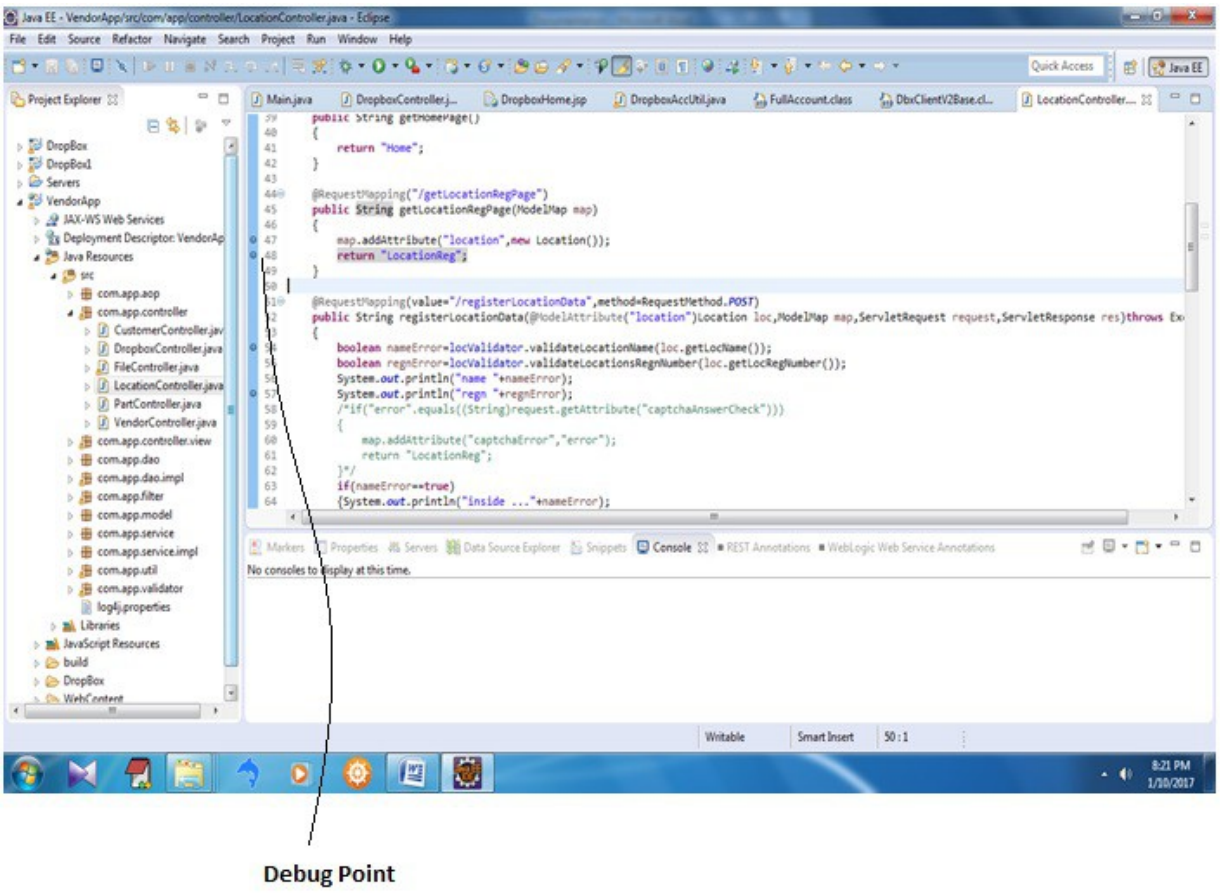
Vendor Data Management System (VDM)

Debug:

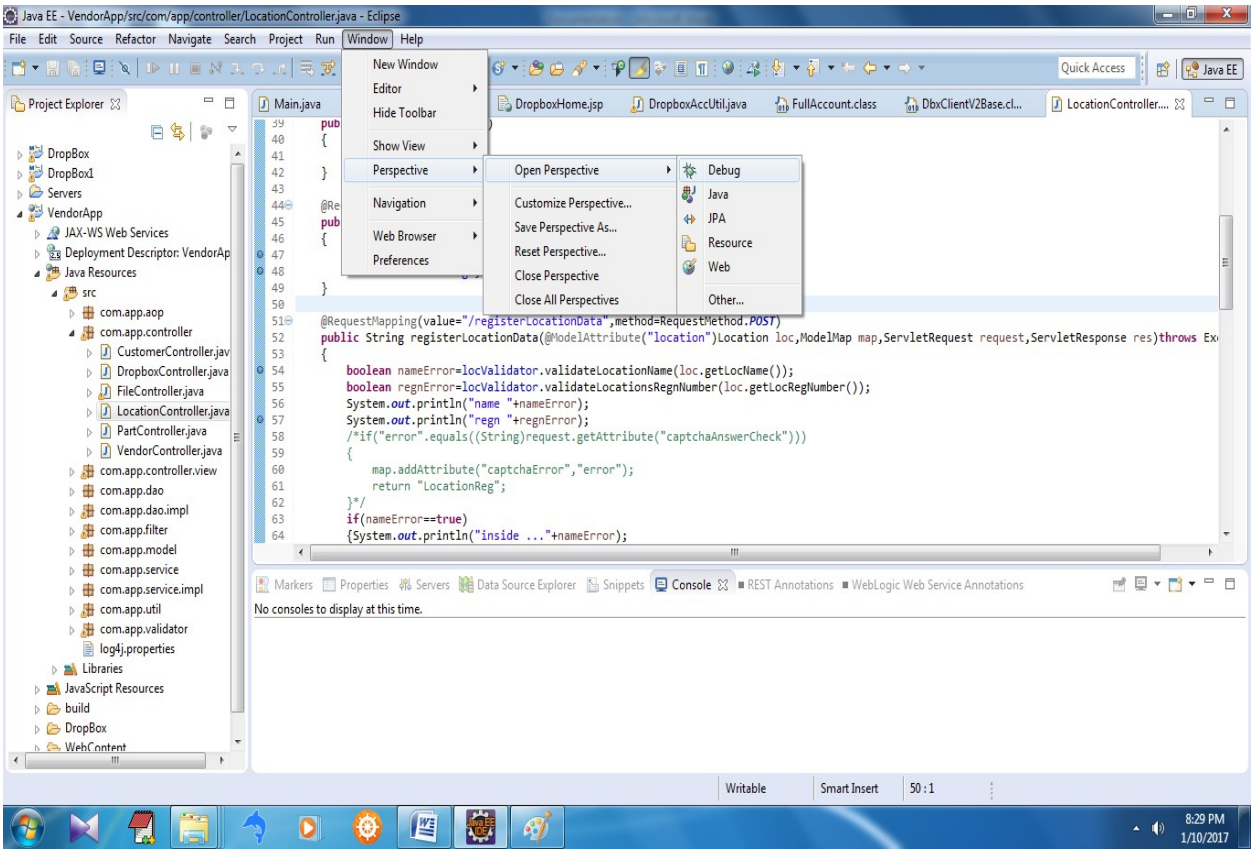
- It is used to test application in step by step mode
- We can observe variable values,expressions,exception traces,object data execution process…

Procedure

1. Double click on blue color(number bar) bar to create or remove break point.

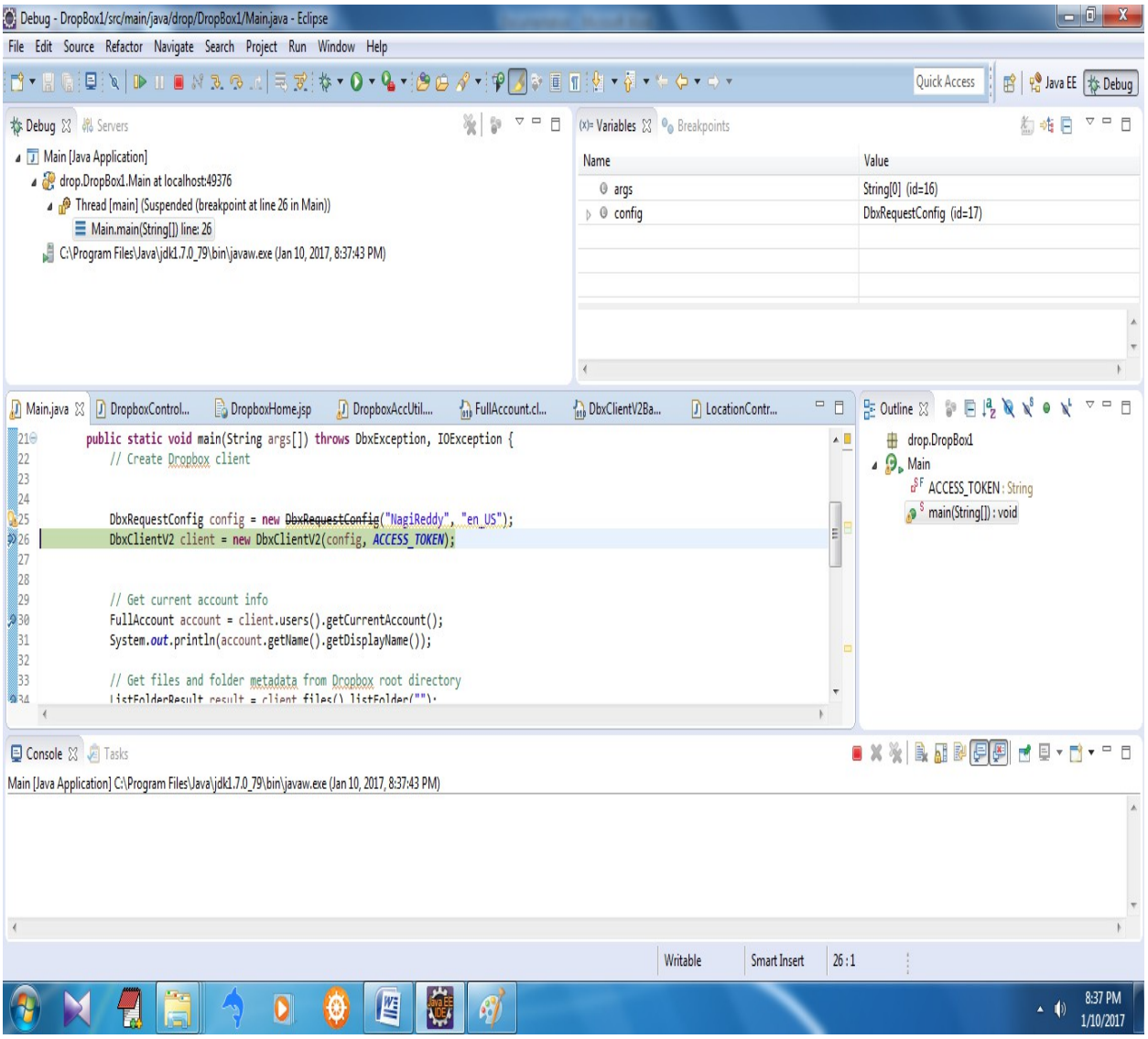


2. Now goto, Window->Perspective->Open Perspective-> Debug.



3. Now you have selected Debug mode,otherwise do **CTRL+F11** to execute your program in debugging mode.

4. Press **F6** to execute next line,**F5** to step into method call,**F7** to come out of method call,**F8** to Resume or to stop.



Relations:

- We can provide relation between two tables in database and between 2 classes in Java also.
- In Java,relation is 2 types.

1. HAS-A

2. IS-A

- IS-A relation means **Inheritance**. i.e extending a class from another class or implementing a class from an interface.

Ex:

```
class A{

}

class B extends A{

}
```

- IS-A relation between 2 classes will give 'Tight coupling' in the application and we can't inherit more than one class behaviours or features.
- HAS-A relation means **Composition**.

Ex:

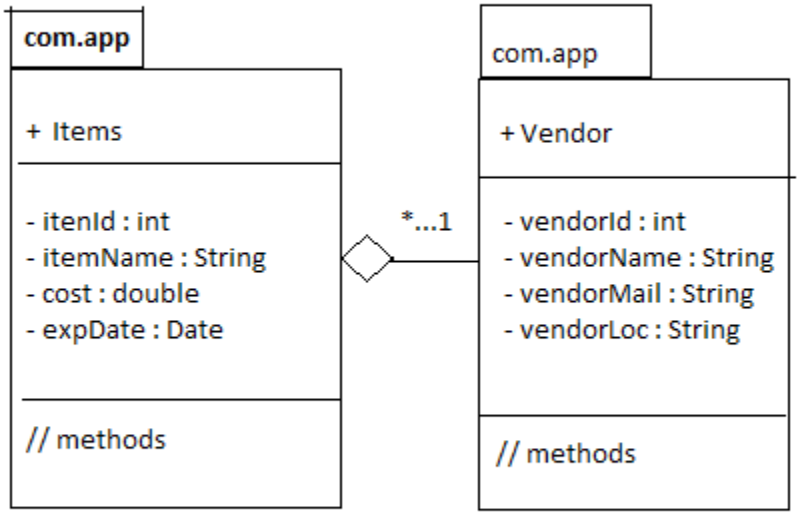
```
Class A{

}

Class B{

    private A a;

}
```

✓ Here, class B HAS-A A.

✓ HAS-A means creating an object of another class from which you want services in the class where you want to use the services.

- By using HAS-A relation, we can get 'Loose coupling' between applications and also a class can get services more than one class.
- The mostly used relation between 2 classes in a project is HAS-A.

UML:



Code: **Items.java**

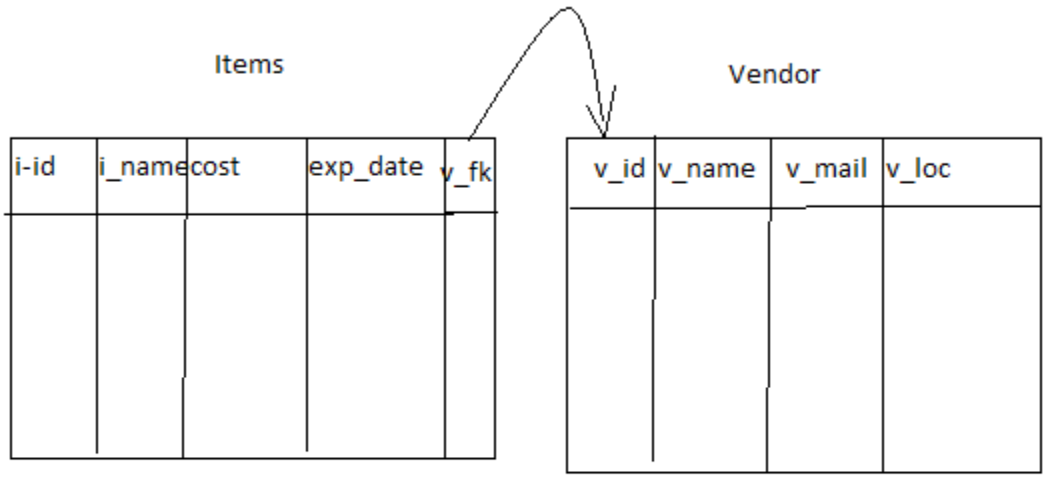
```
package com.app;

public class Items{
```

```
    private int itemId;
    private String itemName;
    private double cost;
    private Date expDate;
}
```

Vendor.java

```
package com.app;
public class Vendor{
    private int vendorId;
    private String vendorName;
    private String vendorMail;
    private String vendorLoc;
}
```

Database:



Types

1.Collection (1….* *….*)

2.Non collection (1….1 *….1)

- For every HAS-A relation, we go for the **POJI** and **POJO** approach.
- To define 2 classes,define child class as member in parent class.
- To design tables,always provide foreign key column at many(*) side.
- In case of 1…..1,use (unique)*….1

Design of App :

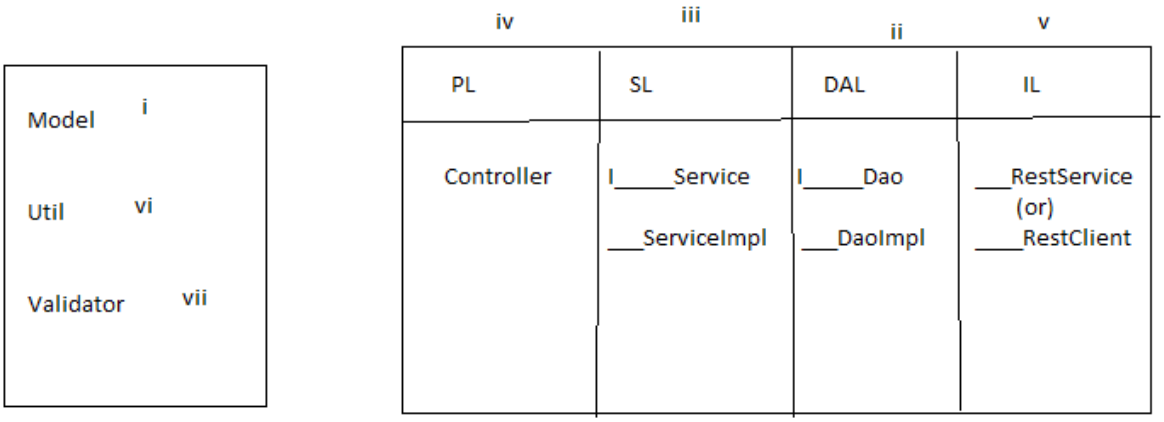
- For every module in a project, we have to create the below classes.

Example for User module:

1. User.java (Model)
2. IUserDao.java (POJI)
3. UserDaoImpl.java (POJO)
4. IUserService.java (POJI)
5. UserServiceImpl .java (POJO)
6. UserController.java
7. UserValidator.java
8. UserUtil.java
9. UserRestService.java (or) UserRestClient.java

- To develop any module which is in all layers,classes needed are

1. Model
2. DAO (POJI,POJO)
3. Service (POJI,POJO)
4. Controller class
5. RestService or RestClient class
6. Util and Validator classes

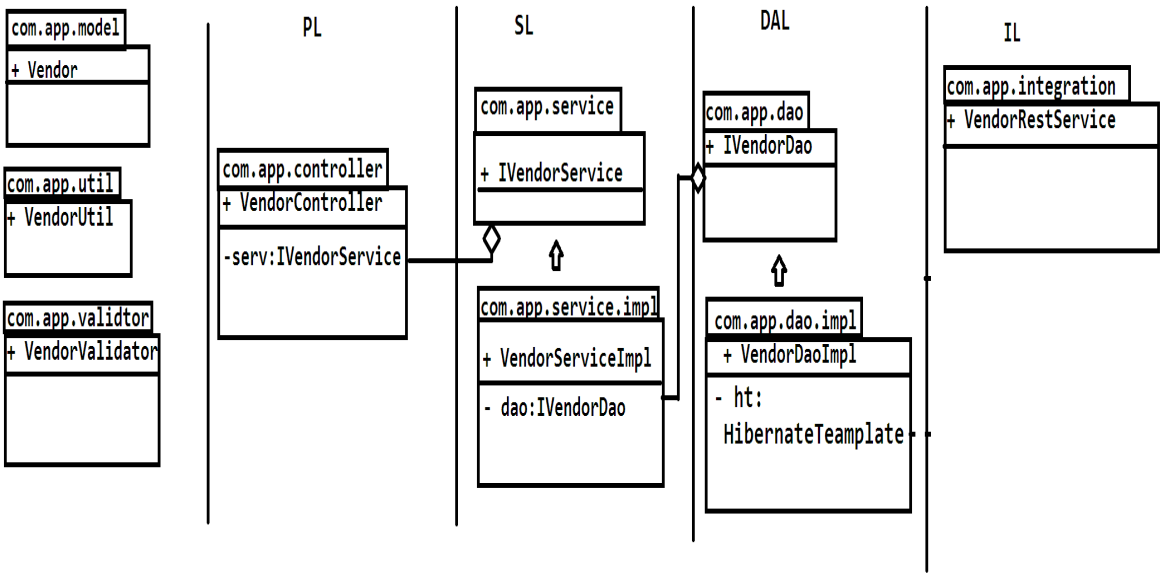


Ex module for Vendor:

1. Vendor.java (Model)
2. IVendorDao.java (POJI)
3. VendorDaoImpl.java (POJO)
4. IVendorService.java (POJI)
5. VendorServiceImpl.java (POJO)
6. VendorController.java
7. VendorRestService.java (or) VendorRestClient.java
8. VendorUtil.java
9. VendorValidator.java

Designing Module :

- Every module contains its supportive classes in all layers. They will be communicated with each other using relations (IS-A,HAS-A)



DAO

- To define all operations in DaoImpl , it needs HibernateTemplate.
- If we use template class object, then we can write logic with less code.

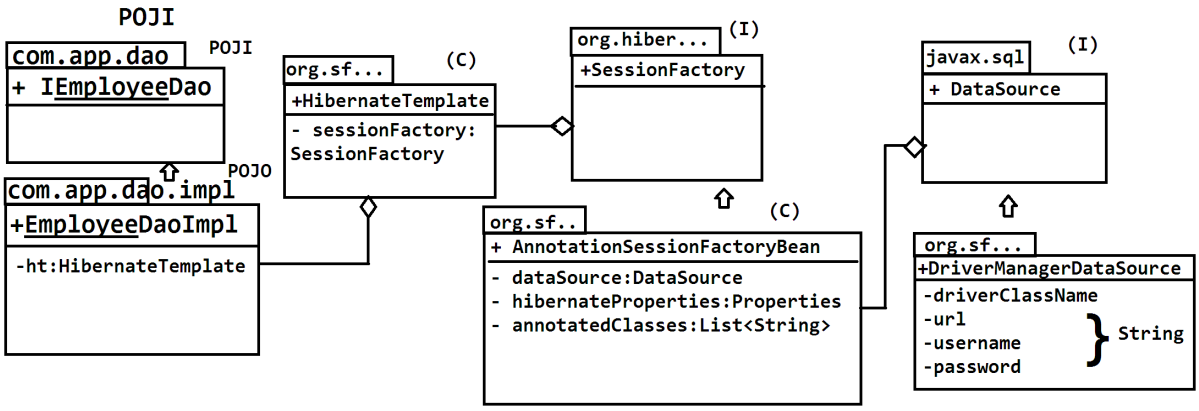
Example:

```
template.save(obj);
```

template.update(obj);

- This template takes care of writing session and transaction management in Hibernate.
- Every DaoImpl class must have HibernateTemplate dependency. HibernateTemplate is pre-defined and must be configured only one time.

DAO Design:



UI Design :

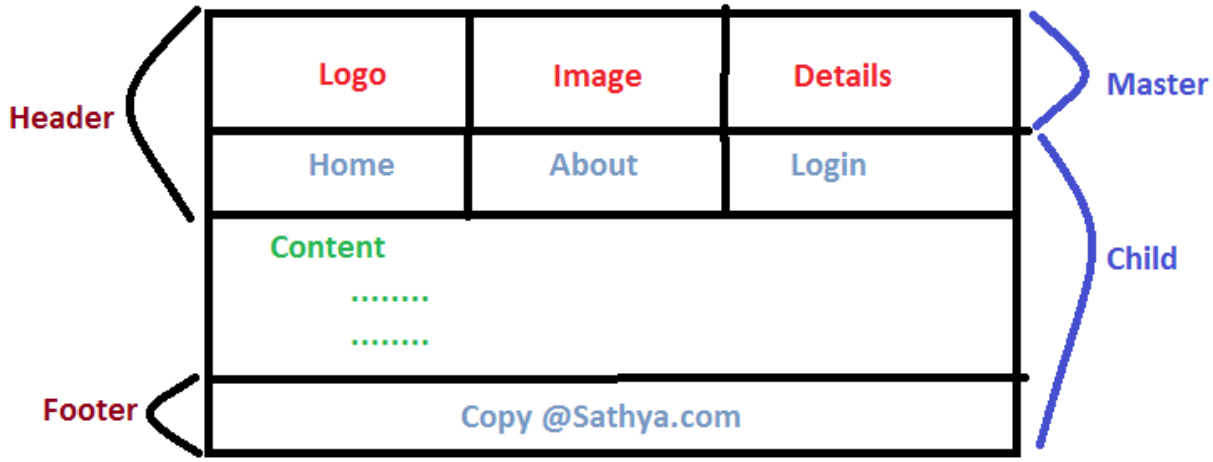
- **UI** (view page) will be constructed using HTML+JSP+JSTL.
- We can also use CSS and Java Script for design and client side validation.
- Every UI page is divided into 2 parts. i.e. Master + Child.

- **Master** can be Header,Footer,Slider(common data).
- **Child** is used to represent page content.

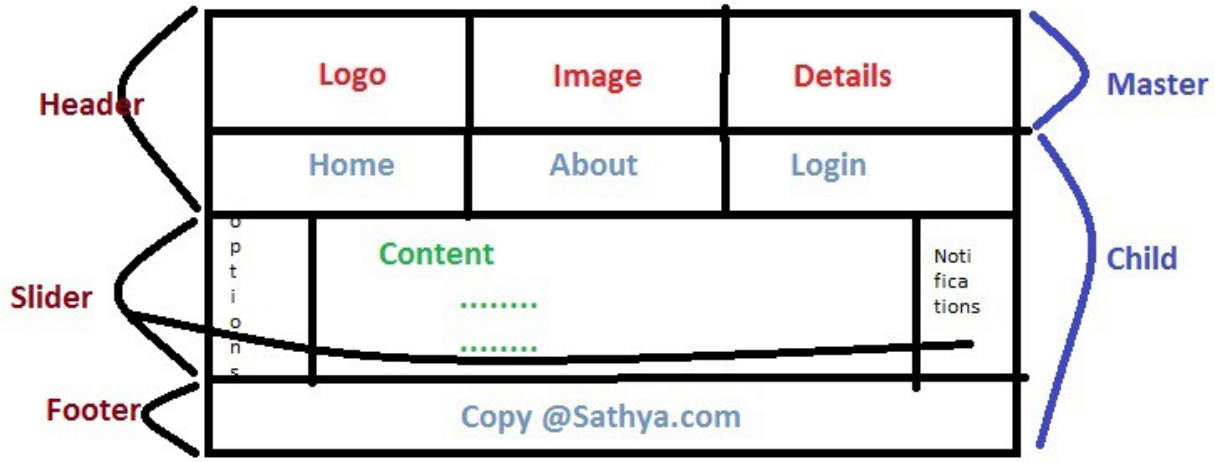
Design :

Ex 1:

Ex 2:


Logo
Image
Details
Master
Header
Home
About
Login
Content
.....
.....
Child
Footer
Copy @Sathya.com

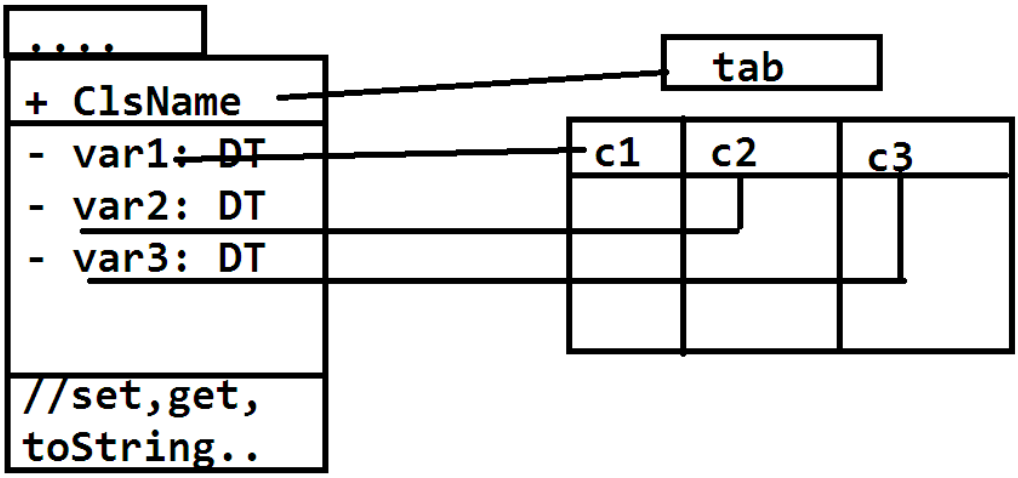
Ex 3:



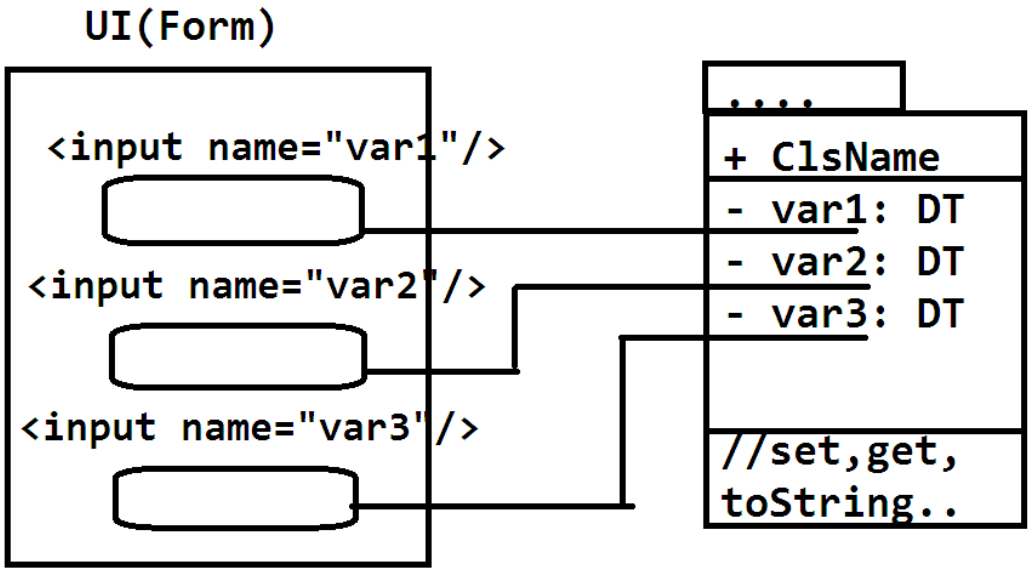
Module Implementation :

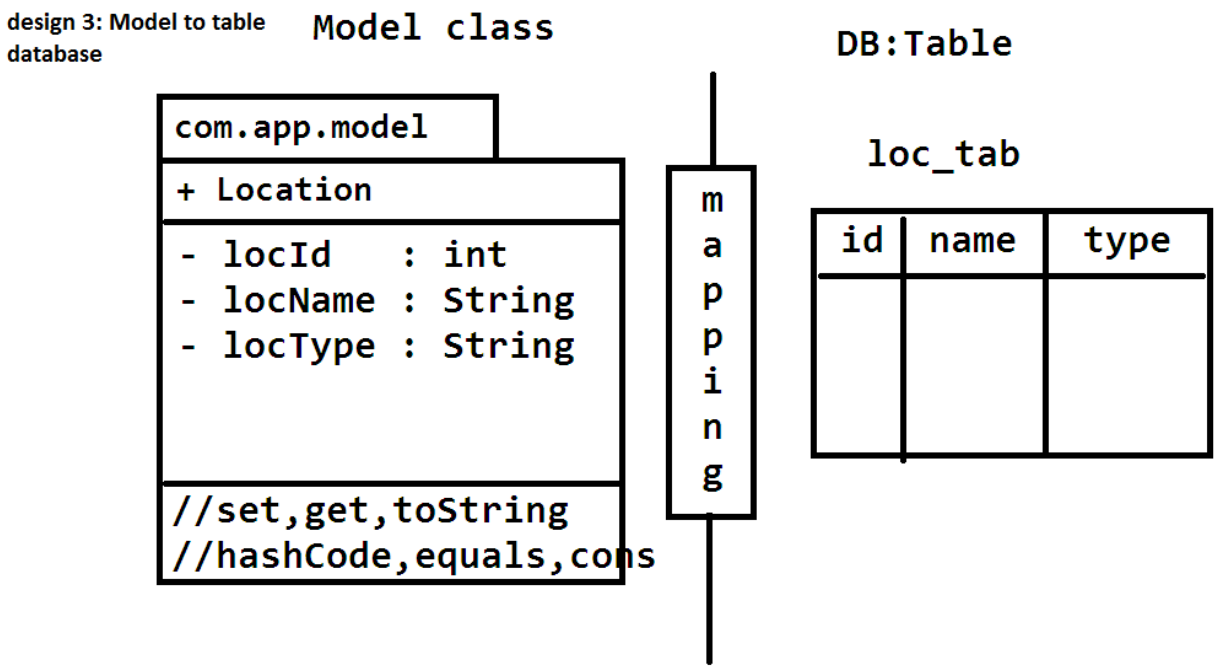
- For every Module,start coding from Model class and its DB table and UI mapping.
- Use **@Entity** , **@Table** , **@Id** and **@Column** to map a class with table.
- Use <input type=" " name="variable-name"/> to map UI input to Model class variable.

design 1: Model to Table Mapping



design 2: UI to model mapping





Code :

```
package  com.app.model;
@Entity
@Table(name=”loc_tab”)
public class Vendor
{
    @Id
    @Column(name=”id”)
    private int locId;
```

```
    @Column(name="name")
    private String locName;
    @Column(name="type")
    private String locType;

    //setters,getters
    //toString
    //hashCode,equals
    //constructor
}
```

Form :

```
<form action=" " method="post">
    <input type=" " name="locId"/>
    <input type=" " name="locName"/>
    <input type=" " name="locType"/>
    ……
    ……
</form>
```

DAL Coding :

✓ After designing Model class, we have to define IDao and DaoImpl coding using below steps.

Step 1 : Create an interface for IDao as I_____Dao

Step 2 : Add abstract methods for save,update,delete,getOne and getMultiple.

Step 1 : Write one impl class for above IDAO as _____DaoImpl implements I_____Dao

Step 3 : Add HibernateTemplate ht dependency in DaoImpl.

Step 5 : Override all IDAO methods in DaoImpl and use ht to define code as

ht.save(obj);

ht.update(obj);

ht.delete(obj);

ht.get(T.class,id) : T

ht.loadAll(T.class) : List<T>

Syntax :

```
package com.app.dao;

public interface I_____Dao

{

   //add abstract methods

}
```

```
package com.app.dao.impl;
public class ___DaoImpl implements I_____Dao
{
    HibernateTemplate ht;
    //override all methods & use ht inside method.
}
```

Example : Address Module

```
package com.app.dao;
public interface IAddressDao
{
    public void saveAddress(Address obj);
    public void updateAddress(Address obj);
    public void daleteAddress(int addrId);
    public Address getAddessById(int addrId);
    public List<Address> getAllAddresses();
}
```

```
package com.app.dao.impl;
public class AddressDaoImpl implements IAddressDao
```

```
{
    private HibernateTemplate ht;
    public void saveAddress(Address obj)
    {
        ht.save(obj);
    }

    public void updateAddress(Address obj)
    {
        ht.update(obj);
    }
    public void daleteAddress(int addrId)
    {
        ht.delete(new Address(addrId));
    }
    public Address getAddressById(int addrId)
    {
        return ht.get(Address.class,addrId);
    }
    public List<Address> getAllAddresses()
```

```
	{
		return ht.loadAll(Address.class);
	}
}
```

Example : Customer Module

```
package com.app.dao;
public interface ICustomerDao
{
	public void saveCustomer(Customer obj);
	public void updateCustomer(Customer obj);
	public void daleteCustomer(int addrId);
	public Customer getAddessById(int addrId);
	public List<Customer> getAllCustomeres();
}

package com.app.dao.impl;
public class CustomerDaoImpl implements ICustomerDao
```

```
{
    private HibernateTemplate ht;
    public void saveCustomer(Customer obj)
    {
        ht.save(obj);
    }

    public void updateCustomer(Customer obj)
    {
        ht.update(obj);
    }
    public void daleteCustomer(int addrId)
    {
        ht.delete(new Customer(addrId));
    }
    public Customer getCustomerById(int addrId)
    {
        return ht.get(Customer.class,addrId);
    }
    public List<Customer> getAllCustomeres()
```

```
	{
		return ht.loadAll(Customer.class);
	}
}
```

Example : Location Module

```
package com.app.dao;
public interface ILocationDao
{
	public void saveLocation(Location obj);
	public void updateLocation(Location obj);
	public void daleteLocation(int addrId);
	public Location getAddessById(int addrId);
	public List<Location> getAllLocationes();
}

package com.app.dao.impl;
public class LocationDaoImpl implements ILocationDao
{
	private HibernateTemplate ht;
```

```
public void saveLocation(Location obj)
{
	ht.save(obj);
}

public void updateLocation(Location obj)
{
	ht.update(obj);
}
public void daleteLocation(int addrId)
{
	ht.delete(new Location(addrId));
}
public Location getLocationById(int addrId)
{
	return ht.get(Location.class,addrId);
}
public List<Location> getAllLocationes()
{
	return ht.loadAll(Location.class);
```

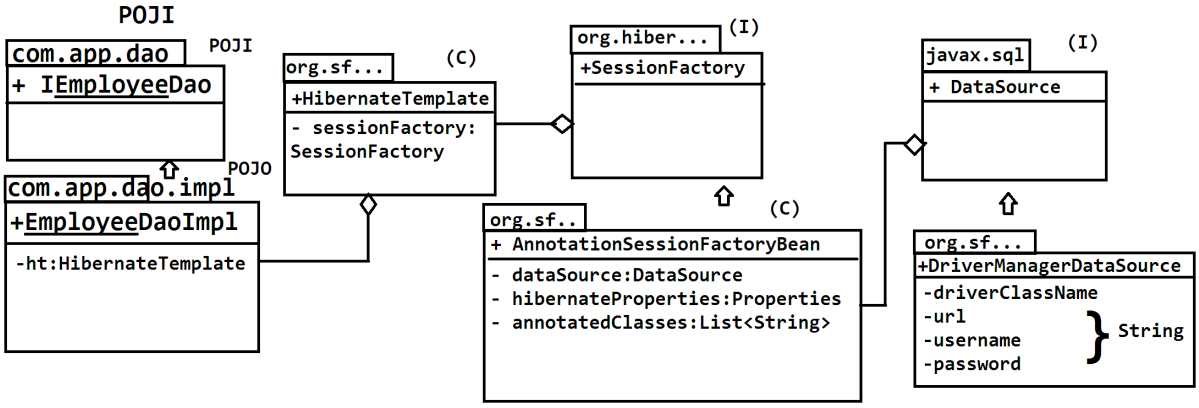
```
        }
}
```

Example : Employee Module

```
package com.app.dao;
public interface IEmployeeDao
{
        public void saveEmployee(Employee obj);
        public void updateEmployee(Employee obj);
        public void daleteEmployee(int addrId);
        public Employee getAddessById(int addrId);
        public List<Employee> getAllEmployeees();
}

package com.app.dao.impl;
public class EmployeeDaoImpl implements IEmployeeDao
{
        private HibernateTemplate ht;
        public void saveEmployee(Employee obj)
        {
                ht.save(obj);
```

```
	}


	public void updateEmployee(Employee obj)
	{
		ht.update(obj);
	}
	public void daleteEmployee(int addrId)
	{
		ht.delete(new Employee(addrId));
	}
	public Employee getEmployeeById(int addrId)
	{
		return ht.get(Employee.class,addrId);
	}
	public List<Employee> getAllEmployeees()
	{
		return ht.loadAll(Employee.class);
	}
}
```

UML



✓ First of all we have to configure the HibernateTemplate in the spring configuration file.

✓ HibernateTemplate is a class from Spring framework that depends on the SessionFactory bean.

✓ SessionFactory is an interface whose object will be produced by AnnotationSessionFactoryBean class.

✓ AnnotationSessionFactoryBean is a factory bean which produces object of another bean.This class have the following 3 dependencies.

1. dataSource : DataSource

2. hibernateProperties : Properties

3. annotatedClasses : List<String>

✓ AnnotationSessionFactoryBean depends on DataSource bean.

✓ DataSource is an interface whose object will be produced by DriverManagerDataSource class.This class is having the following 4 dependencies.

1. driverClassName : String

2. url : String

3. username : String

4. password : String

Configuration in XML

1. **DataSource object**

<bean class=
“org.springframework.jdbc.dataSource.DriverManagerDataSource”
name=”dsObj”

p:driverClassName=” “

p:url=” “

p:username=” “

p:password=” “/>

2. **SessionFactory**

<bean class=”org.springframework.orm.hibernate3.annotation.
AnnotationSessionFactoryBean” name=”sfObj”

p:dataSource-ref=”dsObj”>

<property name=”hibernateProperties”>

<props>

<prop key=”hibernate.dialect”>____</prop>

<prop key=”hibernate.show_sql”>_____</prop>

<prop key=”hibernate.hbm2ddl.auto”>__</prop>

</props>

</property>

<property name=”annotatedClasses”>

<list>

<value>_____</value>

<value>_____</value>

</list>

</property>

3.HibernateTemplate

<bean class=”org.springframework.orm.hibernate3.
HibernateTemplate” name=”htObj”

p:sessionFactory-ref=”sfObj”/>

Sequence :

✓ To generate primary key values at DB side, database supports sequence creation (number generation process).

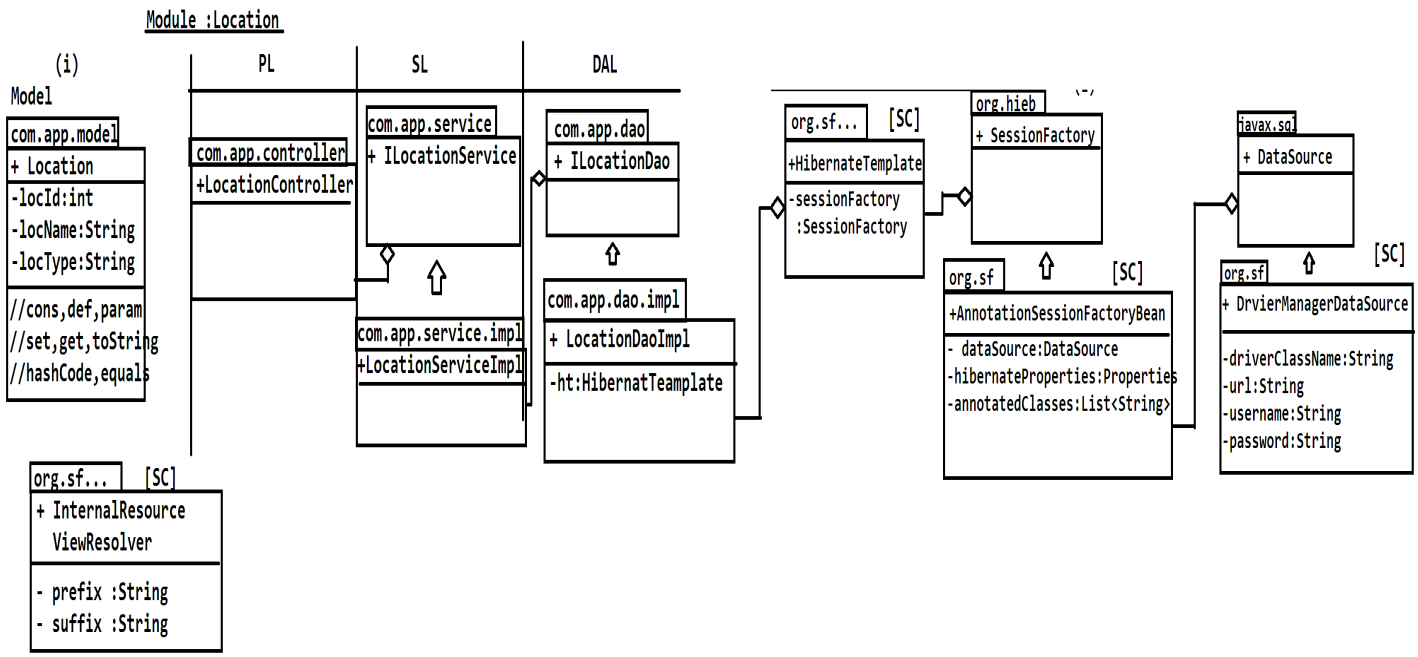
Example : sequence in oracle

```
Create sequence SAMPLE start with 100 increment by 2;
```

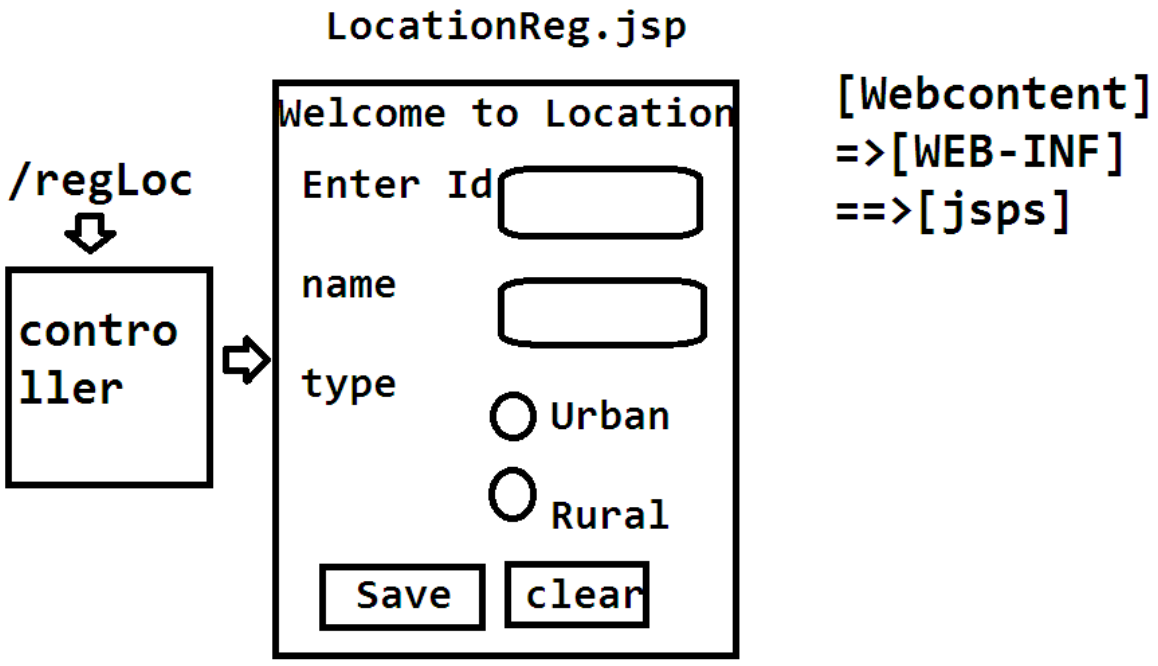
- ✓ By default, Hibernate uses **HIBERNATE_SEQUENCE** .
- ✓ Sequence generates the numbers only on **save()** operation.

Module – Location

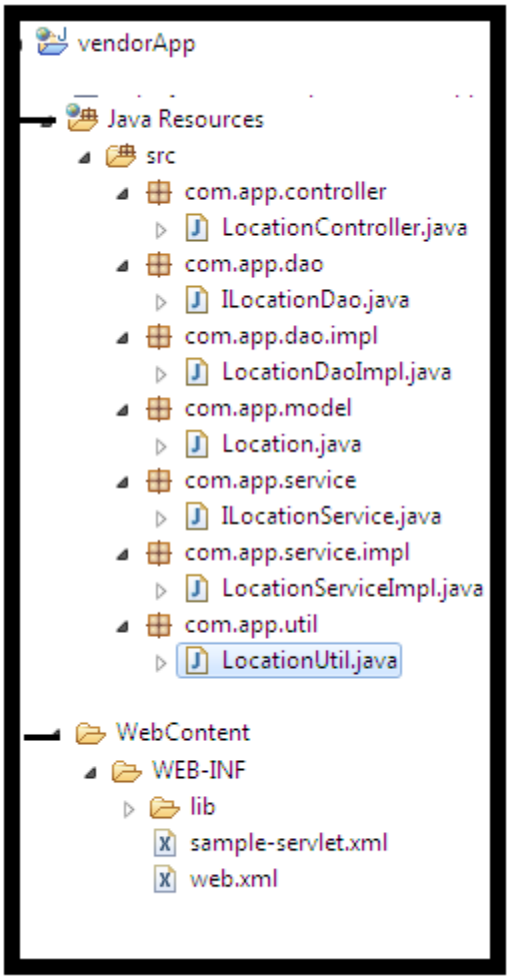
Design 1


Module :Location
(i)
Model
PL
SL
DAL
com.app.model
+ Location
-locId:int
-locName:String
-locType:String
//cons,def,param
//set,get,toString
//hashCode,equals
com.app.controller
+LocationController
com.app.service
+ ILocationService
com.app.service.impl
+LocationServiceImpl
com.app.dao
+ ILocationDao
com.app.dao.impl
+ LocationDaoImpl
-ht:HibernatTeamplate
org.sf... [SC]
+HibernateTemplate
-sessionFactory
:SessionFactory
org.hieb
+ SessionFactory
org.sf [SC]
+AnnotationSessionFactoryBean
- dataSource:DataSource
-hibernateProperties:Properties
-annotatedClasses:List<String>
javax.sql
+ DataSource
org.sf [SC]
+ DrvierManagerDataSource
-driverClassName:String
-url:String
-username:String
-password:String
org.sf... [SC]
+ InternalResource
ViewResolver
- prefix :String
- suffix :String

Design 2


LocationReg.jsp
Welcome to Location
/regLoc
contro
ller
Enter Id
name
type
Urban
Rural
Save
clear
[Webcontent]
=>[WEB-INF]
==>[jsps]

Folder Structure



Module implementation coding steps :

Step 1 Model class

- Design model class and write code. package name should be <p1>.<p2>.<model>

- Here <p1>.<p2> indicates domain name in reverse.

Example :

We are developing application for http://sathya.com then package name should be “ com.sathya.<p3> “.

Code :

```
package com.app.model;
@Entity
@Table(name=”loc_tab”)
public class Location
{
    @Id
    @Column(name=”id”)
    private int locId;
    @Column(name=”name”)
    private String locName;
    @Column(name=”type”)
    private String locType;

    //setters,getters
```

```
        //toString
        //hashCode,equals
        //constructor
}
```

Note :

All imports should be from javax.persistence package. For this , we have to add a jar ‘ hibernate-jpa-2.0-1.0.0.Final.jar’.

Step 2 : Dao layer

- In layer development code should start from DAL POJI and POJO.

Code

POJI

```
package com.app.dao;
public interface ILocationDao
{
}
```

POJO

```
package com.app.dao.impl;
@Repository
public class LocationDaoImpl implements ILocationDao
{
	@Autowired
	private HibernateTemplate ht;
}
```

Step 3 : Service layer

- Design and code Service layer.

Code

POJI

```
package com.app.service;
public interface ILocationService
{
}
```

POJO

```
package com.app.service.impl;
@Service
public class LocationServiceImpl implements ILocationService
{
	@Autowired
	private ILocationDao dao;
}
```

Step 4 : PL

- Design and code Presentation layer.

Code

```
package com.app.controller

@Controller
public class LocationController
{
	@Autowired
	private ILocationService locServ;
}
```

Operation wise Steps:

Step 1:

Define an abstract method which represents your operation. Provide input as parameter and get output as return type.

1. Add this method in I_____Dao
2. Implement this method in ___DaoImpl using ‘ ht ’.

Example

saveLocation

i/p: Location object

o/p: int (pk value)

Dao

```
public int saveLocation(Location loc)
{
    int i=(Integer)ht.save(loc);
    return i;
}
```

Step 2:

Specify above(same) abstract method in I_____Service also. Implement this in ServiceImpl using I___Dao dependency. i.e. Service method should call dao method. Define logic if it is available.

Service

```
public int saveLocation(Location loc)
{
    Dao.saveLocation(loc);
}
```

Step 3:

Design UI and define controller method based on operations like show,delete,update…..

Example **LocationReg.jsp**

```
<form action=”  ” method=”post”>
 Enter Id : <input type=”text” name=”locId”/><br>
 Enter name : <input type=”text” name=”locName”/><br>
```

Type:<input type=”radio” name=”locType” value=”urban”>Urban

<input type=”text” name=”locType” value=”rural”>Rural

<input type=”submit” value=”Save”>

<input type=”clear” value=”Clear”>

</form>

- Controller methods are

1. To display the page

```
@RequestMapping(“/login”)
public String showLocRegPage()
{
    return “LocationReg”;
}
```

2. To save data

```
@RequestMapping(value=”/check”, method= RequestMethod.POST)
public String saveLoc(@ModelAttribute(“location”) Location loc)
{
```

```
        int id=servLoc.saveLocation(loc);
        return "LocationReg";
}
```

✓ **@ModelAttribute** is used to inject the model object to the method parameter.

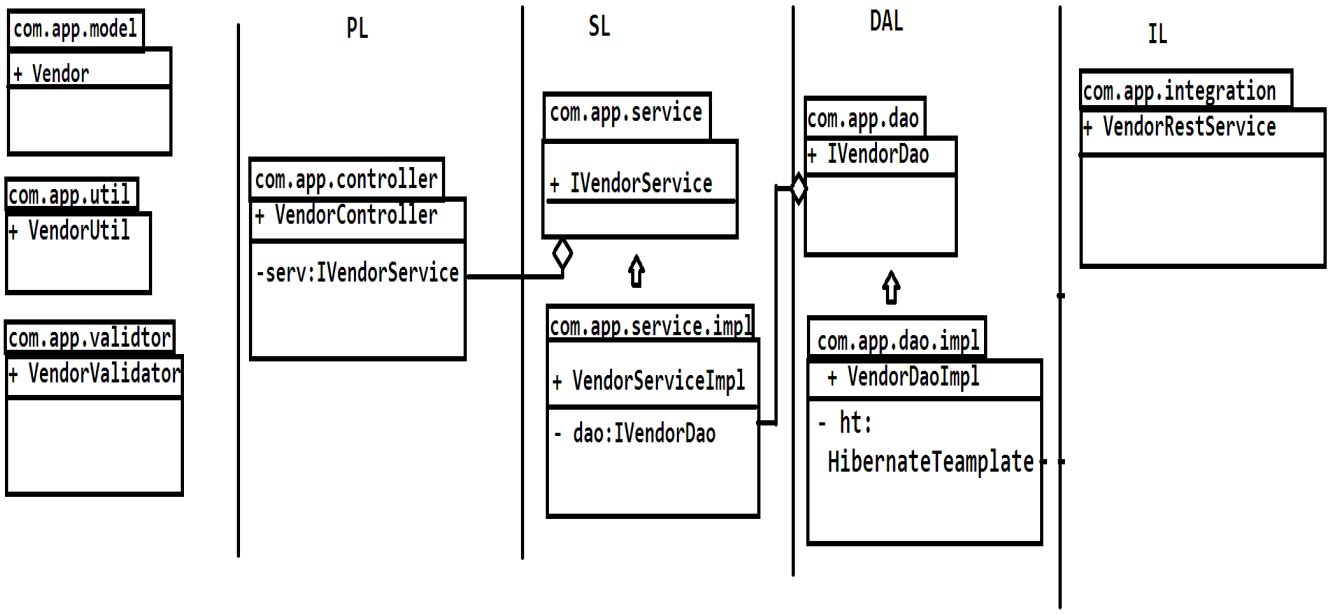
✓ ModelMap is from Spring MVC which is used to send data from Controller to UI. Add data to this object using key-value pair with the help of **addAttribute(key,value)** method.

✓ **ModelMap** is used to place models data (response) into request scope.

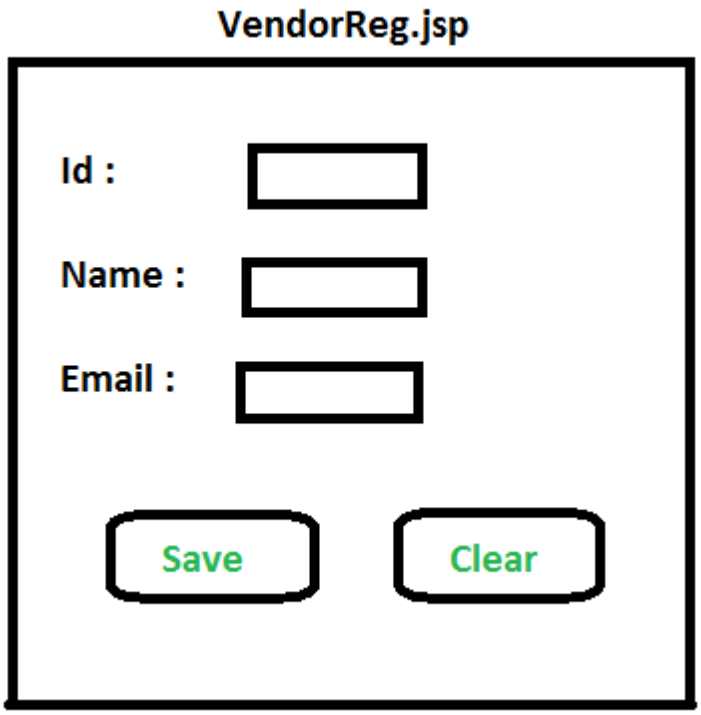
i.e. If we place response into the ModelMap object then **DispatcherServlet** will read the data and place it into the request object with a given key and value. Then from UI, we can read the value using the key.

Module – Vendor

Design 1


com.app.model
+ Vendor
com.app.util
+ VendorUtil
com.app.validtor
+ VendorValidator
PL
com.app.controller
+ VendorController
-serv:IVendorService
SL
com.app.service
+ IVendorService
com.app.service.impl
+ VendorServiceImpl
- dao:IVendorDao
DAL
com.app.dao
+ IVendorDao
com.app.dao.impl
+ VendorDaoImpl
- ht:
HibernateTeamplate
IL
com.app.integration
+ VendorRestService

Design 2


VendorReg.jsp
Id :
Name :
Email :
Save
Clear

Coding :

Model class

```
package com.app.model;
@Entity
@Table(name=”ven_tab”)
public class Vendor
{
    @Id
    @Column(name=”id”)
    private int venId;
    @Column(name=”name”)
    private String venName;
    @Column(name=”email”)
    private String email;

    //setters,getters
    //toString
    //hashCode,equals
    //constructor
```

```
}
```

DAL

POJI

```
package com.app.dao;
public interface IVendorDao
{
	public void saveVendor(Vendor ven);
}
```

POJO

```
package com.app.dao.impl;
@Repository
public class VendorDaoImpl implements IVendorDao
{
	@Autowired
	private HibernateTemplate ht;
	@Override
	public void saveVendor(Vendor ven)
	{
		ht.save(ven);
	}
```

```
}
```

Service

POJI

```
package com.app.service;
public interface IVendorService
{
	public void saveVendor(Vendor ven);
}
```

POJO

```
package com.app.service.impl;
@Service
public class VendorServiceImpl implements IVendorService
{
@Autowired
private IVendorDao dao;
```

```
	@Override
	public void saveVendor(Vendor ven)
	{
		Dao.saveVendor();
	}
}
```

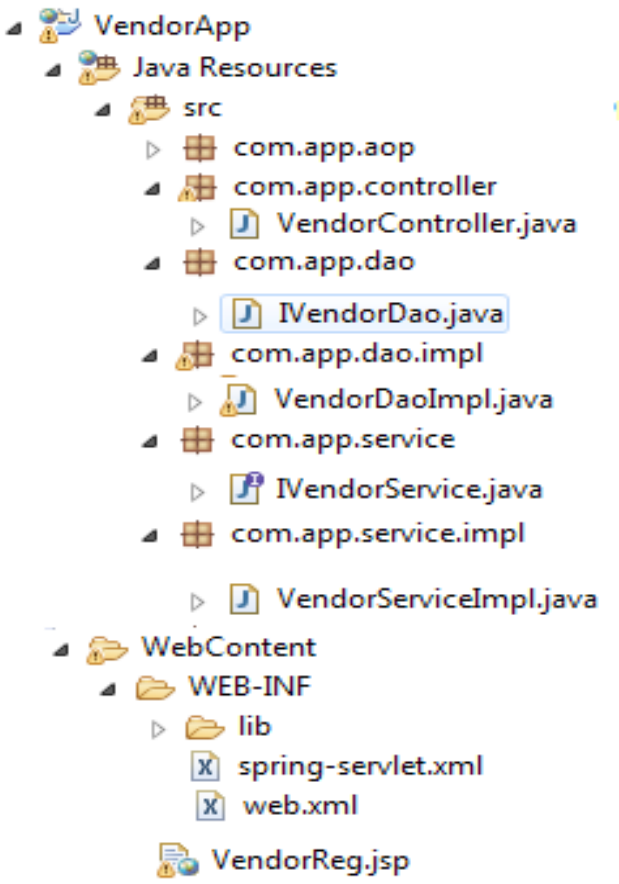
Controller

```
package com.app.controller;
@Controller
public class VendorController
{

	@Autowired
	private IVendorService venServ;
	@RequestMapping(“/venReg”)
	public String getVenRegPage()
	{
		return “VendorReg”;
	}
```

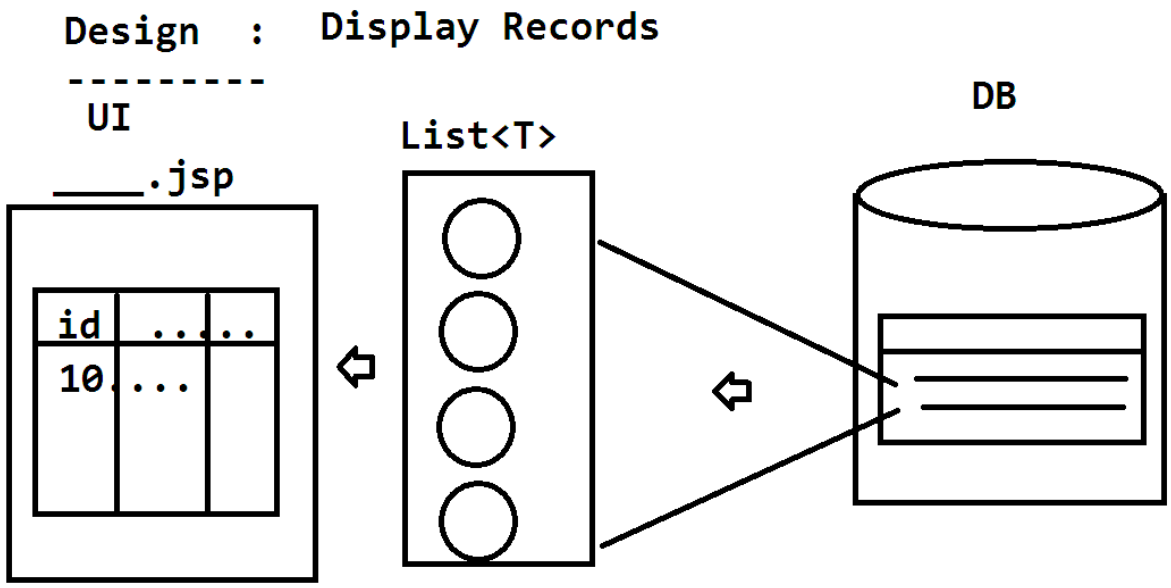
```
@RequestMapping(value="/regSave",
method=RequestMethod.POST)
public String checkReg(@ModelAttribute("vendor")
Vendor  ven,ModelMap map)
{
venServ.saveVendor(ven);
map.addAttribute("msg","successfully registered");
return "VendorReg";
}
}
```

VendorReg.jsp

```
<form action="  " method="POST">
    Id : <input type="text" name="venId">      <br>
    Name : <input type="text" name="venName">    <br>
    Email : <input type="text" name="email">      <br>
  <input type="submit" value="Save">
</form>
${msg}
```



Design : Display Records



- Add **jstl-1.2.jar** in lib folder. Because we are using JSTL for-each concept to display list collection.

Java

```
for(Class var : callObj)
{
    System.out.println(var);
}
```

JSTL

```
<c:forEach items="${callObj}"  var="var">
    <c:out value="${var}"/>
</c:forEach>
```

Step 1

Add a method in I___Dao and implement in ___DaoImpl.

Ex :

ILocationDao.java

```
public List<Location> getAllLocations();
```

LocationDaoImpl.java

```
public List<Location> getAllLocations()
{
    return ht.loadAll(Location.class);
}
```

Step 2

Specify same method in I___Service and implement in ___ServiceImpl as below.

Ex :

ILocationService.java

```
public List<Location> getAllLocations();
```

LocationServiceImpl.java

```
public List<Location> getAllLocations()
{
    return dao.getAllLocations();
}
```

Step 3

Define Controller method that should be mapped with one url.Controller should call service.method() and returned data add to ModelMap to display at UI.

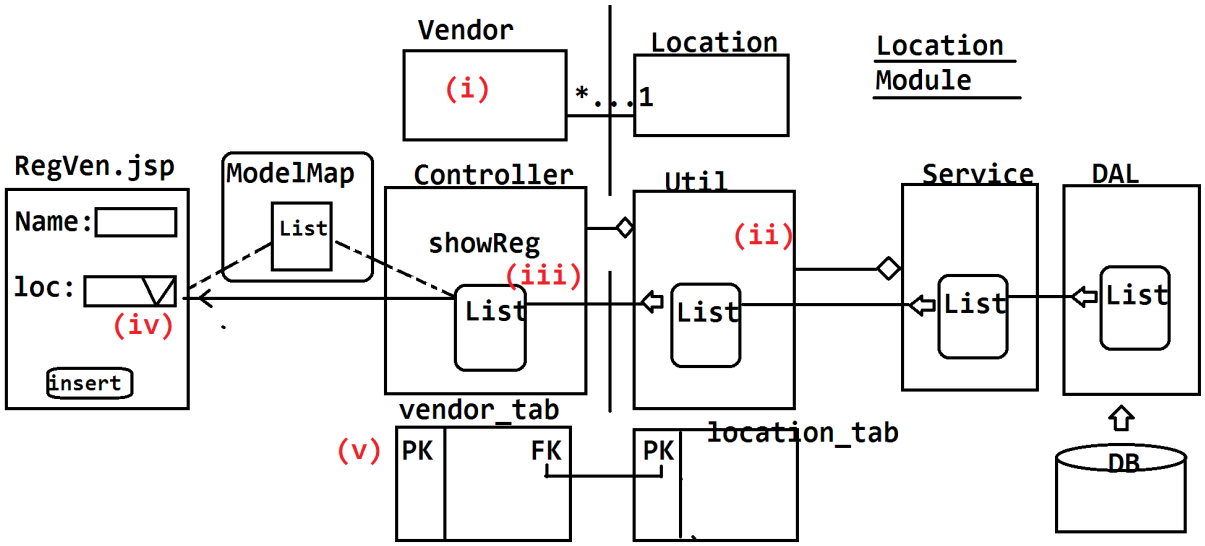
Ex :

LocationController.java

```
@RequestMapping("/showLocs")
public String showLocObjs(ModelMap map)
{
    List<Location> locList=service.getAllLocations();
    //to send to UI,add the list to ModelMap
    map.addAttribute("locListObj",locList);
    return "LocationData";
}
```

Connecting Vendor and Location Modules

Design



Coding

1. Model class relations

Vendor-------<>Location

@Entity

@Table(name=”vendor_tab”)

public class Vendor

{

……

……

@ManyToOne

```
@JoinColumn(name="loc_fk")
private Location location;
//setters,getters
//hashCode,equals
//toString,constructor
}
```

2. Util class of Location should have relation with LocationService interface(POJI).

LocationUtil---------<>ILocationService

Define a method to get all locations from service in util class. Create object using **@Component** and @Autowired at dependency level.

Code

```
@Component
public  class LocationUtil
{
    @Autowired
    private ILocationService service;
    /**
     *write a method to get all locations
    */
```

```
        public List<Location> getAllLocations()
        {
            Return service.getAllLocations();
        }
    }
```

3. Controller of Vendor should use Location Util to get data and to send data to UI.

VendorController ------<> LocationUtil

Code

```
        @Controller
        public class VendorController
        {
            ……..
            @Autowired
            private LocationUtil util;
            ……..
            ……..
            @RequestMapping(“/regVen”)
            public String showVenReg(ModelMap map)
            {
```

```
                List<Location> locList=util.getAllLocations();
                Map.addAttribute("locList",locList);
                return "VendorReg";
            }
        }
```

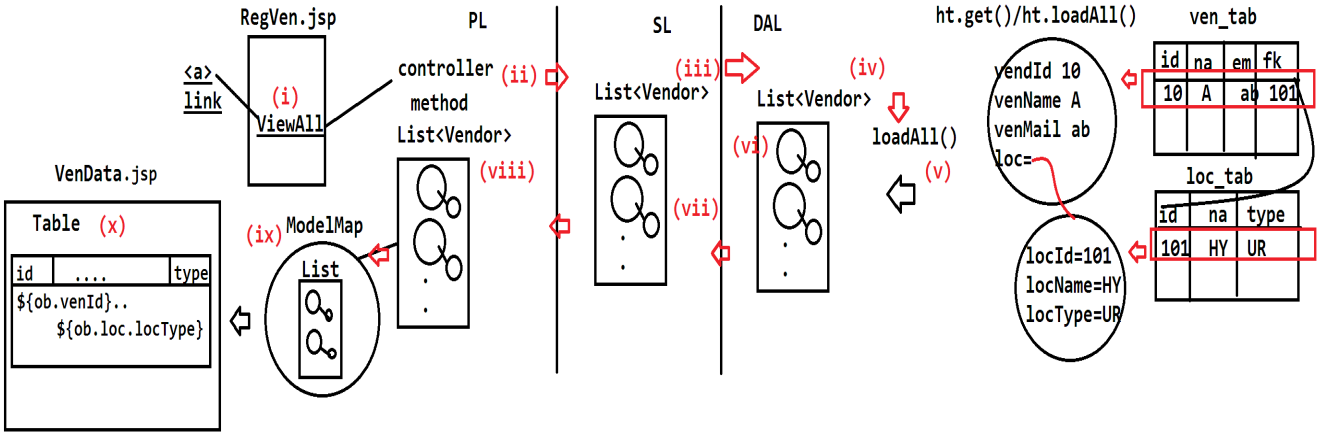
4. UI changes.

Display above list as drop down (<select>), one object as one option (<option value="id">name</option>).

Ex :

```
<form ……..>
    ……..
    ……..
    Location: <select name="location.locId">
                <c:forEach items="${locList}" var="loc">
                    <option value="${loc.locId}">
                        <c:out value="${loc.locName}"/>
                    </option>
                </c:forEach>
              </select>
</form>
```

Displaying Vendors Data



Step 1

Adding link at Vendorreg.jsp page

```
<a href="viewAllVendors">View All</a>
```

Step 2 Controller

Call a methid that access service to get all vendors as list object and send it to UI page using ModelMap.

Code

```
@RequestMapping("/viewAllVendors")
public String showAllVendors(ModelMap map)
{
    List<Vendor> venList=service.loadAllVendor();
    map.addAttribute("venListObj",venList);
    return "VendorData";
}
```

Step 3

Define a method getAllVendors() : List<Vendor> at DAL and SL, use loadAll() to get tha data from DB.

Ex :

```
public List<Vendor> getAllVendors()
{
    return ht.loadAll(Vendor.class);
```

}

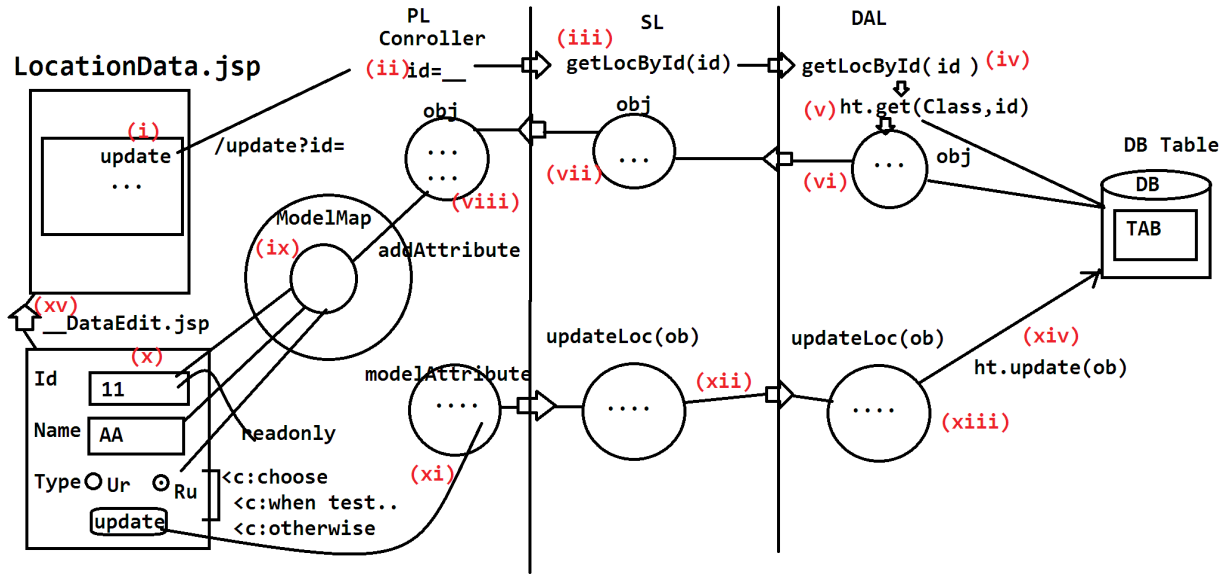
Step 4

Display the data at UI using obj with forEach loop.

Ex :

```
<c:forEach items="${venListObj}" var="obj">
    <c:out value="${obj.venId}"/>
    <c:out value="${obj.venName}"/>
    <c:out value="${obj.email}"/>
    <c:out value="${obj.locName}"/>
    <c:out value="${obj.locType}"/>
</forEach>
```

Updating Location Data



Edit programming

I. On click hyperlink, an object should be loaded from controller and must send to EDIT page.

Ex

 EDIT

Use method getLocById(int) : Location

II. Display them using ModelMap and EL in JSP (Edit page) as values (<input value=”${……}” />)

In case of check box and radio buttons,use below format.

<c:choose>

<c:when test=”${‘urban’ eq loc.locType}”>

<input type=”radio” name=”locType” value=”urban”

checked=" checked"/> Urban

<input type="radio" name="locType" value="rural"/> Rural

</c:when>

<c:otherwise>

<input type="radio" name="locType" value="urban"/> Urban

<input type="radio" name="locType" value="rural"/> checked=" checked"/>Rural

</c:otherwise>

</c:choose>

On click submit (Update),form is converted to object (ModelAttribute).Send this object to DAL using SL and call **ht.update()** to store in DB,then redirect to data jsp.

Validations

- To avoid invalid data to application,we do validations at 2 levels.

1. Client side
2. Application side (Server side)

- Client side validations are done by using JavaScript,JQuery…..

1.Client side validations

Changes in code

i. <form> tag level

<form name=”f1” onSubmit=”return validateInput();” action=” “ method=” “>

ii. <script> tag

Define one function under <script> to do validations.

Ex

```
<script type=”text/javascript”>
    function validateInput()
    {
        ……..
        ……..
    }
```

iii. tag

To display message at input level only ,use this

Ex

```
<span id=”a”></span>
```

To insert message into this, use **.innerHTML=’ ‘**

Ex

document.getElementById(‘a’).innerHTML=’msg’;

To link external file for CSS and JavaScript,use below tags

JavaScript

<script type=”text/javascript” src=”file.js”>

</script>

CSS

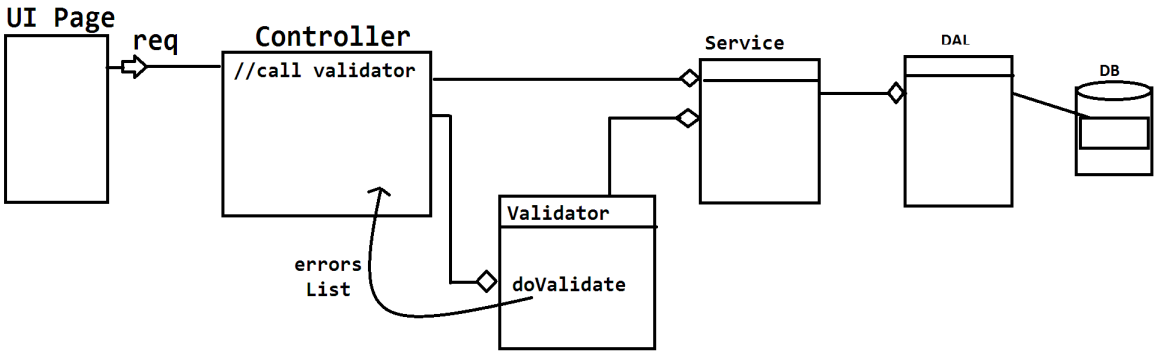
<link type=”text/css” href=”file.css” rel=”stylesheet”>

</link>

2. Server side Validations

- These are also known as DB level validations or logical validations.
- Onsubmit request to server/application,validation will be done before save/update/delete operations.

Design of validation class



Ex : Location name check in DB

DAL

```
public Boolean isLocationNameExist(String locName)
{
return ht.find("from  "+Location.class.getName()+"  where locName=?",locName).size()>0;
}
```

Validator code

```
package com.app.validator;
@Component
public class LocationValidator
{

@Autowired
```

```
        private ILocationService service;
        public List<String> doValidator(Location loc)
        {
            if(…….. isLocNameExist ……..)
            return list;
        }
    }
```

Binding with controller

```
    @Controller
    public class LocationController
    {
        @Autowired
        private LocationValidator validator;
        ……..
        ……..
        public String saveLocation(Location loc)
        {
            List l=validator.doValidate(loc);

            If(l.isEmpty())
```

```
                    {
                        //save data into DB
                    }
                    else
                    {
                        //send errors to UI
                    }
                return "LocationReg";
            }
        }
```

Display Errors at UI page

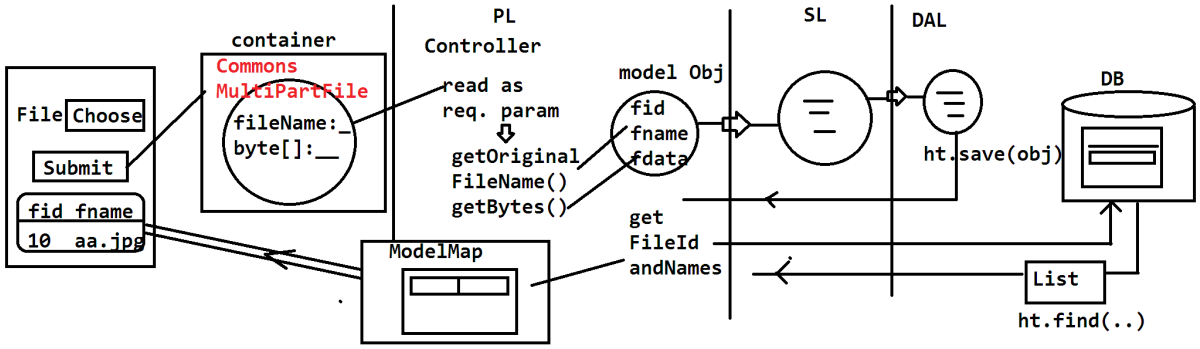
Use <c:forEach> & <c:out>

Join query of HQL

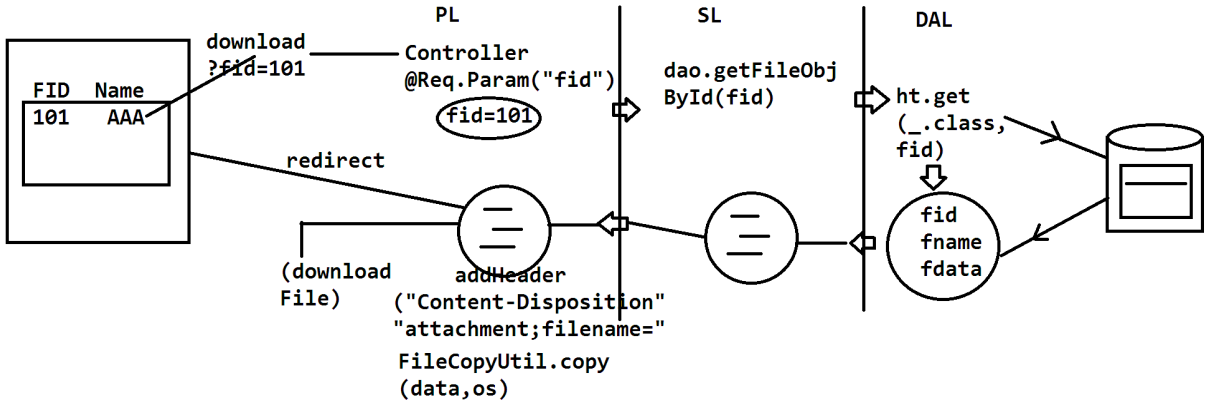
```
from Vendor.class.getName() as ven left join ven.loc as loc
where loc.locId=?
```

File Upload & Download

Design 1 - Upload



Design 2 – Download



- To do upload & download, we have to provide 2 jars.

1. Commons-io.jar
2. Commons-fileupload.jar

- We have to configure **CommonsMultiPartResolver** class in spring config file.

Configuration :

```
<bean id="multipartResolver"
class="org.springframework.web.multipart.commons.
        CommonsMultipartResolver"/>
```

- Every file object is a **CommonsMultipartFile**, it contains originalFileName() and byte[] (file data).

Code

HTML

```
<form action="uploadFile" method="post" enctype="multipart/form-data">
        Select a File : <input type="file" name="filesFromBrowser"/><br>
                        <input type="submit" value="Upload"/><br>
                                <c:out value="${fileUploadStatus}"/>
</form>
```

```
package com.app.controller;
@Controller
public class FileController
{
	@Autowired
	private IFileUploadService fileServ;

	@RequestMapping("/getFileUploadPage")
	public String getFileUploadPage()
	{
		return "FileUpload";
	}
	@RequestMapping(value="/uploadFile",method=  RequestMethod.POST)
	public String uploadFile(@RequestParam("filesFromBrowser")
				CommonsMultipartFile[] file,ModelMap map)
	{
		for(CommonsMultipartFile fileS : file)
		{
			String fileName=fileS.getOriginalFilename();
			//System.out.println(fileS.getContentType());
			//FileUpload is an user defined class
			FileUpload fup=new FileUpload();
			fup.setFileName(fileName);
				//System.out.println
			(".....>>>>>"+URLConnection.guessContentTypeFromName(fileName));
				fup.setFileData(fileS.getBytes());
				boolean success=fileServ.fileUpload(fup);
				if(success)
					map.addAttribute("fileUploadStatus","File uploaded
			Successfully...");
```

```
                    else
                        map.addAttribute("fileUploadStatus","Error got when
            uploading the file...");
                    }
                    return "FileUpload";
                      "redirect:filesData";
                }
            }
        }
```

DAL

```
    package com.app.dao.impl;
import java.util.List;
@Repository
public class FileUploadDaoImpl implements IFileUploadDao
{
        @Autowired
        private HibernateTemplate ht;
        @Override
        public boolean fileUpload(FileUpload upload)
        {
            int success=(int)ht.save(upload);
```

```
            if(success>0)
                return true;
            else
                return false;
        }
    }
```

- While downloading, use HTTP header 'Content-Disposition' which indicates sending 'attachment;filename=" to download data through OutputStream (ServletOutputStream).
- **@Lob** is used to specify byte[] data, level to represent BLOB (Binary Large Object).
- If you want to store only text files then use @Lob with char[] data; known as CLOB (Character Large Object).

Download

Code

HTML

```
<table border="2">
<h3><font color='blue'>Files Uploaded</font></h3>
<tr><th>File Id</th><th>File Name</th><th>Operations</tr>
<c:forEach items="${filesUploaded}" var="files">
    <tr><td><c:out value="${files.fid}"/></td>
        <td><c:out value="${files.fileName}"/></td>
```

```
<td><a href='downloadFile?fid=<c:out value="${files.fid}"/>'>  Download
</a></td>
        </tr>
</c:forEach>
```

Controller

```
package com.app.controller;
@Controller
public class FileController
{
    @Autowired
    private IFileDownloadService fileServ;
    @RequestMapping(value="/filesData")
    public String getFilesData(ModelMap map)
    {
        List<FileUpload> files=fileServ.getAllFiles();
        map.addAttribute("filesUploaded",files);
        return "FilesData";
     }

     @RequestMapping("downloadFile")
```

```
  public String downloadFile(HttpServletResponse
response,ModelMap map,@RequestParam("fid")int fid)
        {
            FileUpload file=fileServ.getFileUploadById(fid);
response.addHeader("content-
disposition","attachment;filename="+file.getFileName());
            try
            {
FileCopyUtils.copy(file.getFileData(), response.getOutputStream());
            }
            catch(IOException e)
            {
                e.printStackTrace();
            }
            map.addAttribute("filesUploaded",fileServ.getAllFiles());
            return "FilesData";
        }
}
```

DAL

```
package com.app.dao.impl;
@Repository
```

```
public class FileUploadDaoImpl implements IFileUploadDao
{
	@Autowired
	private HibernateTemplate ht;

	@Override
	public List<FileUpload> getAllFiles()
	{
		//System.out.println(ht.loadAll(FileUpload.class));
		return ht.loadAll(FileUpload.class);
	}
}
```

File Export

Excel export using Apache POI jar

Step 1

Download POI jar and add to lib folder

Poi-3.16-beta1.jar

Step 2

Create a location class using **AbstractExcelView** class for excel design.

Ex :

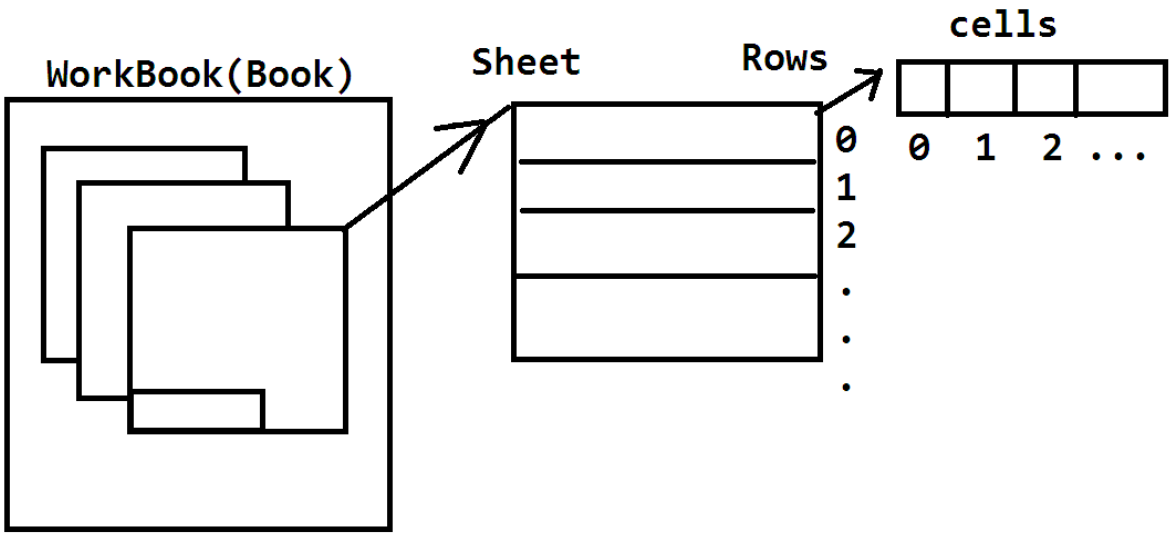
Book (has) sheets

Sheet (has) rows

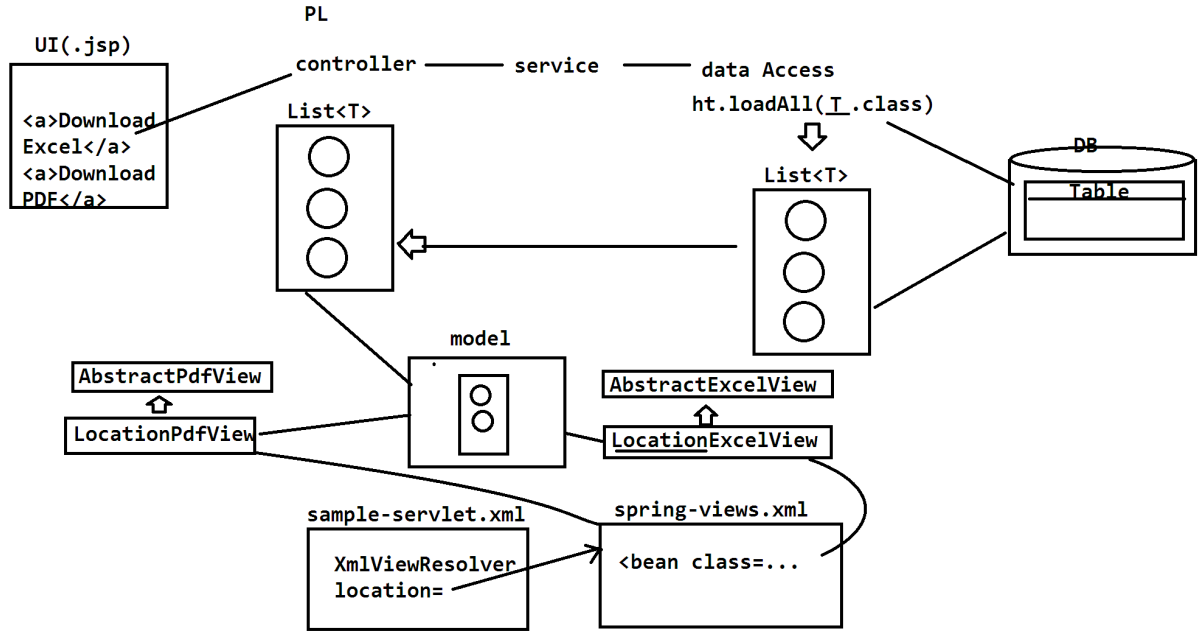
Rows (has) cells

Row and Cell number starts from zero.

Design 1



Design 2



Code

Configuration

✓ Create a separate spring config file with name **spring-views.xml** and write the following configuration.

```
<bean name="locationExcelView"
class="com.app.controller.view.LocationExcelView"/>
```

HTML

```
<a href="LocationsData"><font color='green'>Export Excel</font></a>
```

Controller

```
package com.app.controller;

@Controller
```

```
public class FileController
{
                @Autowired
                 private ILocationService locServ;

                @RequestMapping("LocationsData")
public String exportExcel(ModelMap map,HttpServletResponse res)
                {
                        List<Location> locList=locServ.getAllLocations();
                        map.addAttribute("locListObj",locList);
res.addHeader("content-disposition", "attachment;filename="+"LocationsData");
                        return "locationExcelView";
                }
}
```

Export PDF file using Itext jar

✓ Itext jar (from lowagie) is used to design PDF in the form of document format.

✓ Jar : itext-2.1.7.jar

✓ PDF is designed using Document (file),in this ,to add any child use **add(Element) method.**

✓ An element can be Paragraph, Image, Table, Heading , Attachment, Header, Footer ……

✓ After creating every element, add it to Document.

Code

HTML

```
<a href="LocationsPdfData"><font color='green'>Export PDF</font></a>
```

Configuration

✓ Configure the below code in spring-views.xml file

```
<bean name="locationPdfView"
class="com.app.controller.view.LocationPdfView"/>
```

Controller

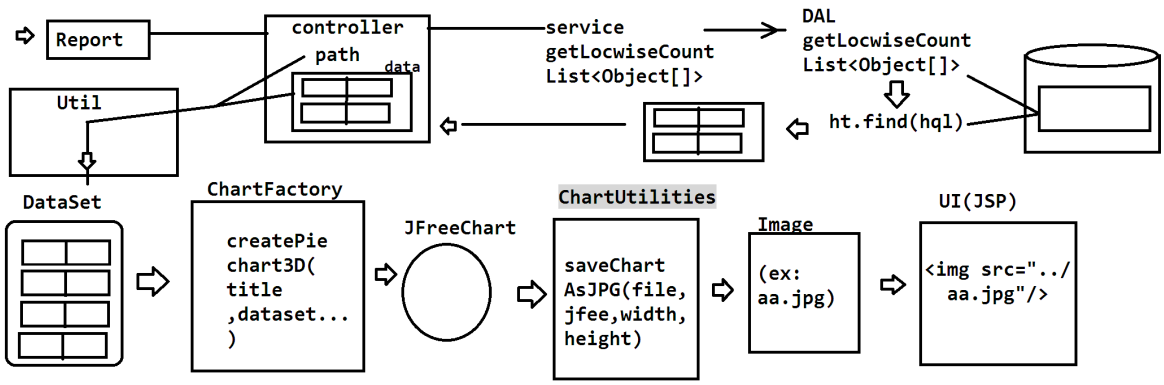
```
package com.app.controller;

@Controller

public class FileController

{

        @Autowired

        private ILocationService locServ;

        @RequestMapping("LocationsPdfData")

        public String exportPdf(ModelMap map,HttpServletResponse res)

        {

                List<Location> locList=locServ.getAllLocations();

                map.addAttribute("locListObj", locList);
```

```
            return "locationPdfView";
        }
}
```

Generating Charts

Generating Pie chart

Design



Jars

jfreechart-1.0.19.jar

jcommon-1.0.23.jar

Step 1

- To get current server (application) path, use context (ServletContext) in controller using

@Autowired

private ServletContext context;

also util class object to call generateAndSavePie()

@Autowired

private LocationUtil util;

- context.getRealPath("/") returns current application path(in server) like root directory.

Step 2

- HQL to get data using ' group by ' from DaoImpl.

```
String hql=" select loc.locType , count(loc.locType) from
"+Location.class.getName()+" loc group by loc.locType";
```

Step 3 DefaultPieDataset

- This class is used to create dataset object for pie chart.
- To add one type and it's value use setValue(type,value) method over dataset object.

```
No. of types= no. of parts of pie chart
```

Step 4 ChartFactory

- This class is used to generate **PieChart** object using createXXX() method which takes Title of chart,dataset,tooltip,legend,……

- It returns **JFreeChart** (super type).

Step 5 ChartUtils

- This class supports converting JFreeChart to image type which also takes dataset and file object with width and height.

Step 6

- To access this image at UI,use

```
<img src="../file.jpg"/>
```

Code

HTML

```
<form action="locPieChart" method="GET">
    <input type="submit" value="Generate PieChart">
```

Controller

```
package com.app.controller;
@Controller
public class LocationController
{
    @Autowired
```

```
	private LocationUtil locUtil;

	@Autowired
	private ServletContext ctx;
	@RequestMapping("locPieChart")
	public String generatePieChart()
	{
		String path=ctx.getRealPath("/");
System.out.println("Path upto Project Root Directory(VendorApp) :\t "+ path);
	List<Object[]> list=locServ.getLocationTypeWiseList();
		locUtil.createPieChart(path,list);
		return "LocationChart";
	}
}
```

DAL

```
	package com.app.dao.impl;
@Repository
public class LocationDaoImpl implements ILocationDao
{
	@Autowired
	private HibernateTemplate ht;
	@Override
```

```
public List<Object[]> getLocationTypeWiseList()
{
String hql="select loc.locType,count(loc.locType) from "+Location.class.getName()
+" loc group by loc.locType";
List<Object[]>  list=ht.find(hql);
return list;
}
}
```

LocationUtil

```
package com.app.util;
@Component
public class LocationUtil
{
@Autowired
private ILocationService locServ;
public void createPieChart(String path,List<Object[]> list)
{
    DefaultPieDataset dataSet=new DefaultPieDataset();
for(Object[] obj : list)
{
dataSet.setValue(obj[0].toString(), new Double(obj[1].toString()));
}
```

```
/*
* create JFreeChart object using ChartFactory by passing dataSet object
*/
JFreeChart chart= ChartFactory.createPieChart3D("Location Chart", dataSet, true, true, false);
try
{
System.out.println(path);
ChartUtilities.saveChartAsJPEG(new File(path+"/Charts/LocType.jpg"), chart, 250, 250);
}
catch(Exception e)
{
e.printStackTrace();
}
}
}
```

Generating Bar chart

- For this, we need to dataset using **DefaultCategoryDataset**.
- Once we get JFreeChart object,call method saveAsJpg() on util object.
- To get JFreeObject,use

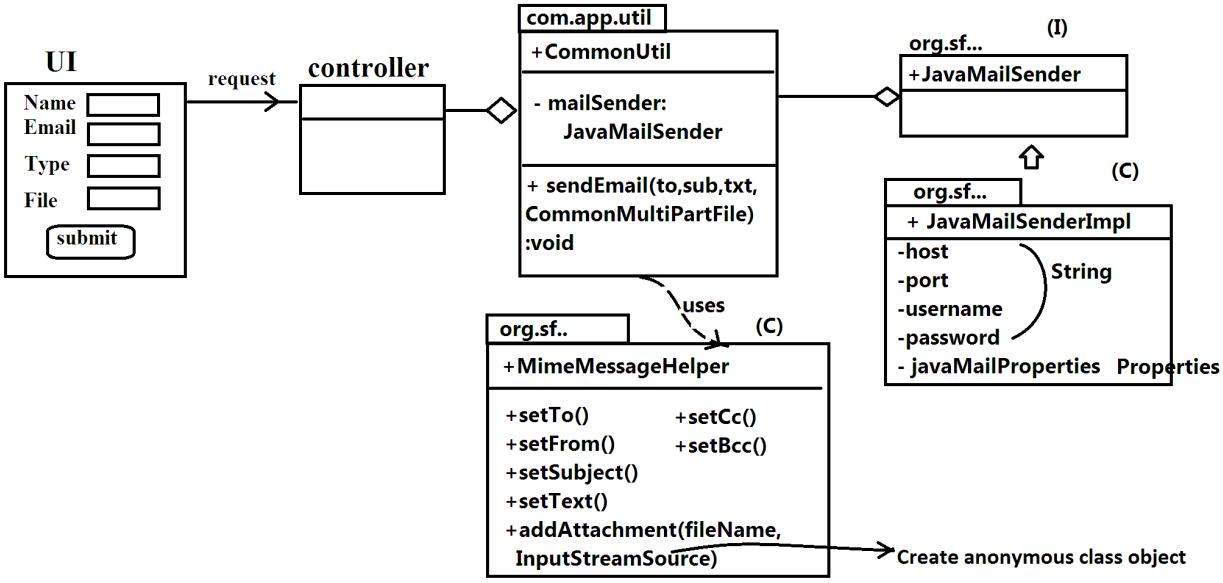
createBarChart3D(file,xAxisLabelName,yAxisLabelName,dateset)

EMAIL

Sending Email using Java Mail implementation by Spring

- Spring supports sending email with attachments by using it's POJI-POJO's JavaMailSender (i) and JavaMailSenderImpl (c).

Design



Note

- ✓ Add mail.jar to lib folder
- ✓ Syntax for Anonymous class

Syntax

```
new [interface](){
    //override all the abstract methods
}
```

Example

```
interface  A{
    m1();
```

```
}


new A(){
        m1(){
        ……..
        ……..
        }
}
```

- Here for adding attachment, we used **InputStreamSource** (i) with getInputStream() : InputStream method.

```
new InputStreamSource(){
    public InputStream getInputStream(){
        return fileObj.getInputStream();
    }
}
```

- Adding attachment code

```
helper.addAttachment(fileObj.getOriginalFileName(),
      new InputStreamSource(){
        public InputStream getInputStream()throws IOException {
              return fileObj.getInputStream();
        }
      }
);
```

Code

HTML

```
<form action=”sendMail” method=”GET”>
      <input type=”submit” value=”Send Mail”>
</form>
```

Controller

```
package com.app.controller;
@Controller
public class CustomerController
{
```

```
@Autowired
private ICustomerService custServ;
@Autowired
private EmailUtil mailUtil;
@RequestMapping("sendMail")
public String sendMailToCustomer(ModelMap map)
{
Customer cust = (Customer) custService.getCustomerById(1000);
String subject="Hello Mr/Ms/Mrs. "+ cust.getCustName()+". This is regarding registration to VDM.";
String message="Hi…This is a test message from VDM";
mailUtil.sendEmailToCustomer(
cust.getCustEmail(),subject, message);
}
map.addAttribute("id", "mail sent to customer ");
return "CustReg";
}
```

EmailUtil

```
package com.app.util;
@Component
public class EmailUtil
{
```

```
@Autowired
private JavaMailSender mailSender;
public void sendMail(String to,String subject,String text,final
CommonsMultipartFile file)
{
MimeMessage mime= mailSender.createMimeMessage();
try{
MimeMessageHelper mimeHelper=new MimeMessageHelper(mime,true);

mimeHelper.setFrom("xyz@gmail.com");
mimeHelper.setTo(to);
mimeHelper.setSubject(subject);
mimeHelper.setText(text);

if(file!=null)
{
mimeHelper.addAttachment(
file.getOriginalFilename(),
new InputStreamSource(){ public InputStream
getInputStream()throws IOException{
return file.getInputStream();}
}
```

```
                    );
                }
                mailSender.send(mime);
            } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Schedulers

- It is an automated task, this will be executed by container without any call by programmer.
- It is also known as Background process.
- One Demon Thread will be created for every task.

Demon Thread

✓ It is a thread runs in background without disturbing other threads or programming execution.

✓ This thread cannot be created by a programmer.

- To activate schedulers, configure **task** schema in spring config file.

```
<beans ……..
        xmlns:task=" http://www.springframework.org/schema/task"
```

```
xmlns:schemaLocation="……
    http://www.springframework.org/schema/task
    http://www.springframework.org/schema/task/spring-task-3.0.xsd">
```

- Activating of annotations for task (or) schedulers

```
<task : annotation-driven />
```

Example

- Define one void and public with zero param method and this must be annotated with **@Scheduler**.

```
@Scheduler(fixedDelay=5000)
public void msg()
{
    System.out.println("hello ……..."+\t + new Date());
}
```

Spring Schedulers :

In spring, scheduling tasks are done using @Scheduled annotation.

The @Scheduled annotation can be added to a method along with delayTimes. @Scheduled annotation has 3 types. They are:

fixedDelay: Incase, if you specify fixed delay as 5 seconds, then the annotated method would be invoked at every 5 seconds with a fixed delay, meaning that the period will be measured from the completion time of each preceding invocation.

fixedRate: Incase, if you specify fixed rate as 5 seconds, then the annotated method would be executed every 5 seconds measured between the successive start times of each invocation.

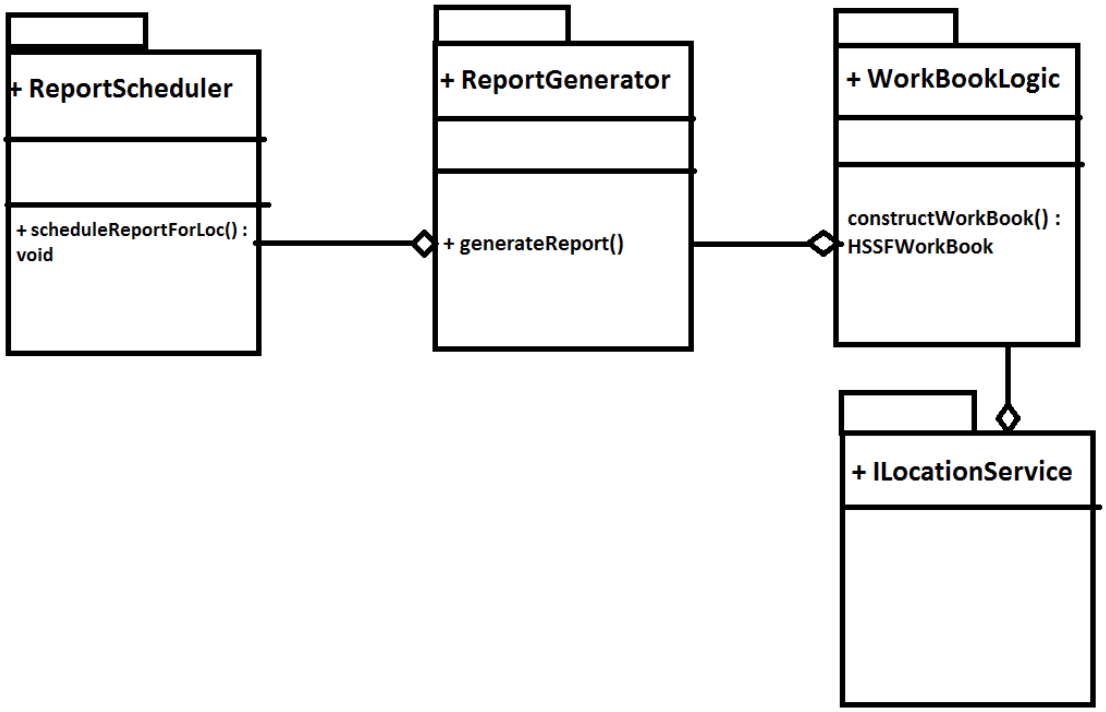
cron: It supports cron expression.

Examples are given for all above cases. Before annotating any method with @Scheduled, make sure that you are making below given xml based configurations. Add task:annotation-driven tag and also register your job bean where your annotated methods are presented.

```
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:task="http://www.springframework.org/schema/task"
 xsi:schemaLocation="    http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
   http://www.springframework.org/schema/task
   http://www.springframework.org/schema/task/spring-task-3.0.xsd">

   <task:annotation-driven />
```

```
</beans>

 @Scheduled(fixedDelay=5000)
   public void updateReportAndGenerate(){
      System.out.println("Started fixed delay job");
      /**
       * add your scheduled logic here
       */
   }
  @Scheduled(cron="*/2 * * * * MON-FRI")
   public void updateReportAndGenerate(){
      System.out.println("Started cron job");
      /**
       * add your scheduled job logic here
       */
   }
```

Using Schedulers for Report generation



Code

ReportScheduler.java

```
@Component
public class ReportScheduler
{
    @Autowired
    private ReportGenerator gen;
    @Scheduled
    public void scheduledReportForLoc()
```

```
                {
                        gen.generateReport();
                }
        }
```

ReportGenerator.java

```
        @Component
        public class ReportGenerator
        {
                @Autowired
                private WorkBookLogic logic;
                public void generateReport()
                {
                        HSSFWorkBook book=logic.constructWorkBook();
                        try{
FileOutputStream fos=new FileOutputStream("d:/reports/file.xls");
book.write(fos);
fos.flush();
fos.close();
                                }
                        catch(Exception e){
                                e.printStackTrace();
```

```
                }
            }
        }
```

WorkBookLogic.java

```
        @Component
        public class WorkBookLogic
        {
            @Autowired
            private ILocationService serv;
            public HSSFWorkBook constructWorkBook()
            {
                HSSFWorkBook book=new HSSFWorkBoo();
                HSSFSheet sheet=book.createSheet("Location");
                List<Location> locList=serv.getAllLocations();
                setHeader(sheet);
                setBody(sheet,locList);
            }
            private void setHeader(HSSFWorkSheet sheet)
            {
                //logic for creating rows & columns
            }
            private void setBody(HSSFWorkSheet sheet,List<Location> list)
```

```
        {
            //logic for appending body
        }
    }
```

Create file using Date & Time

Step 1

Create Date / Calender class object.

Step 2

Use it's method to get date and time with path and extension.

Ex

```
String fileNameWithPath="d:/reports/locFile/"+ (d.getYear()+1900)+"-"+
(d.getMonth()+1)+"-" +d.getDate()+"-" +d.getHours()+ ":" + d.getMinutes()
+":"+d.getSeconds()+".xls";
```

CRON Expression

Cron

✓ It is a date & time representation expression from UNIX, also used in Spring framework to specify schedule Date and Time. Internally used **CronSequenceGenerator**.

✓ It provides details of sec,min,hour,day of month,month,week day…

✓ It also accepts range & possible values using "-" , "," symbols.

✓ Example

0-59 0-59 1-24 1-31 1-12 MON-SUN

A Cron Expressions

A cron expression is a string consisting of six or seven subexpressions (fields) that describe individual details of the schedule.

These fields, separated by white space, can contain any of the allowed values with various combinations of the allowed characters for that field. Table A-1 shows the fields in the expected order.

Table A-1 Cron Expressions Allowed Fields and Values

Name	Required	Allowed Values	Allowed Special Characters
Seconds	Y	0-59	, - * /
Minutes	Y	0-59	, - * /
Hours	Y	0-23	, - * /
Day of month	Y	1-31	, - * ? / L
Month	Y	0-11 or JAN-DEC	, - * /
Day of week	Y	1-7 or SUN-SAT	, - * ? / L #
Year	N	empty or 1970-2099	, - * /

Example A-1 Cron Expressions

Cron expressions can be as simple as * * * * ? * or as complex as 0 0/5 14,18,3-39,52 ? JAN,MAR,SEP MON-FRI 2002-2010.

Here are some more examples:

Expression	**Means**
0 0 12 * * ?	Fire at 12:00 PM (noon) every day
0 15 10 ? * *	Fire at 10:15 AM every day
0 15 10 * * ?	Fire at 10:15 AM every day
0 15 10 * * ? *	Fire at 10:15 AM every day
0 15 10 * * ? 2005	Fire at 10:15 AM every day during the year 2005
0 * 14 * * ?	Fire every minute starting at 2:00 PM and ending at 2:59 PM, every day
0 0/5 14 * * ?	Fire every 5 minutes starting at 2:00 PM and ending at 2:55 PM, every day
0 0/5 14,18 * * ?	Fire every 5 minutes starting at 2:00 PM and ending at 2:55 PM, AND fire every 5 minutes starting at 6:00 PM and ending at 6:55 PM, every day
0 0-5 14 * * ?	Fire every minute starting at 2:00 PM and ending at 2:05 PM, every day
0 10,44 14 ? 3 WED	Fire at 2:10 PM and at 2:44 PM every Wednesday in the month of March
0 15 10 ? * MON-FRI	Fire at 10:15 AM every Monday, Tuesday, Wednesday, Thursday and Friday
0 15 10 15 * ?	Fire at 10:15 AM on the 15th day of every month
0 15 10 L * ?	Fire at 10:15 AM on the last day of every month
0 15 10 ? * 6L	Fire at 10:15 AM on the last Friday of every month
0 15 10 ? * 6L	Fire at 10:15 AM on the last Friday of every month
0 15 10 ? * 6L 2002-2005	Fire at 10:15 AM on every last friday of every month during the years 2002, 2003, 2004, and 2005
0 15 10 ? * 6#3	Fire at 10:15 AM on the third Friday of every month

Expression	Means
0 0 12 1/5 * ?	Fire at 12 PM (noon) every 5 days every month, starting on the first day of the month
0 11 11 11 11 ?	Fire every November 11 at 11:11 AM

StopWatch

- It is a pre-defined class to calculate time taken for a task/sub task.
- It provides time printing based on method/task wise time % , known as prettyPrint() : String
- To create a new Task, stop() the watch and use start(String taskName) method.

Example

```
public String saveData()
{
    StopWatch watch=new StopWatch("Data Saving");
    watch.start("validate");
    ------------
    ------------
    watch.stop();
    watch.start("save data");
    ----------------
    ----------------
```

```
        watch.stop();
        --------------
        --------------
        System.out.println(watch.prettyPrint());
}
```

Output

```
StopWatch 'Data Saving' : running time ('millis') = 259
-----------------------------------------------------------------------------------------
-
ms                    %                    Task Name
-----------------------------------------------------------------------------------------
-
00222                 086%                 Validate
00037                 014%                 Save Data
```

AOP

(Aspect Oriented Programming)

- It is a process of separation of external services and business logic.
- We should never write External service inside the Business classes.
- AOP is also known as Cross cutting Concern (separate at code level and link at runtime) .

Terminology

Aspect

It is a class which represents external service.

Advice

It is a method of Aspect class. It contains external service logic.

Pointcut

It is an expression. It will provide details of Business class method. i.e. it tells which method needs service but not what service it needs.

JoinPoint

It is a combination of Advice + Pointcut .It provides which business method connects to what advice.

Types

- Spring AOP provides 5 types of advices. They are
- Before advice

First advice will be executed followed by Business method.

- After advice

First Business method will be executed followed by advice.

- Around advice

Execution order is, advice→business method→advice

- After Returning advice

Execution order is business method→success→advice

- After Throwing advice

Execution order is, business method→exception→advice

AOP Coding Steps

Step 1

Activate AOP using <aop : aspect-autoproxy > in spring coding file.

We need to add aop schema at <beans > tag level.

Step 2

Define an aspect with pointcut and advice

Step 3

Write some implementation code inside the advice.

Example

```
package com.app.aop;
@Aspect
@Component
public class Logging
{
        @Pointcut(“execution (* com.app.*..*.*(..))”)
        public void point1()
        {
        }


        @Around(“point1()”)
       public Object calculateTime(ProceedingJoinPoint jp)throws Throwable
        {
StopWatch watch=new StopWatch(jp.getTarget().getClass(). getName());
watch.start(jp.getSignature().getName());
```

```
Object ob=jp.proceed();
watch.stop();
System.out.println(watch.prettyPrint());
Return ob;
}
```

- ✓ **jp.proceed()** It is used to call business method.
- ✓ **jp.getTarget()** It indicates Business class object.
- ✓ **jp.getSignature()** It provides Business method signature (i.e. name,parameters).

Here, stop watch is started before Business method is called and stopped after Business method is executed.

Log4j

Logging using Log4j

Log4j Using Spring (Log4j+Commons Logging):-

- To do logging, use following components (classes)

Logger	Appender	Layout

Logger:

- It will enable logging to a class and provides methods like

Methods:

debug < info < warn < error < fatal

debug	Message with values like true/false,id value,count , ...
info	It is normal information like saved or not, connected or not, printed or not,.....
warn	casting problems, no generic types, non-used local & instance or static variables,.....
error	Prints normal exceptions like NullPointerException, NumberFormatException,ArrayIndexOutOfBoundsException,.....
fatal	It indicates high level problems like SecurityException's, server / DB droppings.

✓ private final Log log = LogFactory.getLog(this.getClass());

Appender:

- Appender job is to write the messages into the external file or database or smtp.
- Logger classes generates some statements under different levels right, this Appender takes these logstatements and stores in some files or database

- Appender is an interface , in log4j we have different Appender implementation classes as shown below

- ✓ FileAppender [writing into a file]
- ✓ ConsoleAppender [Writing into console]
- ✓ JDBCAppender [For Databases]
- ✓ SMTPAppender [Mails]
- ✓ SocketAppender [For remote storage]
- ✓ SocketHubAppender
- ✓ SyslogAppendersends
- ✓ TelnetAppender

- Again in FileAppender we have 2 more

- ✓ RollingFileAppender
- ✓ DailyRollingFileAppender

LayOut:-

- It will tell how to print log messages like HTML format ,XML or Pattern.

- The layouts given by log4j are

- ✓ SimpleLayout
- ✓ PatternLayout
- ✓ HTMLLayout
- ✓ XMLLayout

=========

Maintaining Properties File:

By default the file name would be log4j.properties. This properties file stores data in the form of key, values pairs, in this file keys are fixed but values are our own. We can include all the log4j related properties into this file.

log4j.properties:-

log4j.rootLogger=DEBUG,CONSOLE,LOGFILE
log4j.appender.CONSOLE=
log4j.appender.CONSOLE.layout=
log4j.appender.CONSOLE.layout.ConversionPattern=

log4j.appender.LOGFILE=
log4j.appender.LOGFILE.File=
log4j.appender.LOGFILE.MaxFileSize=
log4j.appender.LOGFILE.layout=
log4j.appender.LOGFILE.layout.ConversionPattern=
Example:
 # Root logger option
log4j.rootLogger=DEBUG, stdout, file

# Redirect log messages to console
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n

# Redirect log messages to a log file
log4j.appender.file=org.apache.log4j.RollingFileAppender
#outputs to Tomcat home
log4j.appender.file.File=${catalina.home}/logs/myapp.log
log4j.appender.file.MaxFileSize=5MB

```
log4j.appender.file.MaxBackupIndex=10
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p
%c{1}:%L - %m%n
log4j.logger.org.springframework=OFF
=======
```

Using AOP+Stopwatch+Logging :-

```
package com.app.aop;


import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.springframework.stereotype.Component;
import org.springframework.util.StopWatch;

@Aspect
@Component
public class LoggingAspect {

	private final Log log = LogFactory.getLog(this.getClass());

	@Around("execution(* com.app.*..*.*(..))")
	public Object logTimeMethod(ProceedingJoinPoint joinPoint) throws
Throwable {

			StopWatch stopWatch = new StopWatch();
			stopWatch.start();

			Object retVal = joinPoint.proceed();
```

```
                stopWatch.stop();

                StringBuffer logMessage = new StringBuffer();

        logMessage.append(joinPoint.getTarget().getClass().getName());
                logMessage.append(".");
                logMessage.append(joinPoint.getSignature().getName());
                logMessage.append("(");
                // append args
                Object[] args = joinPoint.getArgs();
                for (int i = 0; i < args.length; i++) {
                        logMessage.append(args[i]).append(",");
                }
                if (args.length > 0) {
                        logMessage.deleteCharAt(logMessage.length() - 1);
                }

                logMessage.append(")");
                logMessage.append(" execution time: ");
                logMessage.append(stopWatch.getTotalTimeMillis());
                logMessage.append(" ms");
                log.info(logMessage.toString());
                return retVal;
        }

        //@Around("execution(* com.app.*..*.*(..))")
        public Object getLogTime(ProceedingJoinPoint jp) throws Throwable{
                StopWatch watch=new
StopWatch(jp.getTarget().getClass().getName());
                watch.start(jp.getSignature().getName());
                Object ob=jp.proceed();
                watch.stop();
```

```
            System.out.println(watch.prettyPrint());
            return ob;
        }

}
```

For AOP JARS:

http://www.mediafire.com/file/2szvk5sjxgoqkdp/Maven.rar

==========

Patterns Example notes:

A flexible layout configurable with pattern string. This code is known to have synchronization and other issues which are not present in org.apache.log4j.EnhancedPatternLayout. EnhancedPatternLayout should be used in preference to PatternLayout. EnhancedPatternLayout is distributed in the log4j extras companion.

The goal of this class is to format a Logging Event and return the results as a String. The results depend on the conversion pattern.

The conversion pattern is closely related to the conversion pattern of the printf function in C. A conversion pattern is composed of literal text and format control expressions called conversion specifiers.

You are free to insert any literal text within the conversion pattern.

Each conversion specifier starts with a percent sign (%) and is followed by optional format modifiers and a conversion character. The conversion character specifies the type of data, e.g. category, priority, date, thread name. The format modifiers control such things as field

width, padding, left and right justification. The following is a simple example.

Let the conversion pattern be "%-5p [%t]: %m%n" and assume that the log4j environment was set to use a PatternLayout. Then the statements

```
Category root = Category.getRoot();
root.debug("Message 1");
root.warn("Message 2");
```

would yield the output

```
DEBUG [main]: Message 1
WARN  [main]: Message 2
```

Note that there is no explicit separator between text and conversion specifiers. The pattern parser knows when it has reached the end of a conversion specifier when it reads a conversion character. In the example above the conversion specifier %-5p means the priority of the logging event should be left justified to a width of five characters. The recognized conversion characters are

Conversion Character	Effect
C	Used to output the category of the logging event. The

	category conversion specifier can be optionally followed by precision specifier, that is a decimal constant in brackets. If a precision specifier is given, then only the corresponding number of right most components of the category name will be printed. By default the category name is printed in full. For example, for the category name "a.b.c" the pattern %c{2} will output "b.c".
C	Used to output the fully qualified class name of the caller issuing the logging request. This conversion specifier can be optionally followed by precision specifier, that is a decimal constant in brackets. If a precision specifier is given, then only the corresponding number of right most components of the class name will be printed. By default the class name is output in fully qualified form. For example, for the class name "org.apache.xyz.SomeClass", the pattern %C{1} will output "SomeClass". WARNING Generating the caller class information is slow. Thus, use should be avoided unless execution speed is not an issue.
D	Used to output the date of the logging event. The date conversion specifier may be followed by a date format specifier enclosed between braces. For

	example, %d{HH:mm:ss,SSS} or %d{dd MMM yyyy HH:mm:ss,SSS}. If no date format specifier is given then ISO8601 format is assumed. The date format specifier admits the same syntax as the time pattern string of the SimpleDateFormat. Although part of the standard JDK, the performance of SimpleDateFormat is quite poor. For better results it is recommended to use the log4j date formatters. These can be specified using one of the strings "ABSOLUTE", "DATE" and "ISO8601" for specifying AbsoluteTimeDateFormat, DateTimeDateFormat and respectively ISO8601DateFormat. For example, %d{ISO8601} or %d{ABSOLUTE}. These dedicated date formatters perform significantly better than SimpleDateFormat.
F	Used to output the file name where the logging request was issued. WARNING Generating caller location information is extremely slow and should be avoided unless execution speed is not an issue.
L	Used to output location information of the caller which generated the logging event. The location information depends on the JVM implementation but usually consists of the fully qualified

	name of the calling method followed by the callers source the file name and line number between parentheses. The location information can be very useful. However, its generation is extremely slow and should be avoided unless execution speed is not an issue.
L	Used to output the line number from where the logging request was issued. WARNING Generating caller location information is extremely slow and should be avoided unless execution speed is not an issue.
M	Used to output the application supplied message associated with the logging event.
M	Used to output the method name where the logging request was issued. WARNING Generating caller location information is extremely slow and should be avoided unless execution speed is not an issue.
N	Outputs the platform dependent line separator character or characters. This conversion character offers practically the same performance as using non-portable line separator strings such as "\n", or "\r\n". Thus, it is the preferred way of specifying a line separator.

P	Used to output the priority of the logging event.
R	Used to output the number of milliseconds elapsed from the construction of the layout until the creation of the logging event.
T	Used to output the name of the thread that generated the logging event.
X	Used to output the NDC (nested diagnostic context) associated with the thread that generated the logging event.
X	Used to output the MDC (mapped diagnostic context) associated with the thread that generated the logging event. The X conversion character must be followed by the key for the map placed between braces, as in %X{clientNumber} where clientNumber is the key. The value in the MDC corresponding to the key will be output. See MDC class for more details.
%	The sequence %% outputs a single percent sign.

By default the relevant information is output as is. However, with the aid of format modifiers it is possible to change the minimum field width, the maximum field width and justification.

The optional format modifier is placed between the percent sign and the conversion character.

The first optional format modifier is the left justification flag which is just the minus (-) character. Then comes the optional minimum field width modifier. This is a decimal constant that represents the minimum number of characters to output. If the data item requires fewer characters, it is padded on either the left or the right until the minimum width is reached. The default is to pad on the left (right justify) but you can specify right padding with the left justification flag. The padding character is space. If the data item is larger than the minimum field width, the field is expanded to accommodate the data. The value is never truncated.

This behavior can be changed using the maximum field width modifier which is designated by a period followed by a decimal constant. If the data item is longer than the maximum field, then the extra characters are removed from the beginning of the data item and not from the end. For example, it the maximum field width is eight and the data item is ten characters long, then the first two characters of the data item are dropped. This behavior deviates from the printf function in C where truncation is done from the end.

Below are various format modifier examples for the category conversion specifier.

Format modifier	left justify	minimum width	maximum width	Comment
%20c	false	20	none	Left pad with spaces if the category name is less than 20 characters long.

%-20c	true	20	none	Right pad with spaces if the category name is less than 20 characters long.
%.30c	NA	none	30	Truncate from the beginning if the category name is longer than 30 characters.
%20.30c	false	20	30	Left pad with spaces if the category name is shorter than 20 characters. However, if category name is longer than 30 characters, then truncate from the beginning.
%-20.30c	true	20	30	Right pad with spaces if the category name is shorter than 20 characters. However, if category name is longer than 30 characters, then truncate from the beginning.

Below are some examples of conversion patterns.

%r [%t] %-5p %c %x - %m%n

This is essentially the TTCC layout.

%-6r [%15.15t] %-5p %30.30c %x - %m%n

Similar to the TTCC layout except that the relative time is right padded if less than 6 digits, thread name is right padded if less than 15 characters and truncated if longer and the category name is left padded if shorter than 30 characters and truncated if longer.

EDI

(Electronic Data Interchange)

- VDM supports Data Interchanges (without UI) can be done using A & A (Authentication-username & password) and (Authorization-Roles like Admin, User).
- Here, we are using

1. Code / String generation programming for accessToken and password.
2. JSON – JACKSON conversion, for object to JSON & JSON to object.
3. CoDec (Coding & Decoding)

It is a process of converting readable format to unreadable format (encoding) and back conversion (decoding).

Modules

1. Customer
2. Item / Parts

1. **Customer**

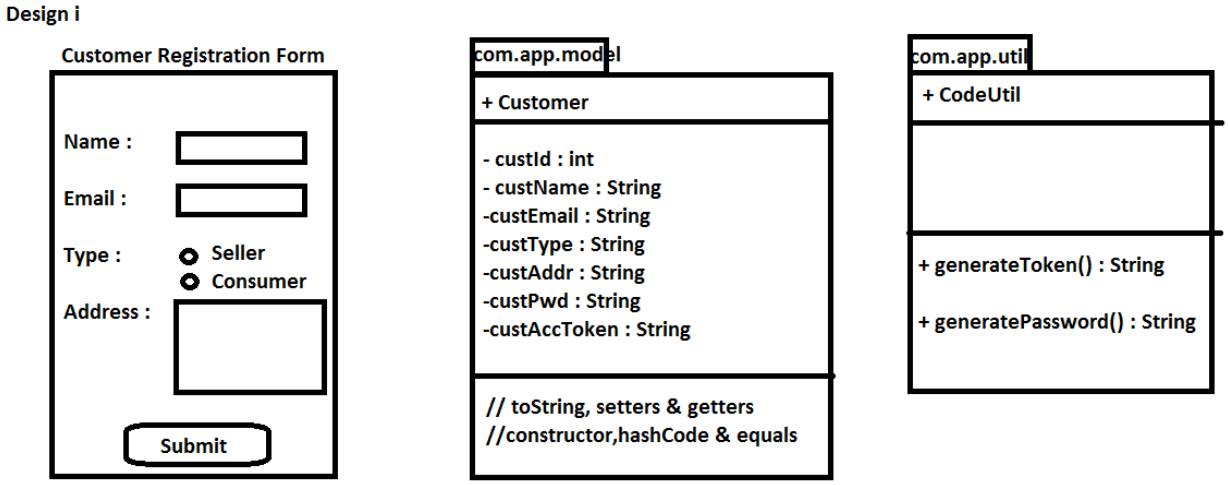
- Customer can be seller/customer.
- Seller can create/modify/remove items from VDB database.
- Consumer can only view data,which is available in VDM DB.

2. **Item / Parts**

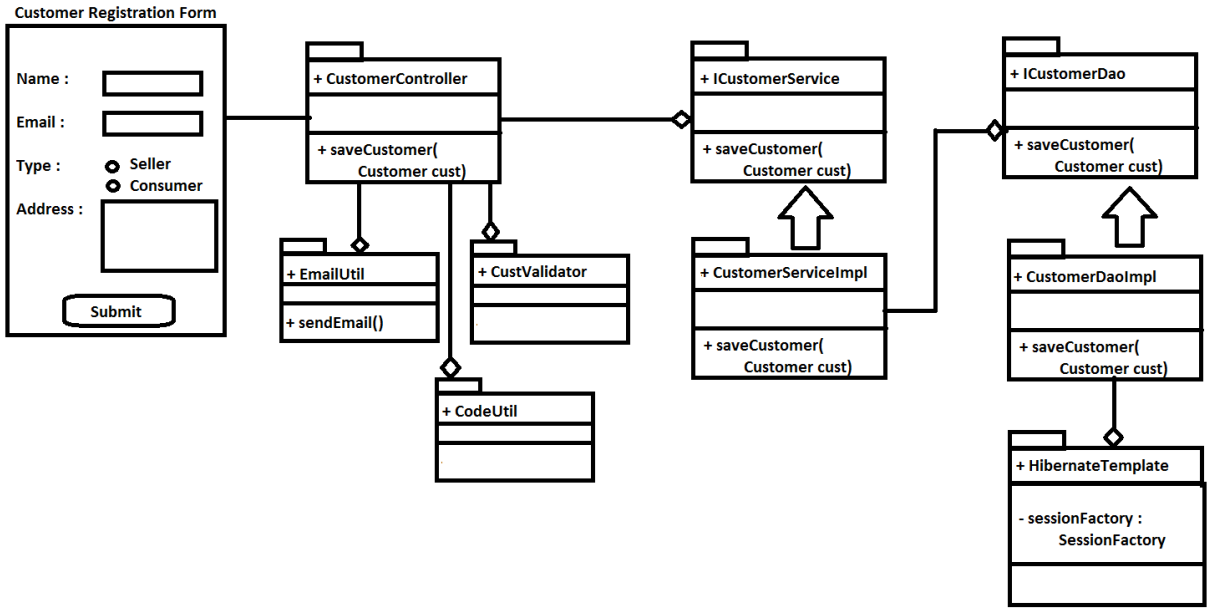
- These will be created by seller.

- Request should be made using header params like user, pwd, authToken.
- Based these details, role will be found at VDM and processed according to that.

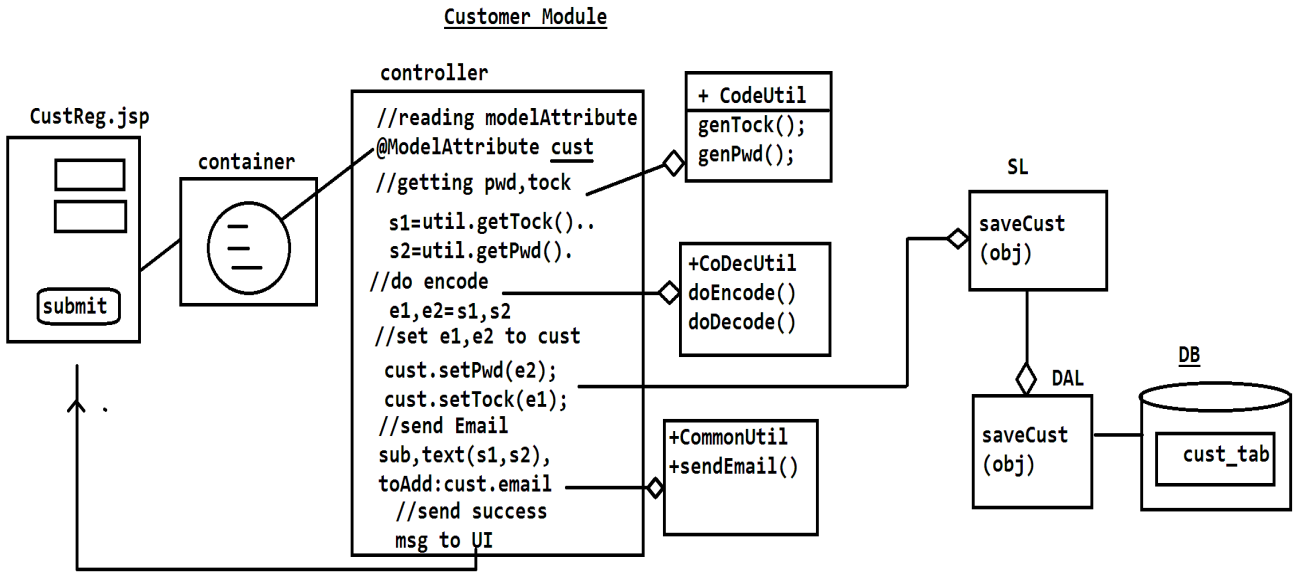
Design



Design 2



- In this module, we have to generate access token and password randomly using **Apache Commons-Codec** .



Encoding process

- Encoding is the process of converting a data which is in readable format into an unreadable data format.
- Encoding process is going to take the following process

Step 1

Convert readable String into byte format. i.e. byte[]

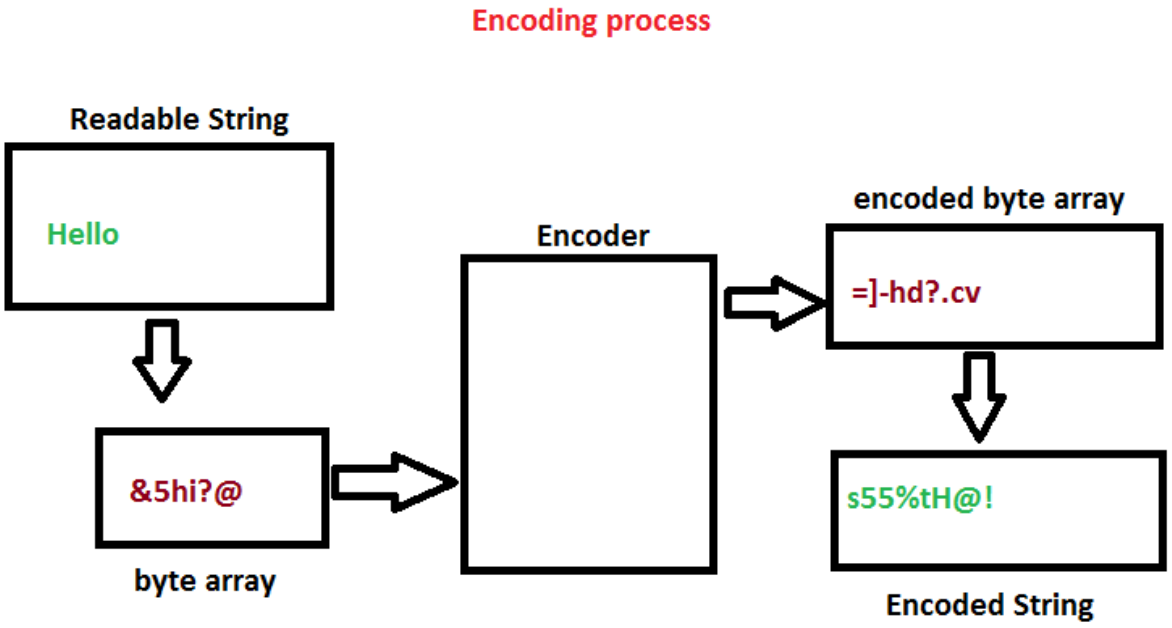
Step 2

Call a method of encoder class by passing byte[].

This method will return an encoded byte array.
i.e. encoded byte[].

Step 3

Convert the encoded byte[] into String, this is the encoded String.



Decoding process

- Decoding is the process of converting an encoded data into readable / original format.
- Decoding is having the following process

Step 1

Convert encoded String into byte format.
i.e. encoded byte[]

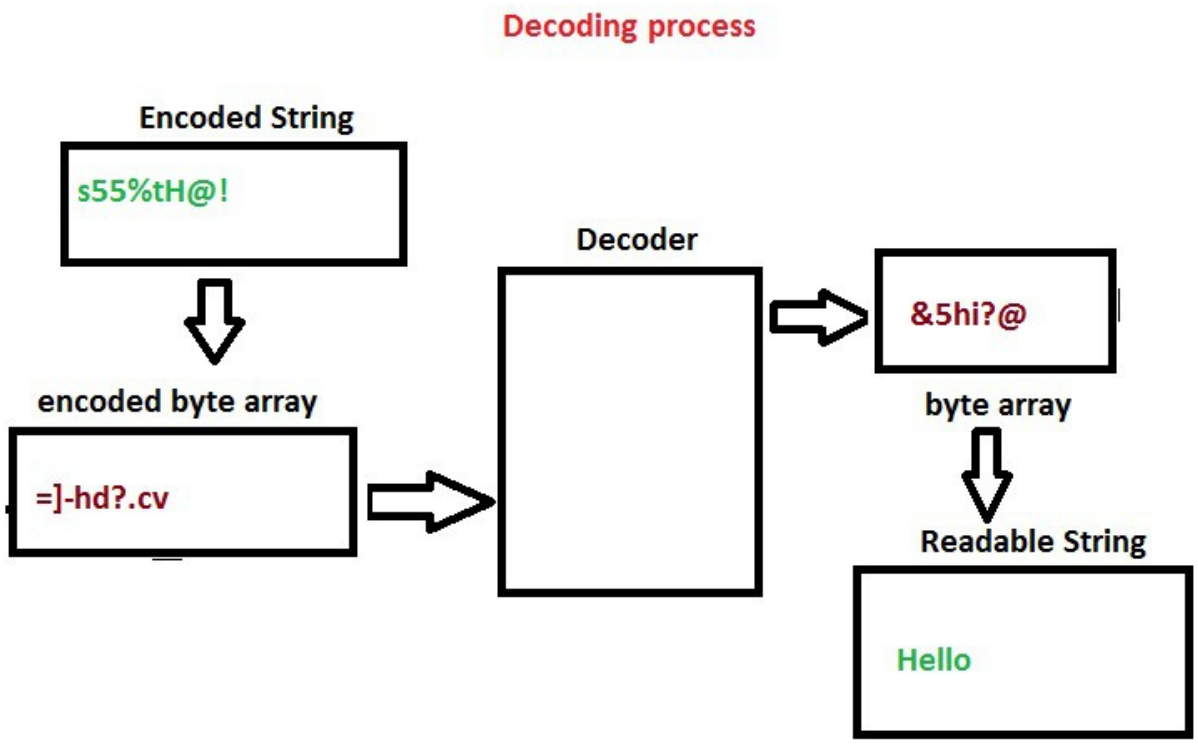
Step 2

Call a method of decoder class by passing encoded byte[].

This method will return an byte array, which is decoded byte array. i.e. decoded byte[].

Step 3

Convert the decoded byte[] into String, this is the decoded String / original String.



Coding

HTML

```
<form action="saveCust" method="post">
    Name : <input type="text" name="custName"><br>
    Email : <input type="text" name="custEmail"><br>
    Type : <select name="custType">
        <option value="Seller">Seller</option>
        <option value="Consumer">Consumer</option>
    </select><br>
Address : <textarea rows="5" cols="15" name="custAddr"></textarea><br>
<input type="submit" value="Register"> <input type="reset"
value="Reset"> <br>

    <c:if test="${id ne null}">
        <c:out value="You are successfully registered with id :"/> 
        <c:out value="${id}"/>
    </c:if>
</form>
```

Controller

```
package com.app.controller;
@Controller
public class CustomerController
{
```

```
@Autowired
private ICustomerService custServ;
@Autowired
private CodeUtil codeUtil;
@Autowired
private CodecUtil codecUtil;
@Autowired
private EmailUtil mailUtil;

@RequestMapping("/getCustomerRegPage")
public String getCustomerRegPage(ModelMap map)
{
	map.addAttribute("customer",new Customer());
	return "CustReg";
}
@RequestMapping("saveCust")
public String saveCust(@ModelAttribute("customer")Customer cust,ModelMap map)
{
	String pwd=codeUtil.generatePwd(6);
	String accToken=codeUtil.generateAccessToken(8);
	/*
```

```
* Here, encode the pwd, accToken and then set to the Customer    object.
                        * Now after setting to customer object,save Customer.
                        */
                        cust.setCustAccToken(codecUtil.doEncode(accToken));
                        cust.setCustPwd(codecUtil.doEncode(pwd));
                        int id=custServ.saveCustomer(cust);
                        if(id!=0)
                        {
String subject="Hello Mr/Ms/Mrs. "+cust.getCustName()+".This is regarding registration to VDM.";
String message="You are Successfully registered as a "+cust.getCustType()+".Your Password is: "+pwd+
                                    " and accessToken is: "+accToken ;
mailUtil.sendEmailToCustomer(cust.getCustEmail(), subject, message);
                        }
                        map.addAttribute("id",id);
                        return "CustReg";
                }
}
```

DAL

```
        package com.app.dao.impl;
@Repository
```

```
public class CustomerDaoImpl implements ICustomerDao
{
		@Autowired
		private HibernateTemplate ht;
		@Override
		public int saveCustomer(Customer cust)
		{
			return (Integer)ht.save(cust);
		}
}
```

CodeUtil

```
	package com.app.util;
@Component
public class CodeUtil
{
		private String generateUUID(int length)
		{
			UUID uuid=UUID.randomUUID();
String value=uuid.toString().replace("-", "").substring(0, length);
			return value;
		}
		public String generateAccessToken(int tokenLength)
```

```
		{
			return generateUUID(tokenLength);
		}
		public String generatePwd(int pwdLength)
		{
			return generateUUID(pwdLength);
		}
}
```

CodecUtil

```
	package com.app.util;
@Component
public class CodecUtil
{
		public String doEncode(String originalString)
		{
			/*
			 * Base64 class is used to encode and to decode the String.
* The method encodeBase64(byte[] data) is used to encode the data. To this method we must pass binary(byte) data.
			*/
byte[] encoded= Base64.encodeBase64(originalString. getBytes());
```

```
                    /*
* here encoded data in byte[] is again converted to String , because to store this encoded data in Database.
                     */
                    return new String(encoded);
            }
            public String doDecode(String encodedString)
            {
                    byte[] decoded=Base64.decodeBase64(encodedString);
                    return new String(decoded);
            }
}
```

ReST WebService for Vendor App

Step 1 Configuration

- Add Jersey jars to lib folder
- Define Front Controller in web.xml (ServletContainer in web.xml) using **/rest/***

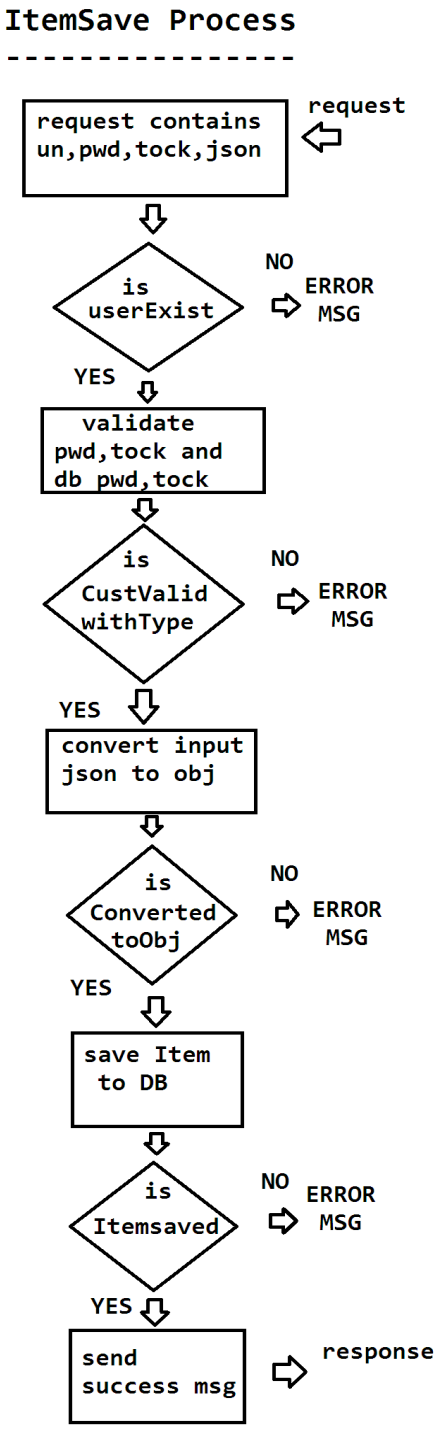
Step 2 Coding

- Define a class represents RestServiceProvider for Customer.

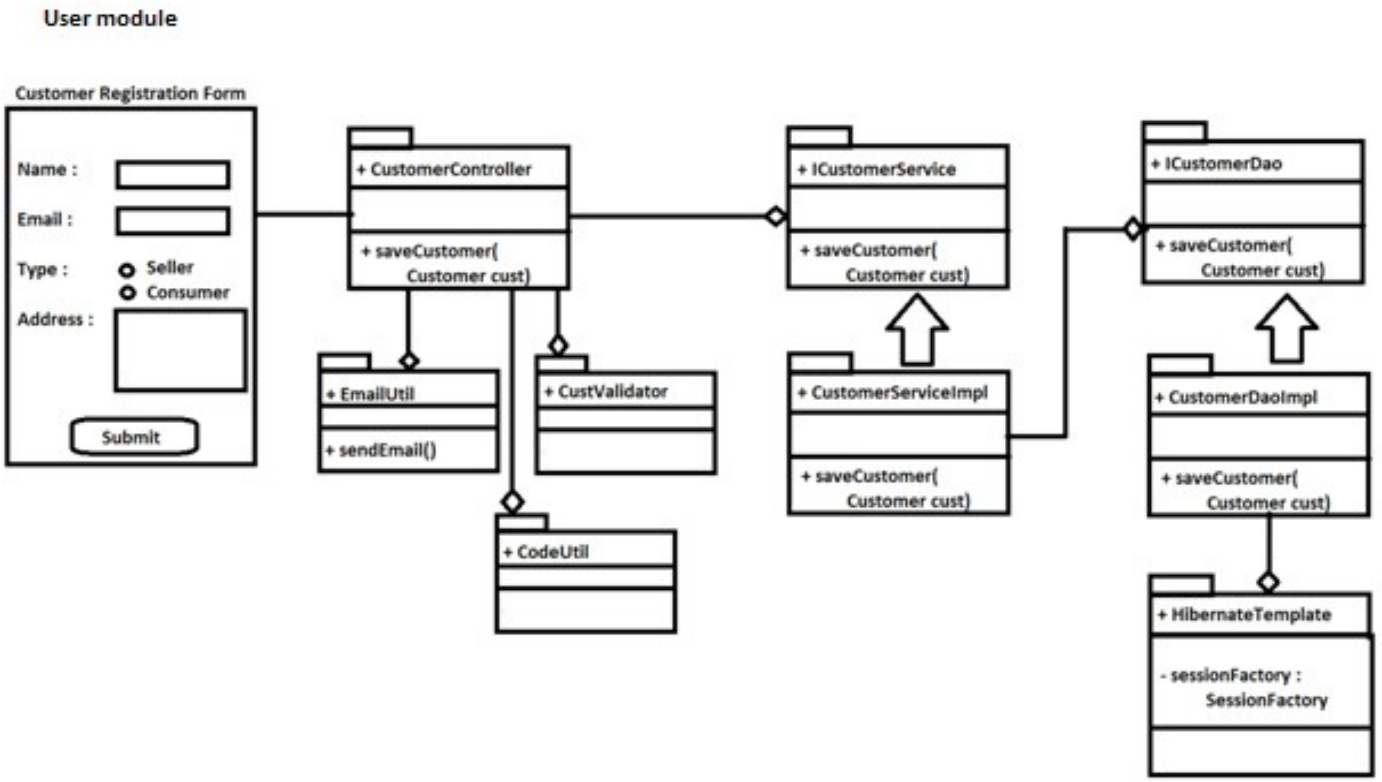
@Path(" url "), @GET, @POST,……

- Read data using Header parameters (userName, password, accessToken).
- Verify user using username, password, accessToken .
- If record exist then check customer type for operation.
- Perform operation if valid customer, then return back to client with message.

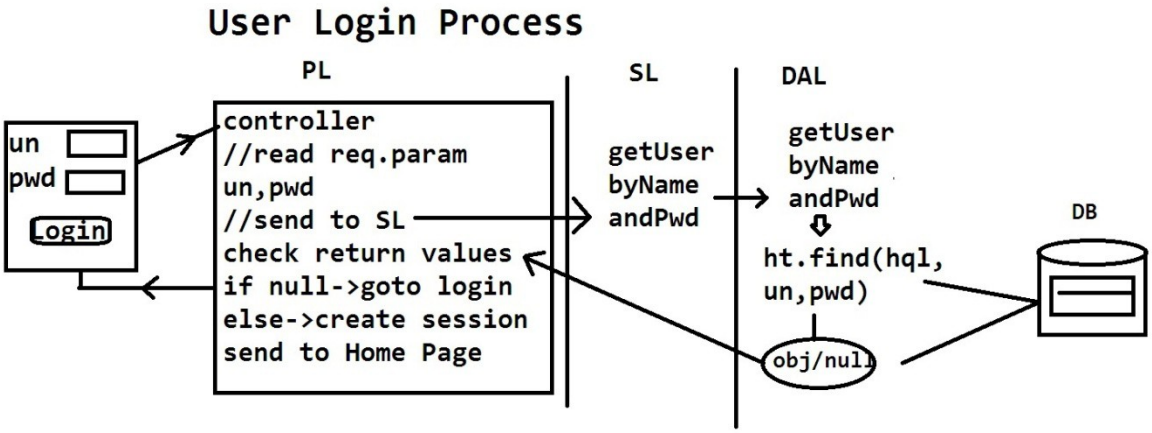
Item save process


ItemSave Process
request contains un,pwd,tock,json
request
is userExist
NO
ERROR MSG
YES
validate pwd,tock and db pwd,tock
is CustValid withType
NO
ERROR MSG
YES
convert input json to obj
is Converted toObj
NO
ERROR MSG
YES
save Item to DB
is Itemsaved
NO
ERROR MSG
YES
send success msg
response

User module


User module
Customer Registration Form
Name :
Email :
Type :
Seller
Consumer
Address :
Submit
+ CustomerController
+ saveCustomer(Customer cust)
+ EmailUtil
+ sendEmail()
+ CustValidator
+ CodeUtil
+ ICustomerService
+ saveCustomer(Customer cust)
+ CustomerServiceImpl
+ saveCustomer(Customer cust)
+ ICustomerDao
+ saveCustomer(Customer cust)
+ CustomerDaoImpl
+ saveCustomer(Customer cust)
+ HibernateTemplate
- sessionFactory : SessionFactory

User login



Filters

- A Filter is an interface. It provides request & response filtering before processing.
- It works based on Servlet URL matching.
- Filter should be configured in web.xml, using Servlet URL pattern. It can /* also to specify all request filtering.
- Filter also contains life-cycle methods

1. init()
2. doFilter()
3. destroy()

- init() and destroy() are executed only one time.
- Filter supports init parameters from web.xml

Creating security filter for session management

Step 1

Define a filter class and configure in web.xml that should filter all **DispatcherServlet** request (i.e. use filter url as /mvc/*).

Step 2

Provide init-param's which doesnot required session check.

Step 3

Read all init-param's into filter and store in List object.

Step 4

In doFilter() method, get current URI using with the request.getRequestURI() and check this with avoid url list.

- If available, then skip session check.
- Else check session and return to home page.
- Here, we can also enable cache clear and this will be executed on invalidation of session. (also disables back button).

Code

Filter

```
package com.app.filter;
public class SessionCheckFilter implements Filter
{
        private List<String> urlsList=null;
        @Override
        public void destroy()
 {
```

```
                System.out.println("Auth:Raghu");
        }
public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain) throws IOException, ServletException
 {
HttpServletRequest req=(HttpServletRequest)request;
                HttpServletResponse res=(HttpServletResponse)response;


                //disable back button
res.setHeader("Cache-Control", "no-cache, no-store, must-revalidate");
                res.setHeader("Pragma", "no-cache");
                res.setDateHeader("Expires", 0);

                //
                String uri=req.getRequestURI();
                System.out.println("Current Path:"+uri);
                boolean allowedRequest=false;

 try
 {
                        String url=req.getRequestURI();
                        System.out.println(req.getRequestURI());
                        if(urlsList.contains(url))
 {
                                allowedRequest = true;
                        }
                        if (!allowedRequest)
 {
                                HttpSession session = req.getSession(false);
if (null == session || session.getAttribute("userName") == null)
 {
                                        res.sendRedirect(req.getContextPath());
                                }
                        }
                }
 catch (Exception e)
```

```
{
                    e.printStackTrace();
                }

                chain.doFilter(request, response);
            }
            public void init(FilterConfig fConfig) throws ServletException
{
                String urls=fConfig.getInitParameter("avoid-urls");
                StringTokenizer tock=new StringTokenizer(urls, ",");
                urlsList=new ArrayList<String>();
                while(tock.hasMoreTokens())
{
                    urlsList.add(tock.nextToken());
                }
                System.out.println(urlsList);
            }
}
```

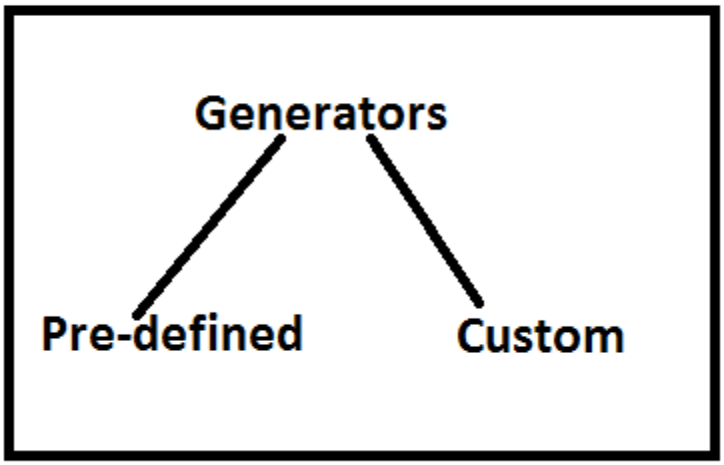
web.xml

```
    <filter>
        <display-name>SessionCheckFilter</display-name>
        <filter-name> SessionCheckFilter</filter-name>
        <filter-class>com.app.filter.SessionCheckFilter</filter-class>
        <init-param>
            <param-name>avoid-urls</param-name>
            <param-value>
/SathyaVDM/,/SathyaVDM,/SathyaVDM/mvc/login,/SathyaVDM/mvc/loginAdmin,
/SathyaVDM/mvc/logout </param-value>
        </init-param>
     </filter>
     <filter-mapping>
        <filter-name>SessionCheckFilter</filter-name>
        <url-pattern>/*</url-pattern>
     </filter-mapping>
```

Generators

Primary key generation

- Hibernate supports generating a primary key value at save operation.
- Every Generator is a class in case of Hibernate.
- These are 2 types



- Incase of Oracle database, most recommended generator is **SEQUENCE** .
- By default, Hibernate provides **HIBERNATE_SEQUENCE** .
- On save operation, it will generate primary key value as select hibernate_sequence from dual ;

Syntax to create sequence at DB

create sequence <sequence-name> start <value> step <value>

Ex

```
@Id
@GeneratedValue(generator="abc")
@GenericGenerator(name="abc" , strategy= "com.app.model.MyGeneratorSample")
private String venId;
```

Eclipse Shortcuts

1. Manage Files and Projects	
Ctrl+N	Create new project using the Wizard
Ctrl+Alt+N	Create new project, file, class, etc.
Alt+F, then .	Open project, file, etc.
Ctrl+Shift+R	Open Resource (file, folder or project)
Alt+Enter	Show and access file properties
Ctrl+S	Save current file
Ctrl+Shift+S	Save all files
Ctrl+W	Close current file
Ctrl+Shift+W	Close all files
F5	Refresh content of selected element with local file system

2. Editor Window

Focus/ cursor must be in Editor Window for these to work.	
F12	Jump to Editor Window
Ctrl+Page Down/Ctrl+Page Up	Switch to next editor / switch to previous editor
Ctrl+M	Maximize or un-maximize current Editor Window (also works for other Windows)
Ctrl+E	Show list of open Editors. Use arrow keys and enter to switch
Ctrl+F6/Ctrl+Shift+F6	Show list of open Editors. Similar to ctrl+e but switches immediately upon release of ctrl
Alt+Arrow Left/Alt+Arrow Right	Go to previous / go to next Editor Window
Alt+-	Open Editor Window Option menu
Ctrl+F10	Show view menu (features available on left vertical bar: bookmarks, line numbers, …)
Ctrl+F10, then n	Show or hide line numbers
Ctrl+Shift+Q	Show or hide the diff column on the left (indicates changes since last save)

3. Navigate in Editor	
Home/End	Jump to beginning / jump to end of indention. Press home twice to jump beginning of line
Ctrl+Home/End	Jump to beginning / jump to end of source
Ctrl+Arrow Right/Arrow Left	Jump one word to the left / one word to the right
Ctrl+Shift+Arrow Down/Arrow Up	Jump to previous / jump to next method
Ctrl+L	Jump to Line Number. To hide/show line numbers, press ctrl+F10 and select 'Show Line Numbers'
Ctrl+Q	Jump to last location edited
Ctrl+./Ctrl+,	Jump to next / jump to previous compiler syntax warning or error
Ctrl+Shift+P	With a bracket selected: jump to the matching closing or opening bracket
Ctrl+[+]/Ctrl+- on numeric keyboard	Collapse / Expand current method or class

Ctrl+[/]/Ctrl+* on numeric keyboard	Collapse / Expand all methods or classes
Ctrl+Arrow Down/Ctrl+Arrow Up	Scroll Editor without changing cursor position
Alt+Page Up/Alt+Page Down	Next Sub-Tab / Previous Sub-Tab

4. Select Text	
Shift+Arrow Right/Arrow Left	Expand selection by one character to the left / to the right
Ctrl+Shift+Arrow Right/Arrow Left	Expand selection to next / previous word
Shift+Arrow Down/Arrow Up	Expand selection by one line down / one line up
Shift+End/Home	Expand selection to end / to beginning of line
Ctrl+A	Select all
Alt+Shift+Arrow Up	Expand selection to current element (e.g. current one-line expression or content within brackets)
Alt+Shift+Arrow Left/Arrow Right	Expand selection to next / previous element
Alt+Shift+Arrow Down	Reduce previously expanded selection by one step

5. Edit Text	
Ctrl+C/Ctrl+X/Ctrl+V	Cut, copy and paste
Ctrl+Z	Undo last action
Ctrl+Y	Redo last (undone) action
Ctrl+D	Delete Line
Alt+Arrow Up/Arrow Down	Move current line or selection up or down
Ctrl+Alt+Arrow Up/Ctrl+Alt+Arrow Down/	Duplicate current line or selection up or down

Ctrl+Delete	Delete next word
Ctrl+Backspace	Delete previous word
Shift+Enter	Enter line below current line
Shift+Ctrl+Enter	Enter line above current line
Insert	Switch between insert and overwrite mode
Shift+Ctrl+Y	Change selection to all lower case
Shift+Ctrl+X	Change selection to all upper case

6. Search and Replace	
Ctrl+F	Open find and replace dialog
Ctrl+K/Ctrl+Shift+K	Find previous / find next occurrence of search term (close find window first)
Ctrl+H	Search Workspace (Java Search, Task Search, and File Search)
Ctrl+J/Ctrl+Shift+J	Incremental search forward / backwards. Type search term after there is now search window
Ctrl+Shift+O	Open a resource search dialog to find any class

7. Indentions and Comments	
Tab/Shift+Tab	Increase / decrease indent of selected text
Ctrl+I	Correct indention of selected text or of current line
Ctrl+Shift+F	Autoformat all code in Editor using code formatter
Ctrl+/	Comment / uncomment line or selection (adds '//')
Ctrl+Shift+/	Add Block Comment around selection (adds '/... */')
Ctrl+Shift+\	Remove Block Comment
Alt+Shift+J	Add Element Comment (adds '/** ... */')

8. Editing Source Code	
Ctrl+Space	Opens Content Assist (e.g. show available methods or field names)
Ctrl+1	Open Quick Fix and Quick Assist
Alt+/	Propose word completion (after typing at least one letter). Repeatedly press alt+/ until reaching correct name
Ctrl+Shift+Insert	Deactivate or activate Smart Insert Mode (automatic indention, auto brackets, etc.)

9. Code Information	
Ctrl+O	Show code outline / structure
F2	Open class, method, or variable information (tooltip text)
F3	Open Declaration: Jump to Declaration of selected class, method
F4	Open Type Hierarchy window for selected item
Ctrl+T	Show / open Quick Type Hierarchy for selected item
Ctrl+Shift+T	Open Type in Hierarchy
Ctrl+Alt+H	Open Call Hierarchy
Ctrl+Shift+U	Find occurrences of expression in current file
Ctrl+move over method	Open Declaration or Implementation

10. Refactoring	
Alt+Shift+R	Rename selected element and all references
Alt+Shift+V	Move selected element to other class or file (With complete method or
Alt+Shift+C	Change method signature (with method name selected)
Alt+Shift+M	Extract selection to method
Alt+Shift+L	Extract local variable: Create and assigns a variable from a selected expr
Alt+Shift+I	Inline selected local variables, methods, or constants if possible (replace

	declarations/ assignment and puts it directly into the statements)

11. Run and Debug	
Ctrl+F11	Save and launch application (run)
F11	Debug
F5	Step Into function
F6	Next step (line by line)
F7	Step out
F8	Skip to next Breakpoint

12. The Rest	
Ctrl+F7/Ctrl+Shift+F7	Switch forward / backward between views (panels). Useful for switching bac Explorer and Editor.
Ctrl+F8/Ctrl+Shift+F8	Switch forward / backward between perspectives
Ctrl+P	Print
F1	Open Eclipse Help
Shift+F10	Show Context Menu right click with mouse