

SPRING FRAMEWORK

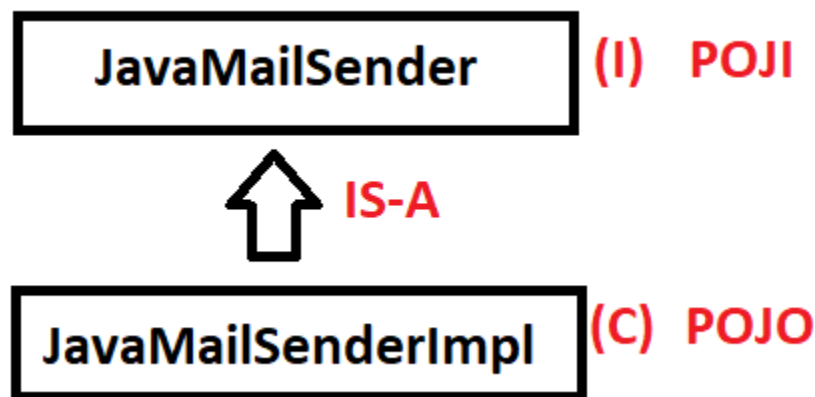
by

Mr. RAGHU

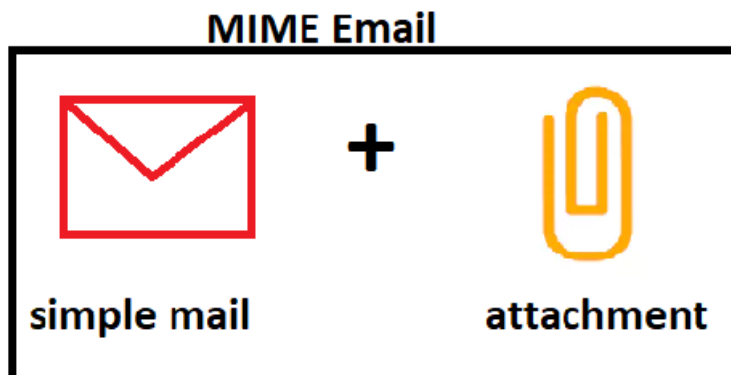
[PART-III]

CHAPTER # 5 SPRING EMAIL

- Spring Email API has been created by Spring Framework using java-mail API and POJI-POJO Design Pattern.
- Spring Email API is a simplified email service which can be implemented and integrated with any spring application easily.
- By using Java mail API (given by Sun MicroSystem) coding and setup is lengthy, it is simplified with POJI-POJO given below :

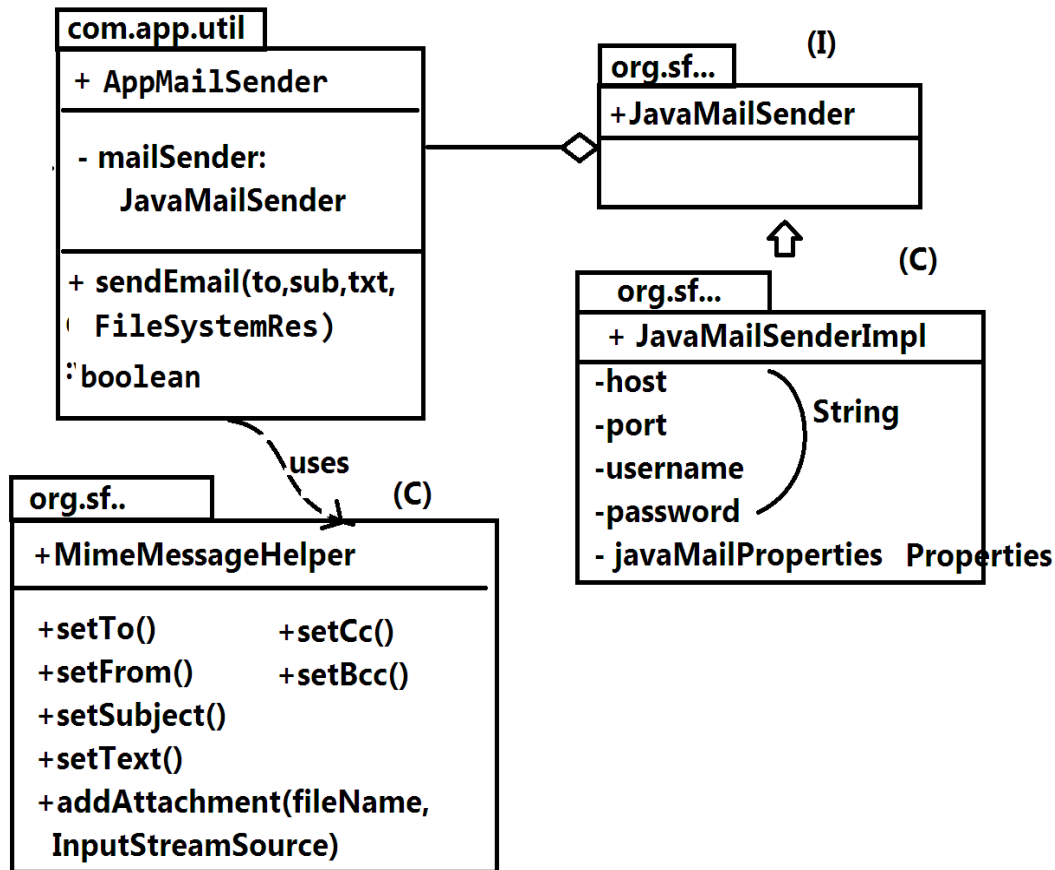


- Spring Email API supports MIME Type Email Sending (Multipurpose Internet Mail Extension). It means “Any kind of file as attachment”,
EX: Video, Audio, Text, Document, Images etc...
I.e shown as :



- Spring Email API also provided one Helper class
"MIMEMessageHelper" to create message(writing code) in less lines of code.
- To provide attachment details use MultiPartFile Concept SystemResource.
(EX: FileSystemResource).

SPRING EMAIL DESIGN:



STEP TO CREATE SPRING EMAIL PROJECT IN ECLIPSE OR STS:-

Step#1: create simple maven project

> File > new > Maven Project > click on checkbox
[v] create simple project (skip archetype.....)

> next button > enter details like:
groupId : org.sathyatech
artifactId : Spring5EmailApp
version : 1.0
> Finish

Step#2: Add <dependencies> and build plugins in pom.xml

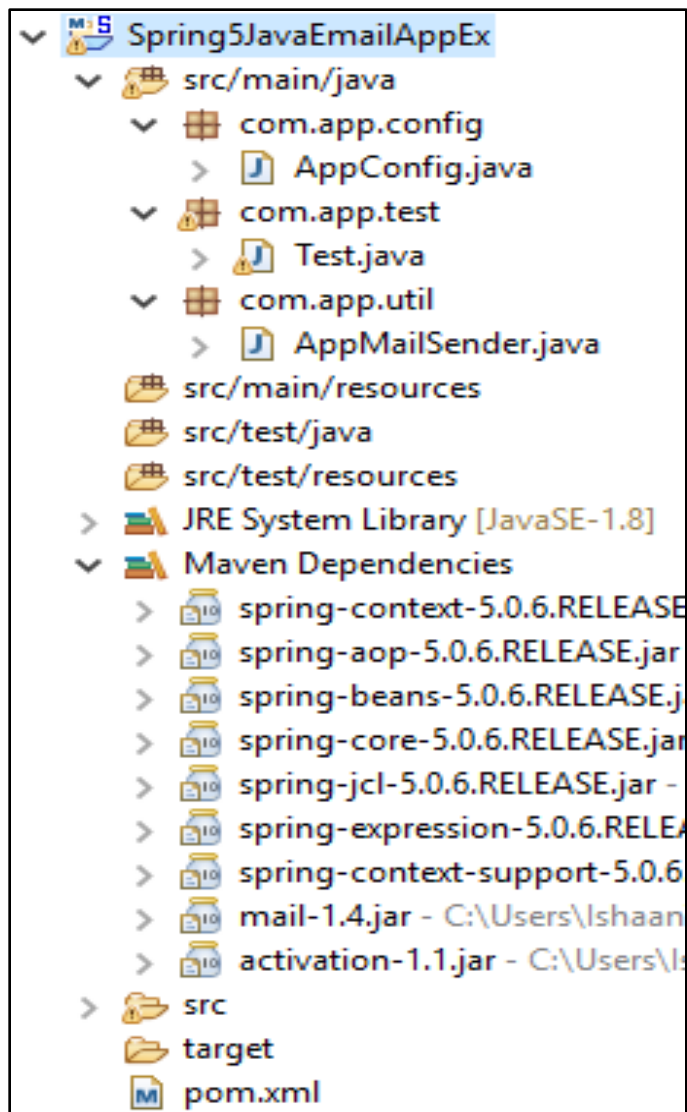
```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context</artifactId>
  <version>5.0.6.RELEASE</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context-support</artifactId>
  <version>5.0.6.RELEASE</version>
</dependency>
<dependency>
  <groupId>javax.mail</groupId>
  <artifactId>mail</artifactId>
  <version>1.4</version>
</dependency>
```

Step#3: Update Maven Project

>Right click on Project > Maven
>Update Project (or alt + F5)

EXAMPLE PROGRAM:-

FOLDER SYSTEM STRUCTURE:-



CODE:

1. Spring Java Configuration File:-

```
package com.app.config;
import java.util.Properties;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.mail.javamail.JavaMailSenderImpl;
```

@Configuration

```
@ComponentScan(basePackages="com.app")
public class AppConfig {
    //JavaMail Sender Impl

    @Bean
    public JavaMailSenderImpl mail() {
        JavaMailSenderImpl mail = new JavaMailSenderImpl();
        mail.setHost("smtp.gmail.com");
        mail.setPort(587);
        mail.setUsername("abc@gamil.com");//enter your emailId.
        mail.setPassword("12345");//enter ur password.
        mail.setJavaMailProperties(props());
        return mail;
    }

    private Properties props() {
        Properties p = new Properties();
        p.put("mail.smtp.auth", "true");
        p.put("mail.smtp.starttls.enable", "true");
        return p;
    }
}
```

2. Mail Sender Util Class:-

```
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 AppMailSender {
    @Autowired
    private JavaMailSender mailsender;
```

```
public boolean sendEmail(String to, String sub, String
text, FileSystemResource file) {
    boolean status = false;
    try {
        // 1. Create Message Object
        MimeMessage message =
mailsender.createMimeMessage();
        // 2. Create helper class Object
        MimeMessageHelper helper = new
MimeMessageHelper(message, file!=null?true:false);
        // 3. Compose Message
        helper.setTo(to);
        helper.setFrom("abc@gamil.com");
        helper.setSubject(sub);
        helper.setText(text);
        helper.addAttachment(file.getFilename(),
file);

        // 4. Send Email
        mailsender.send(message);
        status=true;
    }
    catch (Exception e) {
        status=false;
        e.printStackTrace();
        System.out.println(e);
    }
    return status;
}
}
```

3. Test Class:-

```
package com.app.test;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.annotation.AnnotationConfigAppli
cationContext;
import org.springframework.core.io.FileSystemResource;
import com.app.config.AppConfig;
```

```
import com.app.util.AppMailSender;

publicclass Test {
    publicstaticvoid main(String[] args) {
        //ApplicationContext ac = new
        ClassPathXmlApplicationContext(AppConfig.class);
        ApplicationContext act = new
        AnnotationConfigApplicationContext(AppConfig.class);
        AppMailSender mail = act.getBean("appMailSender",
        AppMailSender.class);
        FileSystemResource file = new
        FileSystemResource("C:/Users/Ishaan/Desktop/ashu.jpg");
        booleanflag = mail.sendEmail("ashuptn92@gmail.com",
        "Hello", "Welcome To Spring Email", file);
        if(flag) {
            System.out.println("Done!!!");
        }else {
            System.out.println("Sorry!!!!");
        }
    }
}
```

4. pom.xml:-

```
<projectxmlns="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>org.sathyatech</groupId>
<artifactId>Spring5JavaEmailAppEx</artifactId>
<version>1.0</version>

    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>5.0.6.RELEASE</version>
        </dependency>
```



```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context-
support</artifactId>
  <version>5.0.6.RELEASE</version>
</dependency>
<dependency>
  <groupId>javax.mail</groupId>
  <artifactId>mail</artifactId>
  <version>1.4</version>
</dependency>
</dependencies>
<build>
<plugins>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-compiler-plugin</artifactId>
<version>3.8.0</version>
<configuration>
<source>1.8</source>
<target>1.8</target>
</configuration>
</plugin>
</plugins>
</build>
</project>
```

OUTPUT:-

Successfully done....

NOTE:-

- Before running above application.
 - a. Disable your antivirus for few minutes.
 - b. Enable less secure App in your gmail account.
- Login to gmail account.
- Top right corner click on your name.

- Choose google account settings options.
- Click on “ sign in add security”.
- Scroll down and “ Enable Less Secure App : ON”

CHAPTER # 6 SPRING SCHEDULING

Spring Scheduler :

To execute a task simultaneously by container(without user interaction) based on "Period of time" or "Point of time" scheduling are used.

Here period of time indicates hours/days and gap but not starting and ending hours or days and Point of time indicates start and end hours and days.

To do scheduling write one method (public, void, zero param) and apply annotation "@Scheduled" over method. Then this method will be called by container automatically.

To activate this process using

(a) XML configuration: <task:@annotation-driven>

[use task schema in <bean> tag]

(b) Java configuration: @EnabledScheduling

[write in AppConfig class]

Example:---

Point of Time

12th JAN

11:00AM

1:32:41PM

[Exact Date and Time]

Period of time

3 days

6 hours

14 min

[Time gap]

Example Code:-

(a) fixedDelay:-

It works based on period of time . Input must be milli seconds[1000mili sec = 1 sec]

On spring container startup, it will call method one , after completing method execution , container wait for give delay then calls one more time. This process is repeated until container is stopped.

Example:-

Consider above method takes 3 sec time to finish work then "time line" is shown below,

(b)fixedRate:-

We can write code is

@Scheduled(fixedRate=1000) over method then max waiting time including method execution time is 1sec.

If limit is crossed ,then once last method call is complete then next method call is made without any gap[gap time=0].

If method has taken less time then given fixedRate wait time is =fixedRate-method execution time.

CRON:-

It is an expression used to specify "Point of Time" or "Period of Time", provided by Unix Operating System and followed by spring scheduler also.

Formate is:-

	<u>0-59</u>	<u>0-59</u>	<u>0-23</u>	<u>1-31</u>	<u>1-12</u>	<u>SUN-SAT</u>	
	Sec	min	hrs	day	month	weak	
➔	*	*	*	*	*	*	

Possible symbol used in expression are:-

* = any Symbol

? =any day/week

- = range

,=Possible values

/= Period of time

Ex#1 0 0 9 * * *

>> Every day morning 9 AM

Ex#2 0 0 9,21 * * *

>> Every day morning 9:00am and 9:00pm

Ex#3 0 30 8-10 * * *

>> Every day morning 8:30 AM ,10:30

Ex#4 0 0/15 16 * * *

>> Every day 4:00PM, with 15 gaps

4:15:00 pm 4:30:00pm, 4:45:00 (only)

Ex#5 15 * * * * *

>> every minute 15 sec only

>> like 9:10:15, next 9:11:15

Ex#6 */15 * * *

>> like 9:10:15, next 9:10:30>9:10:45

Ex#7 0 * 6,7 * * *

>> invalid expression

Ex#8 0 0 9 19 *

>>Sep(9th month) 1st - 9:00:00am

Ex#9 59 59 23 31 12 ?

>> 31 DEC mid-night 11:59:59PM

Ex10 9 9 9 ? 6 ?

>> 6th month every day 09:09:09AM

Ex#11 0 0 6,19 * * *

>> 6:00AM and 7:00PM every day

Ex#12 0 0/30 8-10 * * *

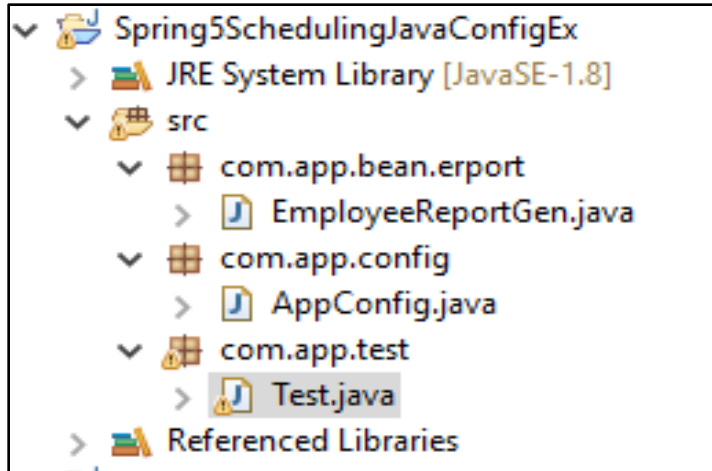
>> 8:00, 8:30 , 9:00, 9:30, 10:00 and 10:30 every day

Ex#13 0 0 9-17 * * MON-FRI

>> on the hour nine-to-five week-days

EXAMPLE PROGRAM:-

Folder Structure:-



CODE

1. AppConfig.java

```
package com.app.config;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.scheduling.annotation.EnableScheduling;

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

}
```

2. EmployeeReportGen.java

```
package com.app.bean.erport;
import java.util.Date;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;

@Component
public class EmployeeReportGen {
    // 1000 milli Second = 1 Second
}
```

```
@Scheduled(fixedDelay=5000)
public void genReport() {
    System.out.println(new Date());
}
}
```

3. TestClass.java

```
package com.app.test;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.annotation.AnnotationConfigAppli
cationContext;
import com.app.config.AppConfig;

public class Test {

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

    }
}
```

RUN TEST CLASS THEN OUTPUT IS:

Sun Oct 07 19:16:12 IST 2018

Sun Oct 07 19:16:17 IST 2018

Sun Oct 07 19:16:22 IST 2018

Sun Oct 07 19:16:27 IST 2018

.
.
.

CHAPTER # 7 SPRING AOP

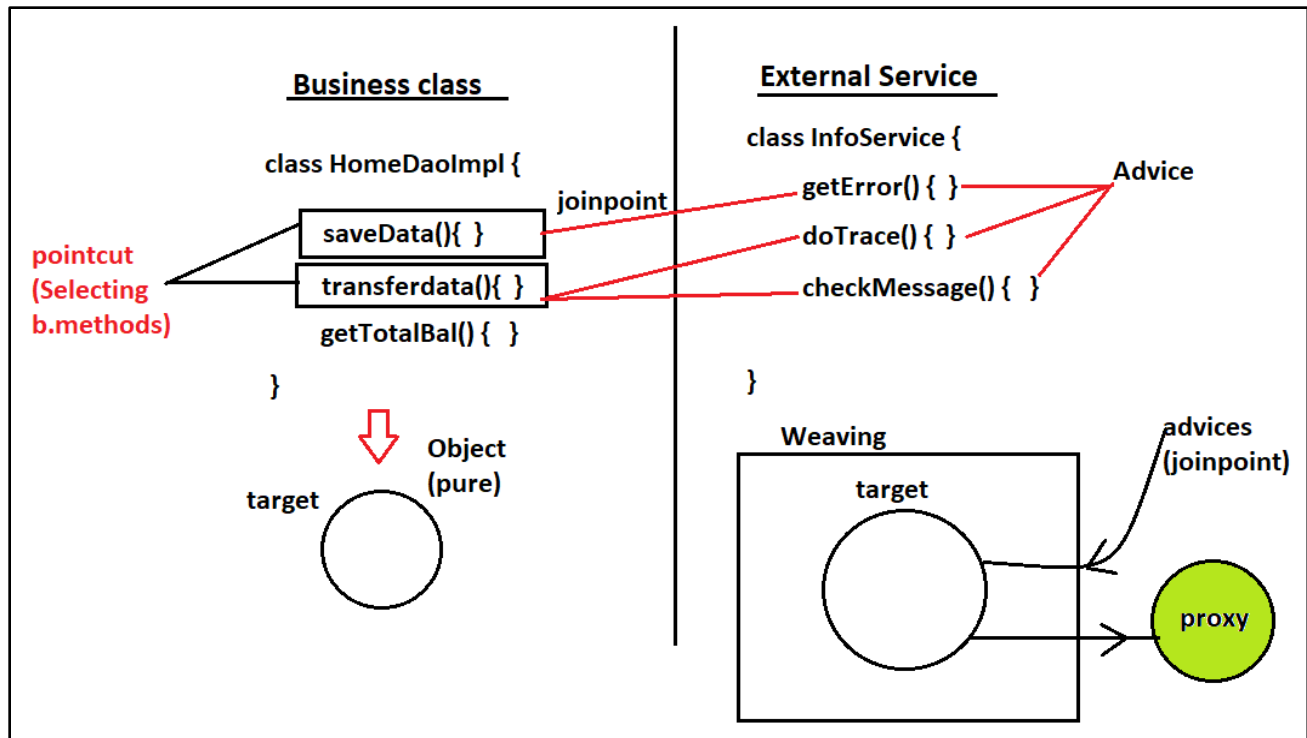
AOP (Aspect Oriented Programming):-

- It is used for “Cross-cutting-concern ” . It means separate business logic and external services.
- External service must behave as plug-in-code, that is without modifying existed application, programmer, should able to add/remove external services Example are :- Log4j,UnitTest,security,JMS,Cryptography,Encode and decode request/response ,filter management, request identity process, etc.....

AOP Terms

1. Aspect:- It is a class, which indicates external services logic.
2. Advice:- It is a method inside Aspect (class). It is also called as implementation of Aspect.
3. Pointcut: It is an expression which select the business class method to connect with advice. But it will not tell which advice it is
4. Joinpoint:- It is a combination of Advice and Pointcut expression . It means “joinpoint says which business class method need what and how many advice.
5. Target :- It is a pure business class object (before adding/without external services logic).
6. Weaving :- It is process done by weaver (sub component of spring container).It will add advice logic to target based on join points.
7. Proxy:- It is a final output of weaving which contains business class logic and selected advices logic.
** ie Code linked at object level, not at class level.

EXAMPLE DESIGN:-



-----Types of Advices in AOP-----

Every method defined in Aspect class must have signed with one of below advice are;-----

1> Before Advice:- Execute advice before business method.

Execution order:

Advice- method ();

b. method (); // Business Method.

2> After Advice:- Execute advice before business method.

Execution order:

b. method ();

Advice- method ();

3> Around advice:- Advice first part is called before business method and second part of advice is called after business method line "Proceed" calls business method from advice.

Execution order:

Advice- method (); --1st part

b. method ();

Advice- method (); 2nd part

4> After Returning:- This is after advice type but it is only called on successful execution of b.method () only.

Execution order:-

b. method (); (if execution successfully)

Advice- method ();

5> AfterThrowing Advices: This is after advice type but it is only called on fail/exception execution of b.method () only.

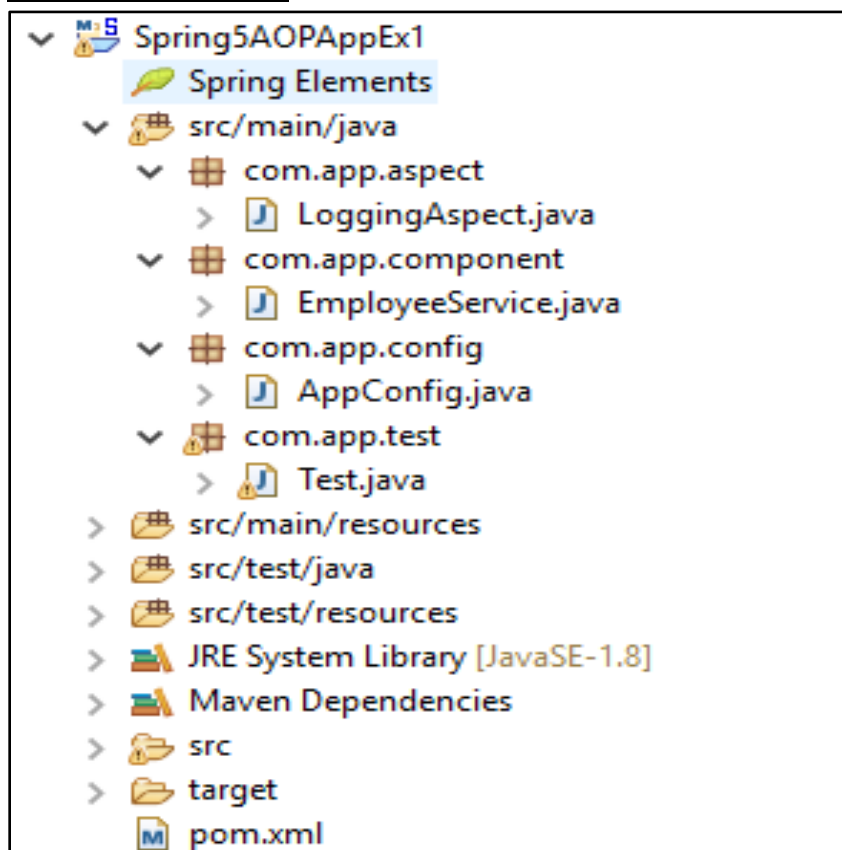
Execution order:-

b. method (); (if throw execution)

Advice- method ();

Example:-

Folder Structure :-



CODE :-

1. LoggingAspect.java:-

```
package com.app.aspect;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.Pointcut;
import org.springframework.stereotype.Component;

@Component
@Aspect
public class LoggingAspect {

    @Pointcut("execution(public * sh*(..))") // joinpoint
    public void point1() {

    }

    @Before("point1()")
    public void showLog() {
        System.out.println("I m from Before Advice()");
    }
}
```

2. EmployeeService.java:-

```
package com.app.component;
import org.springframework.stereotype.Service;

@Service
public class EmployeeService {

    public void showMsg() {
        System.out.println("Hello I m from Business
Methos()");
    }
}
```

3. AppConfig.java:-

```
package com.app.config;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import
org.springframework.context.annotation.EnableAspectJAutoProxy;

@Configuration
@EnableAspectJAutoProxy
@ComponentScan(basePackages="com.app")
publicclass AppConfig {

}
```

4. Test.java:-

```
package com.app.test;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.annotation.AnnotationConfigAppli
cationContext;
import com.app.component.EmployeeService;
import com.app.config.AppConfig;

publicclass Test {

    publicstaticvoid main(String[] args) {
        ApplicationContext ac = new
AnnotationConfigApplicationContext(AppConfig.class);
        EmployeeService e =
ac.getBean("employeeService",EmployeeService.class);
        e.showMsg();
    }
}
```

RUN TEST CLASS THEN OUTPUT IS :-

I m from Before Advice()

Hello I m from Business Methos()

Diff between After, AfterReturning and AfterThrowing Advices:---

After advice is executed either success or failure of b.method(), but

AfterReturning advice is executed only if b.method() is executed successfully and

AfterThrowing advice is executed only on exception thoown by b.method()

Q. >>Can we join one b.method() with multiple advice:--

->Yes possible ,even one advice can also be linked with multiple b.method()

Q.>> Who will create target object:--

->Spring container ,before weaving process

Q.>> Why BeforeReturning/throwing advice are not available in AOP?

-> Containe/JVM can never guess what happens before execution of b.method() ,
So it is not possible to write those.

Q.> Why pointcut is expression are used?

-> To select business(), which are connected to advice next,

Point cut never provides advices details to b.method().

Pointcut:-

It is an expression used to select business class methods which needs advices. But it will not provide details of advices.

Point cut follows below format:--

AS RT PACK.CLS.MN(PARAMETERS)

All are optional in this , we can use wild card character like *(star).dot-dot(..), dot(.) only.

AS (access specifier)

MN(method name)

RT(Return type)

PMTRS(Parameters)

- Consider below business class method

1. +getData(int): void
2. +get(): void
3. +get(double): String
4. +getModel():String
5. +getFormate(int): void
6. +set(): void
7. +setData(int): void
8. +set (double): String
9. +setModel():String
- 10.+setFormat(int): void

-----Point Cut Expression-----

1>>Public * get(..)

Result:→Here two dots(..) indicates any number of parameters in any order and * in place of return type indicates any return type is accepted.

Selected method:-- 2,3

2>> public void * t*()

Result :-- method name should contain one letter 't' nay place (starting/ending/middle) and must have zero param and void type method

Selected method :-----> 5,6,7,2

3>> public * *()

Selected method:--> 2,4,7,9

4>> public String *Data(..)

Selected method--> No method selected

5>> public * get *()

Selected method-->2,4

```
6>> public * get(..)
```

Selected method-->2

Selected method-->2,3

```
7>> public int *et(..)
```

Selected method-->not matched

```
8>>public void *o*()
```

Selected method-->

```
9>>public String *(..)
```

Selected method-->3,4,8,9

```
10>>public * *(..)
```

Selected method-->All method are selected.

AOP Programming using Aspect:---

Spring has provided AOP basic model and implement by vender "Aspect" (3rd party) using AOP annotations. Given few example :--

@Aspect,@before,@After,@Around,@AfterReturning,@AfterThrowing,
@Pointcut..... etc

To enable these annotation in case of

1>>XML:--><aop:aspect-autoproxy>

2>>JAVA:-- @EnableAspectjAutoProxy

Steps to write AOP Example :-

#1: define one component (Service / controller / Repository / RestController) class which behaves like business class.

#2: Define Aspect (class) which indicates External Services.

#3: Define Spring Configuration file (XML / JAVA) To Enable AOP Program.

#4: Write one Test class to call only Business Method .

*****Expected output :Advice must be called automatically.**

EXAMPLE CODE :-

#1:Create one simple maven project.

Details:

Group-Id : org.sathyatech

Artifact-Id : Spring5AOPEx1

Version : 1.0

#2:AddSpring and Aspects Dependencies for jars download.

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context</artifactId>
  <version>5.0.6.RELEASE</version>
</dependency>
<dependency>
  <groupId>org.aspectj</groupId>
  <artifactId>aspectjrt</artifactId>
  <version>1.8.7</version>
</dependency>
<dependency>
  <groupId>org.aspectj</groupId>
  <artifactId>aspectjweaver</artifactId>
  <version>1.8.7</version>
</dependency>
```

#3:Update Maven Project (alt+F5)

#4: Create one java config file under src/main/java folder.

-----AppConfig.java-----

```
Package com.app.config;
//ctrl+shift+o(imports)
@EnableAspectJAutoProxy
```



```
@Configuration
@ComponentScan(basePackages="com.app")
publicclass AppConfig { }
```

#5: Create one Business class.

-----EmployeeService.java-----

```
Package com.app.component;
//ctrl+shift+o(imports)
@Service
publicclass EmployeeService {

    publicvoid showMsg() {
        System.out.println("Hello I M from Business
Method...");
    }
}
```

#6: Write one Aspect with Advices and Pointcut.

-----LoggingAspect.java-----

```
Package com.app.aspect;
//ctrl+shift+o(imports)
@Aspect
@Component
publicclass LoggingAspectA {
    @Pointcut("execution(public * s*(..))")
    publicvoid point1() {
        System.out.println("Hello LoggingA Pointcut");
    }

    @Before("point1()")
    publicvoid showLogA() {
        System.out.println("From Before Advice");
    }

    @After("point1()")
    publicvoid showLogB() {
        System.out.println("From After Advice");
    }
}
```

```
    }
}
```

#7: Test class

-----Test.java-----

```
Package com.app.test;
//ctrl+shift+o(imports)
public class Test {
    public static void main(String[] args) {
        ApplicationContext ac = new
AnnotationConfigApplicationContext(AppConfig.class);
        EmployeeService emp =
ac.getBean("employeeService", EmployeeService.class);
        emp.showMessage();
    }
}
```

Example #2:- Types of Advices Example

➔ All files are same as above example only aspect is different.

CASE #1Before , After Advice Types:

```
package com.app.aspect;
// ctrl+shift+o (imports)
@Aspect
@Component
Public class LoggingAspect {
    @Pointcut ("execution(public * s*(..))")
    public void point1() { }
    @Before("point1()")
    public void showLogA(){
        System.out.println("From Before Advice")
    }
    @After("point1()")
    public void showLogB(){
        System.out.println("From After Advice")
    }
}
```

```
}
```

CASE#2: Around Advice

Here use `ProceedingJoinPoint` to call `proceed()` method. It throws checked exception, must handle using try – catch.

```
package com.app.aspect;
// ctrl+shift+o(imports)
@Aspect
@Component
Public class LoggingAspectB {
    @Pointcut ("execution(public * sh*(..))")
    public void point1() { }
    @Around("point1()")
    public void getMsg(ProceedingJoinPoint jp){
        System.out.println("From 1st part of Around");
        Try{ // b.method call
            Object ob = jp.proceed();
        } catch(Throwable t) {
            System.out.println(t);
        }
        System.out.println("From 2nd part of Around");
    }
}
```

CASE#3: After Returning and Throwing:

➔ For Returning we should provide pointcut, return type and Throwing provide pointcut, throw (execution) type.

```
package com.app.aspect;
// ctrl+shift+o(imports)
@Aspect
@Component
Public class LoggingAspectC {
    @Pointcut ("execution(public * s*(..))")
    public void point1() { }
    @AfterReturning(pointcut="point1()", returning="ob")
```

```

public void onSuccess(Object ob){
    System.out.println("After Success : "+ob);
}
@AfterThrowing(pointcut="point1()", throwing="th")
public void onFail(Throwable th) {
    System.out.println("After Failure :: "+th.getMessage());
}
}

```

##In above any one advice is executed , to see onFail advice output add below code in b.method.

Int x = 9;

If(x > 0) throw new RuntimeException ("Test Exception");

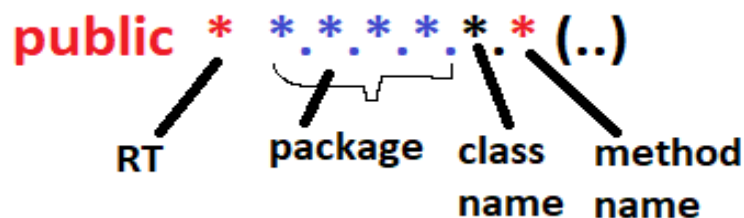
➔ If we don't specify any package and class name then expression considers all packages and all classes. To sort required classes and packages provide expression as :

AS RT PACK.CLS.MN(PARAMETER)

NOTE :-

1. In expression beside parameters next (right to left) is method name, next dot position is class name, next all dots positions are package name.

EX Format :-

public * * * * * (..)

 RT package class name method name

2. Always consider last level package for finding classes.

EX: *.*.*.*.*(..) Here it is 4th level package is considered.

EX: com.app.one.two.*.*(..)

Consider classes only from two package.

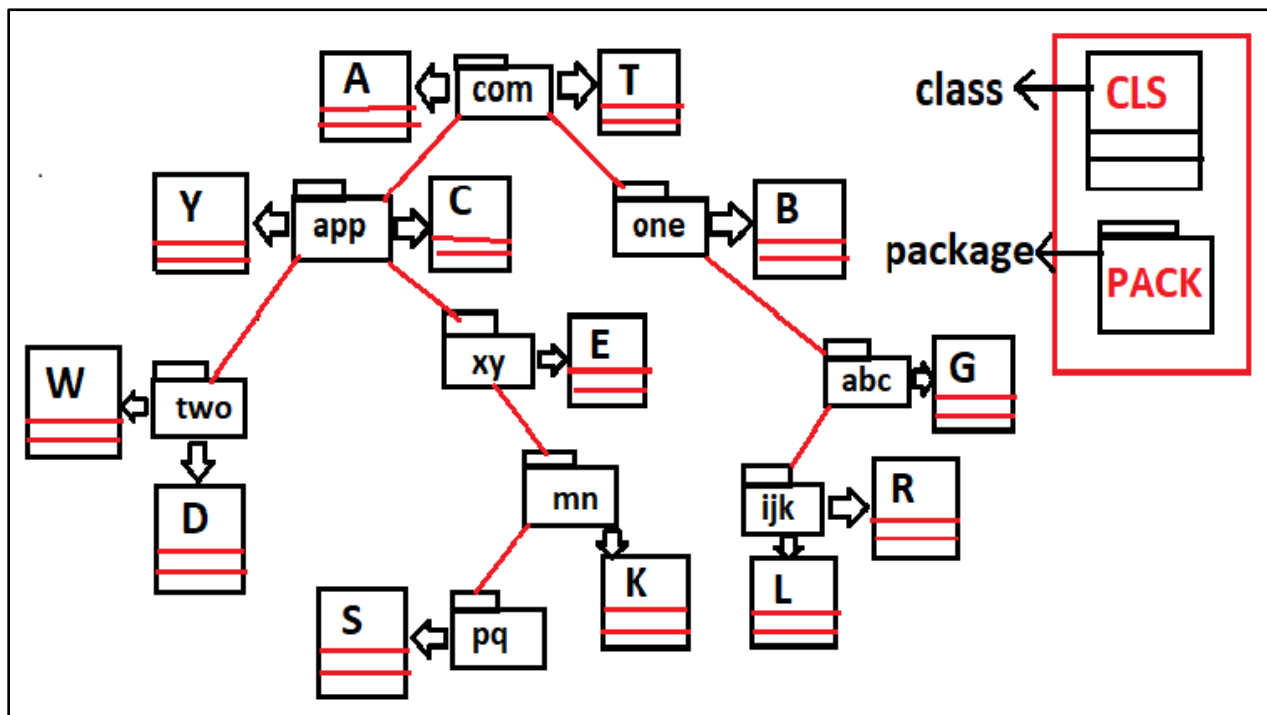
3. For last level package if we provide two dots “**pack..**” then it indicates current package and all it's sub package classes are selected.

EX: com.app..*()

Here app package classes and all it's sub package classes. (not super package)

4. Symbol ‘*’ indicates any (package, class, method or return type).
5. Symbol ‘..’ (dot dot) can be used for current and sub package classes and any parameter type.

Consider below Tree for Expression :-



Consider above all classes having below all methods (methods in every classes).

1. +getData():void
2. +get():void
3. +getModel():String
4. +getCode():String
5. +get(double):int
6. +getModel(int):int

- 7. +set():void
- 8. +setModel():String
- 9. +setCode(int):String
- 10.+set(double):int

#1. Expression

public * com.*.*.*.*()
 classes(4) X method(4)
 W,D,E,G | 2,3,7,8
 Total = 16.

#2. Expression

public void com.app.*.*.*()
 classes(7) X method (2)
 Y,C,W,D,E,S,K | 2,7
 Total = 14.

#3. Expression

public String com.one.*.*.*get*()
 classes(1) X method(1)
 D | 3
 Total = 1.

#4. Expression

public int com.*.*.*.*set()
 classes (8) X method (0)
 W,D,E,F,R,S,K,L | 0
 Total = 0.

#5. Expression

public * com.*.*.*.*.*et()
 classes (3) X method (2)

K,L,R | 2,7
Total = 6.

#7. Expression

public void com..*.set()
classes (13) X method (1)
All classes | 7
Total = 13.

#8. Expression

public * *..*.*(..)
classes (13) X method (10)
All classes | 10
Total = 130;

SPECIAL EXPRESSION IN AOP :

1. within() expression:

➔ This is used to write one pointcut expression which indicates select all methods in given package classes or given classes.

EX:- expression and equal meaning is :

a. within(com.app.*)

equal meaning is all classes in com.app package (and all methods in that).

b. within (com.app.Employee)

only Employee class method.

c. within (com.app..*)

app package classes and all sub – package classes (all methods in those).

2. args () expression:

➔ To indicate only parameters not other thing in pointcut use this expression.

EX :-

a. **args(int)** -->all classes methods having int type parameter is selected.

b. **args(..)** -->method having any parameter is **OK**.

c. **args(String, ..)** --> First parameter of method must be String, 2nd onward anything is **OK**.

3. **this()** expression:

→ To specify exact class methods (not multiple classes).

EX:-

a. **this(com.app.Employee)** --> means only Employee class methods are selected.

Pointcut Joins:-

→ One advice can be connected to multiple pointcuts using AND AND(&&) OR OR (||) symbols.

EX#1:-

```
@Pointcut ("execution (_____)")
public void p1() { }
```

```
@Pointcut("execution (_____)")
Public void p2() { }
```

Here p1, p2 are two Pointcuts then we can write as p1() && p2().

P1() || p2() for a JoinPoint (Advice)

EX#2:-

P1() -----> within (com.app..*)

P2() -----> args(int)

Advice --> p1() && p2()

→ A method which exist in class that is available in app package and method must have int param is selected and connected with advice.

SPECIAL CASE IN ACCESS SPECIFIER :-

Access Specifiere	Return Type	Valid / Invalid
public	void	✓
public	*	✓
*	void	✗
* (dashed box)	* (dashed box)	(only one *)

AS RT PACK.CLS.M(PMTR);

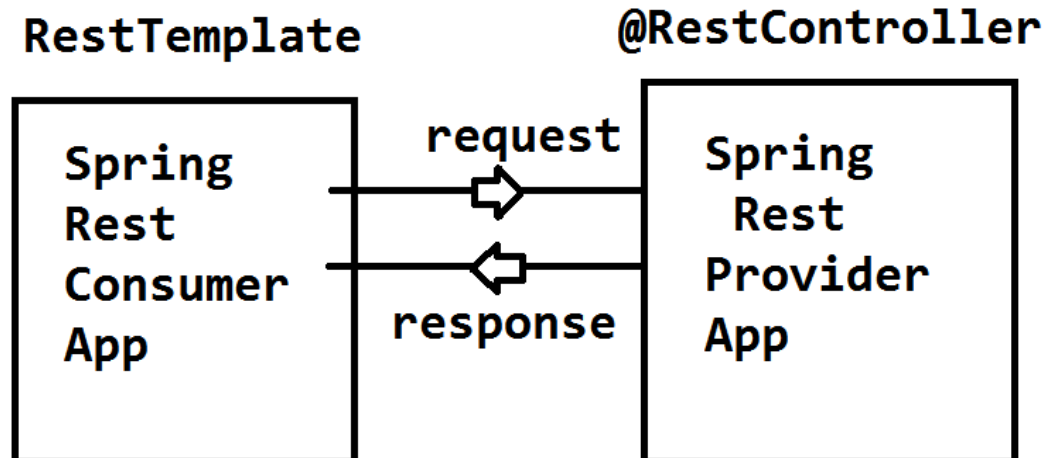
 * (Only one *(star))

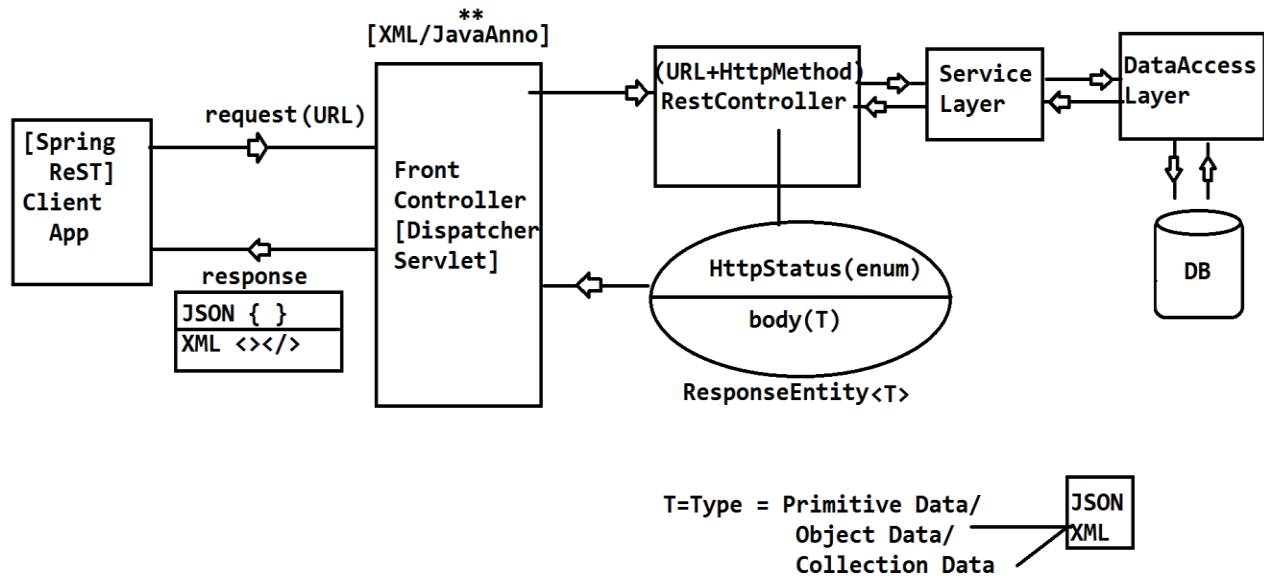
Ex:- * *..*.*(..)

CHAPTER # 8 SPRING REST WEBSERVICES

Spring 5.x Rest Webservice :-

- ➔ Spring 5.x provides ReST Webservices implementation using spring ReST light weight Design, which can be implemented in RAD model (Rapid Application Development) using Template Design Pattern.
- ➔ Spring ReST API contains classes, interfaces, annotations and enums (pre-defined).
- ➔ To implement Spring ReST Application we should create two projects.
Those are :
 1. Spring ReST Provider Application
 2. Spring ReST Consumer ApplicationBoth communicates using Http Protocol (HttpRequest / HttpResponse).



SPRING REST PROVIDER APPLICATION DESIGN**# RestController class coding :-**

- ➔ This is written in IL (Integration Layer) using Spring Annotations and wnums. Every RestController method must be bounded to one HttpMethod Type using it's equal annotations. Few are given as :

HttpMethod (enum)	Spring Annotation
GET	@GetMapping
POST	@PostMapping
PUT	@PutMapping
DELETE	@DeleteMapping
PATCH	@PatchMapping

Format of writing ReST Controller class :-

```
package [packageName];  
  
// ctrl+shift+o (imports)  
  
@RestController  
  
@RequestMapping("/url") // optional  
  
public class [className] {  
  
    // HttpMethod + URL  
  
    @GetMapping("/url")  
  
    public ResponseEntity<?> [methodName] () {  
  
        // logic...  
  
        return ResponseEntity(Body, HttpStatus);  
  
    }  
  
}
```

NOTE :-

1. **@RestController :- [Spring 4.x]**

It is a 5th Stereotype Annotation ie which detect the class and creates the object in Spring Container.

It must be applied on class level.

It internally follows @Controller and @RestController

2. **ResponseEntity<T> :-**

It is a class provide by spring ReST API. It is used as method return type which should contain body (GenericType) and HttpStatus (enum).

3. **@RequestMapping :-**

It is used to provide path(URL) at class / method level. class level it is optional.

4. Method Level,

Path and HttpMethod Type can be provided using HttpMethod Annotations using @XXMapping,

Ex :- @GetMapping, @PostMapping ... etc.

Spring Provider Coding Steps Part # 1:-

Step#1:-

Create One Maven Project (webapp)

File > New > Maven Project

****** do not select any checkbox > next> search for “webapp” and choose

“maven-archtype-webapp” > next > Enter Details:

groupId : org.sathyatech.app

artifactId : Spring5ProviderApp

version : 1.0

>Finish.

Step#2:-

Provide dependencies, build plugins in pom.xml

Spring WebMVC, fasterxml(JSON, XML)

Using JACKSON, maven compiler plugin,

Maven war plugin.

Step#3:-

Assign one webserver to Application

>Right click on project > build path...

>Configure Buildpath > Library Tab

>Add Library > Server runtime

>choose Apache Tomcat(7.x/8.x/ 9.x) > Apply > Apply and close

Step#4:-

Update maven project (alt+F5)

>Right click on project > maven

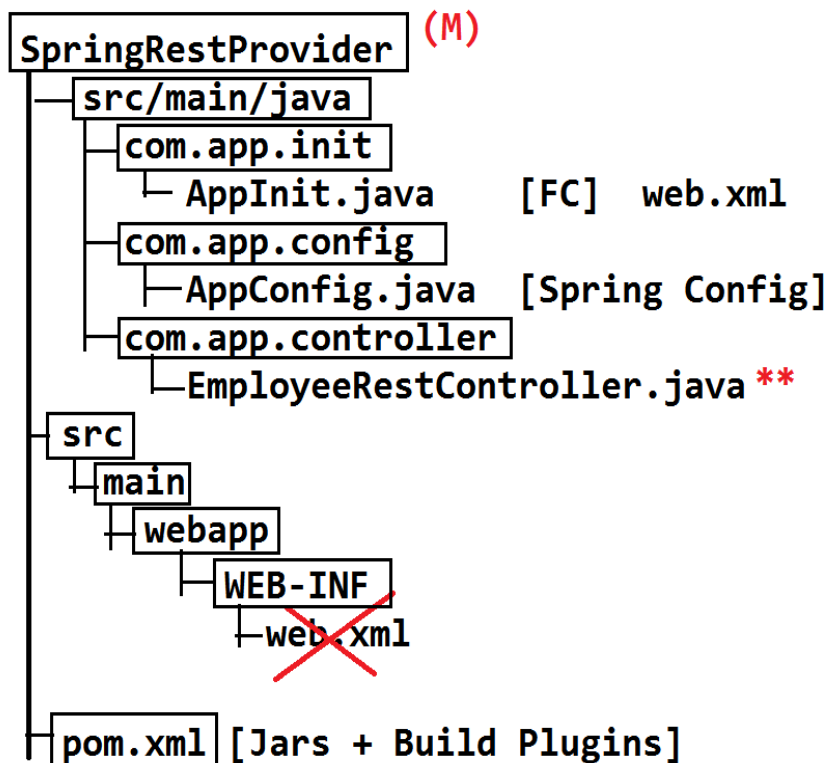
>update project.

Step#5:-

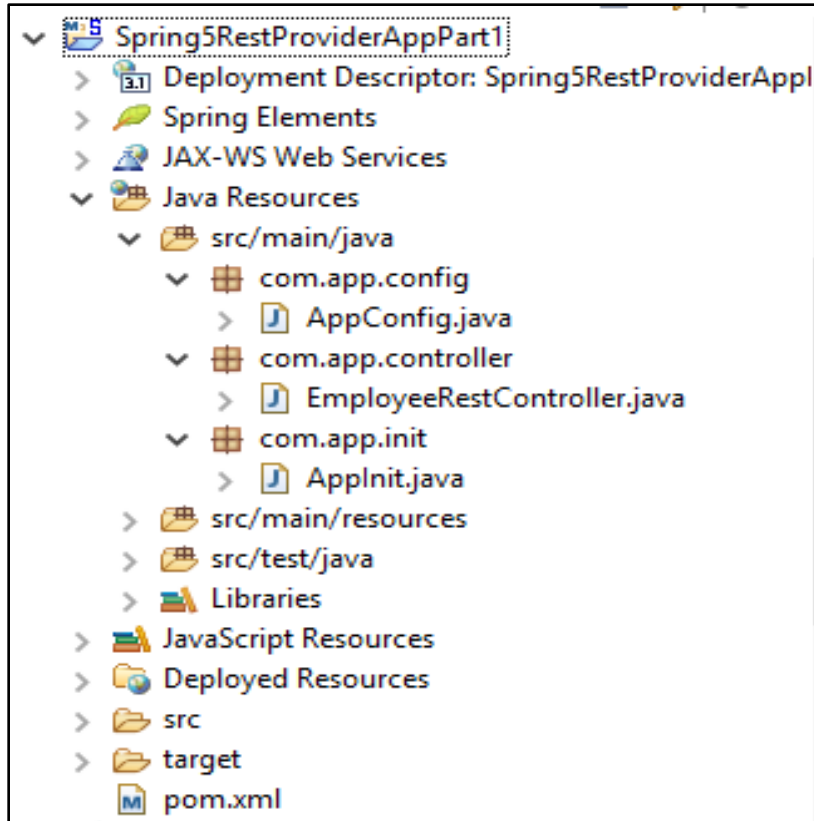
Write code in below order under src/main/java folder, also delete web.xml and index.jsp file.

-----Folder Structure-----

Provider Application



-----IN ECLIPSE OR STS FOLDER STRUCTURE-----



-----JAVA CODE PART #1-----

1. 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/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.sathyatech.app</groupId>
  <artifactId>Spring5RestControllerApp1</artifactId>
  <packaging>war</packaging>
  <version>1.0</version>
  <name>Spring5RestControllerApp1 MavenWebapp</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
```

```
        <version>5.0.6.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
        <version>2.9.5</version>
    </dependency>
</dependency>

    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-
xml</artifactId>
        <version>2.9.5</version>
    </dependency>
</dependencies>
<build>
    <plugins>
        <plugin>

            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-
plugin</artifactId>
            <version>3.7.0</version>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
            </configuration>
        </plugin>
        <plugin>

            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-war-plugin</artifactId>
            <version>2.6</version>
            <configuration>

                <failOnMissingWebXml>false</failOnMissingWebXml>
            </configuration>
        </plugin>
    </plugins>
```



```
</build>
</project>
```

2. AppConfig.java:-

```
package com.app.config;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import
org.springframework.web.servlet.config.annotation.EnableWebM
vc;

@Configuration
@EnableWebMvc
@ComponentScan(basePackages="com.app")
publicclass AppConfig {

}
```

3. AppInit.java:-

```
package com.app.init;
import
org.springframework.web.servlet.support.AbstractAnnotationCo
nfigDispatcherServletInitializer;
import com.app.config.AppConfig;

publicclass AppInit extends
AbstractAnnotationConfigDispatcherServletInitializer {

    @Override
    protected Class<?>[] getRootConfigClasses() {
        returnnew Class[] {AppConfig.class};
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        returnnull;
    }
}
```

```
@Override
protected String[] getServletMappings() {
    return new String[] {"/"};
}
```

4. EmployeeRestController.java:-

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

@RestController
@RequestMapping("/employee") // Optional
public class EmployeeRestController {

    // Method
    // HttpMethod + Method URL
    @GetMapping("/show")
    public ResponseEntity<String> showMsgA(){
        String body = "Welcome To GET Method Spring Rest Application!!";
        HttpStatus status = HttpStatus.OK;
        ResponseEntity<String> entity = new
ResponseEntity<String>(body,status);
        return entity;
    }

    @PostMapping("/show")
    public ResponseEntity<String> showMsgB(){
        String body = "Welcome To POST Method Spring Rest Application!!";
        HttpStatus status = HttpStatus.OK;
        ResponseEntity<String> entity = new
ResponseEntity<String>(body,status);
    }
}
```

```
        return entity;
    }

    @PutMapping("/show")
    public ResponseEntity<String> showMsgC(){
        String body = "Welcome To PUT Method Spring Rest Application!!";
        HttpStatus status = HttpStatus.OK;
        ResponseEntity<String> entity = new
ResponseEntity<String>(body,status);
        return entity;
    }

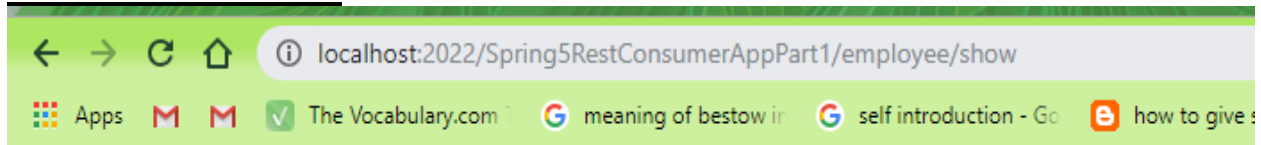
    @DeleteMapping("/show")
    public ResponseEntity<String> showMsgD(){
        String body = "Welcome To DELETE Method Spring Rest
Application!!";
        HttpStatus status = HttpStatus.OK;
        ResponseEntity<String> entity = new
ResponseEntity<String>(body,status);
        return entity;
    }
}
```

Running on server:

Run as > run on server

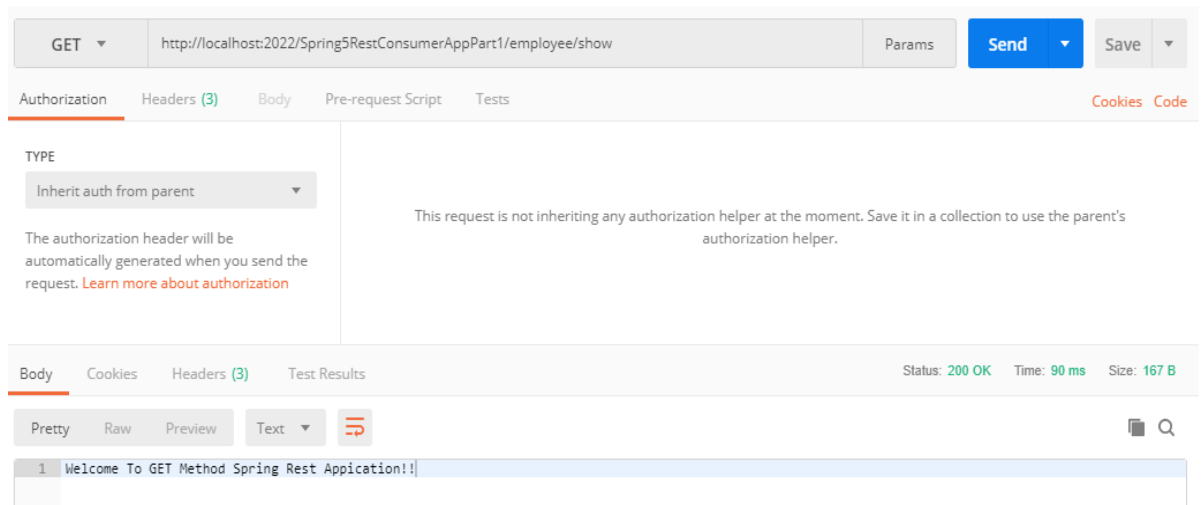
<http://localhost:2022/Spring5RestConsumerAppPart1/employee/show>

OUTPUT ON BROWSER:-



Welcome To GET Method Spring Rest Application!!

-----POSTMAN SCREEN-----



SPRING REST CONSUMER APPLICATION:-

- ➔ Use RestTemplate to make HTTP Request calls to provider application from consumer application.
- ➔ Template is a Design Pattern used to reduce common lines of code (duplicate code/ boiler plate code...).

RestTemplate takes care of :

- >Creating client objects
- >web resources object
- > Default HTTP methods with Header
- > Making call to Provider
- > Auto conversion of response to ResponseEntity<T>

-----Client Application Steps-----

1. Create simple maven project
 - > File > New > Maven Project
 - > Choose checkbox *** > next
 - > Enter Details like:
groupId : org.sathyatech.app
artifactId : Spring5ConsumerApp

version : 1.0

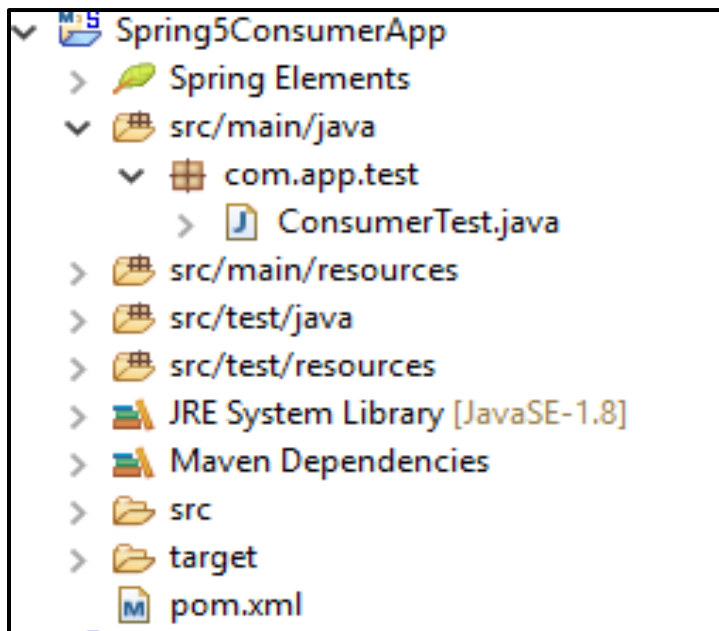
>Finish.

2. Open pom.xml and provide jars and build plugins.
3. Update Maven Project (alt+F5)
 - > Right click on project > Maven
 - > update project.
4. Create one class "ClientTest" under src/main/java Folder.

Coding Steps are:-

- a. Create object to RestTemplate
- b. Create String (Provider) URL
- c. Make call (as HTTPRequest)
- d. Get Response in ResponseEntity
- e. Print or use result (body / status)

FOLDER STRUCTURE:-



Example Code:

1. 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>org.sathyatech.app</groupId>
  <artifactId>Spring5ConsumerApp</artifactId>
  <version>1.0</version>

  <dependencies>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>5.0.6.RELEASE</version>
    </dependency>
    <dependency>
      <groupId>com.fasterxml.jackson.core</groupId>
      <artifactId>jackson-databind</artifactId>
      <version>2.9.5</version>
    </dependency>
    <dependency>
      <groupId>com.fasterxml.jackson.dataformat</groupId>
      <artifactId>jackson-dataformat-xml</artifactId>
      <version>2.9.5</version>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.7.0</version>
        <configuration>
          <source>1.8</source>
```

```
        <target>1.8</target>
    </configuration>
</plugin>
</plugins>
<finalName>SpringRestProvider</finalName>
</build>
```

```
</project>
```

2. ConsumerTest.java

```
package com.app.test;
import org.springframework.http.ResponseEntity;
import org.springframework.web.client.RestTemplate;

public class ConsumerTest {

    public static void main(String[] args) {
        //1.create object to RestTemplate
        RestTemplate rt = new RestTemplate();
        //2.Provider URL
        String
url="http://localhost:2022/Spring5RestProviderAppPart1/employee/show";
        //3.make call (http request)
        //4.get Response in ResponseEntity
        ResponseEntity<String>entity=rt.getForEntity(url,
String.class);

        //5.Print or use result
        System.out.println(entity.getBody());
        System.out.println(entity.getStatusCode().name());
        System.out.println(entity.getStatusCodeValue());
    }
}
```

OUTPUT:-

Welcome To GET Method Spring Rest Application!!

OK

200

SPRING REST-PROVIDER USING XML CONFIGURATION PART- #2

➔ Here FC (FrontController) must be configured in web.xml using directory match url pattern.

Ex: /rest/* , /* , / (only slash) , a/b/c/*

Code look likes : (web.xml)

```
<web-app>
<servlet>
<servlet-name>sample</servlet-name>
<servlet-class>
    org.springframework.web.servlet.DispatcherServlet
</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>sample</servlet-name>
    <url-pattern>/rest/*</url-pattern>
</servlet-mapping>
</web-app>
```

➔ Spring xml configuration must be created under /WEB-INF/ with naming rule : [<servlet-name>]-servlet.xml in above ex:<servlet-name> = sample, Then File name : **sample-servlet.xml**

It should contain code for:

a. Activation of annotations

b. Activation of MVC/ReST process.

Code looks like:

```
<beans...>

    <context:component-scan base-package="com.app" />

    <mvc:annotation-driven />

</beans>
```

Working with MediaType (Global Format):-

MediaType:-

It is a concept for data representation, language data (object) can be convert to global format and reverse using MediaType Annotations.

Those are:

@RequestBody :Must be applied at method parameter by programmer.

@ResponseBody: Autoapplied by RestController for every method return type.

➔ HttpRequest/Response holds data in global format in body area at same time we should add Header key "**Content-Type**" to indicate what type of data Body holds.

- **For JSON Content-Type : application/json**
- **For XML Content-Type : application/xml**

➔ Request Header should also have Header key "**Accept**" (with JSON/XML) which indicates what type of response Body is expected by consumer.

➔ To enable JSON conversion in ReST in pom.xml

Add dependency :

Artifact-Id : Jackson-databind
(or any it's equal)

To enable XML conversion in ReST,

In pom.xml add dependency :

Artifact-Id : Jackson-dataformat-xml (or any it's equal)

➔ If both are added then default **"Accept : application/xml"**
(with high priority) , if any one is added JSON/XML , then that is only default
"Accept".

CASE#1:-

Provider supports both XML,JSON and request Body is JSON, Content-Type is application/xml. Then HttpStatus **400 Bad Request**.

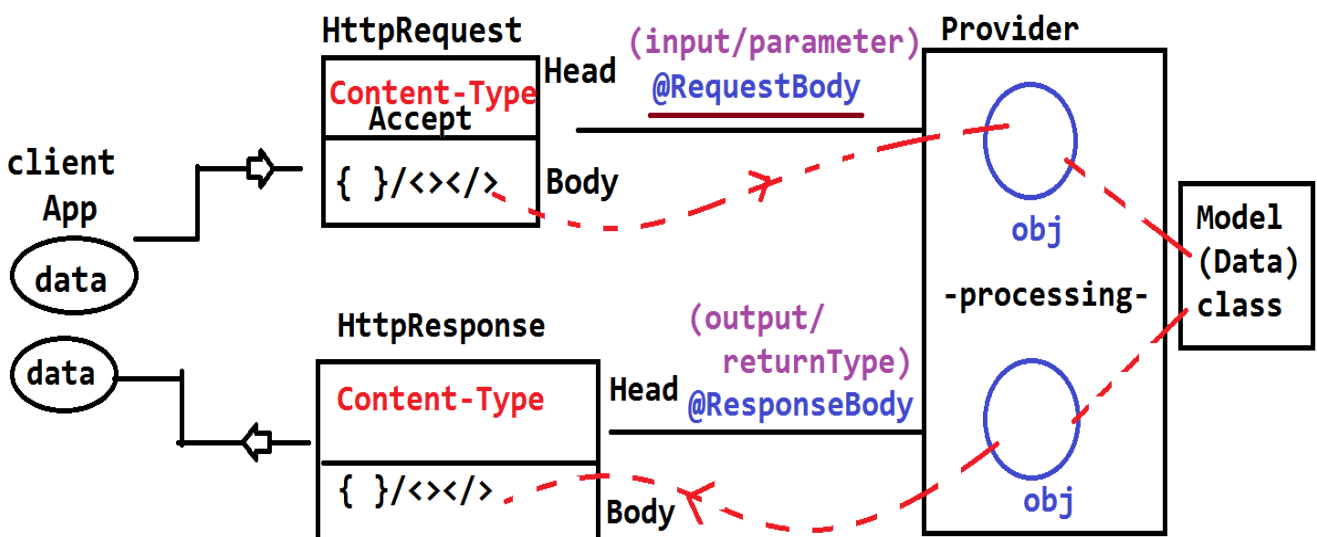
CASE#2:-

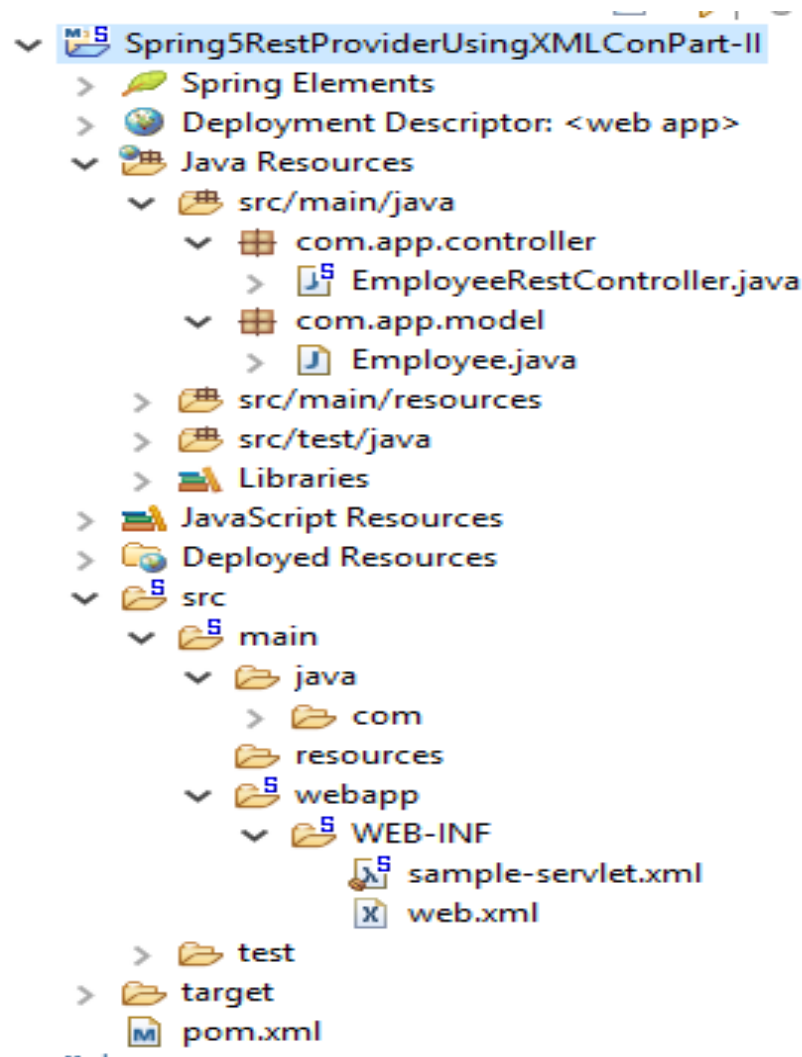
Provider supports XML only. Request Body JSON. Then HttpStatus **415Unsupported MediaType**.

CASE#3:-

Provider supports JSON only Request Body JSON. Then HttpStatus **200 OK**.

-----WORKING FLOW OF MEDIATYPE-----



PROGRAM EXAMPLE CODE :-**# Folder Structure Maven Project:-****Example Code:-****1. 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/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.sathyatech.app</groupId>
```

```
<artifactId>Spring5RestProviderUsingXMLConPart-II</artifactId>
<packaging>war</packaging>
<version>1.0</version>
<name>Spring5RestProviderUsingXMLConPart-II MavenWebapp</name>
<url>http://maven.apache.org</url>

<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>5.0.6.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
        <version>2.9.5</version>
    </dependency>
    <dependency>
        <groupId>com.fasterxml.jackson.dataformat</groupId>
        <artifactId>jackson-dataformat-xml</artifactId>
        <version>2.9.5</version>
    </dependency>
</dependencies>
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.7.0</version>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
            </configuration>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-war-plugin</artifactId>
```

```
        <version>2.6</version>
        <configuration>

        <failOnMissingWebXml>false</failOnMissingWebXml>
        </configuration>
    </plugin>
</plugins>
</build>

</project>
```

2. Employee Model Class

```
package com.app.model;

public class Employee {

    private int empId;
    private String empName;
    private double empSal;

    public Employee() {
        super();
    }

    public int getEmpId() {
        return empId;
    }

    public void setEmpId(int empId) {
        this.empId = empId;
    }

    public String getEmpName() {
        return empName;
    }

    public void setEmpName(String empName) {
        this.empName = empName;
    }
}
```

```
    }

    public double getEmpSal() {
        return empSal;
    }

    public void setEmpSal(double empSal) {
        this.empSal = empSal;
    }

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

3. EmployeeRestController class:

```
package com.app.controller;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.app.model.Employee;

@RestController
@RequestMapping("/employee")
public class EmployeeRestController {

    @PostMapping("/data")
    public ResponseEntity<Employee> processData(@RequestBody Employee emp){
        emp.setEmpSal(emp.getEmpSal()*4);
        ResponseEntity<Employee> entity = new ResponseEntity<Employee>(emp,
        HttpStatus.OK);
    }
}
```

```
        return entity;
    }
}
```

4. Web.xml file:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
    xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-
app_3_1.xsd" id="WebApp_ID" version="3.1">

    <servlet>
        <servlet-name>sample</servlet-name>
        <servlet-
class>org.springframework.web.servlet.DispatcherServlet</ser
vlet-class>
        </servlet>

        <servlet-mapping>
            <servlet-name>sample</servlet-name>
            <url-pattern>/rest/*</url-pattern>
        </servlet-mapping>

</web-app>
```

5. Sample-servlet.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:context="http://www.springframework.org/schema/con
text"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xmlns:p="http://www.springframework.org/schema/p"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context.xsd
    http://www.springframework.org/schema/mvc
    http://www.springframework.org/schema/mvc/spring-mvc.xsd
">

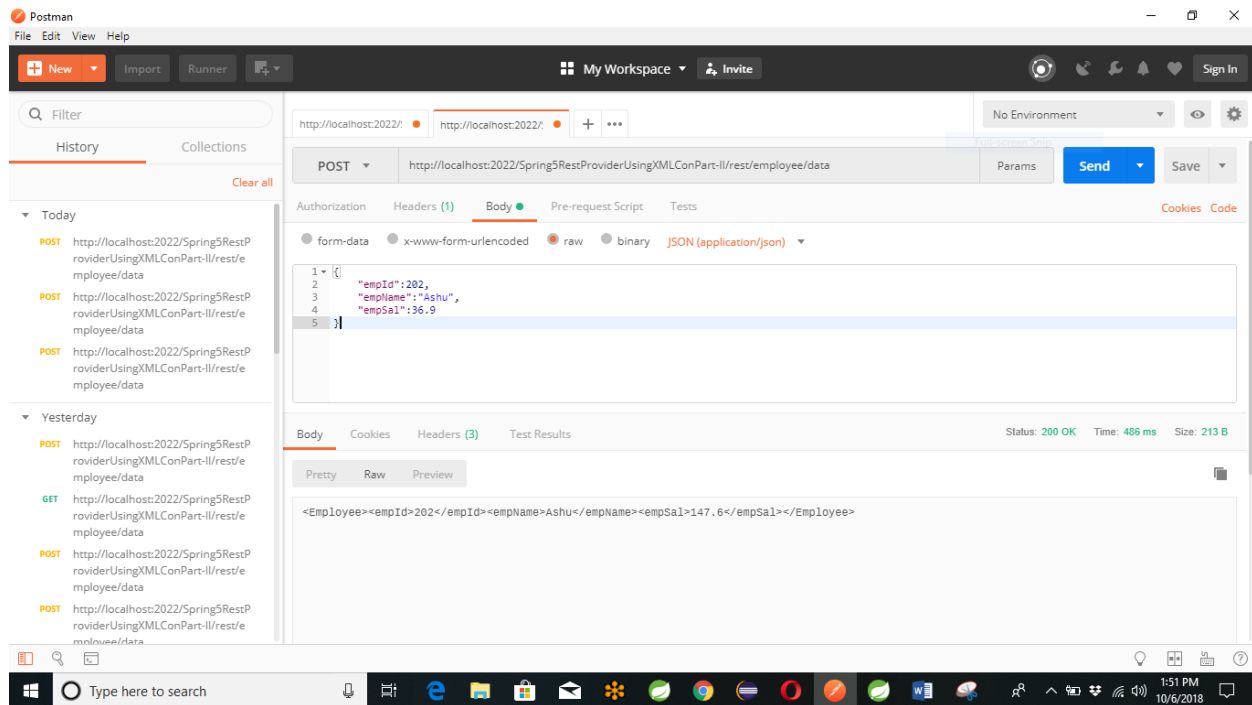
<!-- 1. Activation of Annotations -->
<context:component-scan base-package="com.app"/>

<!-- 1. MVC of Annotations -->
<mvc:annotation-driven/>

</beans>
```

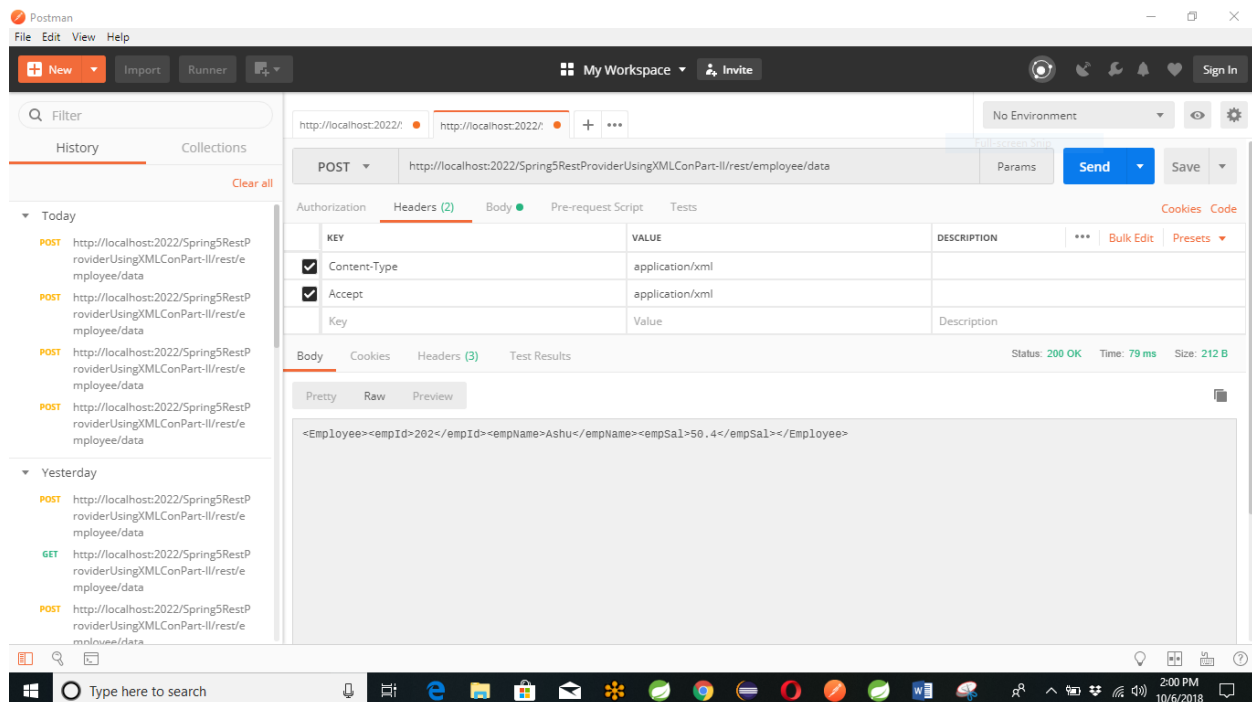
Run on server add make Request Using POSTMAN

POSTMAN SCREEN #1 BODY DETAILS



Run on server add make Request Using POSTMAN

POSTMAN SCREEN #2 HEADER DETAILS



MAKING REQUEST USING SPRING REST CLIENT

1. Create HttpHeaders with 2 header params. Those are Content-Type , Accept.

Code Sample :-

```
HttpHeaders headers = new HttpHeaders();  
Headers.add("Content-Type", "_____");
```

```
Headers.add("Accept", "_____");
```

2. Create HttpEntity with two parts Body(String) and headers(HttpHeaders).

Code Sample :-

```
HttpEntity<String> entity = new HttpEntity<String>();
```

3. Create RestTemplate Object.

```
RestTemplate template = new RestTemplate();
```

4. Make Request call (get()/post()/put().....) with inputs like URL, Entity, ResponseType.
5. Store Response data back into ResponseEntity<T> Object.

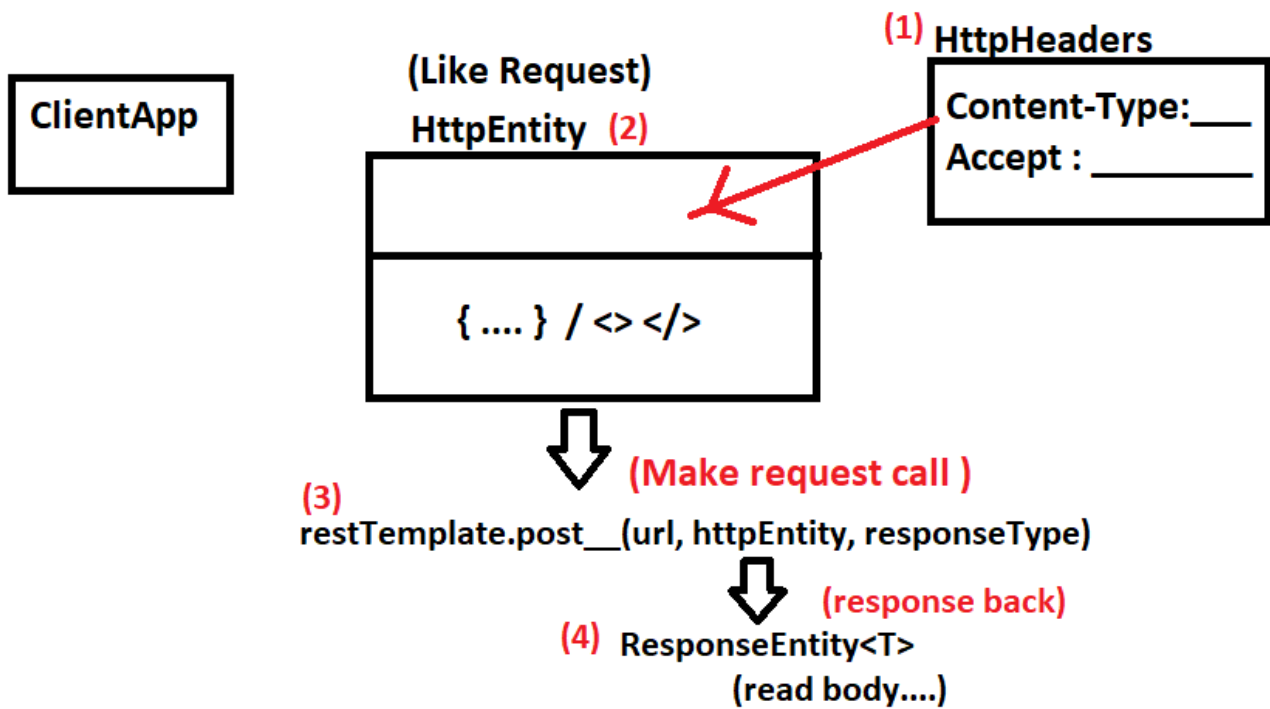
Code Sample :-

```
ResponseEntity<String> re = template.postForEntity(url, entity, String.class);
```

6. Print HttpStatus and Body

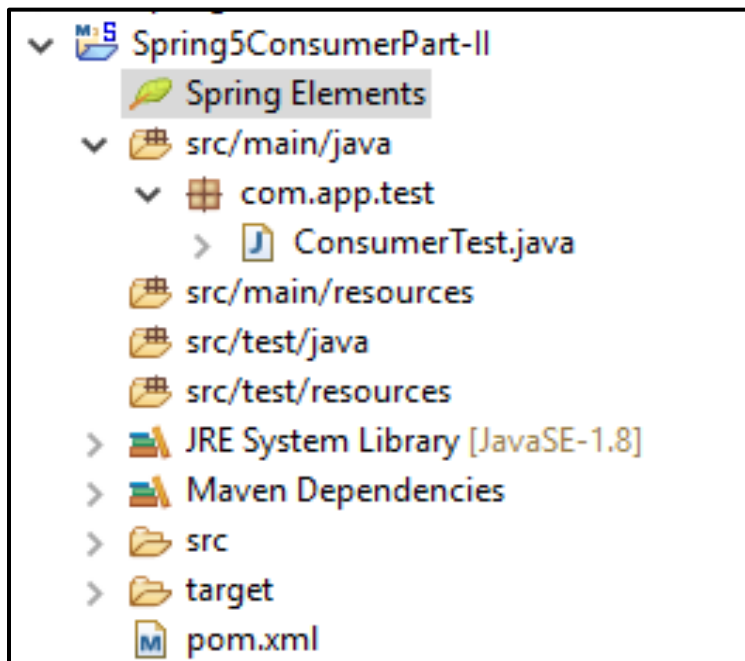
```
System.out.println(re.getStatusCodeValue());  
System.out.println(re.getStatusCode().name());  
System.out.println(re.getBody());
```

-----**DESIGN**-----



Example Program :-

Folder Structure :-



CODE :-

1. 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>org.sathyatech.app</groupId>
  <artifactId>Spring5ConsumerPart-II</artifactId>
  <version>1.0</version>

  <dependencies>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>5.0.6.RELEASE</version>
    </dependency>
    <dependency>
      <groupId>com.fasterxml.jackson.core</groupId>
      <artifactId>jackson-databind</artifactId>
      <version>2.9.5</version>
    </dependency>
    <dependency>
      <groupId>com.fasterxml.jackson.dataformat</groupId>
      <artifactId>jackson-dataformat-xml</artifactId>
      <version>2.9.5</version>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>

        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-
plugin</artifactId>
        <version>3.7.0</version>
```

```

        <configuration>
            <source>1.8</source>
            <target>1.8</target>
        </configuration>
    </plugin>
</plugins>
<finalName>Spring5RestProviderUsingXMLConPart-
II</finalName>
</build>

</project>

```

2. ConsumerTest.java :-

```

package com.app.test;
import org.springframework.http.HttpEntity;
import org.springframework.http.HttpHeaders;
import org.springframework.http.ResponseEntity;
import org.springframework.web.client.RestTemplate;

public class ConsumerTest {

    public static void main(String[] args) {
        //String body =
        "{\"empId\":\"202\",\"empName\":\"Ashu\",\"empSal\":\"36.9\"}";
        String body =
        "<Employee><empId>999</empId><empName>Ashutosh</empName><empSal>88.99</empSal></Employee>";
        String url =
        "http://localhost:2022/Spring5RestProviderUsingXMLConPart-II/rest/employee/data";

        // 1. Add Header
        HttpHeaders headers = new HttpHeaders();
        //headers.add("Content-Type", "application/json");
        headers.add("Content-Type", "application/xml");
        headers.add("Accept", "application/json");
    }
}

```

```
// 2. Entity
HttpEntity<String>he = new HttpEntity<String>(body,
headers);

// 3. Make Request call
RestTemplate template = new RestTemplate();

// 4. Get Response back also
ResponseEntity<String>re =
template.postForEntity(url, he, String.class);
System.out.println(re.getStatusCodeValue());
System.out.println(re.getStatusCode().name());
System.out.println(re.getBody());
    }
}
```

CHAPTER # 9 SPRING JMS

⇒ **JMS** : Java Message Service.

This concept is used to exchange (send / receive) messages (data) between two (or more) java applications.

⇒ These java application can run in same computer (same vm) or different computers (different vms).

[VM = Virtual Machine]

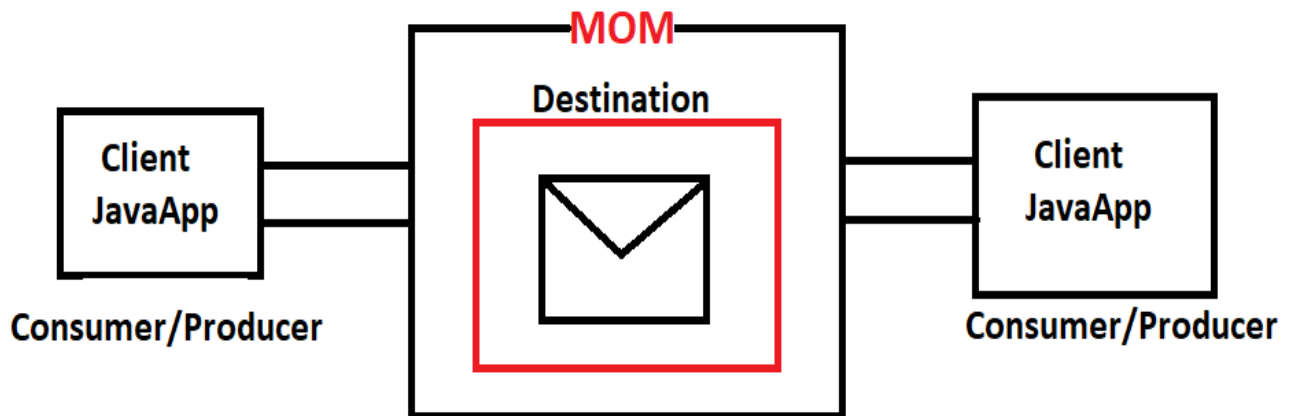
⇒ JMS is applicable only for java applications (any type web, stand alone, mobile , etc...).

⇒ One java application is called as client which can be acting as consumer or producer.

⇒ Two clients communicates using **MOM (Message Oriented Middleware)**.

- **Ex : Apache ActiveMQ , WebLogic MQs.**

⇒ MOM contains special memory to hold messages called as **"Destination"**. It is like a storage location for messages.



Types of JMS Communication:-

1. P2P (Peer-To-Peer) Communication:-

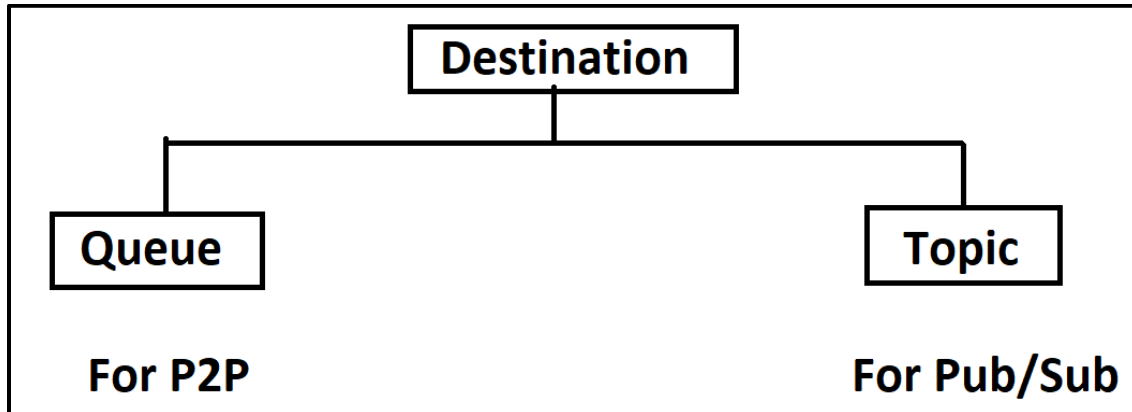
⇒ If message is taken (read) by only one Consumer (Client) then it is known as P2P Communication.

2. Pub/Sub (Publish-and-Subscribe) Communication :-

⇒ If one message is taken (read) by multiple Consumer (Clients) [Same copy] then it is known as Pub/Sub Communication.

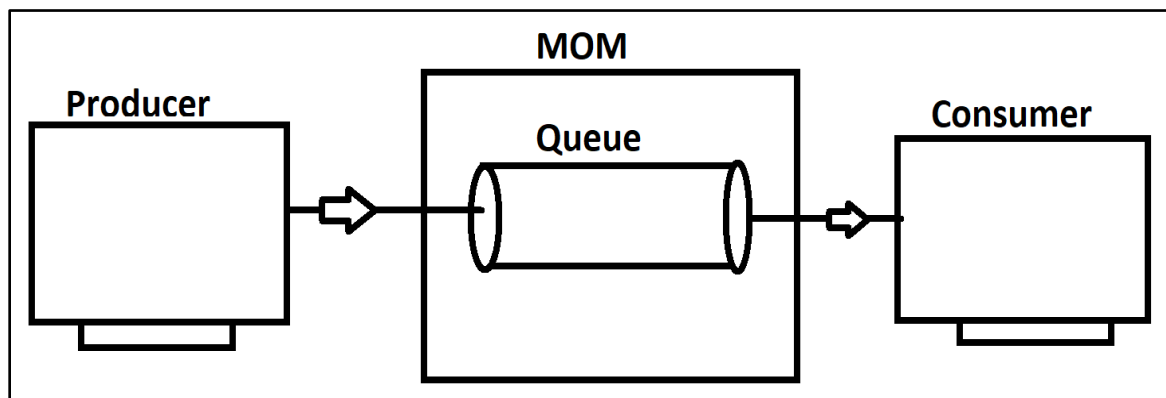
##Destination :-

⇒ It is a memory created in MOM to hold messages. It is two types based on Communications.



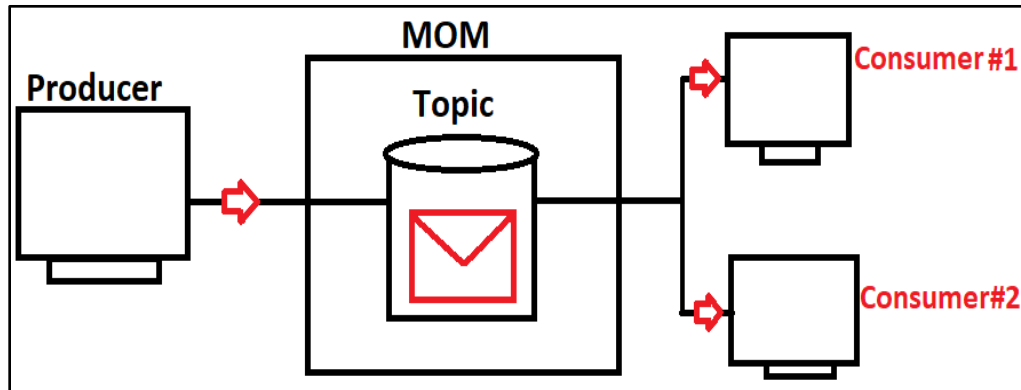
#) Queue :-

⇒ It is used to hold message in case of P2P Communication. It must be identified using one name **Ex: myqueue12**.



#) Topic :-

⇒ It is used to hold message given by Producer, this message will be broadcasted to (given to multiple) Consumers . one message multiple copies are created in memory.



#JMS DESIGN PROVIDED BY SUNMICROSYSTEM :-

#1:Download and setup MOM

(Ex: Apache ActiveMQ Software)

#2: Identify http port and tcp port:

Ex: Http port = **8161**

Tcp port = **61616**

⇒ Http port is used to view admin console of MOM, to see “how many Topic and Queue are created and their status” Tcp port is used to send/receive messages to/from MOM using Java JMS Application.

#3: Write code for client (Message Producer) and for client (Message Consumer).

******* Create multiple message Consumer in case of Pub/Sub (Topic).

-----Coding Steps-----

#a:Create Connection Factory Object using MOM URL.

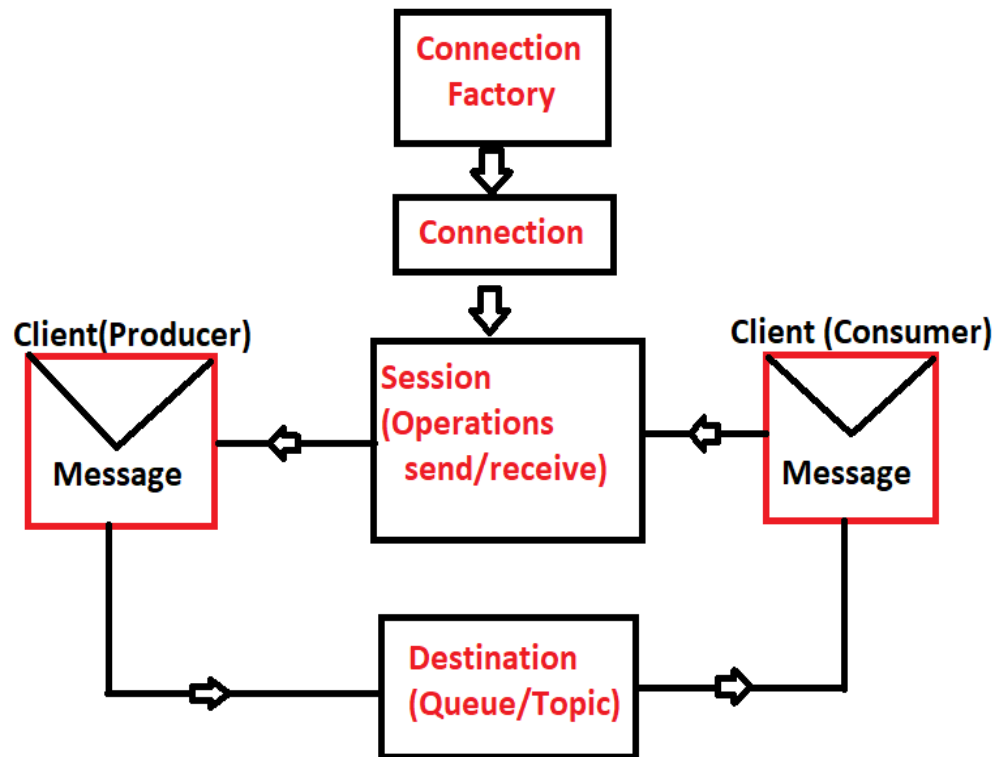
(BROKER URL)

For ActiveMQ: tcp://localhost:61616

#b:Create Connection between JMS App and MOM.

- #c: Create Session to do operations (send/recieve) and at same time Destination also (any order).
- #d: Destination must be Queue for P2P and Topic for Pub/Sub.
- #e: Use session to create send message to destination for producer app.
- #f: Use Session to read message from destination for Consumer Application.

-----GENERIC JMS DESIGN BY SUN-----



Spring JMS :-

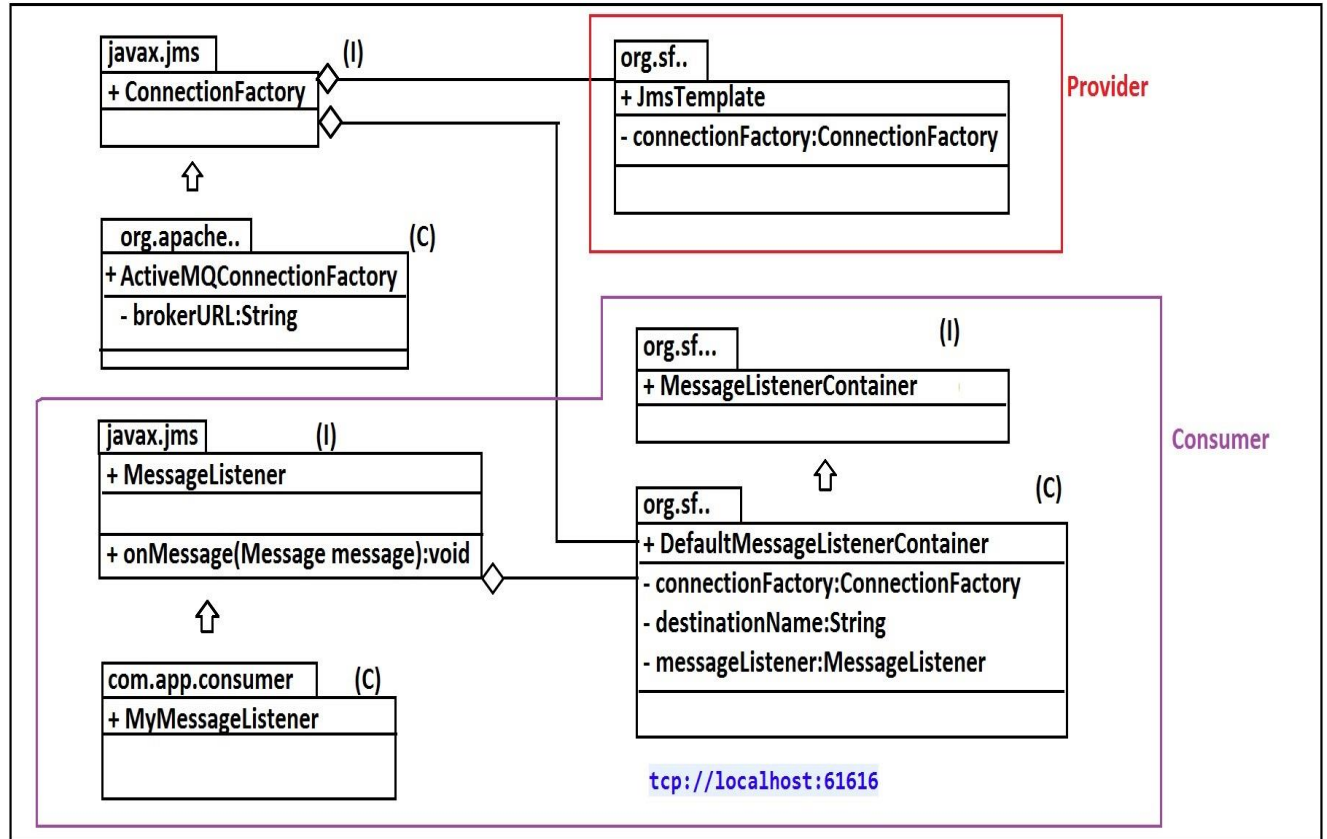
- ⇒ Spring JMS reduces coding lines provided by SUN JMS, using Template Design Pattern given as JMSTemplate.

JmsTemplate :-

- ⇒ Auto Creates Connection, Session, Destination and manages application link with MOM.

⇒ This Template supports for both Consumer and Producer but mainly used for Producer.

-----JMS Template Design-----



STEPS TO IMPLEMENT JMS PRODUCER CLIENT:-

#1: Create One Maven Project

> File > New > Maven Project

> Click checkbox

[v] Create Simple Project

> next > enter details:

groupId : org.sathyatech

artifactId : Spring5JMSProviderApp

version : 1.0

> Finish

#2:in pom.xml add dependencies for spring context, spring jms, spring ActiveMQ and build plugins for maven compiler.

#3:Update Maven Project (alt+F5)

> Right click on Project > Maven

> Update Project

#4:Write AppConfig.java folder under src/main/java folder

*) Configure 2 Beans here, Those are :

a. ActiveMQConnectionFactory

b. JmsTemplate

#5:Use JmsTemplate in Test class and send Message using method

send(String destination, messageCreate)

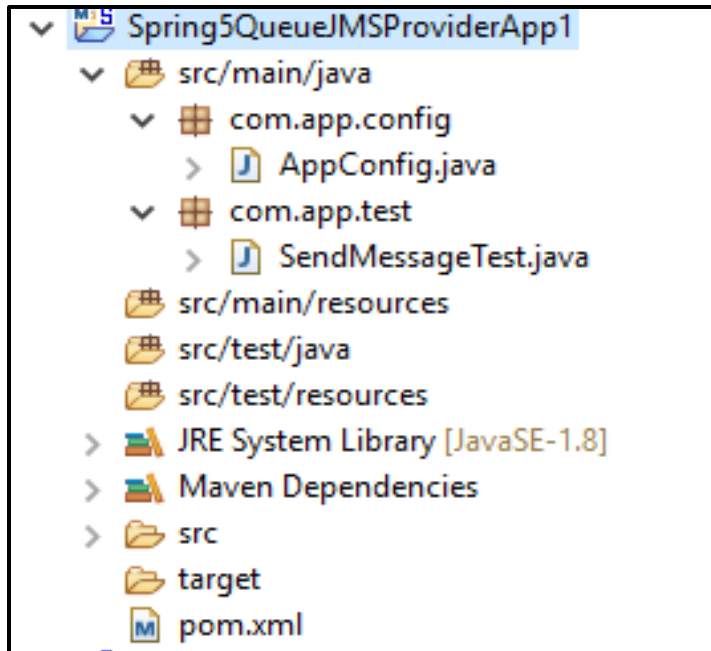
Here Default Destination type is Queue messageCreator is Functional interface (having only one abstract method) , so we can write lambda expression.

#6:send() method code looks like :-

JmsTemp.send("my-test-queue" , (ses)-> ses.createTextMessage("Hello Ashu!!!"));

Example Program:-

Folder System:-

**CODE :-****1. 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>org.sathyatech</groupId>
  <artifactId>Spring5JMSProvider</artifactId>
  <version>1.0</version>
  <dependencies>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-jms</artifactId>
      <version>5.0.6.RELEASE</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>5.0.6.RELEASE</version>
```

```
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>5.0.6.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.apache.activemq</groupId>
    <artifactId>activemq-spring</artifactId>
    <version>5.15.4</version>
</dependency>
</dependencies>
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.7.0</version>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
            </configuration>
        </plugin>
    </plugins>
</build>
</project>
```

2. AppConfig:-

```
package org.sathyatech.app.config;
import javax.jms.ConnectionFactory;
import org.apache.activemq.ActiveMQConnectionFactory;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.jms.core.JmsTemplate;
```

@Configuration

```
public class AppConfig {

    @Bean
    public ConnectionFactory connectionFactory() {
        ActiveMQConnectionFactory cf=new
ActiveMQConnectionFactory();
        cf.setBrokerURL("tcp://localhost:61616");
        return cf;
    }

    @Bean
    public JmsTemplate jmsTemplate() {
        JmsTemplate jt=new JmsTemplate();
        jt.setConnectionFactory(connectionFactory());
        return jt;
    }

}
```

3. Test class:-

```
package org.sathyatech.app.test;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.Session;
import org.sathyatech.app.config.AppConfig;
import
org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.jms.core.JmsTemplate;
import org.springframework.jms.core.MessageCreator;

public class SendMessageTest {

    public static void main(String[] args) {
        AnnotationConfigApplicationContext c=new
AnnotationConfigApplicationContext(AppConfig.class);
        JmsTemplate jt=c.getBean(JmsTemplate.class);
```

```
jt.send("my-test-spring", new MessageCreator() {  
  
    @Override  
    public Message createMessage(Session ses) throws  
JMSException {  
        return ses.createTextMessage("SAMPLE ONE");  
    }  
});  
c.close();  
}  
}
```

***** Start ActiveMQ before run above application :-**

Steps are:-

- >Goto Folder Apache-activemq-5.15.4
- > bin folder
- > choose os version win32/win64
- > click on bat file active.bat
- > ** wait for 3 minutes
- > Goto browser and enter URL
http://localhost:8161/admin
username, password : admin
- > click on menu option : Queues
- > observe details

➔ To Convert above client (Producer) code from **P2P (Queue)** concept to **pub/sub (Topic)** use below code in AppConfig over JmsTemplate Object.

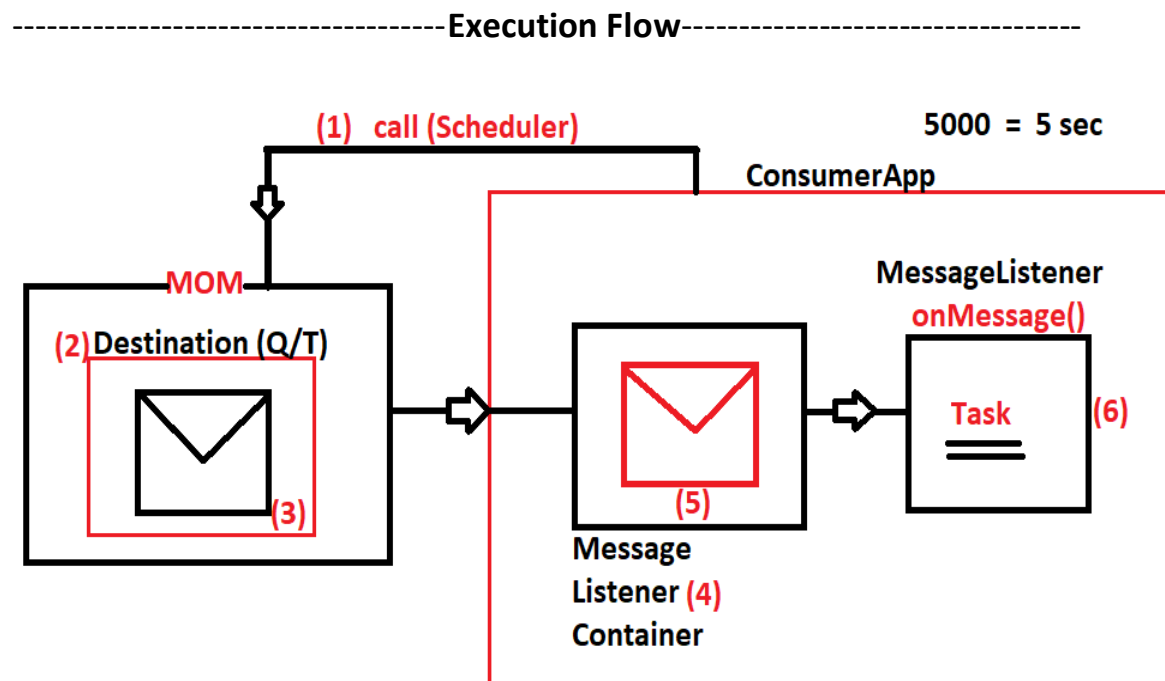
```
JmsTemplate jt = new JmsTemplate();  
Jt.setPubSubDomain(true); [or]  
Jt.setPubSubDomain(Boolean.TRUE);
```

#CONSUMER APPLICATION USING SPRING JMS:-

➔ Here (Client) Consumer Application make call to MOM (using schedulers with gap of 5000 mili sec = 5 sec by default).

This call gets executed in below order :-

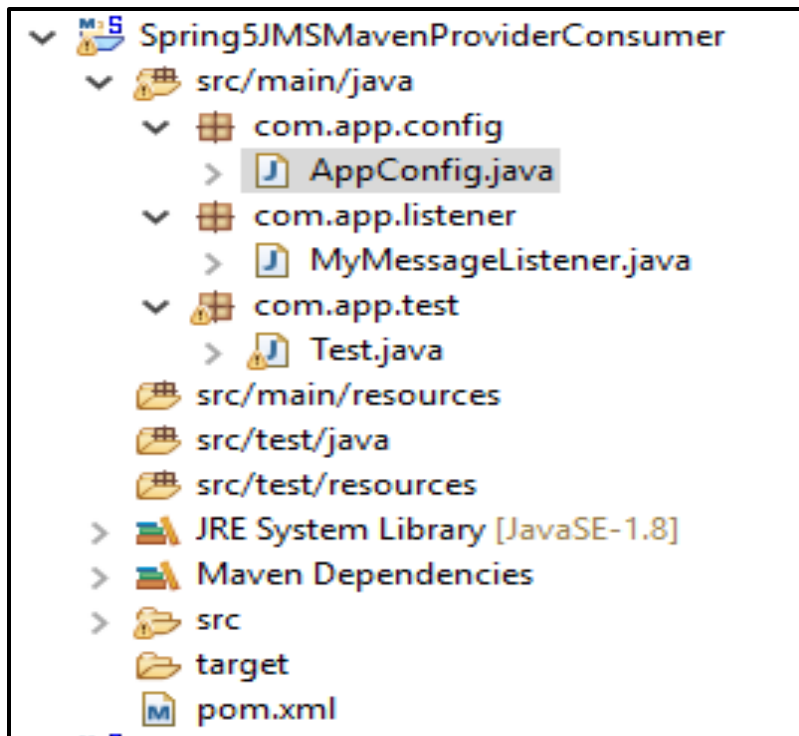
- #1: Identify MOM using ConnectionFactory.
- #2: Identify Destination in MOM and read Message if exist.
- #3: Copy Message into Consumer App memory known as **“MessageListenerContainer”**.
- #4: Once copied from MOM to Consumer hand over to **“MessageListener” (I)**.
- #5: **onMessage() method** executed task by taking Message as Input.



➔ If Message is received , then after executing onMessage() again Consumer makes a call with 5 sec gap (default).

Example Program :-

Folder System Structure:-



1. 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>org.sathyatech.app</groupId>
  <artifactId>Spring5JMSMavenProviderConsumer</artifactId>
  <version>1.0</version>

  <dependencies>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-jms</artifactId>
      <version>5.0.6.RELEASE</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
```

```

        <version>5.0.6.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>5.0.6.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>org.apache.activemq</groupId>
        <artifactId>activemq-all</artifactId>
        <version>5.15.4</version>
    </dependency>
</dependencies>
<build>
    <plugins>
        <plugin>

            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-
plugin</artifactId>
            <version>3.7.0</version>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
            </configuration>
        </plugin>
    </plugins>
</build>

</project>

```

2. AppConfig.java:-

```

package com.app.config;
import javax.jms.ConnectionFactory;
import javax.jms.MessageListener;
import org.apache.activemq.ActiveMQConnectionFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;

```

```
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.jms.annotation.EnableJms;
import org.springframework.jms.listener.DefaultMessageListenerContainer;
import org.springframework.jms.listener.MessageListenerContainer;

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

    @Autowired
    private MessageListener messageListener;

    @Bean
    public ConnectionFactory connectionFactory() {
        ActiveMQConnectionFactory c=new ActiveMQConnectionFactory();
        c.setBrokerURL("tcp://localhost:61616");
        return c;
    }

    @Bean
    public MessageListenerContainer listenerContainer() {
        DefaultMessageListenerContainer m=new
DefaultMessageListenerContainer();
        m.setConnectionFactory(connectionFactory());
        m.setDestinationName("my-test-spring");
        m.setMessageListener(messageListener);
        return m;
    }
}
```

3. MyMessageListener.java:-

```
package com.app.listener;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.TextMessage;
```

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

@Component
public class MyMessageListener implements MessageListener {

    @Override
    public void onMessage(Message message) {
        TextMessage tm = (TextMessage) message;
        try {
            System.out.println(tm.getText());
        } catch (JMSEException e) {
            e.printStackTrace();
        }
    }
}
```

4. Test.java:-

```
package com.app.test;
import
org.springframework.context.annotation.AnnotationConfigApplicationContext;
import com.app.config.AppConfig;
import com.app.listener.MyMessageListener;

public class Test {

    public static void main(String[] args) {
        AnnotationConfigApplicationContext ac = new
        AnnotationConfigApplicationContext(AppConfig.class);
        MyMessageListener ms = ac.getBean(MyMessageListener.class);
    }
}
```

EXECUTION STEPS #:

#1: Start Apache ActiveMQ

URL : <http://localhost:8161/admin>

Username/password : admin/admin

#2: Run Consumer Application

(Create multiple Consumers in multiple work spaces, just modify
oneMessage() method data)

#3: Run Provider Application.

****** Make sure that Consumer Destination name must match with Provider
Destination name Else no output.

CHAPTER#10 SPRING SECURITY (JAAS)

JAAS: (Java Authentication and Authorization Service)

- ⇒ It is a concept used to secure application URL's
- ⇒ It will secure URL's in 2 level
 - a. User Identity [un , pwd]
 - b. Role verification

Authentication :

Work on store and retrieve use identity details like username , password , role it will compare only name and password not role with end user input.

Authorization :

It will security work on login and role management of application using JAAS.

- ⇒ Spring security works on login and role management of application using JAAS.
- ⇒ Security is provided to URL using managers.
 - a. Authentication Manager.
 - b. Authorization Manager.

a) Authentication Manager:

It will store details of user in RAM on DB and verifies when user try to login.

Types of Authentication

1. InMemoryAuthentication

Storing details in RAM.

2. JdbcAuthentication

Storing details (un ,pwd , role) in DB using JDBC.

3. UserDetailsService:

Storing details (un ,pwd , role) in DB using ORM.

b) AuthorizationManager:

It will provide details of URL's "who can access what URL? " provided types as:

1. permitAll

everyone can access no login & no role required.

2. hasAuthority

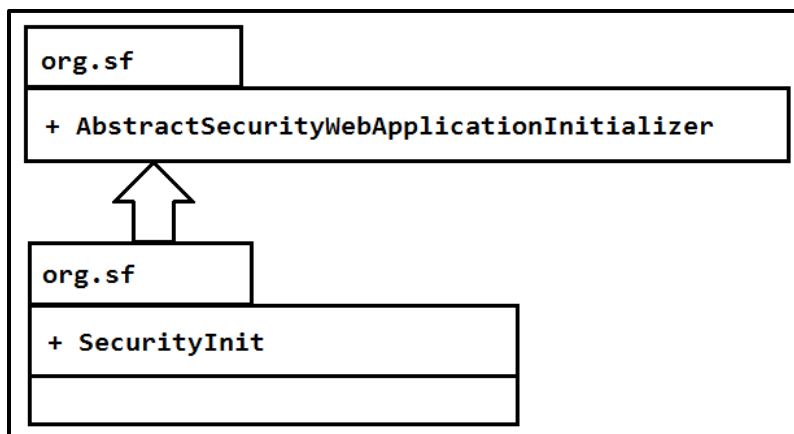
URL can be accessed by users who must login and should have expected role. If login/role failed cannot access URL.

3. authentication

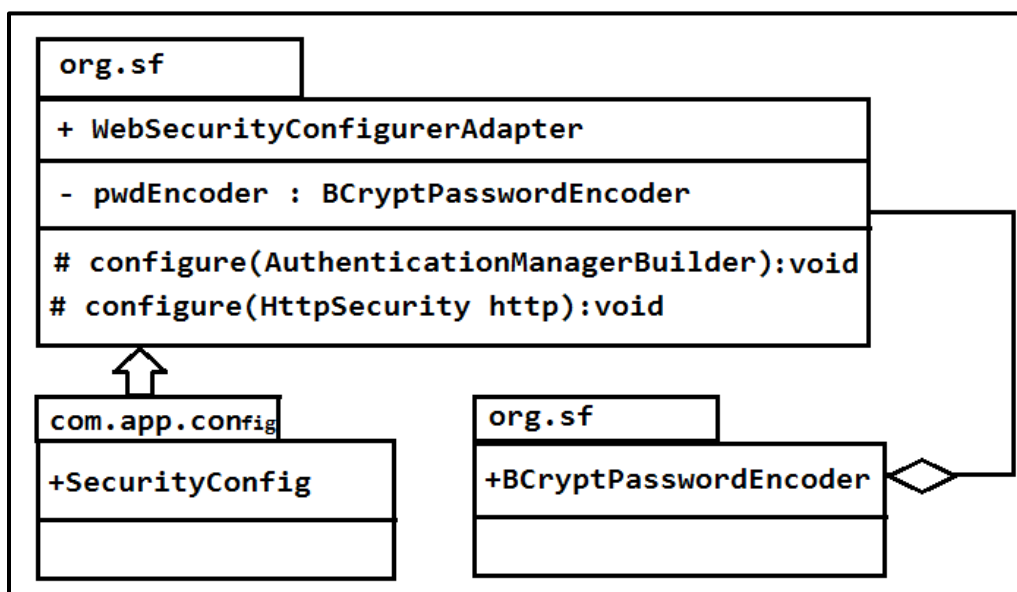
URL can be accessed by users who must login only. Role check not require

Spring Security Design

Level 1: Enable Security Filter



Level 2: Configure Authentication / Authorization manager.



Step 1: write one web application using WEB-MVC with multiple URL method.

Step 2: write on spring config file to provide authentication and authorization manager details

EX: SpringSecurity which should extends class WebSecurityConfigurerAdapter and Override 2 method.

Step 3: use any password encoder for securing password

- a. NoOperationPwdEncoder
- b. BCryptPasswordEncoder. (Binary Cryptography) etc.

Step 4: Enable security filter by writing one class. That extends "AbstractSecurityWebApplicationInitializer"

1. InMemoryAuthentication

It will store data in RAM , it is used only for testing process , if DB is not installed in system. This is best way to test application.

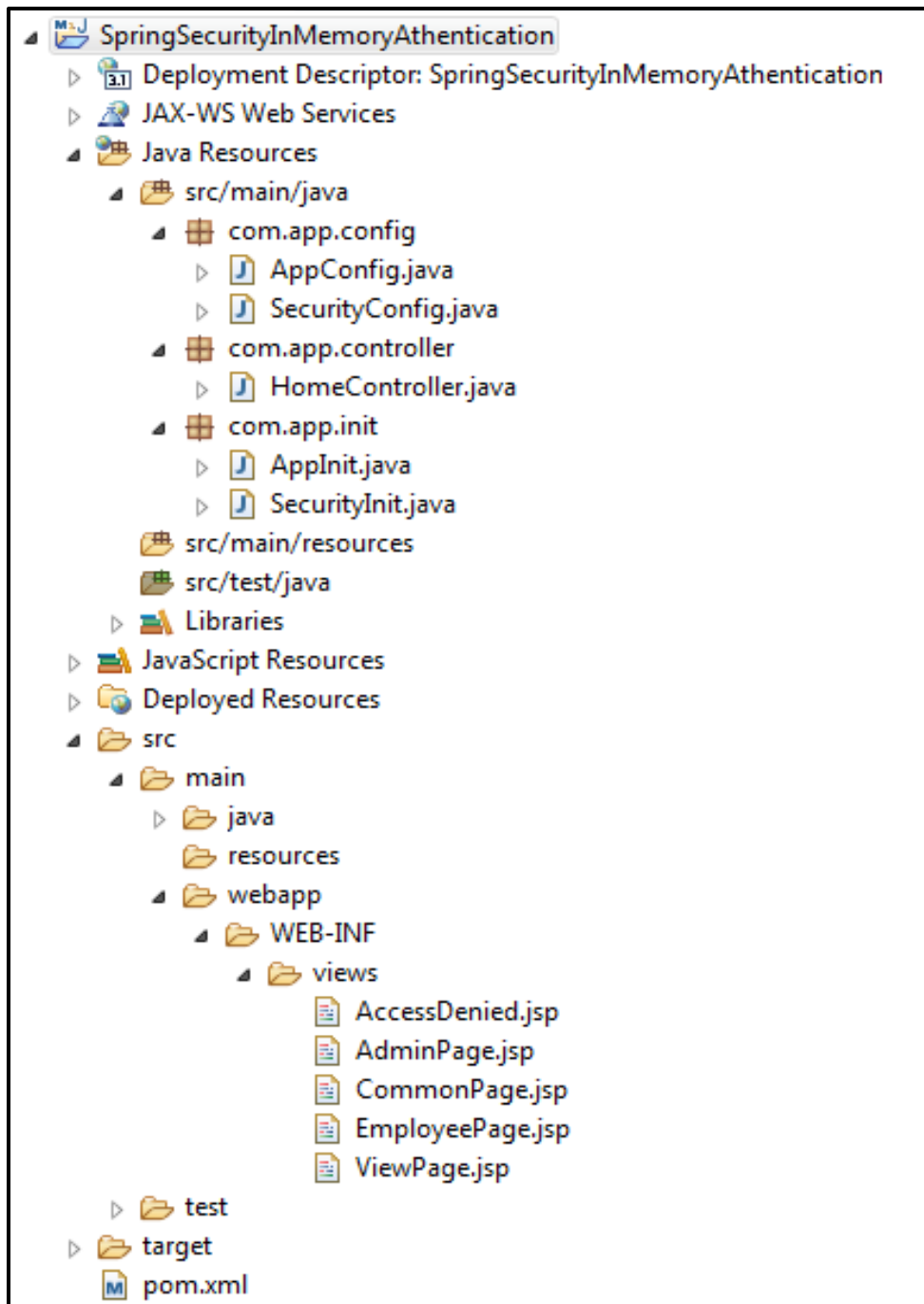
- ⇒ At runtime we cannot create new user.
- ⇒ Again stop server , modify code and start.
- ⇒ Stores data in RAM , on every re-start of server memory deleted and created again.
- ⇒ Format look like:

USER	PASSWORD	AUTHORITIES
SAM	ddf*26*	ADMIN
RAM	\$#ed3f	EMP , ADMIN
VICKY	2734dd	STUDENT

Create one web application using spring WEB-MVC in below format.

URL	OUTPUT PAGES	AUTHORITY
/all	commonpage.jsp	permitAll
/view	viewPage.jsp	authentication
/emp	EmployeePage.jsp	hasAuthority
/admin	AdminPage.jsp	hasAuthority
/denied	AccessDenied.jsp	Invalid Access

SETUP



1. 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/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
```

```
<groupId>org.sathyatech</groupId>
<artifactId>SpringSecurityInMemoryAthentication</artifactId>
<packaging>war</packaging>
<version>0.0.1-SNAPSHOT</version>
<name>SpringSecurityInMemoryAthenticationMavenWebapp</name>
<url>http://maven.apache.org</url>

<dependencies>
  <dependency>

  <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-web</artifactId>
    <version>5.0.2.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.0.3.RELEASE</version>
  </dependency>
  <dependency>

  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-config</artifactId>
    <version>5.0.2.RELEASE</version>
  </dependency>
</dependencies>
<build>
  <plugins>
    <plugin>

    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
      <version>3.7.0</version>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
      </configuration>
    </plugin>
```

```
        <plugin>
            <artifactId>maven-war-plugin</artifactId>
            <version>2.4</version>
            <configuration>

        <failOnMissingWebXml>>false</failOnMissingWebXml>
        </configuration>

    </plugin>
</plugins>
</build>
</project>
```

2. Config File (AppConfig.java)

```
package com.app.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Import;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.view.InternalResourceViewResolver;

@Configuration
@ComponentScan(basePackages = "com.app")
@Import(SecurityConfig.class)
@EnableWebMvc
public class AppConfig {
    @Bean
    public BCryptPasswordEncoder pwdEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```

```
@Bean
public InternalResourceViewResolver ivr() {
    InternalResourceViewResolver ivr =
        new InternalResourceViewResolver();
        ivr.setPrefix("/WEB-INF/views/");
        ivr.setSuffix(".jsp");
        return ivr;
    }
}
```

3. Spring Flie (SecurityConfig.java)

```
package com.app.config;
```

```
import org.springframework.beans.factory.annotation.Autowired;
;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.authentication
n.builders.AuthenticationManagerBuilder;
import
org.springframework.security.config.annotation.web.builders.
HttpSecurity;
import
org.springframework.security.config.annotation.web.configura
tion.EnableWebSecurity;
import
org.springframework.security.config.annotation.web.configura
tion.WebSecurityConfigurerAdapter;
import
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import
org.springframework.security.web.util.matcher.AntPathRequest
Matcher;
```

```
@EnableWebSecurity
@Configuration
```

```
public class SecurityConfig extends WebSecurityConfigurerAdapter
{
    @Autowired
    private BCryptPasswordEncoder pwdEnc;

    @Override
    protected void
    configure(AuthenticationManagerBuilder auth) throws Exception
    {
        auth.inMemoryAuthentication().withUser("Sam")
            .password(pwdEnc.encode("Sam")).authorities("EMP");

        auth.inMemoryAuthentication().withUser("Ram")
            .password(pwdEnc.encode("Ram")).authorities("ADMIN");

        auth.inMemoryAuthentication().withUser("Vicky")
            .password(pwdEnc.encode("Vicky")).authorities("STUDENT" ,
            "MGR");
    }

    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
            .antMatchers("/all").permitAll()
            .antMatchers("/emp").hasAuthority("EMP")
            .antMatchers("/admin").hasAuthority("/ADMIN")
            .anyRequest().authenticated()
            .and().formLogin().defaultSuccessUrl("/view")
            .and().logout().logoutRequestMatcher
                (new AntPathRequestMatcher("/logout"))

            .and().exceptionHandling().accessDeniedPage("/denied");
    }
}
```

4. Init File (AppInit.java)

```
package com.app.init;

import
org.springframework.web.servlet.support.AbstractAnnotationCo
nfigDispatcherServletInitializer;
```

```
import com.app.config.AppConfig;

public class AppInit extends AbstractAnnotationConfigDispatcherServletInitializer {

    @Override
    protected Class<?>[] getRootConfigClasses() {
        return new Class[] { AppConfig.class };
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    protected String[] getServletMappings() {
        return new String[] { "/" };
    }
}
```

5. Init File (SecurityInit.java)

```
package com.app.init;

import org.springframework.security.web.context.AbstractSecurityWebApplicationInitializer;

public class SecurityInit extends AbstractSecurityWebApplicationInitializer { }
```

6. HomeController.java

```
package com.app.controller;

import org.springframework.stereotype.Controller;
```



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

@Controller
public class HomeController {
    @RequestMapping("/all")
    public String all() {
        return "CommonPage";
    }

    @RequestMapping("/emp")
    public String emp() {
        return "EmployeePage";
    }

    @RequestMapping("/view")
    public String view() {
        return "ViewPage";
    }

    @RequestMapping("/admin")
    public String admin() {
        return "AdminPage";
    }

    @RequestMapping("/denied")
    public String denied() {
        return "AccessDenied";
    }
}
```

7. JSP Files

a) AccessDenied.jsp

```
<%@page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
```

```
<html>
<head>
<metahttp-equiv="Content-Type"content="text/html;
charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h1>Youcan not access this URL!!</h1>
<ahref="Logout">Goto Home</a>
</body>
</html>
```

b) AdminPage.jsp

```
<%@pagelanguage="java"contentType="text/html; charset=ISO-
8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPEhtmlPUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<metahttp-equiv="Content-Type"content="text/html;
charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h1>welcome to Admin page!!</h1>
<ahref="Logout">Goto Home</a>
</body>
</html>
```

c) CommonPage.jsp

```
<%@pagelanguage="java"contentType="text/html; charset=ISO-
8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPEhtmlPUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h1>Welcome to All!!</h1>

</body>
</html>
```

d) EmployeePage.jsp

```
<%@page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h1>welcome to Employee Page!!</h1>
<a href="Logout">Goto Home</a>
</body>
</html>
```

e) ViewPage.jsp

```
<%@page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
```

```
<title>Insert title here</title>
</head>
<body>
<h1>Welcome to View Page!! </h1>
<a href="Logout">Goto Home</a>
</body>
</html>
```

antMatchers("/urlPattern"):

This method is used to provide URL-Patterns and their security levels.

1. Here we can use symbol like

*	::	Any Character
**	::	Multi-level path
/*	::	one level path

2. Use security level method like:

.permitAll() :: every one can access.

.hasAuthority("r1")::only given role(R1) can view this after login.

.authenticated() ::indicates only login , no role required.

.hasAnyAuthority("r1" , "r2" , "r3") ::user should have any role in given list and can view after.

3. **anyRequest()** ::it is used to provide all URL's which are not specified in configuration.

Ex:IN URL's only /emp , /admin provided in config , to indicate remaining 198 URL's use anyRequest()

1. **Every URL can be accessed by everyone.**

Ans) **.anyRequest().permitAll()**

2. **/emp can be accessed by ADMIN or EMPLOYEE roles after login.**

Ans) `.antMatchers("/emp").hasAnyAuthority("ADMIN", "EMPLOYEE")`

3. /home can be accessed by everyone.

Ans) `.antMatchers("/home").permitAll()`

**4. All product operations are accessed by Employee only
[/product/view , /product/get , /product/edit , /product/save]**

Ans) `.antMatchers("/product**").hasAuthority("Employee")`

Or

`.antMatchers("/product/view", "/product/get", "/product/edit",
"/product/save")`

- Spring provide default login form without writing (JSP / HTML) code by programmer.

`.and().formLogin()`

- To specify "after login , go to default URL " code is:

`.and().formLogin().defaultSuccessUrl("/view")`

- To specify logout URL Pattern

`.and().logout.logoutRequestMatcher
(newAntPathRequestMatcher("/logout"))`

- To specify access denied error page.

`.and().exceptionHandling().accessDeniedPage("/denied")`

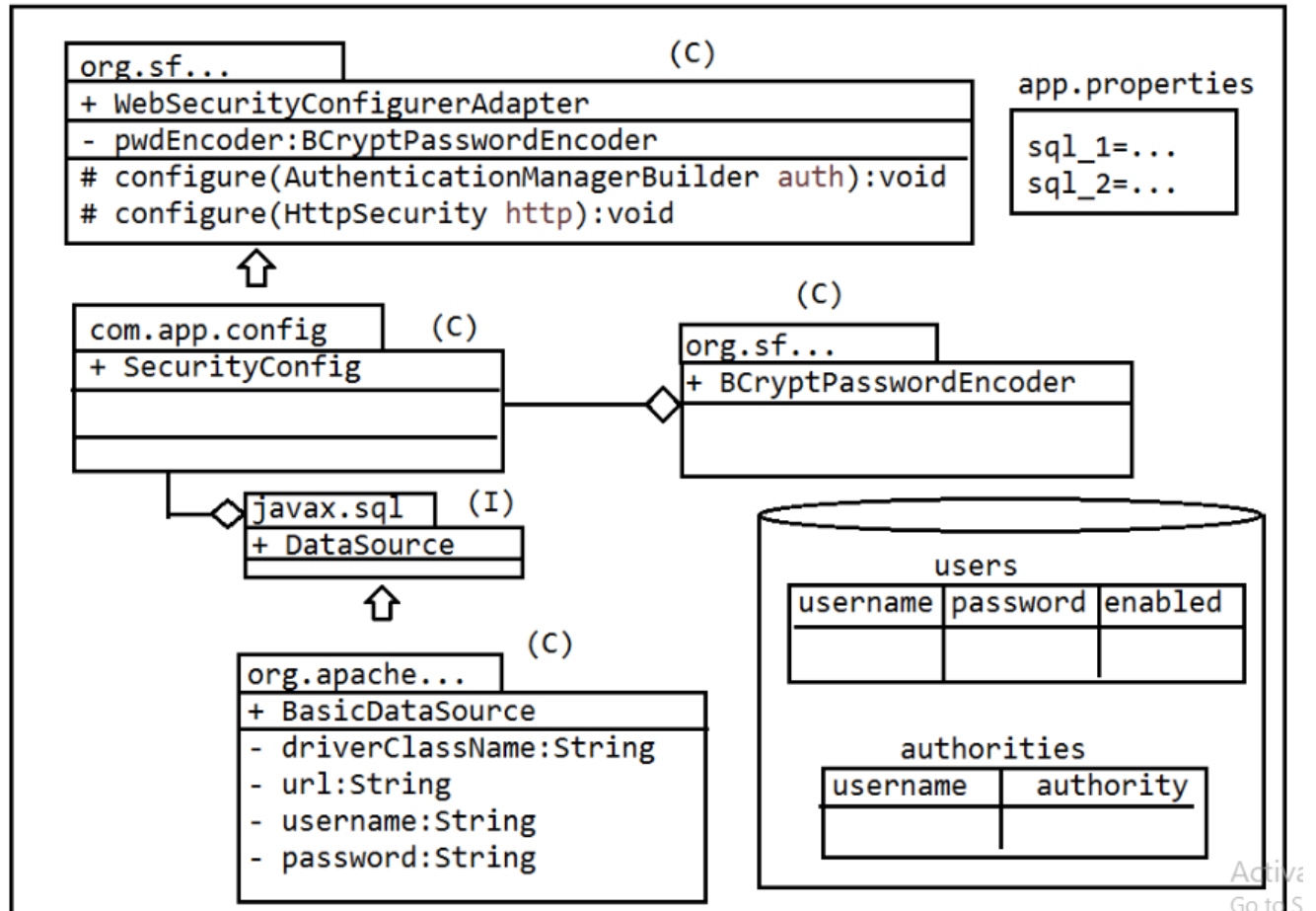
2) JDBCAuthentication:-- It will store data in DB tables(2) table-1 stores user details and table-2 stores authorities.

=>Communicate to DB table using Spring JDBC uses DataSource (javax.sql) interface with two special SQL queries.

SQL#1>Load user by username.

SQL#2>Load Authorities by username.

Design:--



Coding Steps:--

#1>Configure DataSource(I) any one Impl class object in AppConfig. Example use DriverManagerDataSource, BasicDataSource etc...

#2>Define two SQL Queries which will be executed on click login button. These SQLs gets data from 2 DB tables one is "users" based on username and other one is "authorities" based on username.

Ex:-- SQL>select username, password, enable from users where username=?

SQL>select username, authority from authorities where username=?

#3>use passwordEncoder and app.properties (Environment) [optional].

#4>Create two table tables as given in Database also insert few rows I both tables.

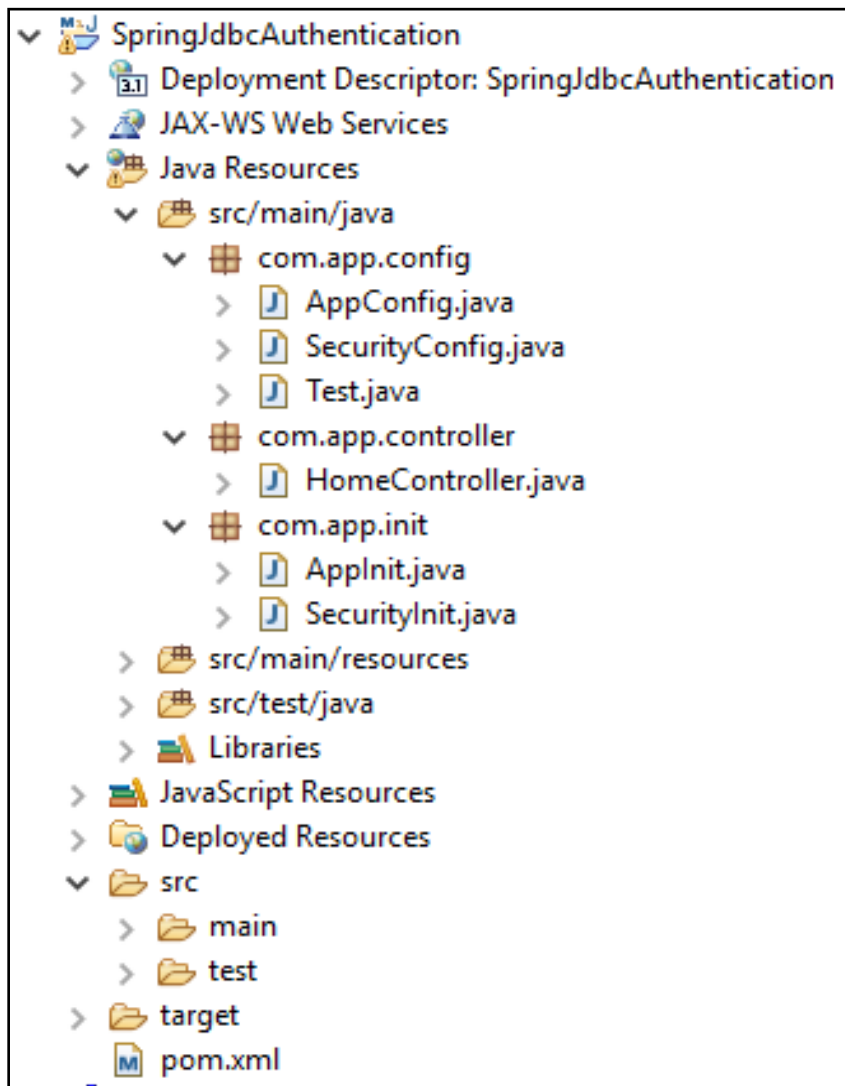
#5>Configure JDBC Authentication Manager using userSQL, AuthoritiesSQL, DataSource and Password Encoder. Code looks like.

protected void configure(AuthenticationManagerBuilder auth) throws Exception

```
{  
    auth.jdbcAuthentication()  
    .usersByUsernameQuery("SQL_1")  
    .authoritiesByUsernameQuery("SQL_2")  
    .dataSource(ds)  
    .passwordEncoder(pwdEncoder);  
}
```

#6>Configure URLs-Role Management using configure (HttpSecurity) method

Folder Structure:--



Code:--**1>app.properties:--**

dc=com.mysql.jdbc.Driver

url=jdbc:mysql://localhost:3306/test

un=root

pwd=root

prefix=/WEB-INF/views/

suffix=.jsp

usernameSQL=select username, password, enabled from users where
username=?

authSQL=select username, authority from authorities where username=?

2>AppInit:--

package com.app.init;

import org.springframework.web.servlet.support.

AbstractAnnotationConfigDispatcherServletInitializer;

import com.app.config.AppConfig;

```
public class AppInit extends AbstractAnnotationConfigDispatcherServletInitializer
{
    @Override
    protected Class<?>[] getRootConfigClasses() {
        return new Class[] {AppConfig.class};
    }
    @Override
    protected Class<?>[] getServletConfigClasses() {
        return null;
    }
    @Override
    protected String[] getServletMappings() {
        return new String[] {"/"};
    }
}
```


3>AppConfig:--

```
package com.app.config;
import org.apache.commons.dbcp2.BasicDataSource;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Import;
import org.springframework.context.annotation.PropertySource;
import org.springframework.core.env.Environment;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.view.InternalResourceViewResolver;
```

@Configuration

@ComponentScan(basePackages="com.app")

@EnableWebMvc

@PropertySource("classpath:app.properties")

@Import({SecurityConfig.class})

public class AppConfig

{

 @Autowired

 private Environment env;

 @Bean

 public BCryptPasswordEncoder pwdEncoder() {

 return new BCryptPasswordEncoder();

 }

 @Bean

 public BasicDataSource ds() {

 BasicDataSource ds=new BasicDataSource();

 ds.setDriverClassName(env.getProperty("dc"));

```
        ds.setUrl(env.getProperty("url"));
        ds.setUsername(env.getProperty("un"));
        ds.setPassword(env.getProperty("pwd"));
        return ds;
    }
    @Bean
    public InternalResourceViewResolver viewResolver() {
        InternalResourceViewResolver v=new
InternalResourceViewResolver();
        v.setPrefix(env.getProperty("prefix"));
        v.setSuffix(env.getProperty("suffix"));
        return v;
    }
}
```

4>HomeController:--

```
package com.app.controller;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
@Controller
public class HomeController
{
    @RequestMapping("/admin")
    public String showAdmin() {
        return "AdminPage";
    }
    @RequestMapping("/all")
    public String showAll() {
        return "CommonPage";
    }
    @RequestMapping("/emp")
    public String showEmp() {
        return "EmployeePage";
    }
}
```

```
@RequestMapping("/view")
public String showView() {
    return "ViewPage";
}

@RequestMapping("/denied")
public String showDenied() {
    return "AccessDenied";
}
}
```

5>SecurityInit:--

```
package com.app.init;
import org.springframework.security.web.context.
AbstractSecurityWebApplicationInitializer;

public class SecurityInit extends AbstractSecurityWebApplicationInitializer
{ }
```

6>SecurityConfig:--

```
package com.app.config;
import javax.sql.DataSource;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;
import org.springframework.core.env.Environment;
import org.springframework.security.config.annotation.authentication.builders.
AuthenticationManagerBuilder;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.
EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.
WebSecurityConfigurerAdapter;
```

```
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.web.util.matcher.AntPathRequestMatcher;
```

```
@Configuration
```

```
@EnableWebSecurity
```

```
@PropertySource("classpath:app.properties")
```

```
public class SecurityConfig extends WebSecurityConfigurerAdapter
```

```
{
```

```
    @Autowired
```

```
    private Environment env;
```

```
    @Autowired
```

```
    private BCryptPasswordEncoder pwdEncoder;
```

```
    @Autowired
```

```
    private DataSource dataSource;
```

```
    @Override
```

```
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
```

```
        auth.jdbcAuthentication()
```

```
        .dataSource(dataSource)
```

```
        .usersByUsernameQuery(env.getProperty("usernameSQL"))
```

```
        .authoritiesByUsernameQuery(env.getProperty("authSQL"))
```

```
        .passwordEncoder(pwdEncoder);
```

```
    }
```

```
    @Override
```

```
    protected void configure(HttpSecurity http) throws Exception {
```

```
        http.authorizeRequests()
```

```
            .antMatchers("/all").permitAll()
```

```
            .antMatchers("/admin").hasAuthority("ADMIN")
```

```
            .antMatchers("/emp").hasAuthority("EMP")
```

```
            .anyRequest().authenticated()
```

```
        .and()
```

```
.formLogin()
.defaultSuccessUrl("/view",true)

.and()
.logout()
.logoutRequestMatcher(new AntPathRequestMatcher("/logout"))

.and()
.exceptionHandling()
.accessDeniedPage("/denied");
    }
}
```

7>Test Class:--

```
package com.app.config;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

public class Test
{
    public static void main(String[] args)
    {
        BCryptPasswordEncoder enc=new BCryptPasswordEncoder();
        String str=enc.encode("uday");
        System.out.println(str);
    }
}
```

JSP Pages:--

1>AdminPage.jsp:--

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

```
<html> <head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title> Welcome to Admin Page</title>
</head> <body>
Welcome to Admin Page
<br/>
<a href="logout">Logout</a>
</body> </html>
```

2>ViewPage.jsp:--

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html> <head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title> Welcome to View Page</title>
</head> <body>
Welcome to All Page
</body> </html>
```

3>CommonPage.jsp:--

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html> <head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title> Welcome to Common Page </title>
</head> <body>
Welcome to Common Page
<br/>
<a href="logout">Logout</a>
```

```
</body> </html>
```

4>EmployeePage.jsp:--

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html><head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title> Welcome to Employee Page</title>
</head><body>
Welcome to Employee Page
<br/>
<a href="logout">Logout</a>
</body></html>
```

5>AccessDenied.jsp:--

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html> <head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Welcome to AccessDenied Page</title>
</head><body>
We are sorry!!
<br/>
<a href="logout">Go Home</a>
</body></html>
```

Execution Process:--

Step#1:--Create Bellow table and insert the value inside the following table.

##Tables##

```
SQL>create table users (username varchar (50) not null primary key,  
password varchar (100) not null, enabled boolean not null);  
SQL>create table authorities (username varchar(50) not null, authority  
varchar(50) not null, constraint authusersFk foreign key(username) references  
users(username));  
SQL>create unique index usernameauthindex on authorities (username,  
authority);
```

Step#2:--**##BASCI DATA##**

```
SQL>insert into users (username, password, enabled) values ('admin',  
'$2a$10$A.Gbvyy89BD7tJAa0Q8cxe8S/Zh5b8nrxWrJLk8IKmXEQB4UAvery',true);
```

```
SQL>insert into authorities (username, authority) values ('admin', 'ADMIN');
```

```
SQL>insert into users (username, password, enabled) values  
('sam','$2a$10$OdTpsFqtYyA7n3GQYit7s.bFMaO.n0fEnxG4WwJ8ZI7IRYOF1srnS',t  
rue);
```

```
SQL>insert into authorities (username, authority) values ('sam', 'EMP');
```

```
SQL>insert into users (username, password, enabled) values  
('ram','$2a$10$0KCebvliUM7SJbxzaO/xn.79QdA9ImQuhNDnXDTuFsvr.UQyN/s1W  
,true);
```

```
SQL>insert into authorities (username, authority) values ('ram', 'STD');
```

##APP Queries##

```
SQL>select username, password, enabled from users where username=?
```

```
SQL>select username, authority from authorities where username=?
```

Step#3:--Run on server and provider user and password as (admin):

```
=>http://localhost:3030/Spring5SecurityInMemotyAuth/admin
```

```
=>http://localhost:3030/SpringJdbcAuthentication/view
```

```
=>http://localhost:3030/ SpringJdbcAuthentication/all
```

```
=>http://localhost:3030/ SpringJdbcAuthentication/emp
```


3) Spring Security Using ORM:--

=>To implement this process, we should follow two stages.

#a>User Module Implementation (User Register Process]

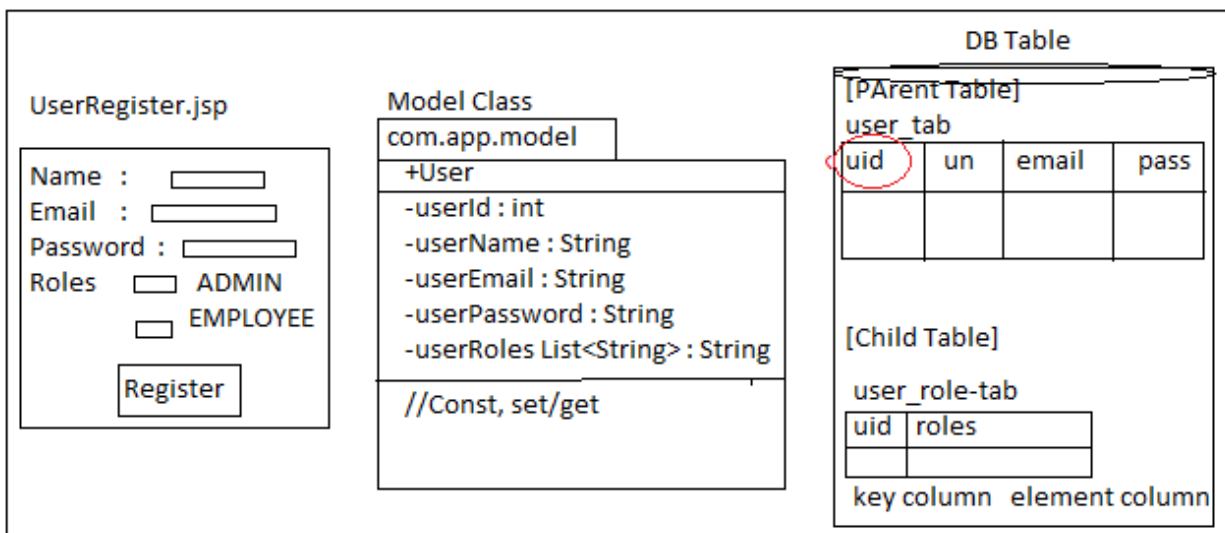
#b>User Module Login Process with Security Adapter

#a>User Module Implementation:--Here, define one JSP (UI) Page for User Register process. This data should be stored in DB table using ORM (Model class).

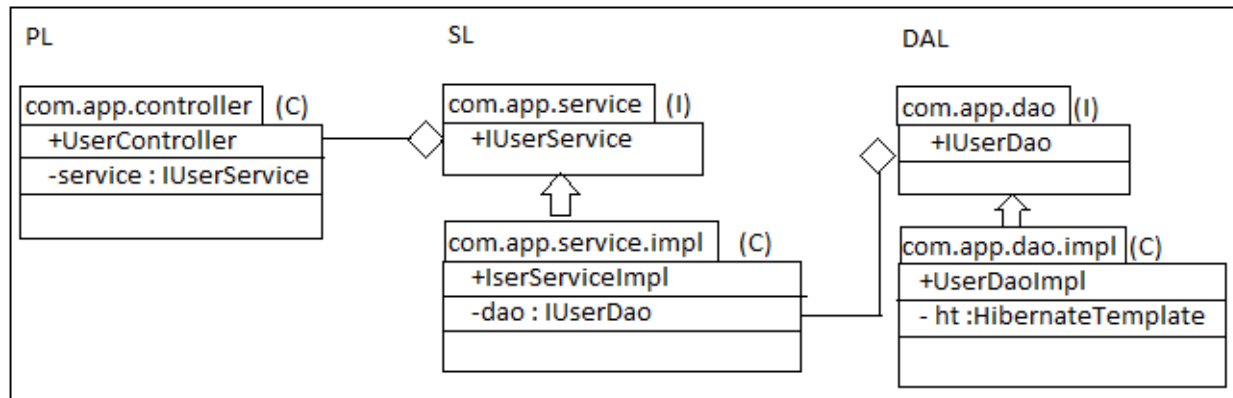
=>Data inserted into two tables, user basic details are inserted into parent table and roles (Authorities) are inserted into child table.

=>For this we need to define Layer design of USER Module using PL, SL and DAL.

-----Designs-----



Layers Design:--



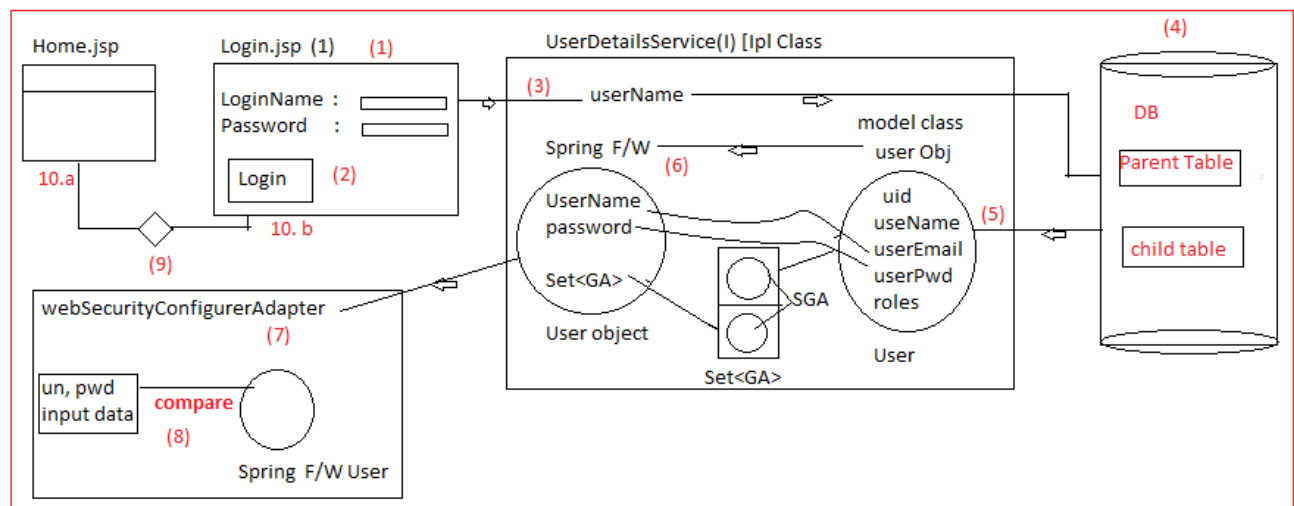
#b> User, define one Login.jsp page with basic input like username and password.

=>On click Login button, read user details from DB based on Username input using “UserDetailsService”(I) impl class, which also converts Model class.

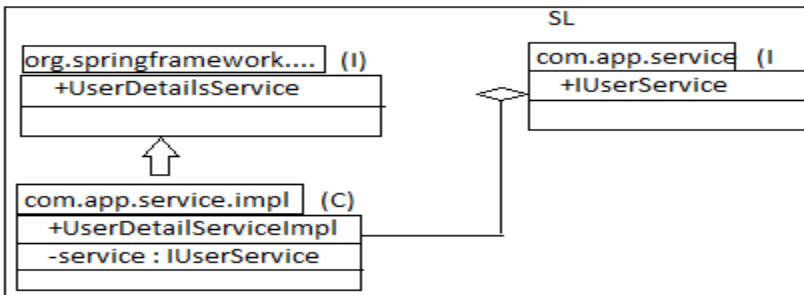
User (com.app.model) object to Spring f/w User (org.springframework.security.core.userdetails).

=>Here roles are converted to Set of GranteAuthority. Implementation used SimpleGrantedAuthority given by Spring F/W.

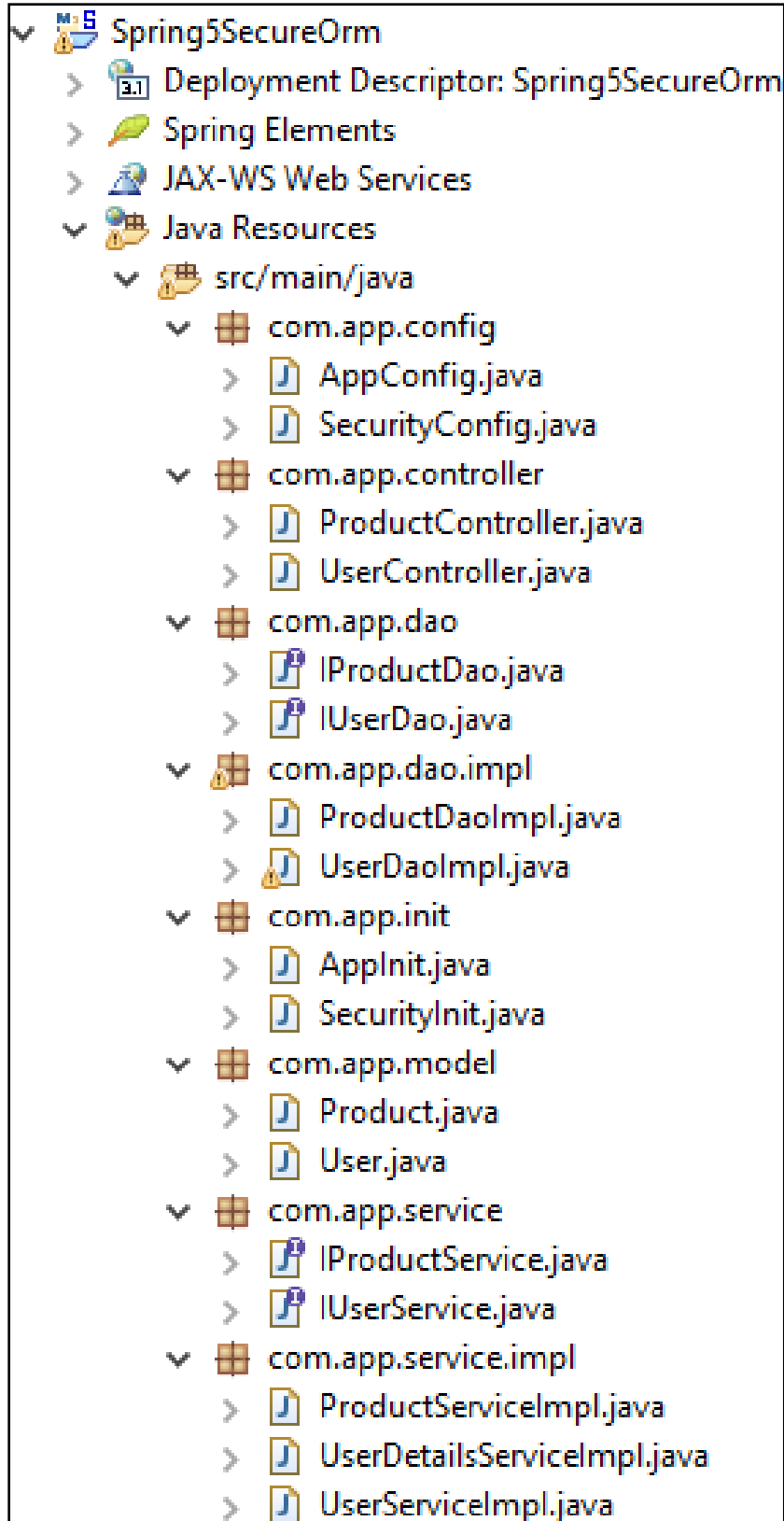
=>WebSecurityConfigurerAdaptor checks given input data and Spring F/W user data, if matched then goto Success URL (Home Page) else goto Login Page with Error message.

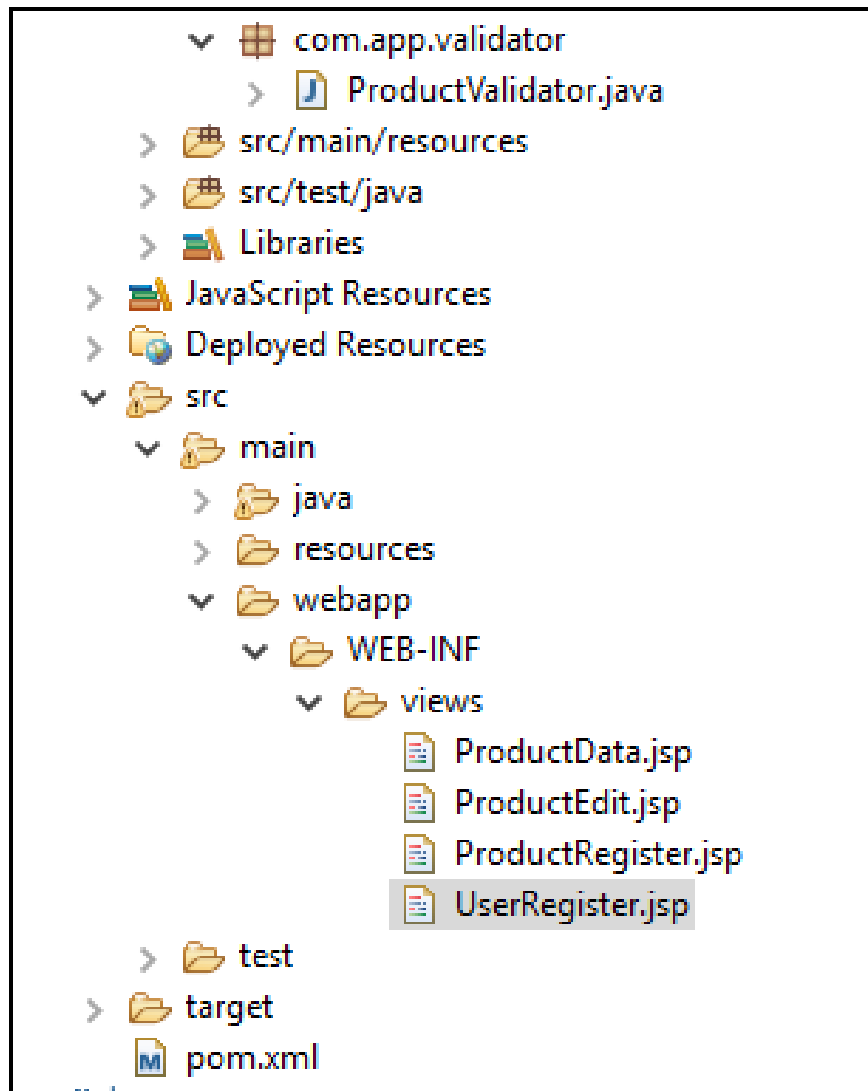


=>Here, we should write one impl class for Interface UserDetailsService having one abstract method loadUserByUsername (String username) and return Spring F/W User class object it uses our IService (SL) to fetch data from DB.



Folder Structure:--





Code:--

1>AppInit:--

```
package com.app.init;

import org.springframework.web.servlet.support.
AbstractAnnotationConfigDispatcherServletInitializer;
import com.app.config.AppConfig;

public class AppInit extends AbstractAnnotationConfigDispatcherServletInitializer
{
    @Override
    protected Class<?>[] getRootConfigClasses() {
        return new Class[] {AppConfig.class};
    }
}
```

```
    }  
    @Override  
    protected Class<?>[] getServletConfigClasses() {  
        return null;  
    }  
    @Override  
    protected String[] getServletMappings() {  
        return new String[] {"/"};  
    }  
}
```

2>AppConfig:--

```
package com.app.config;  
import java.util.Properties;  
import org.apache.commons.dbcp2.BasicDataSource;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.ComponentScan;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.context.annotation.Import;  
import org.springframework.context.annotation.PropertySource;  
import org.springframework.core.env.Environment;  
import org.springframework.orm.hibernate5.HibernateTemplate;  
import org.springframework.orm.hibernate5.HibernateTransactionManager;  
import org.springframework.orm.hibernate5.LocalSessionFactoryBean;  
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;  
import  
org.springframework.transaction.annotation.EnableTransactionManagement;  
import org.springframework.web.servlet.config.annotation.EnableWebMvc;  
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;  
import org.springframework.web.servlet.view.InternalResourceViewResolver;  
import com.app.model.Product;  
import com.app.model.User;
```

```
@Configuration
@EnableWebMvc //MVC
@EnableTransactionManagement//Tx
@PropertySource("classpath:app.properties")
@ComponentScan(basePackages="com.app")
@Import(SecurityConfig.class)
public class AppConfig implements WebMvcConfigurer
{
    //load properties
    @Autowired
    private Environment env;

    @Bean
    public BCryptPasswordEncoder encoder()
    {
        return new BCryptPasswordEncoder();
    }
    //DataSource
    @Bean
    public BasicDataSource dsObj()
    {
        BasicDataSource ds=new BasicDataSource();
        ds.setDriverClassName(env.getProperty("dc"));
        ds.setUrl(env.getProperty("url"));
        ds.setUsername(env.getProperty("un"));
        ds.setPassword(env.getProperty("pwd"));
        ds.setInitialSize(1);
        ds.setMaxIdle(10);
        ds.setMinIdle(5);
        ds.setMaxTotal(10);
        return ds;
    }
    //SessionFactory
```

```
@Bean
public LocalSessionFactoryBean sfObj()
{
    LocalSessionFactoryBean sf=new LocalSessionFactoryBean();
    sf.setDataSource(dsObj());
    sf.setAnnotatedClasses(Product.class,User.class);
    sf.setHibernateProperties(props());
    return sf;
}
private Properties props()
{
    Properties p=new Properties();
    p.put("hibernate.dialect", env.getProperty("dialect"));
    p.put("hibernate.show_sql", env.getProperty("showsql"));
    p.put("hibernate.format_sql", env.getProperty("fmtsql"));
    p.put("hibernate.hbm2ddl.auto", env.getProperty("ddlauto"));
    return p;
}
//HT
@Bean
public HibernateTemplate htObj()
{
    HibernateTemplate ht=new HibernateTemplate();
    ht.setSessionFactory(sfObj().getObject());
    return ht;
}
//Tx manager
@Bean
public HibernateTransactionManager htx() {
    HibernateTransactionManager htm=new
HibernateTransactionManager();
    htm.setSessionFactory(sfObj().getObject());
    return htm;
}
```



```
    }  
    //view resolver  
    @Bean  
    public InternalResourceViewResolver ivr() {  
        InternalResourceViewResolver v=new  
InternalResourceViewResolver();  
        v.setPrefix("/WEB-INF/views/");  
        v.setSuffix(".jsp");  
        return v;  
    }  
}
```

3>Product class:--

```
package com.app.model;  
import java.lang.Integer;  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.GeneratedValue;  
import javax.persistence.Id;  
import javax.persistence.Table;
```

```
@Entity  
@Table(name = "producttab")  
public final class Product  
{  
    @Id  
    @GeneratedValue  
    @Column(name = "id")  
    private Integer id;  
    @Column(name="pname")  
    private String productName;  
    @Column(name="pcost")  
    private double productCost;
```

```
@Column(name="pdtls")
private String productDetails;

public Product() {
    super();
}

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

public Product(Integer id, String productName, double productCost,
                String productDetails)
{
    super();
    this.id = id;
    this.productName = productName;
    this.productCost = productCost;
    this.productDetails = productDetails;
}

public Integer getId() {
    return id;
}

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

public String getProductName() {
    return productName;
}
```

```
public void setProductName(String productName) {
    this.productName = productName;
}

public double getProductCost() {
    return productCost;
}

public void setProductCost(double productCost) {
    this.productCost = productCost;
}

public String getProductDetails() {
    return productDetails;
}

public void setProductDetails(String productDetails) {
    this.productDetails = productDetails;
}

@Override
public String toString()
{
    return "Product [id=" + id + ", productName=" + productName + ",
productCost=" + productCost + ", productDetails=" + productDetails + "];"
}
}
```

4>IProductDao:--

```
package com.app.dao;
import com.app.model.Product;
import java.lang.Integer;
import java.util.List;
```

```
public interface IProductDao
{
    Integer saveProduct(Product product);
    void updateProduct(Product product);
    void deleteProduct(Integer id);
    Product getOneProduct(Integer id);
    List<Product> getAllProducts();
}
```

5>ProductDaoImpl:--

```
package com.app.dao.impl;
import com.app.dao.IProductDao;
import com.app.model.Product;
import java.lang.Integer;
import java.lang.Override;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.orm.hibernate5.HibernateTemplate;
import org.springframework.stereotype.Repository;
```

@Repository

```
public class ProductDaoImpl implements IProductDao
```

```
{
```

```
    @Autowired
```

```
    private HibernateTemplate ht;
```

```
    @Override
```

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

```
        return (Integer)ht.save(product);
```

```
}
```

```
    @Override
```

```
    public void updateProduct(Product product) {
```

```
ht.update(product);  
}
```

```
@Override  
public void deleteProduct(Integer id) {  
    ht.delete(new Product(id));  
}
```

```
@Override  
public Product getOneProduct(Integer id) {  
    return ht.get(Product.class,id);  
}
```

```
@Override  
public List<Product> getAllProducts() {  
    return ht.loadAll(Product.class);  
}  
}
```

6>IProductService:--

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

```
public interface IProductService  
{  
    Integer saveProduct(Product product);  
    void updateProduct(Product product);  
    void deleteProduct(Integer id);  
    Product getOneProduct(Integer id);  
    List<Product> getAllProducts();  
}
```

7>ProductValidator:--

```
package com.app.validator;
import com.app.model.Product;
import java.lang.Class;
import java.lang.Object;
import java.lang.Override;
import org.springframework.stereotype.Component;
import org.springframework.validation.Errors;
import org.springframework.validation.Validator;
```

@Component

```
public class ProductValidator implements Validator
{
    @Override
    public boolean supports(Class<?> clazz)
    {
        return Product.class.equals(clazz);
    }
    @Override
    public void validate(Object target, Errors errors) {
    } }
}
```

8>ProductController:--

```
package com.app.controller;
import static org.springframework.web.bind.annotation.RequestMethod.POST;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.validation.Errors;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
```

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

import com.app.model.Product;
import com.app.service.IProductService;
import com.app.validator.ProductValidator;

@Controller
@RequestMapping("/product")
public class ProductController
{
    @Autowired
    private IProductService service;

    @Autowired
    private ProductValidator validator;

    @RequestMapping("/register")
    public String regProduct(ModelMap map)
    {
        map.addAttribute("product",new Product());
        return "ProductRegister";
    }

    @RequestMapping(value = "/save", method = POST)
    public String saveProduct(@ModelAttribute Product product, Errors errors,
    ModelMap map) {
        validator.validate(product, errors);
        if(!errors.hasErrors())
        {
            Integer id=service.saveProduct(product) ;
            map.addAttribute("message","Product created with Id:"+id);
            map.addAttribute("product",new Product()) ;
        }
    }
}
```

```
return "ProductRegister";  
}
```

```
@RequestMapping( value = "/update", method = POST)  
public String updateProduct(@ModelAttribute Product product, ModelMap  
map)  
{  
    service.updateProduct(product);  
    map.addAttribute("message","Product updated");  
    List<Product> prods=service.getAllProducts();  
    map.addAttribute("products", prods);  
    return "ProductData";  
}
```

```
@RequestMapping("/delete")  
public String deleteProduct(@RequestParam Integer id,ModelMap map)  
{  
    service.deleteProduct(id);  
    List<Product> prods=service.getAllProducts();  
    map.addAttribute("products", prods);  
    return "ProductData";  
}
```

```
@RequestMapping("/edit")  
public String editProduct(@RequestParam Integer id, ModelMap map)  
{  
    Product ob=service.getOneProduct(id);  
    map.addAttribute("product",ob);  
    return "ProductEdit";  
}
```

```
@RequestMapping("/getAll")
```



```
public String getAllProducts(ModelMap map)
{
    List<Product> prods=service.getAllProducts();
    map.addAttribute("products", prods);
    return "ProductData";
}
}
```

Spring Security code:--

1>AppInit:--

```
package com.app.init;
import org.springframework.security.web.context.
AbstractSecurityWebApplicationInitializer;

public class SecurityInit extends AbstractSecurityWebApplicationInitializer
{
}
}
```

JSP Pages:--

1>ProductRegister:--

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html> <head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head> <body>
<h2>Register Product Here</h2>
<form:form action="save" method="POST" modelAttribute="product">
<pre>
NAME : <form:input path="productName"/>
```

```
COST : <form:input path="productCost"/>
DETAILS : <form:textarea path="productDetails"/>
<input type="submit" value="Create"/>
</pre>
</form:form>
${message}
</body> </html>
```

2>ProductData:--

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html> <head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head> <body>
<h2>Welcome to Product Data</h2>
<table border="1">
<tr>
    <th>ID</th>
    <th>NAME</th>
    <th>COST</th>
    <th>NOTE</th>
    <th colspan="2">OPERATIONS</th>

</tr>
<c:forEach items="${products}" var="p">
<tr>
    <td><c:out value="${p.id}"/></td>
    <td><c:out value="${p.productName}"/></td>
    <td><c:out value="${p.productCost}"/></td>
```

```

        <td><c:out value="${p.productDetails}"/></td>
        <td>
            <a href="delete?id=${p.id}">DELETE</a>
        </td>
        <td>
            <a href="edit?id=${p.id}">EDIT</a>
        </td>
    </tr>
</c:forEach>
</table>
${message}</body></html>

```

3>ProductEdit:--

```

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html> <head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head> <body>
<h2>Edit Product Here</h2>
<form:form action="update" method="POST" modelAttribute="product">
<pre>
    ID : <form:input path="id" readonly="true"/>
    NAME : <form:input path="productName"/>
    COST : <form:input path="productCost"/>
    DETAILS : <form:textarea path="productDetails"/>
<input type="submit" value="Modify"/>
</pre>
</form:form>
</body> </html>

```

1>UserRegister:--

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html> <head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head> <body>
<h2>User Register Page</h2>
<form:form action="save" method="POST" modelAttribute="user">
<pre>
NAME    : <form:input path="userName"/>
EMAIL    : <form:input path="userEmail"/>
PASSWORD : <form:password path="userPwd"/>
ROLES    :
    <form:checkbox path="roles" value="ADMIN"/> ADMIN
    <form:checkbox path="roles" value="EMPLOYEE"/> EMPLOYEE
    <input type="submit" value="Create User"/>
</pre>
</form:form>
${message}
</body> </html>
```

2>SecurityConfig:--

```
package com.app.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.authentication.builders.
AuthenticationManagerBuilder;
```

```
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.
EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.
WebSecurityConfigurerAdapter;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.web.util.matcher.AntPathRequestMatcher;
```

@Configuration

@EnableWebSecurity

```
public class SecurityConfig extends WebSecurityConfigurerAdapter
{
```

```
    @Autowired
```

```
    private UserDetailsService service;
```

```
    @Autowired
```

```
    private BCryptPasswordEncoder encoder;
```

```
    @Override
```

```
    protected void configure(AuthenticationManagerBuilder auth) throws
Exception {
        auth.userDetailsService(service).passwordEncoder(encoder);
    }
```

```
    @Override
```

```
    protected void configure(HttpSecurity http) throws Exception
{
        http.authorizeRequests()
            .antMatchers("/product/register","/product/getAll")
            .hasAnyAuthority("ADMIN","EMPLOYEE")
            .antMatchers("/product/delete","/product/update","/product/edit")
            .hasAuthority("ADMIN")
    }
```

```
.antMatchers("/user/*").permitAll()
.anyRequest().authenticated()

.and()
.formLogin()
.defaultSuccessUrl("/")

.and()
.logout()
.logoutRequestMatcher(new AntPathRequestMatcher("logout"))

.and()
.exceptionHandling()
.accessDeniedPage("/denied");
}
}
```

3>User class:--

```
package com.app.model;
import java.util.Set;

import javax.persistence.CollectionTable;
import javax.persistence.Column;
import javax.persistence.ElementCollection;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.Table;

import org.hibernate.annotations.GenericGenerator;

@Entity
```

```
@Table(name="usrtab")
public class User
{
    @Id
    @Column(name="uid")
    @GeneratedValue(generator="ugen")
    @GenericGenerator(name="ugen",strategy="increment")
    private int userId;

    @Column(name="uname")
    private String userName;
    @Column(name="uemail")
    private String userEmail;
    @Column(name="upwd")
    private String userPwd;

    @ElementCollection
    @CollectionTable(name="usr_roles_tab",
    joinColumns=@JoinColumn(name="uid"))
    @Column(name="roles")
    private Set<String> roles;

    public User() {
        super();
    }
    public User(int userId) {
        super();
        this.userId = userId;
    }
    public int getUserId() {
        return userId;
    }
    public void setUserId(int userId) {
```

```
        this.userId = userId;
    }
    public String getUsername() {
        return userName;
    }
    public void setUsername(String userName) {
        this.userName = userName;
    }
    public String getEmail() {
        return userEmail;
    }
    public void setEmail(String userEmail) {
        this.userEmail = userEmail;
    }
    public String getPwd() {
        return userPwd;
    }
    public void setPwd(String userPwd) {
        this.userPwd = userPwd;
    }
    public Set<String> getRoles() {
        return roles;
    }
    public void setRoles(Set<String> roles) {
        this.roles = roles;
    }
    @Override
    public String toString()
    {
        return "User [userId=" + userId + ", userName=" + userName + ",
userEmail=" + userEmail + ", userPwd=" + userPwd + ", roles=" + roles + "];"
    }
}
```


4>IUserDao:--

```
package com.app.dao;
import com.app.model.User;

public interface IUserDao
{
    public int saveUser(User user);
    public User getUserByEmail(String userEmail);
}
```

5>UserdaoImpl:--

```
package com.app.dao.impl;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.orm.hibernate5.HibernateTemplate;
import org.springframework.stereotype.Repository;
import com.app.dao.IUserDao;
import com.app.model.User;
```

```
@Repository
public class UserDaoImpl implements IUserDao
{
    @Autowired
    private HibernateTemplate ht;

    @Override
    public int saveUser(User user)
    {
        return (Integer)ht.save(user);
    }

    @SuppressWarnings("deprecation")
```

```
@Override
public User getUserByEmail(String userEmail)
{
    User user=null;
    String hql="from "+User.class.getName()+" where userEmail=?";
    List<User> userList=(List<User>) ht.find(hql, userEmail);
    if(userList!=null && userList.size()>0) {
        user=userList.get(0);
    }
    return user;
}
}
```

6>UserService:--

```
package com.app.service;
import com.app.model.User;

public interface IUserService
{
    public int saveUser(User user);
    public User getUserByEmail(String userEmail);
}
```

7>UserServiceImpl:--

```
package com.app.service.impl;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import com.app.dao.IUserDao;
import com.app.model.User;
import com.app.service.IUserService;
```

```
@Service
public class UserServiceImpl implements IUserService
{
    @Autowired
    private IUserDao dao;
    @Autowired
    private BCryptPasswordEncoder encoder;

    @Transactional
    public int saveUser(User user) {
        //read UI password
        String pwd=user.getUserPwd();
        //encode password
        pwd=encoder.encode(pwd);
        //set back to model object
        user.setUserPwd(pwd);
        //save user to DB
        return dao.saveUser(user);
    }

    @Transactional(readOnly=true)
    public User getUserByEmail(String userEmail) {
        return dao.getUserByEmail(userEmail);
    }
}
```

8>UserDetailsServiceImpl:--

```
package com.app.service.impl;
import java.util.HashSet;
import java.util.Set;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
```

```
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import
org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import com.app.model.User;
import com.app.service.IUserService;
@Service
public class UserDetailsServiceImpl implements UserDetailsService{
    @Autowired
    private IUserService service;

    @Transactional(readOnly=true)
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException
    {
        //take login username and get DB Model class obj
        User user=service.getUserByEmail(username);

        //convert model class object roles to SGA and add to Set
        Set<GrantedAuthority> roles=new HashSet<>();
        for(String role:user.getRoles()) {
            roles.add(new SimpleGrantedAuthority(role));
        }

        //convert to Spring f/w user object
        return new org.springframework.security.core.userdetails
            .User(
                user.getUserEmail(),
                user.getUserPwd(),
                roles);
    }
}
```

```
}
```

9>UserController:--

```
package com.app.controller;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import com.app.model.User;
import com.app.service.IUserService;
@Controller
@RequestMapping("/user")
public class UserController
{
    @Autowired
    private IUserService service;

    @RequestMapping("/register")
    public String showReg(ModelMap map) {
        map.addAttribute("user",new User());
        return "UserRegister";
    }
    @RequestMapping(value="/save",method=RequestMethod.POST)
    public String saveUser(@ModelAttribute User user,ModelMap map)
    {
        int userId=service.saveUser(user);
        String msg="User '"+userId+"' create successfully ";
        map.addAttribute("message",msg);
        return "UserRegister";
    }
}
```