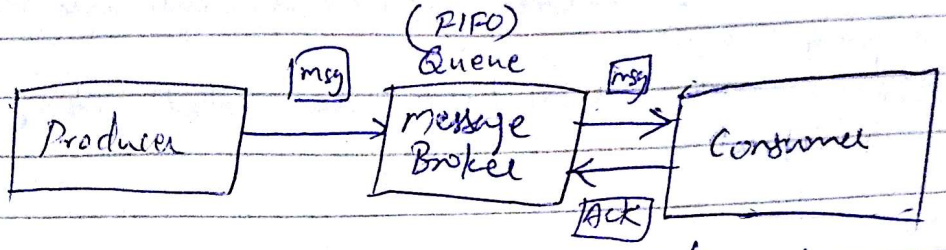
S/w - Message Broker
Active MQ

Tool - Activemq Broker

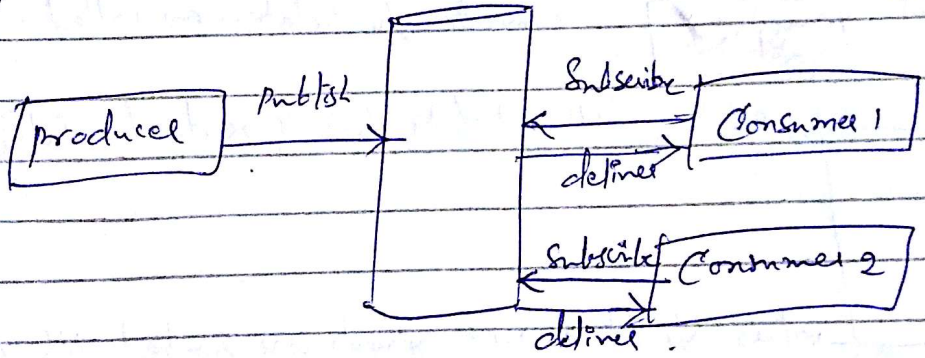
JMS

① Point-to-point



once Ack is received / timeout msg is removed from Queue.

② publish / subscribe — TOPIC (FIFO) — same as Queue except that there are multiple Consumers



(i) synchronous

using start() and receive() methods
↳ waits until the message is received

using start() and receive(1) methods — timeout
↳ waits until timeout.

(ii) asynchronous

using MessageListener — implement MessageListener interface and override onMessage() method.

For subscribers who unsubscribed and wants to subscribe later and still want to receive the missed messages use durable subscribers (using clientID).
— client id works as a session id to identify the subscriber later.
For example, if subscriber disconnects and gets connected later all the missed messages are delivered. (Cisco ISE uses this for replication)

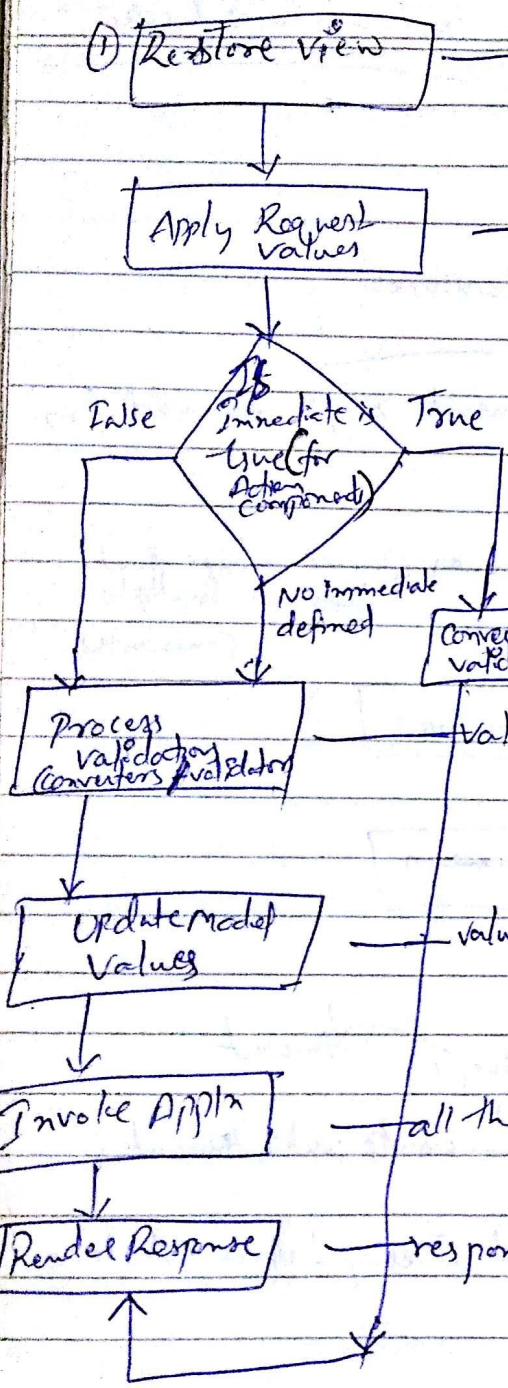
RAP UIR - if immediate = "false"

RAR - if immediate = "true"

JSF

Lifecycle phases

<f:view> components <H:view>



1. Restore View — view object with all components is created and rendered to the client.
   - Converters, validators, Renderers are restored.
2. Apply Request values — user input values are applied on the components.
3. If immediate is true (for Action component) — Action events called.
4. Conversion & validations — conversions/validations are called.
5. Process validations — validations will not take place if rendered = false
6. Update model values — values stored in UI components are synced with model object/ BackingBeans.
7. Invoke Applns — all the action listeners/actionevents will be called that contain appln specific logic
8. Render Response — response rendered to the client.

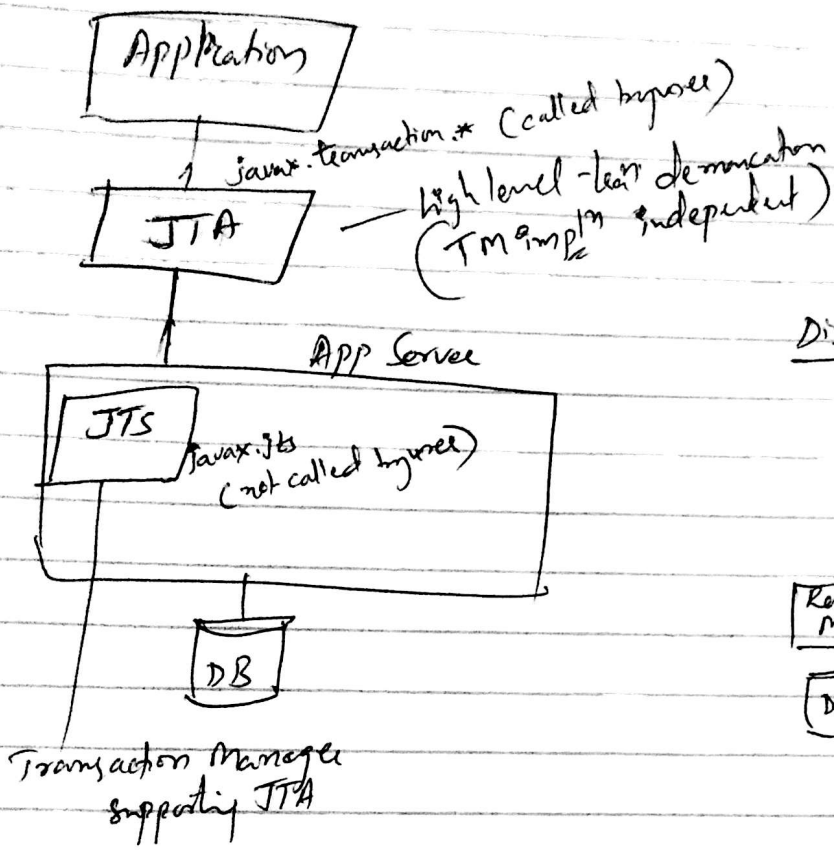
<t:saveState> — to persist beans and values longer than request scope but shorter than session scope.

immediate → when we don't want to validate entire page (ex, list states when country is selected)

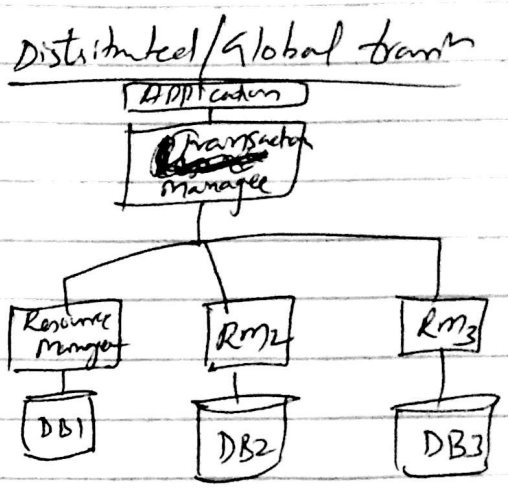
PhaseListener → to run some custom code before/after a particular phase. It is fired before/after every phase.

rendered → attribute used to conditionally render a particular component. If the condition is true then rendered if false then not.

JTA



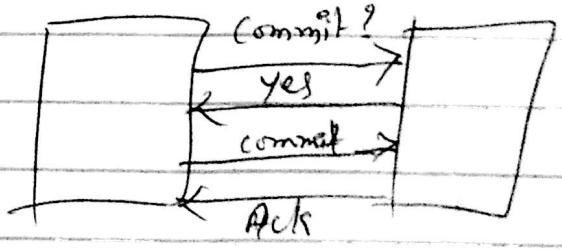
local tran" - within same db
Global/Distributed/JTA/XA tran" - spans multiple db's in same/diff locations



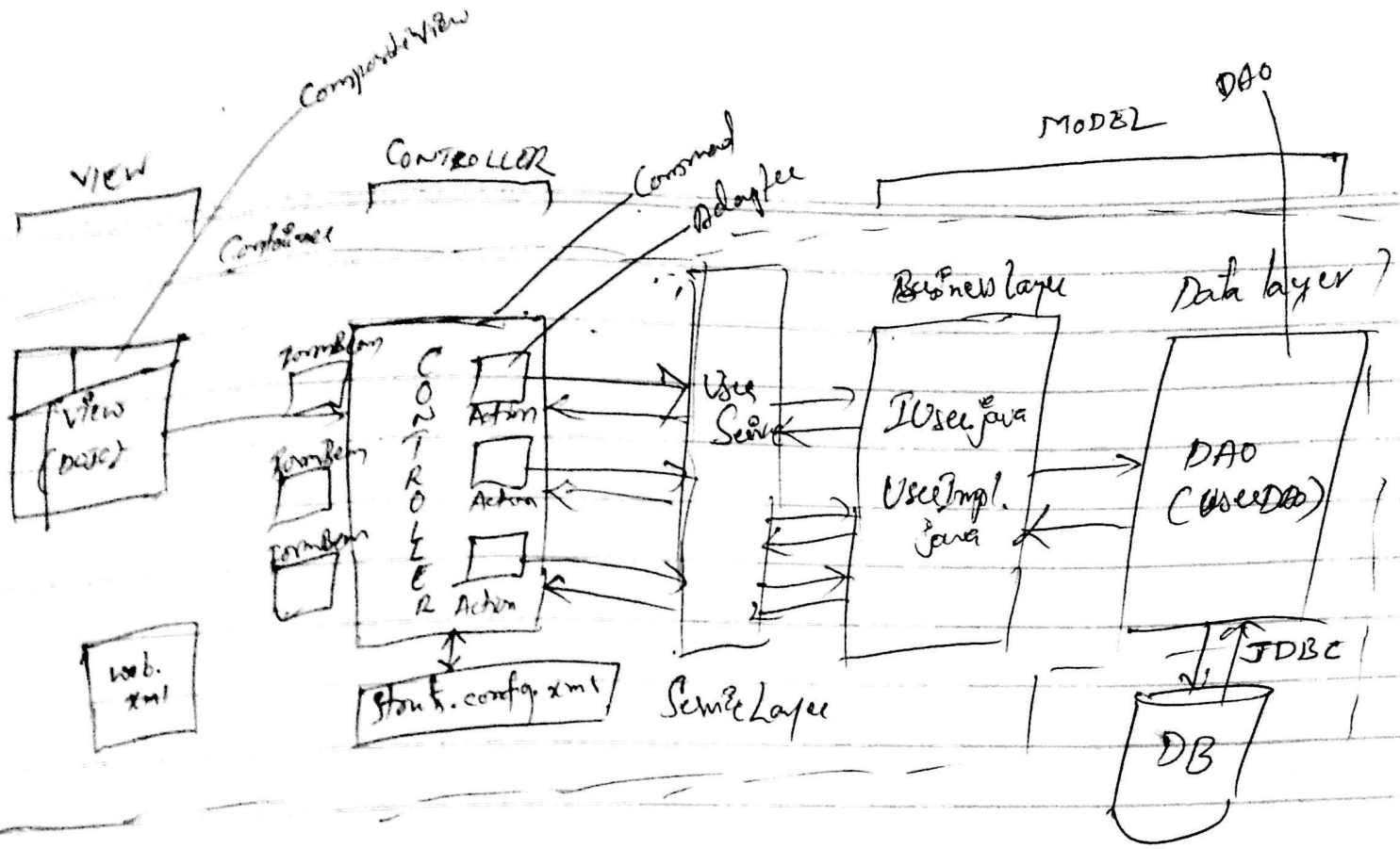
XA - Xtended Architecture - specifies how a TM will combine a tran" with diff databases in an atomic tran" and execute with 2 phase commit protocol.

2PC - is a distributed algorithm that coordinates with all participants to either commit/rollback.

phases - Commit request - voting
commit phase - decide

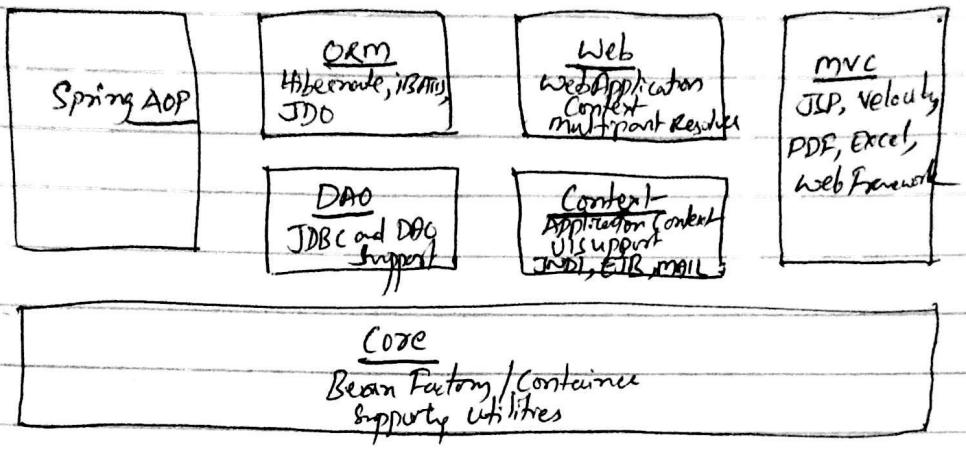


3pc - unlike 2PC is non-blocky. It places an upper bound on time taken to commit. If expires then locks are released.



DESIGN PATTERNS IN STRUTS

Spring Modules



(1.2)

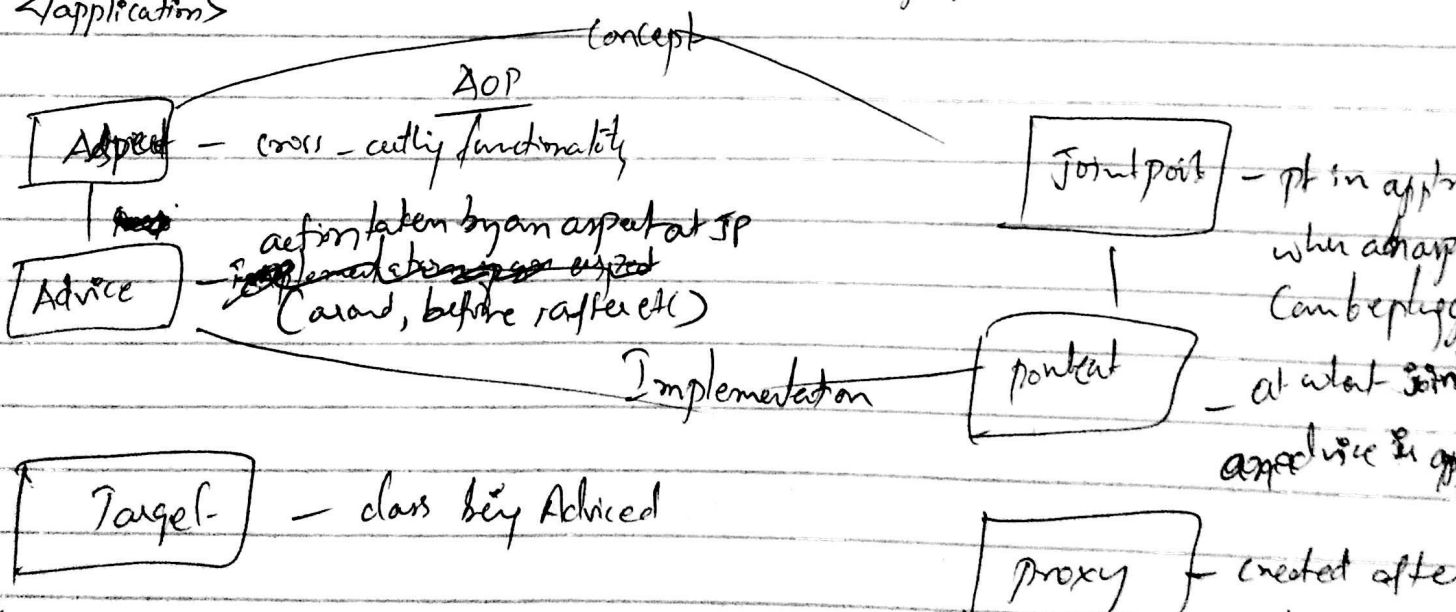
JSF & Spring :-

faces-config.xml :-

① Add DelegatingVariableResolver

② web.xml
Add ContextLoaderListener + FacesServlet

```
<application>
    <variable-resolver> org.springframework.web.jsf.DelegatingVariableResolver </variable-resolver>
</application>
```



JSF 2.0 + Spring

(1) Add SpringBeanFacesELResolver to faces-config.xml

```
<application>
    <el-resolver> org.springframework.web.jsf.el.SpringBeanFacesELResolver </el-resolver>
</application>
```

(2) web.xml

Add ContextLoaderListener + FacesServlet ~~DispatcherServlet~~

(javax.faces.webapp.FacesServlet)

~~Dispatcher~~ RequestContextListener / RequestContextFilter.

(X) For scoping of beans at request / session and global levels while using JSF / Struts use additional config in web.xml (If Servlet 2.4 + container)

```
<listener>
    <listener-class> org.springframework.web.context.request.RequestContextListener
    </listener-class>
</listener>
```

if using 2.3 Servlet container use filter impl:

```
<filter>
    <filter-name> requestContextFilter </filter-name>
    <filter-class> org.springframework.web.filter.RequestContextFilter </filter-class>
</filter>
```

NOTE: (1) These extra configs are not reqd if using only 'singleton' & 'prototype' scopes

(2) If using Spring MVC ie, DispatcherServlet / DispatcherPortlet then No extra configs are reqd.

Listener / Servlet / Filter	– Scopes
ContextLoaderListener	– singleton + prototype
RequestContextListener / RequestContextFilter	– request + session + global
DispatcherServlet / DispatcherPortlet	– all scopes (can be used)

Struts + Spring

① Add ContextLoaderPlugIn to struts-config.xml

```
<plug-in className = "org.springframework.web.struts.ContextLoaderPlugIn">
    <set-property property = "contextConfigLocation" value = "/WEB-INF/classes/
                                                        SpringBeans.xml" />
</plug-in>
```

② In Action class extend "ActionSupport" of spring and get Spring's beans via getWebApplicationContext() method.

```
CustomerBO customerBO = (CustomerBO) getWebApplicationContext().getBean("customerBO");
```

Programmatic Transaction Handling

① Create TransactionDefinition & TransactionStatus in StudentJDBCTemplate.java

```
TransactionDefinition def = new DefaultTransactionDefinition();
TransactionStatus status = transactionManager.getTransaction(def);
try {
    String SQL = "insert into Student(name, age) values (?,?)";
    jdbcTemplate.update(SQL, name, age);
    transactionManager.commit(status);
} catch (Exception e) {
    transactionManager.rollback(status);
}
```

(redAll)

② Define transactionManager in applicationContext.xml

```
<bean id = "transactionManager" class = "org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name = "dataSource" ref = "dataSource" />
</bean>

<bean id = "dataSource" class = "org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name = "driverClassName" value = "" />
                                = "url"
                                = "username"
                                = "password"
</bean>
```

```
<bean id="studentjdbcTemplate" class="com.pksumna.StudentJDBCTemplate">
    <property name="dataSource" ref="dataSource" />
    <property name="transactionManager" ref="transactionManager" />
</bean>
```

① Cont'd. StudentJDBCTemplate.java

TM
```
public void setTransactionManager(PlatformTransactionManager transactionManager) {
    this.transactionManager = transactionManager;
}
```

DS
```
public void setDataSource(DataSource dataSource) {
    this.dataSource = dataSource;
    this.jdbcTemplate = new JdbcTemplate(dataSource);
}
```

Declarative Tran n mgt :-

① No transaction manager in JdbcTemplate
② DataSourceTransaction" def same as above
③ DriverManagerDataSource def same as above
④ create Advice and AOP Config (pointcut + advisor)

```
<tx:advice id="txAdvice" transaction-manager="transactionManager">
    <tx:attributes>
        <tx:method name="create" />
    </tx:attributes>
</tx:advice>

<aop:config>
    <aop:pointcut id="createOperation" expression="execution(* com.pksumna.
        StudentJDBCTemplate.create(..))" />
```

```
<aop:advisor advice-ref="txAdvice" pointcut-ref="createOperation" />
</aop:config>
```

Spring + Hibernate (using Annotations)

① Create Entity

```
public class EmployeeEntity implements Serializable {
    @Id
    @GeneratedValue
    @Column(name="ID")
    private Integer id;
    @Column(name="NAME")
    private String name;
}
```

② Create applicationContext.xml with dataSource + sessionFactory + transactionManager.

```
<bean id="dataSource"
    class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close"
    p:driverClassName="${jdbc.driverClassName}" p:url="${jdbc.url}"
    p:username="${jdbc.username}" p:password="${jdbc.password}"></bean>

<bean id="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
    <property name="dataSource" ref="dataSource"></property>
    <property name="configLocation">
        <value>classpath:hibernate.cfg.xml</value>
    </property>
    <property name="configurationClass">
        <value>org.hibernate.cfg.AnnotationConfiguration</value>
    </property>
```

```
<property name = "hibernateProperties">
    <props>
        <prop key = "hibernate.dialect">${jdbc.dialect}</prop>
        <prop key = "hibernate.show_sql">true</prop>
    </props>
</property>
</bean>

<tx:annotation-driven />
<bean id = "transactionManager" class = "org.springframework.orm.hibernate3.
                                        HibernateTransactionManager">
    <property name = "sessionFactory" ref = "sessionFactory"></property>
</bean>
```

(3) hibernate.cfg.xml

```
<hibernate-configuration>
    <session-factory>
        <mapping class = "com.pkvarma.EmployeeEntity"></mapping>
    </session-factory>
</hibernate-configuration>
```

(4) use AnnotationConfiguration to get sessionFactory

```
return new AnnotationConfiguration().configure().buildSessionFactory();
```

Spring + Hibernate (without Annotations)

① create Entity as before without annotations

② create applicationContext.xml same as before except for the change in the below config:

```
<bean id="sessionFactory" class="org.springframework.orm.hibernate3.
                                    LocalSessionFactoryBean">
    <property name="dataSource" ref="dataSource"></property>

    <property name="hibernateProperties">
        <prop key="...">...</prop>   same as before
    </property>

    <property name="mappingResources">
        <list>
            <value>Employee.hbm.xml</value>
        </list>
    </property>
</bean>
```

③ Same as befor except:

```
<mapping resource="Employee.hbm.xml"></mapping>
```

④ Employee.hbm.xml :

```
<hibernate-mapping>
    <class name="com.pksusma.EmployeeEntity" table="employee" catalog="pksusma">
        <id name="id" type="java.lang.Integer">
            <column name="ID" />
            <generator class="identity" />
        </id>
        <property name="name" type="string">
            <column name="NAME" length="10" not-null="true" unique="true" />
        </property>
    </class>
</hibernate-mapping>
```

④ use Configuration class to get SessionFactory:

return new Configuration().configure().buildSessionFactory();

Spring WS / Apache CXF / Apache Axis2:

Apache CXF: supports several standards like SOAP, WS-I Basic profile, WSDL, WS-Addressing, WS-Policy, WS-ReliableMessaging, WS-Security, WS-Security Policy and WS-Secure Conversation.

- offers both contract-last (start with java) and Contract-first (start with WSDL) approaches.
- CXF implements JAX-WS and JAX-RS.

Spring WS:

- offers only contract-first, start with an XSD schema.
- supports SOAP, WS-Security, WS-Addressing (transport-neutral security mechanism)

so Spring WS is a minimal web services framework

Apache Axis2:

- offers both contract-first and Contract-last approaches.
- supports WS-Atomic Transaction, WS-Business Activity, WS-Coordination, WS-Eventing, WS-Transfer.
- not fully compliant for JAX-WS and JAX-RS
- more code required/generated.

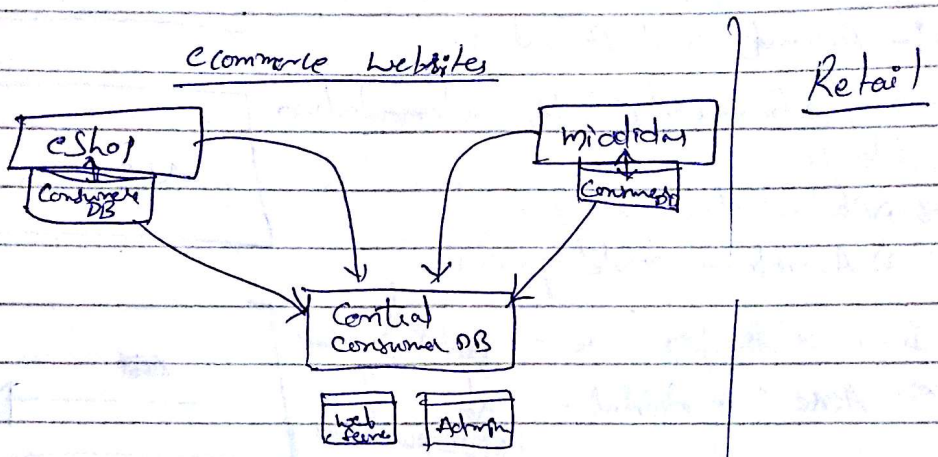
Functional

Domain Flow

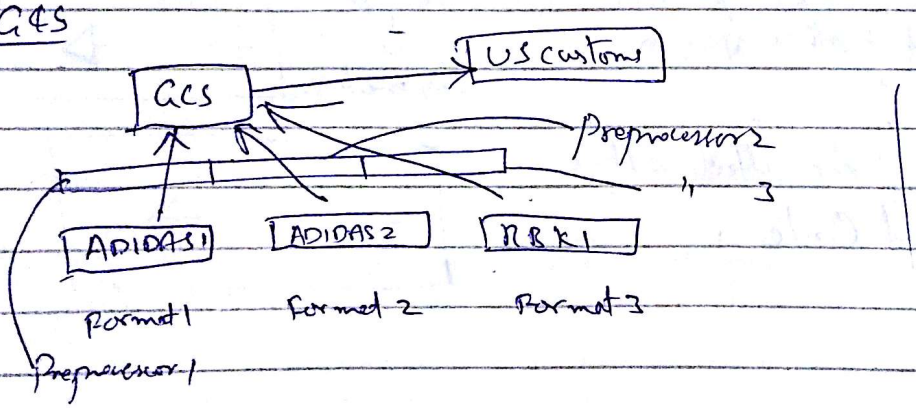
AGILE Methodology

Individual scrums

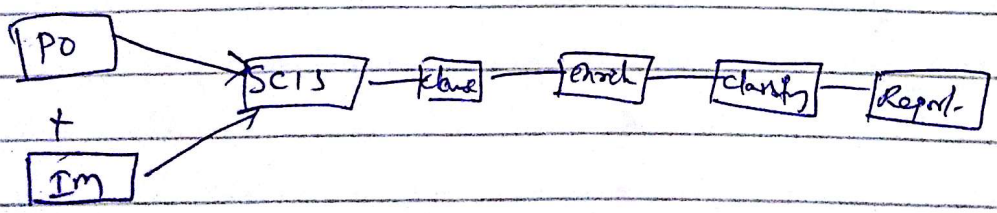
TCCP

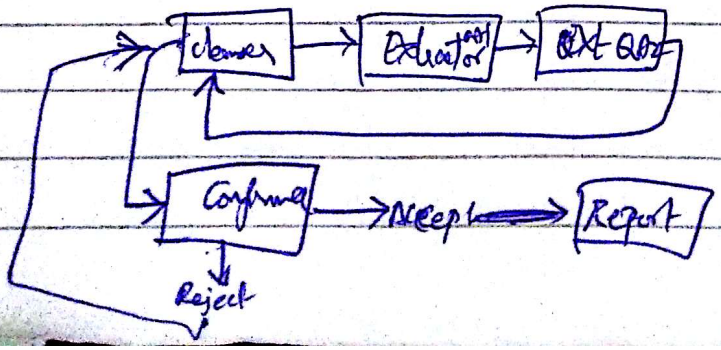


GCS

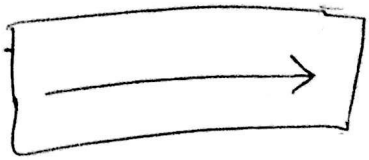


SCIS

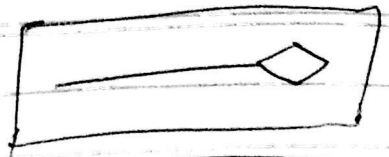




Association — a relationship (one-to-one, one-to-many etc)



Aggregation — "has-a" rel^n ship



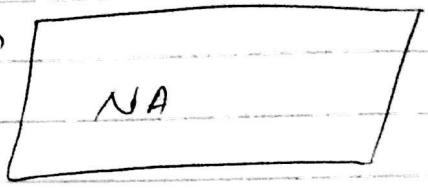
Composition — same as aggregation except that the contained object cannot exist without container

ex:- Account and AccountNo.

Abstraction — specifying a framework & hiding implementation details.

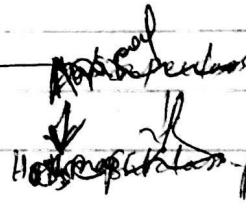
ex:- 1) Abstract class / interface

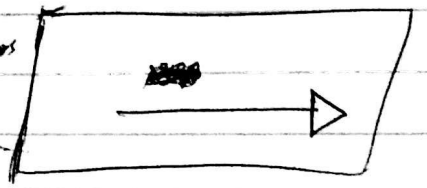
2) A wireframe model of a car.



Generalization — "is-a" relationship

ex:- Horse is a Animal.



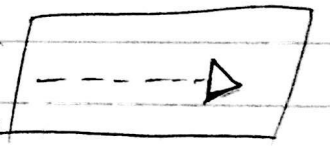


Realization — rel^n ship in which one implements (realizes) the behaviour that other specifies

Interface

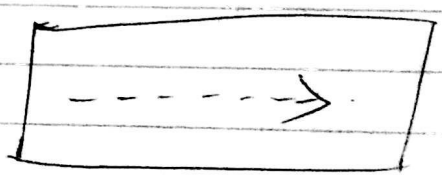
↓

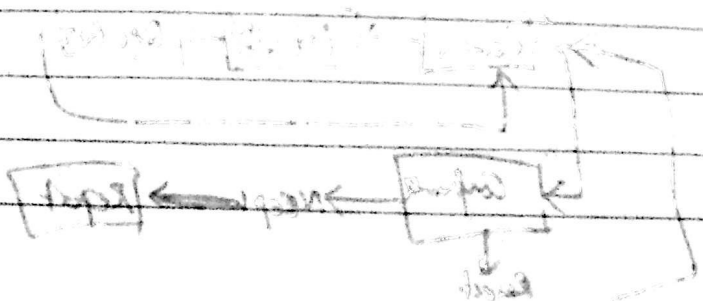
Implementation



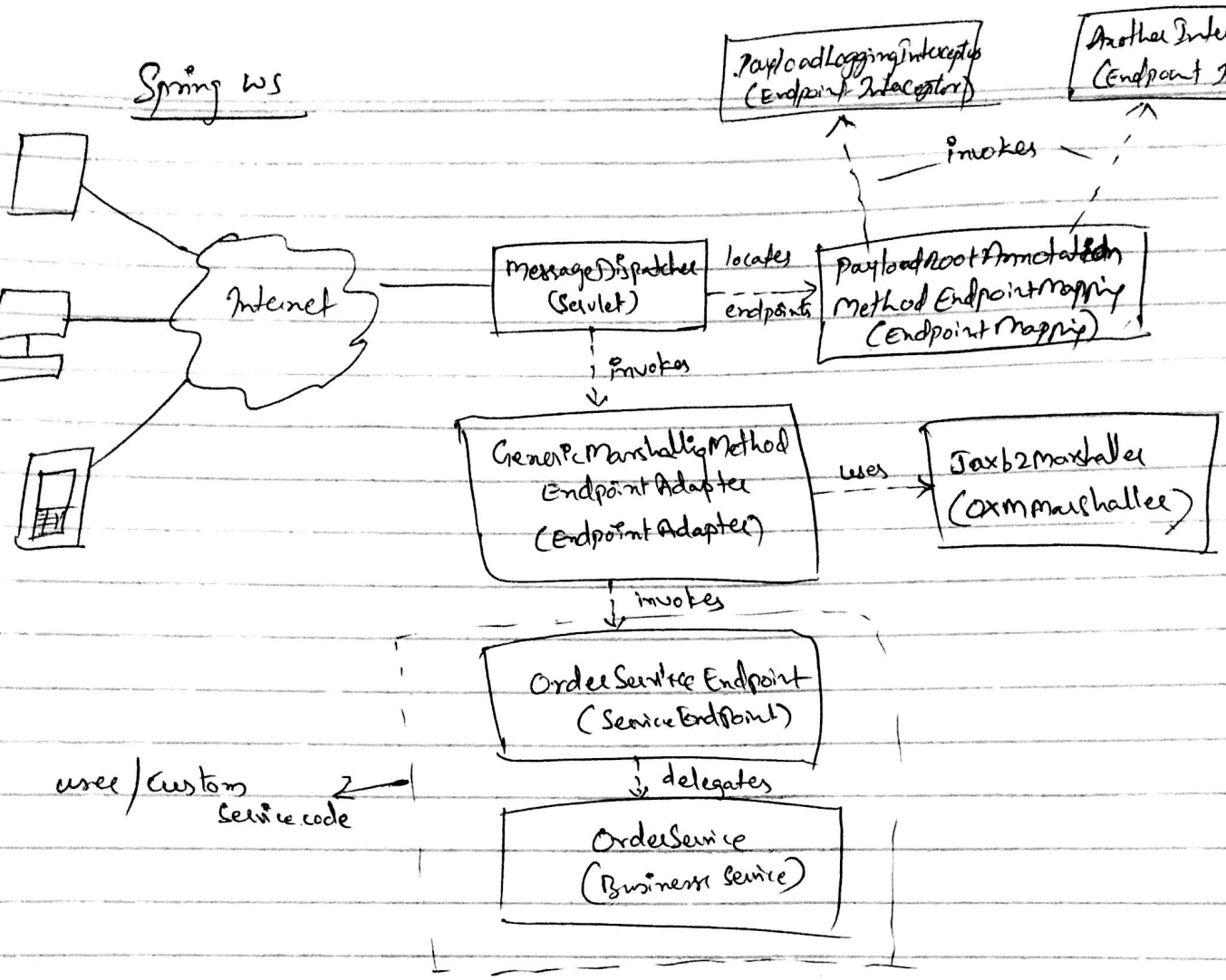
Dependency — change in structure of one affects other

ex:- Shape and Circle.





Spring WS



NOTE:- To easily remember above diagram, remember only the ones in braces.

axis 2

(1) Service.xml . Copy this to META-INF dir. and create a jar as:

jar cvf SimpleService.aar

```
<ServiceGroup>
    <service name="AxisSample" targetNamespace="http://url/samples/">
        <description> Sample service </description>
        <parameter name="ServiceClass" locked="false"> com.pkusma.AxisSample
                                                        </parameter>
        <operation name="getSampleId">
            <messageReceiver class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
        </operation>
        <operation name="getSampleName">
            <messageReceiver class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
        </operation>
    </service>
</ServiceGroup>
```

(2) web.xml .

```
<servlet>
    <servlet-name> AxisServlet </servlet-name>
    <display-name> Apache-Axis Servlet </display-name>
    <servlet-class> org.apache.axis2.transport.http.AxisServlet </servlet-class>
    <load-on-startup> 1 </load-on-startup>
</servlet>
```

(1) (2) (3)

apache CXF - JAX-WS

(1) web.xml.

```
<servlet-name> CXFServlet </servlet-name>
<servlet-class> org.apache.cxf.transport.servlet.CXFServlet </servlet-class>
```

(1) (2) (3)

(2) applicationContext.xml

```
<jaxws:endpoint id="helloworldws" implementor="#helloworldService" address="/">
<bean id="helloworldService" class="com.pkusma.HelloWorldServiceImpl" />
```

(3) Annotations : @WebService @WebMethod @WebParam etc.

apache CXF – JAX-RS

(1) web.xml – same as before (for JAX-WS)

(2) applicationContext.xml

```
<jaxrs:server address="/">
    <jaxrs:serviceBeans>
        <ref bean="helloWorld"/>
        <ref bean="helloUniverse"/>
    </jaxrs:serviceBeans>
    <jaxrs:extensionMappings>
        <entry key="xml" value="application/xml"/>
    </jaxrs:extensionMappings>
</jaxrs:server>
```

"if multiple services are to be defined.

```
<jaxrs:server id="helloWorld" serviceClass="com.pkuroma.HelloWorldImpl" address="/" />
```

(3) (1) (2)

If single service is to be defined.

(3) Annotations: @Path @PathParam etc.

(4) SessionBindingListener – to write code to track a session or to do things when session binds/unbinds.