, data modifications... etc) Processor returns output type Object.

=>`JdbcBatchItemWriter` collects all Items from `ItemProcessor` into `List<T>` based on **chunk (int)** size.

=>Provide one `DataSource` (DB Connection) to communicate with DB Tables.

=>Define one SQL which inserts data into table based on Parameter (Variable Name) SourceProvider.

=>Multiple SQLs are converted to one batch and send to DB table.

*** Here Batch Size = Chunk Size.

Example chunk size = 150 (int value)

=>Chunk indicates maximum Items to be sent to Writer in one network call from Step.

=>For last network call chunk may contain few Items (no. of Items <chunk size).

=>In application.properties add below key-value pairs :

spring.batch.job.enabled=false

spring.batch.initialize-schema=always

=>Here **job.enabled=false** will disable execution of job by one time on app starts by Starter class.

=>**Initialize-schema=always** will allow Spring Batch to communicate DB to hold its Repository details (like Job, Step, current status details...).

=>***In case of Embedded Database initialize-schema not required.

Coding Order for Batch :- csv to ORACLE

1> Model class

2> Processor class***

3> BatchConfig

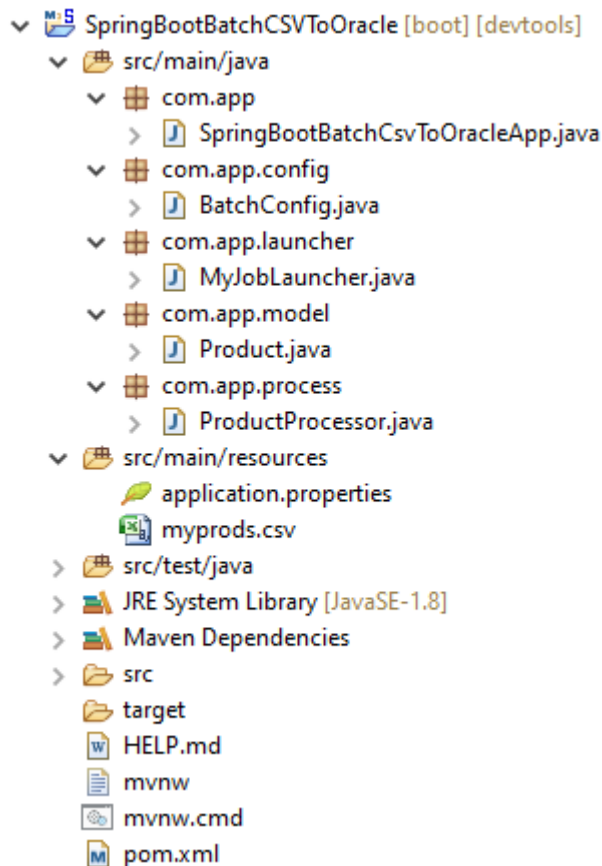
4> ConsoleRunner

5> Application.properties

6> Starter class

7> Pom.xml

#31. Folder Structure of Spring Boot Batch transferring data from csv file to DB:--



application.properties:--

```
spring.batch.job.enabled=false
spring.batch.initialize-schema=always
spring.datasource.driverClassName=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe
spring.datasource.username=system
spring.datasource.password=system
```

1. Model class (Product.java):--

```
package com.app.model;
import lombok.Data;
```

```
@Data
```

```
public class Product {
    private Integer prodId;
    private String prodName;
    private Double prodCost;
    private Double prodGst;
    private Double prodDisc;
}
```

2. ProductProcess.java:--

```
package com.app.process;
import org.springframework.batch.item.ItemProcessor;
import com.app.model.Product;

public class ProductProcessor implements ItemProcessor<Product, Product> {

    @Override
    public Product process(Product item) throws Exception {
        item.setProdGst(item.getProdCost()*12/100.0);
        item.setProdDisc(item.getProdCost()*25/100.0);
        return item;
    }
}
```

3. BatchConfig.java:--

```
package com.app.config;
import javax.sql.DataSource;
import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.configuration.annotation.
EnableBatchProcessing;
import org.springframework.batch.core.configuration.annotation.JobBuilderFactory;
import org.springframework.batch.core.configuration.annotation.StepBuilderFactory;
import org.springframework.batch.core.launch.support.RunIdIncrementer;
import org.springframework.batch.item.ItemProcessor;
import org.springframework.batch.item.ItemReader;
import org.springframework.batch.item.ItemWriter;
import org.springframework.batch.item.database.
BeanPropertyItemSqlParameterSourceProvider;
import org.springframework.batch.item.database.JdbcBatchItemWriter;
import org.springframework.batch.item.file.FlatFileItemReader;
import org.springframework.batch.item.file.mapping.BeanWrapperFieldSetMapper;
import org.springframework.batch.item.file.mapping.DefaultLineMapper;
import org.springframework.batch.item.file.transform.DelimitedLineTokenizer;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.io.ClassPathResource;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
import com.app.model.Product;
import com.app.process.ProductProcessor;
```

@EnableBatchProcessing

@Configuration

public class BatchConfig

{

//STEP

@Autowired

private StepBuilderFactory sf;

@Bean

public Step stepA() {

return sf.get("stepA")

.<Product, Product> chunk(3)

.reader(reader())

.processor(processor())

.writer(writer())

.build();

}

//JOB

@Autowired

private JobBuilderFactory jf;

@Bean

public Job jobA() {

return jf.get("jobA")

.incrementer(new RunIdIncrementer())

.start(stepA())

.build();

}

}

//dataSource -- Creates DB connection

@Bean

public DataSource dataSource() {

DriverManagerDataSource ds = new DriverManagerDataSource();

ds.setDriverClassName("oracle.jdbc.driver.OracleDriver");

ds.setUrl("jdbc:oracle:thin:@localhost:1521:xe");

ds.setUsername("system");

ds.setPassword("system");

return ds;

}

//1. Item Reader from CSV file

/** Code snippet for CSV to ORACLE DB**/

@Bean

```
public ItemReader<Product> reader() {  
    FlatFileItemReader <Product> reader = new FlatFileItemReader<Product>();  
    //--reader.setResource(new    FileSystemResource("d:/abc/products.csv"));  
    //--loading file/Reading file  
    reader.setResource(new ClassPathResource("myprods.csv"));  
    //--read data line by line  
    reader.setLineMapper(new DefaultLineMapper<Product>()){  
    //--make one into multiple part  
    setLineTokenizer (new DelimitedLineTokenizer() { {  
    //-- Store as variables with names  
    setNames ("prodId", "prodName", "prodCost");  
    }});  
    setFieldSetMapper(new BeanWrapperFieldSetMapper<Product>() { {  
    //--Convert to model class object  
    setTargetType (Product.class);  
    }});  
    }});  
    return reader;  
}
```

//2. Item Processor

@Bean

```
public ItemProcessor<Product, Product> processor() {  
    //return new ProductProcessor();  
    return (p)-> {  
        p.setDisc(p.getCost()*3/100.0);  
        p.setGst(p.getCost()*12/100.0);  
        return p;  
    }  
}
```

//3. Item Writer

@Bean

```
public ItemWriter<Product> writer() {  
    JdbcBatchItemWriter<Product> writer = new JdbcBatchItemWriter<>();  
    writer.setDataSource(dataSource ());  
    writer.setItemSqlParameterSourceProvider(new  
    BeanPropertyItemSqlParameterSourceProvider<Product>());  
}
```

```
        writer.setSql("INSERT INTO PRODSTAB(PID, PNAME, PCOST, PGST, PDISC)
VALUES(:prodId, :prodName, :prodCost, :prodGst, :prodDisc)");
        return writer;
    }
}
```

4. MyJobLauncher.java:--

```
package com.app.launcher;
import org.springframework.batch.core.Job;
import org.springframework.batch.core.JobParametersBuilder;
import org.springframework.batch.core.launch.JobLauncher;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

@Component
public class MyJobLauncher implements CommandLineRunner{

    @Autowired
    private JobLauncher jobLauncher;
    @Autowired
    private Job job;

    @Override
    public void run(String... args) throws Exception {
        jobLauncher.run(job,new JobParametersBuilder()
            .addLong("time",System.currentTimeMillis())
            .toJobParameters());
    }
}
```

DB Table:-- Execute below SQL query before execution of program to create table.
SQL>CREATE TABLE prodstab (PID **number** (10), PNAME varchar2 (50), PCOST number, PGST number, PDISC number);

Create a file under src/main/resources folder:--

myprods.csv : under src/main/Resources.

10, PEN, 2.36
11, CAR, 8.8
12, BUS, 99.56

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

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

Task:--

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

Task#1: (JDBC ->CSV)

JdbcCursorItemReader<T>

FlatFileItemWriter<T>

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

Task#2: MongoDB to JSON file

MongoItemReader<T>

JsonFileItemWriter<T>

Task#2: ORACLE to CSV

(ORM -> CSV)

HibernateCursorItemReader<T>

FlatFileItemWriter<T>

Note:-- Every step contains

a> ItemReader

b> ItemProcessor

c> ItemWriter

=>All these are functional Interface (contains only one (1) abstract method.
So Logic can be provided to these interfaces using Lambda Expression.

=>Lambda coding syntax:--

```
(parameters) -> {  
    method body;  
}
```



```

interface Sample { #1
    void m1();
}

class A implements Sample { #2

    public void m1() {
        System.out.println("OK");
    }
}

Sample s = new A(); #3

s.m1(); #4

```



```

interface Sample { #1
    void m1();
}

#2 & #3 = Lambda
Sample s=
    () -> {
        System.out.println("OK");
    }

s.m1(); #4

```

=>Writing Lambda expression

Simple code:--

```

public class MyProcessor implements ItemProcessor<Product, Product> {
    public Product process(Product p) throw Exception {
        double cost=p.getCost();
        p.setDisc(cost*3/100.0);
        p.setGst(cost*12/100.0);
        return p;
    }
}

@Bean
public ItemProcessor<Product, Product>
    process() {
        return new MyProcessor();
    }

```

Lambda Exp:--

```

@Bean
ItemProcessor<Product, Product> process() {
    return (p) -> {
        double cost=p.getCost();
        p.setDisc(cost*3/100.0);
        p.setGst(cost*12/100.0);
        return p;
    };
}

```

***Different ways of creating object and calling method:--

Consider below class:--

```
class Sample {  
  
    Sample() {  
        System.out.println("Constructor");  
    }  
    void show () {  
        System.out.println("Method");  
    }  
}
```

Text class:--

```
public class WayOfCreatingObject  
{  
    public static void main(String[] args) {  
  
        //1. Creating object and calling method  
        Sample s = new Sample();  
        s.show();  
  
        //2. Creating Object and calling method  
        new Sample().show();  
  
        //3. Creating object (add extra code, override methods) and calling method  
        new Sample () {  
            public void show() {  
                System.out.println("NEW LOGIC");  
            }  
        }.show();  
  
        //4. While creating object invoke method  
        new Sample() {  
            //Instance initialize block  
            {  
                show();  
            }  
        };  
    }  
}
```

Ex:-- **Code Extension and instance block:--**

```
package com.app;
```

```
public class Sample {  
    System.out.println("From instance-inside");  
}  
public sample() {  
    System.out.println("From Constructor");  
}  
    //getters setters.. toString  
}
```

```
package Com.app;
```

```
public class Test
```

```
Public static void main {
```

```
    //1. Create object then call method
```

```
    Sample s1 = new Sample();
```

```
    s1.setSid(10);
```

```
    //2. calling method while creating obj
```

```
    Sample s2 = new Sample() { {
```

```
        setSid(10);
```

```
    };
```

```
    //3. Code Extension for current object
```

```
    Sample s3 = new Sample() {
```

```
        void m1() {
```

```
            System.out.println("OK");
```

```
        }
```

```
        {
```

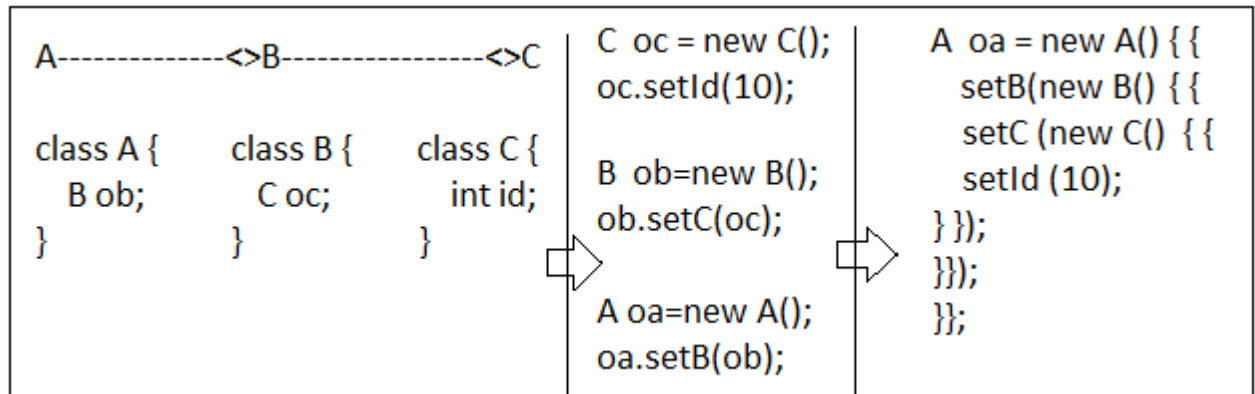
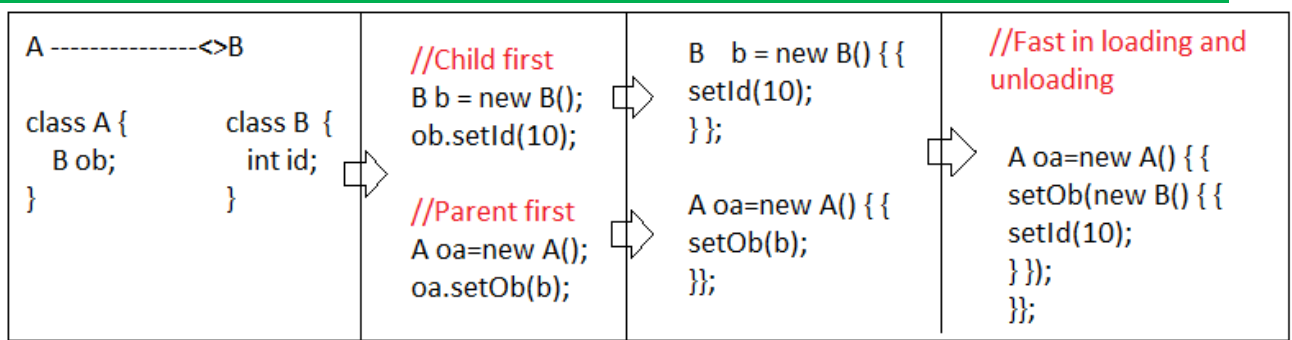
```
            setSid(10);
```

```
            m1();
```

```
        }
```

```
    };
```

```
}
```

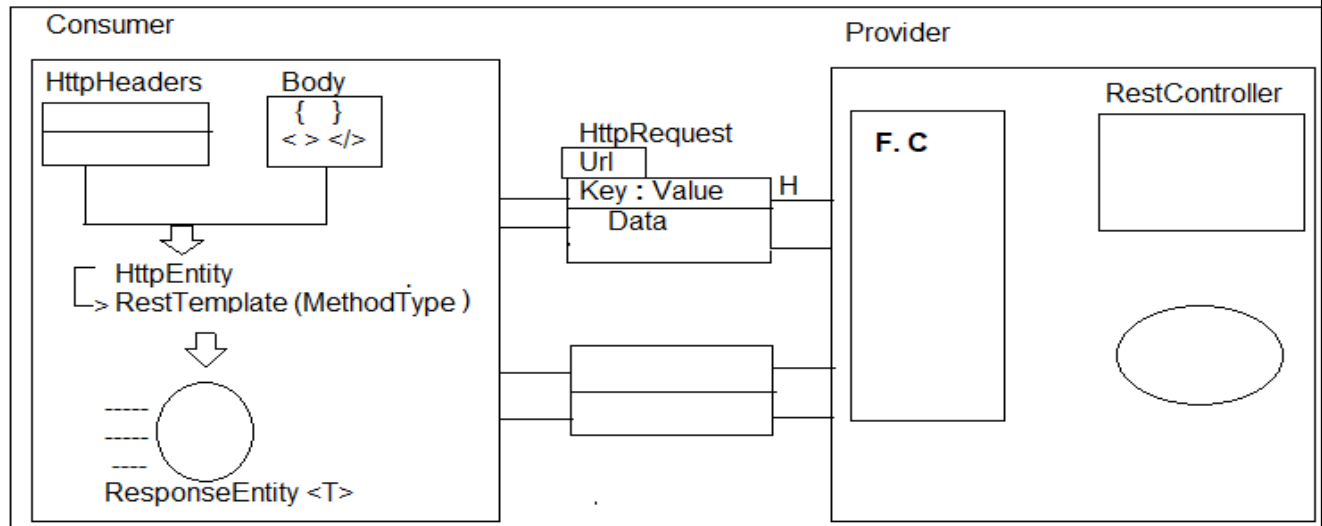


CHAPTER#5: SPRING BOOT REST (PROVIDER & CONSUMER)

1. Introduction:--

=>To implement ReSTful webservices API using simple annotations and Templates Spring boot ReST F/w has been introduced.

=>Even to implement Microservices design Spring boot ReST API is used.



NOTE:--

- Consumer and Provider interchange the data Primitive (String format only), Class Type (Object) and Collections.
- Data will be converted to either **JSON** or **XML** format also known as Global Formats
- String data will be converted to any other type directly.
- ReST Controller supports 5 types of Request Methods Handling. Those are HTTP Method : GET, POST, PUT, DELETE and PATCH.
- Controller class level use annotations `@RestController` (Stereotype), `@RequestMapping` (for URL).
- Controller methods level use annotations.

| TYPE | Annotations |
|--------|-----------------------------|
| GET | <code>@GetMapping</code> |
| POST | <code>@PostMapping</code> |
| PUT | <code>@PutMapping</code> |
| DELETE | <code>@DeleteMapping</code> |
| PATCH | <code>@PatchMapping</code> |

g.>To provide Inputs to Request Methods in RestController, use annotations.

| TYPE | Annotation |
|--------------|-----------------|
| url?key=val | @RequestParam |
| /url/val/ | @PathVariable |
| /url/key=val | @MatrixVariable |
| Http Header | @RequestHeader |
| Http Body | @RequestBody |

***All above annotations works on HttpRequest only, supports reading input data.

h.>By default Matrix Parameter is disabled (may not work properly). To enable this write below code in MVC config file.

Example:--

```
package com.app.controller;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.web.servlet.config.annotation.PathMatchConfigurer;  
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;  
import org.springframework.web.util.UrlPathHelper;
```

@Configuration

```
public class AppConfig implements WebMvcConfigurer
```

```
{
```

```
    @Override
```

```
    public void configurePathMatch(PathMatchConfigurer configure)
```

```
    {
```

```
        UrlPathHelper helper= new UrlPathHelper();
```

```
        helper.setRemoveSemicolonContent(false);
```

```
        configure.setUrlPathHelper(helper);
```

```
    }
```

```
}
```

i.>DispatcherServlet is autoConfigured in Spring Boot with URL Pattern mapped to /.

j.>AutoConfiguration Provided even for @EnableWebMvc,

@ComponentScan(Startsclass), @PropertySource("ClassPath:application.property")

k.>Embedded server.

#1. Spring Boot ReST Provider Example #1

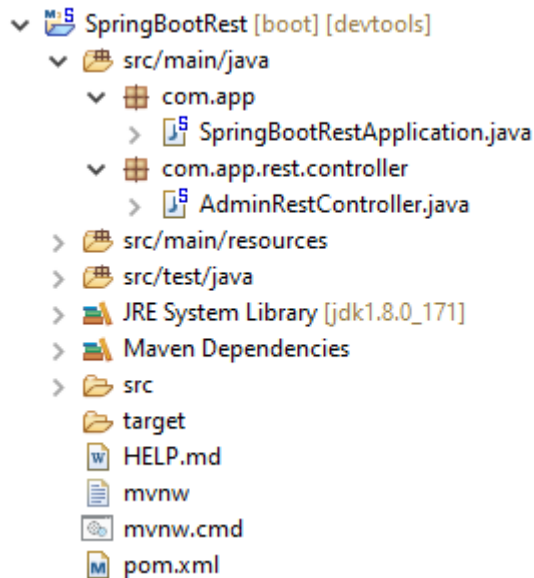
Step#1 :-- Create Starter Project using web and devtools Dependencies.Details are

GroupId : com.app

ArtifactId : SpringBootRestProvider

Version : 1.0

#32. Folder Structure of Spring Boot RestController with possible HttpRequest:--



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

```
package com.app.controller;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PatchMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
```

```
@RestController
```

```
@RequestMapping("/admin") //Optional
```

```
public class AdminRestController {
    @GetMapping("/show")
    public String helloMsgGet () {
        return "Hello From GET";
    }
}
```

```

    }
    @PostMapping("/show")
    public String helloMsgPost () {
        return "Hello From POST";
    }
    @PutMapping("/show")
    public String helloMsgPut () {
        return "Hello From PUT";
    }
    @DeleteMapping("/show")
    public String helloMsgDelete () {
        return "Hello From DELETE";
    }
    @PatchMapping("/show")
    public String helloMsgPatch () {
        return "Hello From PATCH";
    }
}

```

application.properties:--

server.port=2019

server.servlet.context-path=/myApp

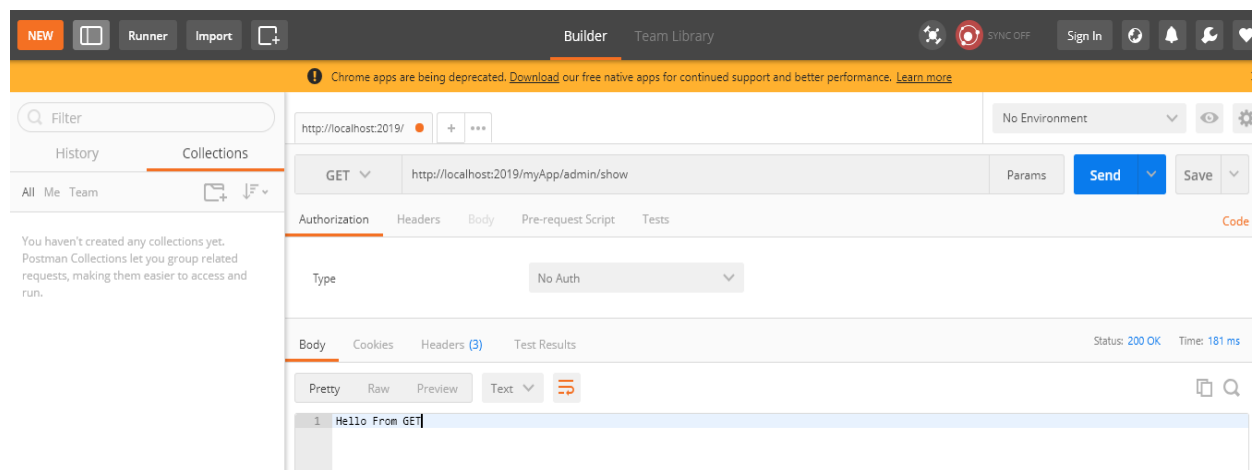
*** Run Starter class and Test Application using POSTMAN

.....POSTMAN SCREEN.....

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

SEND

Postman Screen:--



NOTE:--

a. URL is case-sensitive, same URL can be used at multiple (GET, POST, PUT...) must be different.

b. Difference between PUT and Patch :

PUT :-- It indicates modify complete (full) data, based on Identity (ID-PK).

PATCH :-- It indicates modify partial (few data, based on Identity (ID-PK).

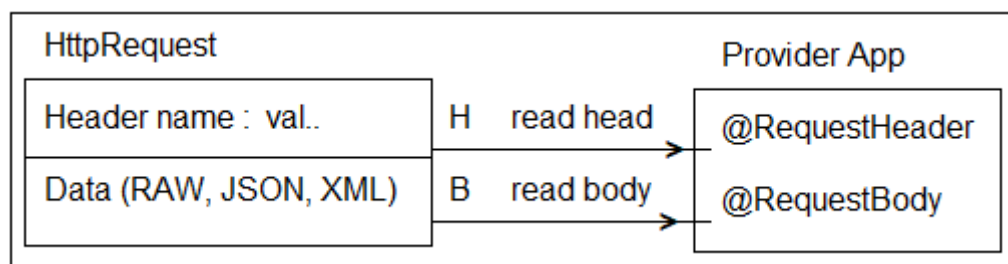
Example#2 Read Data from Http Request (Head and Body).

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

=>To Read any Header Parameter use code

- ❖ @RequestHeader DataType localVariable
- ❖ @RequestHeader (required=false) DataType localVariable
- ❖ @RequestHeader ("key") DataType localVariable

=>To read data from Body (RAW Data) use code @RequestBody

**Code:--**

```
package com.app.rest.controller;
```

```
import org.springframework.web.bind.annotation.PostMapping;
```

```
import org.springframework.web.bind.annotation.RequestBody;
```

```
import org.springframework.web.bind.annotation.RequestHeader;
```

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

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

```
@RestController
```

```
@RequestMapping("/admin")
```

```
public class AdminRestController {
```

```
@PostMapping("/head")
```

```
    public String readHead(@RequestHeader(required=false) String dept,
```

```
    @RequestHeader ("Content-Type") String type, @RequestBody String mydata) {
```

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

```
    } }
```

```

POST Http://localhost:2019/admin/head SEND
  head
  dept      SAMPLE
  Content-Type  application/json

POST Http://localhost:2019/admin/head SEND
  Body
  (*) raw  application/json
{"message" : This is from Request Body}

```

NOTE:--

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

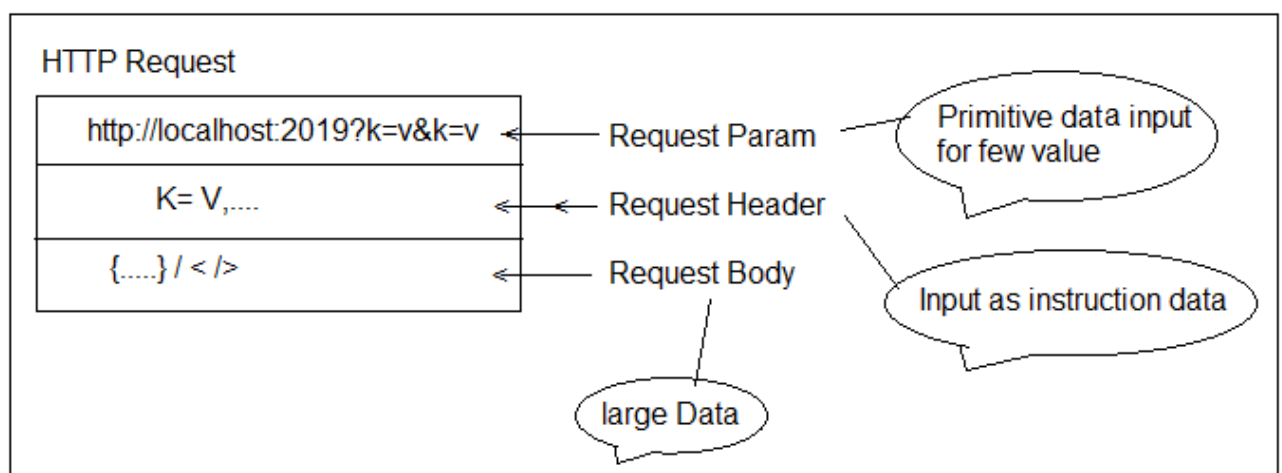
2. Passing Input to RestController:--

=>RestController will read input data either in primitive or in Type (Object/Collection).

=>To pass data/input to rest controller Spring has provided different concepts.

Those are :--

- 1>Request Header
- 2>Request Body ***
- 3>Request Parameter
- 4>Path Variable ***
- 5>Matrix Variable (disable mode by default)



1>Request Header:-- It provides data in key=value format (both are String type).

=>Request header also called as Header Parameters.

=>These are used to provide instructions to server (application/Controller).

Ex:-- Content-Type, Accept, host, Date, Cookies...

2>Request Body:-- To send large/bulk data like objects and collections data in the form of JSON/XML.

=>Even supported large text (Raw data).

=>Spring boot enables JSON conversion by default, but not XML (no JAXB API).

3>Request Parameter:-- To pass primitive data (like id, name, codes ... etc) as input, request parameters are used.

=>format looks like url?key=val&key=val...

=>Both key and value are String type by default.

Request Parameter: Format:--

```
@RequestParam(  
    value="key",  
    required=true/false,  
    defaultValue="-value-")  
DataType localVariable
```

Syntax #1:-

```
@RequestParam("key") DataType localVariable
```

Ex#1: @RequestParam("sname") String sn

Syntax #2:-

```
@RequestParam DataType localVar
```

Ex#2: @RequestParam String sname

=>If key name and local variable name are same then key is not

Syntax #3:-

```
@RequestParam(required=false) DT localVariable
```

=>To make key-value is optional @RequestParam (required=false) String sname

Syntax #4:-

```
@RequestParam (required=false, defaultValue="-DATA-")DT localVariable
```

=>To change default value from null to any other value.

```
@RequestParam(required=false, defaultValue="No DATA") String sname
```

Controller code:--

```
package com.app.rest.controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class AdminRestController
{
    @GetMapping("/show")
    public String showMsg(@RequestParam (value="sname", required=false,
                                     defaultValue="NO DATA") String sid) {
        return "Hello:" +sid;
    }
}
```

Path Variable [Path Parameters]:--

=>We can send data using URL (path).

=>It supports only Primitive Data.

Paths are two types:--

a.>Static Path [format:/url]

b.>Dynamic Path [format:/{key}]

=>Static Path indicates **URL**, where as Dynamic Path indicates **Data** at runtime.

=>While sending data using Dynamic Path key should not be used. Only data

=>Order must be followed in case of sending multiple Parameters.

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

Syntax:--

@PathVariable datatype keyName

Controller code:--

```
package com.app.controller.rest;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;
```

@RestController

public class StudentController

{

 @GetMapping("/show/{sid}/{sname}/{sfee}")

public String showMsg(

 @PathVariable **int** sid,

 @PathVariable String sname,

 @PathVariable **double** sfee)

 {

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

 }

}

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

3. RestController : Method ReturnType:--

=>We can use String (for primitive Data), A classType or Any CollectionType(List, Set, Map...) as Method ReturnType.

=>If return types String then same data will be send to Controller.

=>If return type is non-String (Class or Collection Type) then Data converted to Global Format (ex: JSON/XML).

=>Default conversion Type supported by Boot is JSON (Java Script Object Notation).

=>Even "ResponseEntity<T>" can be used as Return Type which holds Body (T) and status (HttpStatus enum).

Possible Http Status are (5):--

| Code | Types |
|------|-------------------|
| 1xx | Informational |
| 2xx | Success |
| 3xx | Redirect |
| 4xx | Client Side Error |
| 5xx | Server Side Error |

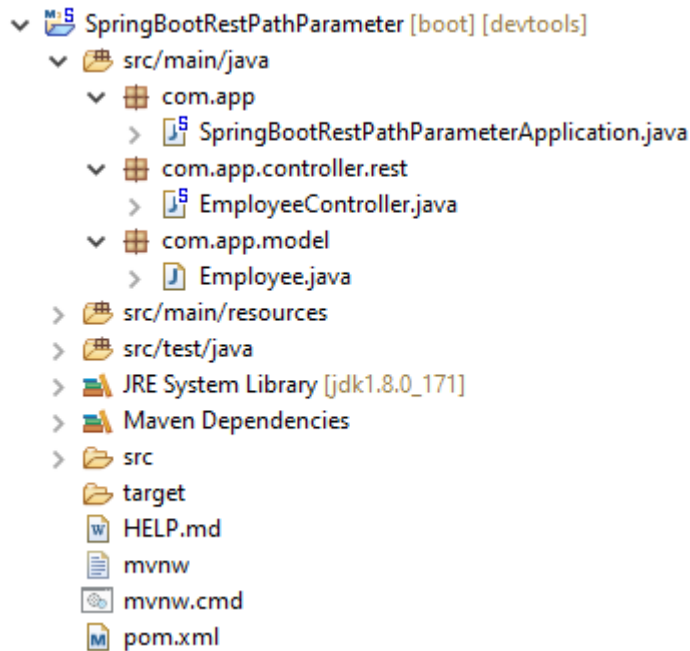
=>To Convert Object to JSON (and JSON to Object) SpringBoot uses JACKSON API.

Step #1:- Create one SpringBoot starter App with web, devtools dependencies.

Step #2:- Add below dependency in pom.xml for XML (JAXB) supports.

```
<dependency>
  <groupId>com.fasterxml.jackson.dataformat</groupId>
  <artifactId>jackson-dataformat-xml</artifactId>
</dependency>
```

#33. Folder Structure of RestController with HttpStatus methods:--



Step#3 Model class:--

package com.app.model;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement

public class Employee

{

private Integer empId;

private String empName;

private double empSal;

public Employee() {

super();

 }

```
public Employee(Integer empld, String empName, double empSal) {  
    super();  
    this.empld = empld;  
    this.empName = empName;  
    this.empSal = empSal;  
}  
public Integer getEmpld() {  
    return empld;  
}  
public void setEmpld(Integer empld) {  
    this.empld = empld;  
}  
public String getEmpName() {  
    return empName;  
}  
public void setEmpName(String empName) {  
    this.empName = empName;  
}  
public double getEmpSal() {  
    return empSal;  
}  
public void setEmpSal(double empSal) {  
    this.empSal = empSal;  
}  
  
@Override  
public String toString() {  
return "Employee [empld=" + empld + ", empName=" + empName + ", empSal=" +  
empSal + "];"  
}  
}
```

Step #4 Controller **class**:--

```
package com.app.controller.rest;  
import java.util.Arrays;  
import java.util.HashMap;  
import java.util.List;  
import java.util.Map;
```

```
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
import com.app.model.Employee;
```

```
@RestController
```

```
public class EmployeeController
```

```
{
```

```
    @GetMapping("/showA")
```

```
    public String showA() {
```

```
        return "Hello-String";
```

```
    }
```

```
    @GetMapping("/showB")
```

```
    public Employee showB() {
```

```
        return new Employee(22, "UDAY", 3.8);
```

```
    }
```

```
    @GetMapping("/showC")
```

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

```
        return Arrays.asList(new Employee(22, "Uday", 6.8),
                           new Employee(23, "Neha", 6.8),
                           new Employee(24, "Ramu", 6.8)
        );
```

```
    }
```

```
    @GetMapping("/showD")
```

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

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

```
        map.put("e1", new Employee(22, "UDAY", 4.6));
```

```
        map.put("e1", new Employee(23, "NEHA", 8.2));
```

```
        map.put("e1", new Employee(24, "RAJA", 9.5));
```

```
    return map;
```

```
    }
```

```
    @GetMapping("/showE")
```

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

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

```
        return resp;
    }
}
```

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

- 1> <http://localhost:2019/showA>
- 2> <http://localhost:2019/showB>
- 3> <http://localhost:2019/showC>
- 4> <http://localhost:2019/showD>

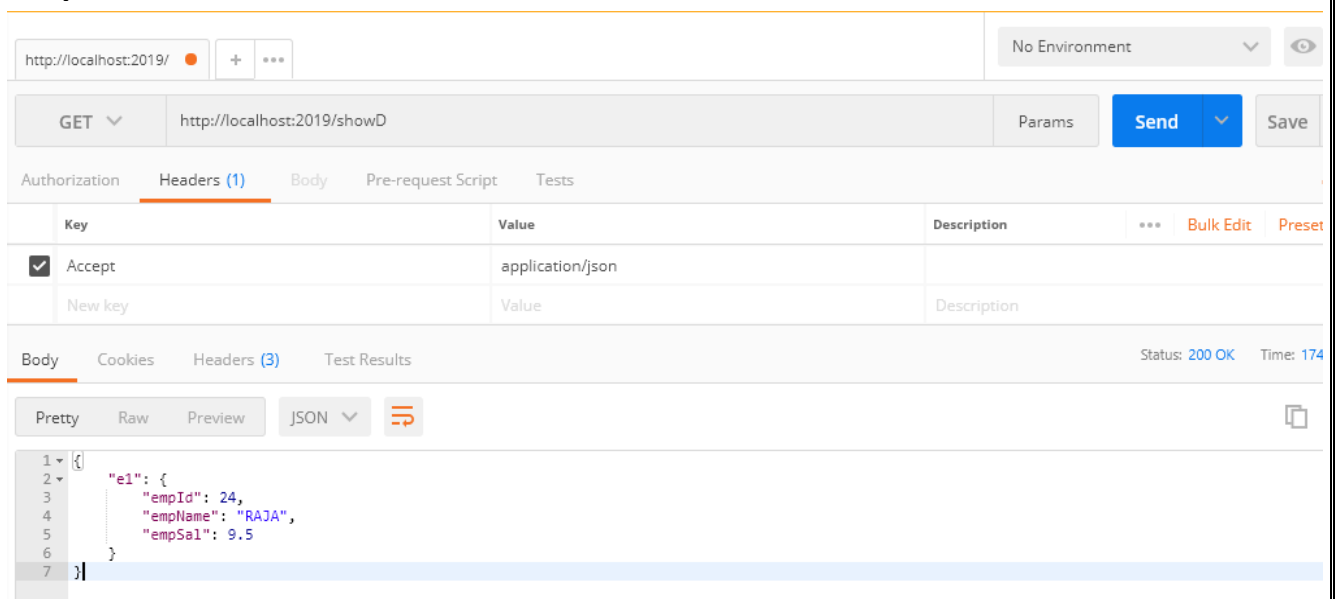
POSTMAN SCREEN :--

GET <http://localhost:2019/showD> SEND

Headers

| Key | Value |
|--------|-----------------|
| Accept | application/xml |

Output Screen:--



Spring Boot ReST + Data JPA +MySQL CRUD Operations (Rest API with Swagger):--

=>Here, define ReST Collector which works for JSON and XML Data input/output.

=>For Primitive inputs use PathVariable.

=>ReST Controller must return output type as `ResponseEntity<T>` which holds `Body(T)` and `HttpStatus` Ex:-- 500, 404, 200, 400 etc..

| Operation Type | Http Method Annotation |
|----------------|------------------------|
| Save | @PostMapping |
| Update | @PutMapping |
| Delete | @DeleteMapping |
| getOne | @GetMapping |
| get all | @GetMapping |

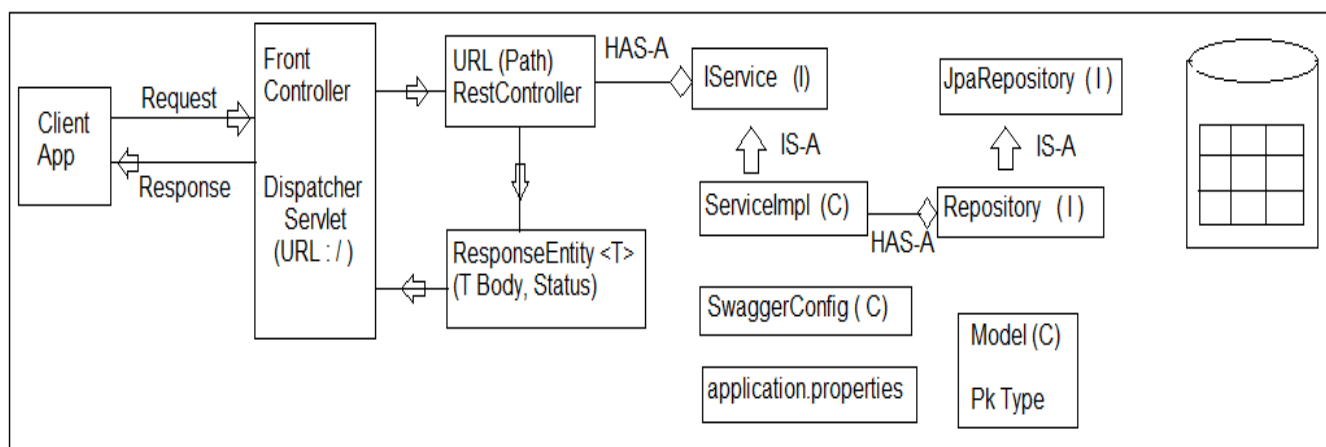
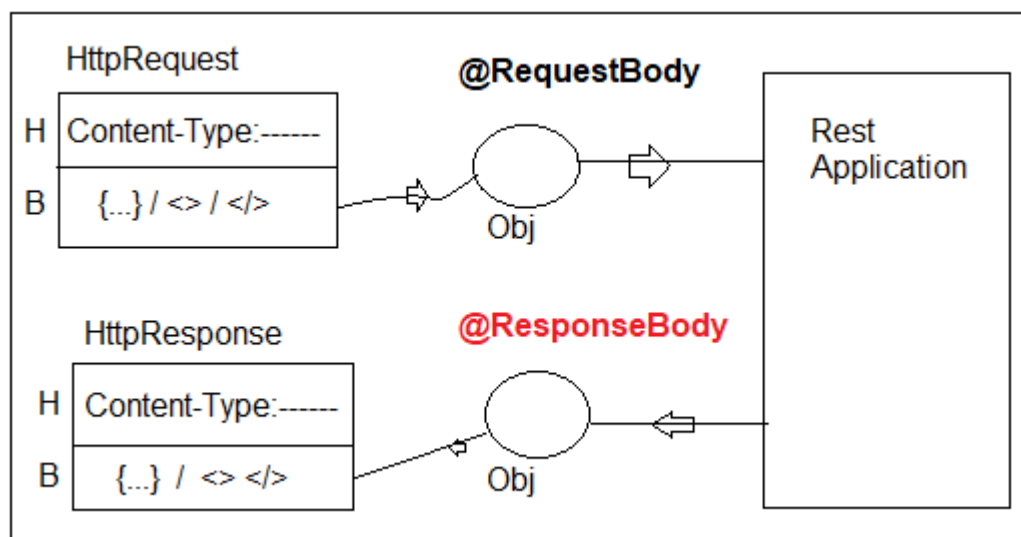
MediaType Conversion annotations are:--

=>@RequestBody and @ResponseBody are the MediaType annotations.

=>Here, @ResponseBody is applied when we write @RestController annotation over class (**Not handled by programmer). It converts **Class/Collection** type to JSON/XML.

=>@RequestBody should be handled by programmer. It converts JSON/XML Data to Object format.

Spring Boot MediaType:--



Coding Order:--

Step#1:- Create Project with web, Jpa, devtools, mysql dependencies.

Step#2:- add jackson-dataformat-xml in pom.xml.

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

=>Student.java = Model

=>StudentRepository.java = Repository

=>IStudentService.java = IService

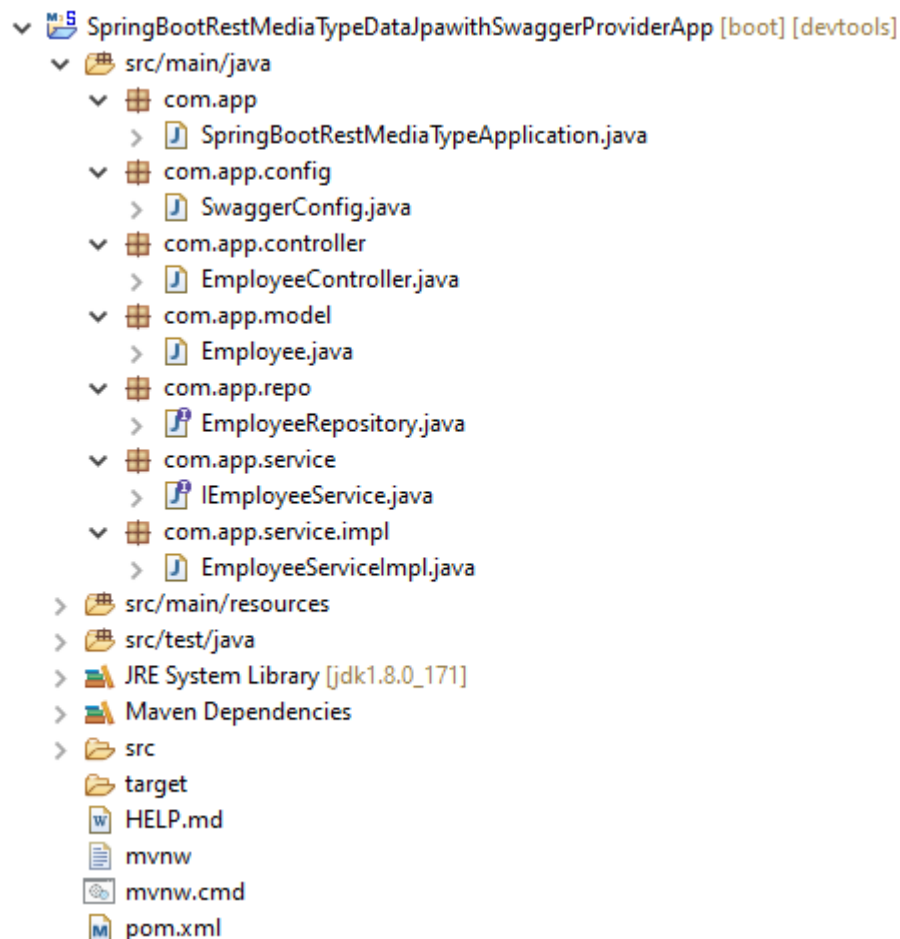
=>StudentServiceImpl.java = Impl

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

Step#5:- Define RestController

#34. Folder Structure of Spring Boot Rest with DataJpa and Swagger

Implementation:--



1. application.properties:--

server.port=2019

DataSource Config

spring.datasource.driverClassName=oracle.jdbc.driver.OracleDriver

spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe

spring.datasource.username=system

spring.datasource.password=system

Hibernate Config

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=update

spring.jpa.database-platform=org.hibernate.dialect.Oracle10gDialect

#2 Model class (Employee.java):--

package com.app.model;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.Table;

import javax.xml.bind.annotation.XmlRootElement;

@Entity

@XmlRootElement

@Table(name="emp_tab")

public class Employee {

 @Id

 @GeneratedValue

 @Column(name="emp_id")

 private Integer empId;

 @Column(name="emp_name")

 private String empName;

 @Column(name="emp_sal")

 private double empSal;



```
public Employee() {
    super();
}
public Employee(Integer empId, String empName, double empSal) {
    super();
    this.empId = empId;
    this.empName = empName;
    this.empSal = empSal;
}
public Integer getEmpId() {
    return empId;
}
public void setEmpId(Integer empId) {
    this.empId = empId;
}
public String getEmpName() {
    return empName;
}
public void setEmpName(String empName) {
    this.empName = empName;
}
public double getEmpSal() {
    return empSal;
}
public void setEmpSal(double empSal) {
    this.empSal = empSal;
}

@Override
public String toString() {
    return "Employee [empId=" + empId + ", empName=" + empName + ",
empSal=" + empSal + "]\n";
}
}
```

#3 Repository Interface (EmployeeRepository.java):--

```
package com.app.repo;
import org.springframework.data.jpa.repository.JpaRepository;
import com.app.model.Employee;

public interface EmployeeRepository extends JpaRepository<Employee, Integer> { }
```

#4 Service Interface (IEmployeeService.java):--

```
package com.app.service;
import java.util.List;
import java.util.Optional;
import com.app.model.Employee;

public interface IEmployeeService {

    public Integer saveEmployee(Employee e);
    public void updateEmployee(Employee e);
    public void deleteEmployee(Integer id);
    public Optional<Employee> getOneEmployee(Integer id);
    public List<Employee> getAllEmployees();
    public boolean isPresent(Integer id);
}
```

#5 ServiceImp class(EmployeeServiceImpl.java):--

```
package com.app.service.impl;
import java.util.List;
import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import com.app.model.Employee;
import com.app.repo.EmployeeRepository;
import com.app.service.IEmployeeService;

@Service
public class EmployeeServiceImpl implements IEmployeeService {
```

```
@Autowired
private EmployeeRepository repo;

@Transactional
public Integer saveEmployee(Employee e) {
    return repo.save(e).getEmpId();
}

@Transactional
public void updateEmployee(Employee e) {
    repo.save(e);
}

@Transactional
public void deleteEmployee(Integer id) {
    repo.deleteById(id);
}

@Transactional(readOnly=true)
public Optional<Employee> getOneEmployee(Integer id) {
    return repo.findById(id);
}

@Transactional(readOnly = true)
public List<Employee> getAllEmployees() {
    return repo.findAll();
}

@Transactional(readOnly = true)
public boolean isPresent(Integer id) {
    return repo.existsById(id);
}
}
```

#6 Controller class (EmployeeController.java):--

```
package com.app.controller;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.DeleteMapping;
```

```
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import com.app.model.Employee;
import com.app.service.IEmployeeService;
```

```
@Controller
```

```
@RequestMapping("/rest/employee")
```

```
public class EmployeeController {
```

```
    @Autowired
```

```
    private IEmployeeService service;
```

```
    //1. save student data
```

```
    @PostMapping("/save")
```

```
    public ResponseEntity<String> save(@RequestBody Employee employee){
```

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

```
        try {
```

```
            Integer id=service.saveEmployee(employee);
```

```
            resp=new ResponseEntity<String>("Employee '"+id+"' created", HttpStatus.OK);
```

```
        } catch (Exception e) {
```

```
            resp=new ResponseEntity<String>(e.getMessage(),
```

```
            HttpStatus.INTERNAL_SERVER_ERROR);
```

```
            e.printStackTrace();
```

```
        }
```

```
        return resp;
```

```
    }
```

```
    //2. get All Records
```

```
    @GetMapping("/all")
```

```
    public ResponseEntity<?> getAll(){
```

```
        ResponseEntity<?> resp=null;
```

```
        List<Employee> list=service.getAllEmployees();
```

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

```
        String message="No Data Found";
        resp=new ResponseEntity<String>(message,HttpStatus.OK);
    } else {
        resp=new ResponseEntity<List<Employee>>(list,HttpStatus.OK);
    }
    return resp;
}

//3. delete based on id , if exist
@DeleteMapping("/delete/{id}")
public ResponseEntity<String> deleteById(@PathVariable Integer id)
{
    ResponseEntity<String> resp=null;
    //check for exist
    boolean present=service.isPresent(id);
    if(present) {
        //if exist
        service.deleteEmployee(id);
        resp=new ResponseEntity<String>("Deleted '"+id+"' successfully",HttpStatus.OK);
    } else { //not exist
        resp=new ResponseEntity<String>(" '"+id+"' Not
Exist",HttpStatus.BAD_REQUEST);
    }
    return resp;
}

//4. update data
@PutMapping("/update")
public ResponseEntity<String> update(@RequestBody Employee employee){
    ResponseEntity<String> resp=null;

    //check for id exist
    boolean present=service.isPresent(employee.getEmpId());
    if(present) { //if exist
        service.updateEmployee(employee);
        resp=new ResponseEntity<String>("Updated Successfully",HttpStatus.OK);
    } else {
```

```
//not exist
resp=new ResponseEntity<String>("Record '"+employee.getEmpId()+" ' not
found",HttpStatus.BAD_REQUEST);
    }
    return resp;
}
}
```

#1. POSTMAN SCREEN for POST Method:--

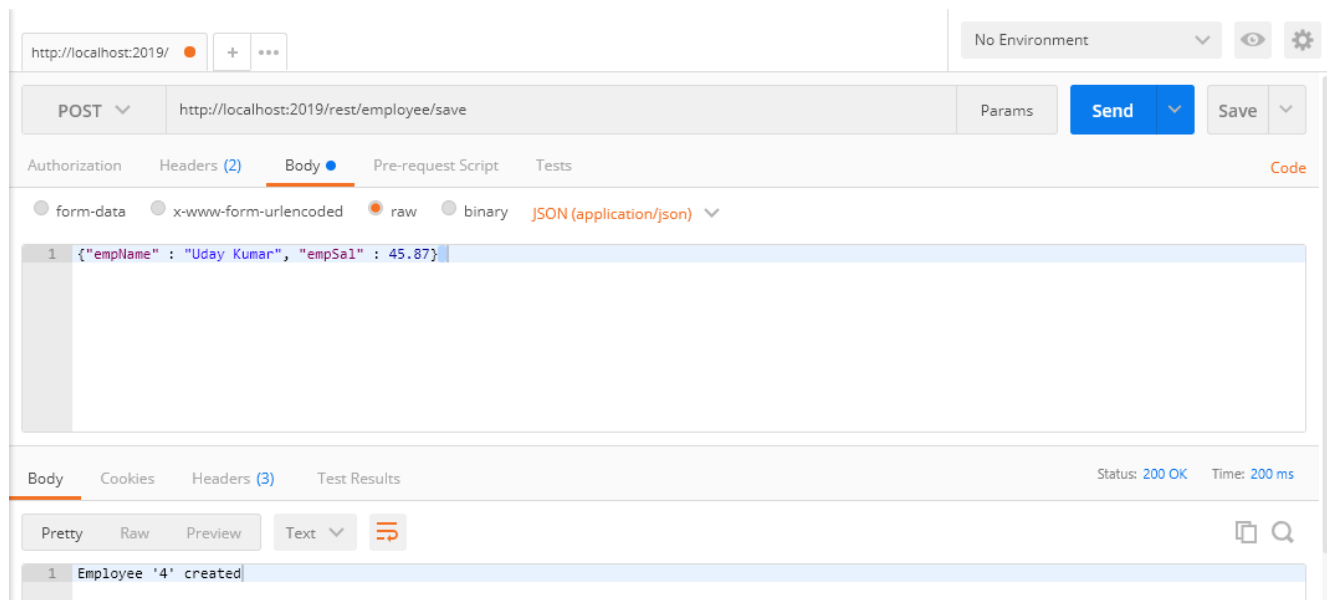
POST <http://localhost:2019/rest/employee/save> **SEND**

Body

raw application/Json

```
{
  "empName" : "Uday Kumar",
  "empSal"   : 5563.3
}
```

POSTMAN OUTPUT SCREEN for POST Method:--



#2. POSTMAN SCREEN for GET Method:--

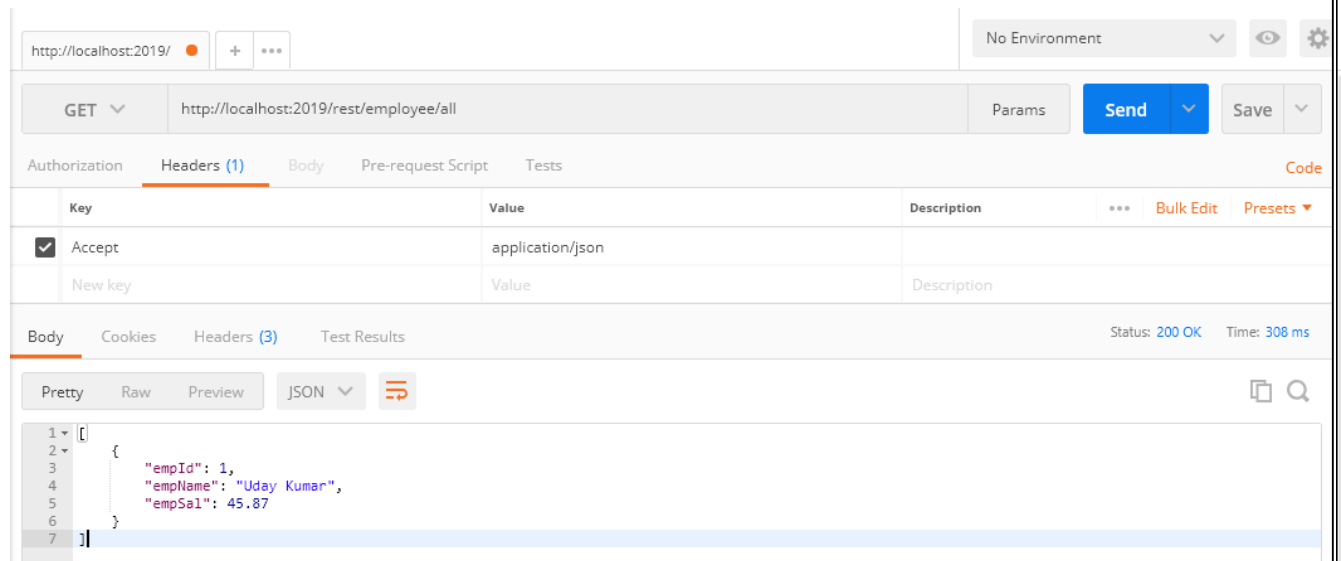
GET <http://localhost:2019/rest/employee/all> **SEND**

Header

Key Value

Accept application/json

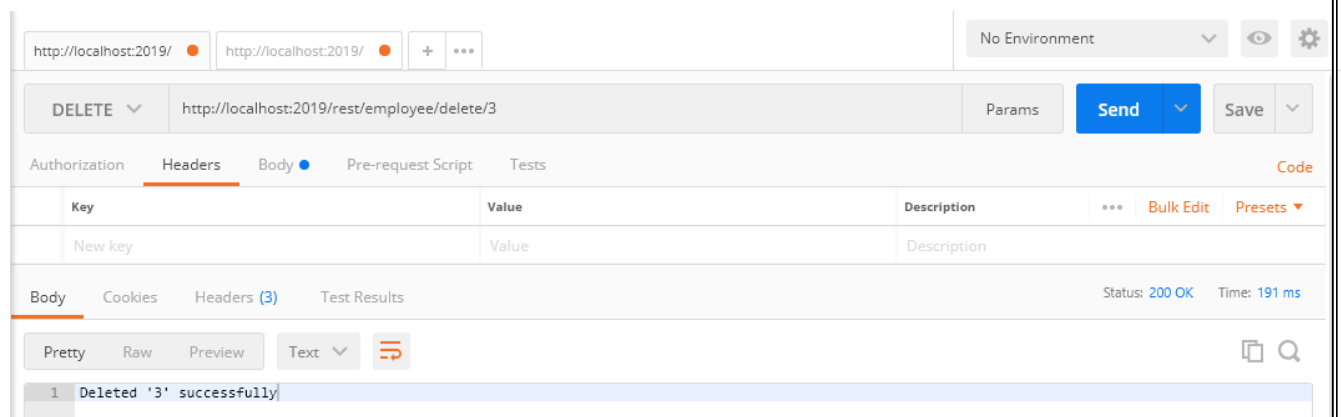
POSTMAN OUTPUT SCREEN for GET Method:--



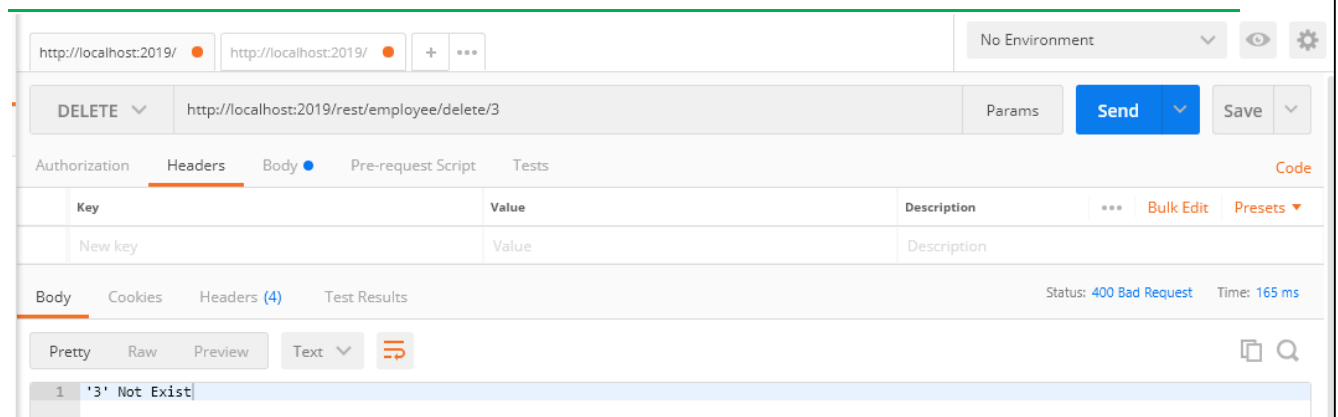
#3. POSTMAN SCREEN for DELETE Method:--

DELETE <http://localhost:2019/employee/delete/4> **SEND**

POSTMAN SCREEN for DELETE Method:--



NOTE:-- If request Id is not present then Return Http Status – **400 BAD-REQUEST**



#4. POSTMAN SCREEN for PUT Method:--

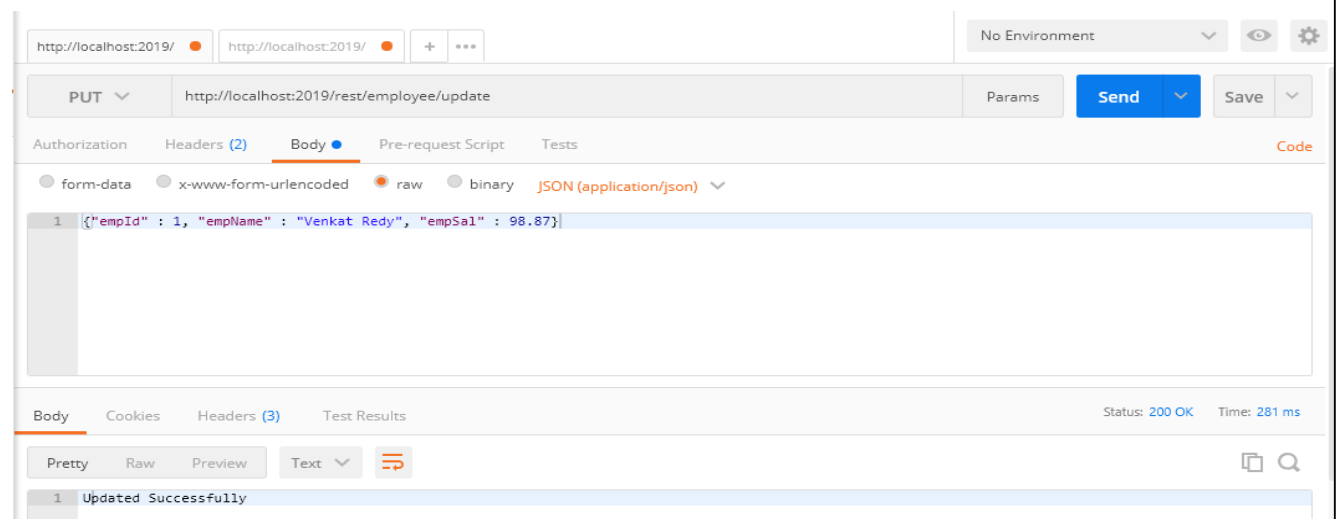
PUT <http://localhost:2019/employee/update> **SEND**

Body

raw application/json

```
{
  "empId": 1,
  "empName": "Venkat Redy",
  "empsal": 67.8
}
```

POSTMAN SCREEN for PUT Method:--



4. Enable Swagger UI in Spring Boot ReST Application:--

=>Compared to all other tools Swagger is a RichSet API provides dynamic UI based on code written for Rest Controller with common Paths.

Step#1:- Add below dependencies in pom.xml

```
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.7.0</version>
</dependency>
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.7.0</version>
</dependency>
```

| Flow | Meaning |
|-----------------------|--------------------------|
| ->Docket () | =>Create Docket |
| ->select () | =>Choose Rest classes |
| ->apis(basepackage()) | =>Classes are in package |
| ->paths (regex()) | =>Having common path |
| ->build() | =>Create final output |

Step#2:- Define Swagger Configuration class in Application (SwaggerConfig.java):--

```
package com.app.config;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spring.web.plugins.Docket;
import springfox.documentation.swagger2.annotations.EnableSwagger2;
import springfox.documentation.builders.RequestHandlerSelectors;
import springfox.documentation.builders.PathSelectors;
```

```
@Configuration
@EnableSwagger2
public class SwaggerConfig {
```

@Bean

```
public Docket myApi() {  
    return new Docket (DocumentationType.SWAGGER_2)  
        .select()  
        .apis(RequestHandlerSelectors.basePackage("com.app.controller.rest"))  
        .paths(PathSelectors.regex("/rest.*"))  
        .build();  
}
```

** basePackage () is a static method defined in RequestHandlerSelectors (C) and in same way regex() is a static method defined in PathSelectors (C).

Step#3:- Run starter class and enter URL :

http://localhost:2019/swagger-ui.html

<http://localhost:2019/rest/employee/save>

{ "empId" : 10, "empName" : "Uday Kumar", "empSal" : 45.87}

Output:--

The screenshot shows a web browser window with the Swagger UI interface. The address bar shows 'localhost:2023/swagger-ui.html#/'. The browser's address bar and tabs are visible at the top. The Swagger UI header is green with the 'swagger' logo and a dropdown menu set to 'default (/v2/api-docs)'. Below the header, the title 'Api Documentation' is displayed, followed by 'Api Documentation' and 'Apache 2.0'. The main content area shows the 'employee-controller : Employee Controller' with a 'Show/Hide' button, 'List Operations', and 'Expand Operations' links. Below this, there is a table of API endpoints:

| Method | Path | Action |
|--------|----------------------------|------------|
| GET | /rest/employee/all | getAll |
| DELETE | /rest/employee/delete/{id} | deleteById |
| POST | /rest/employee/save | save |
| PUT | /rest/employee/update | update |

At the bottom, it says '[BASE URL: / , API VERSION: 1.0]'.

SCREEN#1:-- Swagger Screen for Save Operation

POST /rest/employee/save save

Response Class (Status 200)
string

Response Content Type */*

Parameters

| Parameter | Value | Description | Parameter Type | Data Type | | | | |
|-----------|--|-------------|----------------|--|-------|---------------|--|---|
| employee | <pre>{ "empId": 34, "empName": "Uday Kumar", "empSal": 55,000 }</pre> <p>Parameter content type: application/json</p> | employee | body | <table><thead><tr><th>Model</th><th>Example Value</th></tr></thead><tbody><tr><td></td><td><pre>{ "empId": 0, "empName": "string", "empSal": 0 }</pre></td></tr></tbody></table> | Model | Example Value | | <pre>{ "empId": 0, "empName": "string", "empSal": 0 }</pre> |
| Model | Example Value | | | | | | | |
| | <pre>{ "empId": 0, "empName": "string", "empSal": 0 }</pre> | | | | | | | |

Response Messages

| HTTP Status Code | Reason | Response Model | Headers |
|------------------|--------------|----------------|---------|
| 201 | Created | | |
| 401 | Unauthorized | | |
| 403 | Forbidden | | |
| 404 | Not Found | | |

Try it out!

SCREEN#2:-- Swagger Screen for Delete Operation.

DELETE /rest/employee/delete/{id} deleteById

Response Class (Status 200)
string

Response Content Type */*

Parameters

| Parameter | Value | Description | Parameter Type | Data Type |
|-----------|------------|-------------|----------------|-----------|
| id | (required) | id | path | integer |

Response Messages

| HTTP Status Code | Reason | Response Model | Headers |
|------------------|--------------|----------------|---------|
| 204 | No Content | | |
| 401 | Unauthorized | | |
| 403 | Forbidden | | |

Try it out!