1. WS has provided as the following component,
2. WSDL
3. Skeleton
4. Stub
5. SOAP
6. HTTP
7. UDDI🡺 Universal Description Discovery and Integration
8. WS is for applications interaction
9. A Simple flow Explanation🡺 see the notes🡺 provider has Skeleton and Consumer has Stub where Stub and Skeleton(IS A SERVLET CLASS) has Same Method Declaration as SEI but Logic is to covert Java req and Res to SOAP req and res
10. We have 4 implementers of WS namely,(again this 4 are specifications only we need implementation)
11. JAX-RPC
12. JAX-M🡺 is for async
13. JAX-WS
14. JAX-RS
15. Eclipse will support only 3 implementations it seems for remaining we need some plugins,
16. Axis1
17. Axis2
18. Apache CXF
19. AXIS-1 example🡺 is the implementer of JAX\_RPC
20. Service endpoint Url syntax🡺http://localhost:8080/webappName/skeletonUrl/unique Name of the Service
21. AXIS-1 uses server-config.wsdd(will be present in web-inf folder and this file also contains class ,method and parameters details of a service) file in order to map unique name to the Service

Syntax🡺

<ns:service name=”Unique name of Service”-------- >

<ns1:parameter name=”**allowedMethods**” value=”add sub multiply”/>

1. WSDL GENERATION TOOL is 🡺org.apache.axis.wsdl.java2wsdl 🡺 this is a class that is acting as a WSDL generator
2. java2wsdl WSDL generator needs following information to generate WSDL
3. Service Class
4. endpoint url
5. wsdl file name
6. targetnamespace
7. web.xml file will have skeleton information
8. Steps to create WS provider and Consumer in STS see the notes
9. Final url to be hit in the Browser🡺 <http://localhost:8080/webappName/skeletonUrl/unique?wsdl>
10. In WS we have 2 types of Clients(a client app can be Standalone or web app)
11. Proxy based Client🡺 see the notes
12. DII client🡺 Dynamic Invocation Interface🡺 here we will not generate stub , ws implementors provide few set of classes that acts as a stub to call ws methods
13. Ms to invoke Service class methods we need its reference type, for getting the ref type we will use Service class of Java and URL class of Java🡺URL will have a wsdl location and Service class will have some sort of reference to SEI.
14. Axis2 Example🡺
15. Here Skeleton used is 🡺org.apache.**axis2**.transport.http.AxisServlet
16. Stub and WSDL generation see the notes

SOAP

1. SOAP🡺 Simple Object access protocol
2. IS a Messaging protocol
3. IN WS errors/exceptions are called as Faults
4. Soap contains 3 parts
5. Soap-envelope
6. Soap-header
7. Soap-Body
8. If req is failed then Soap Message will contain 4th Section called🡺 Soap-fault
9. Syntax🡺

<?xml version=”1.0”?>

<soap: Envelope xmlns: soap=”http:------------“

Soap:encodingstyle=”http:-------“>

<soap:header>

------

</soap:header>

<soap:body>

-------

<soap:fault>

------

</soap:fault>

</soap:body>

</soap: Envelope>

1. Here Servlet class used is (I don’t know the Exactly which implementation is this)

<servlet-class>Com.sun.xml.ws.transport.http.servlet.WSServlet</servlet-class>

Note🡺 **for this servlet some Listener class is also used**

Rest Service

1. Rest🡺Representational state transfer
2. REST is an architecture
3. REST service don’t have such WSDL specification file thus URL or URI itself is a specification for rest services to access the web service
4. In case of rest client will directly use the uri to get response
5. SOAP is a Protocol and REST is a architecture
6. Soap permits XML data format only but REST allows different formats like HTML, XML, JSON and etc...
7. SOAP supports different types of protocols like http protocol, RMI protocols. But REST supports only Http and https
8. REST do not need client generation, wsdl generation and etc..
9. Jersey is the implementation of JAX-RS
10. Apache CXF also provides implementation for JAX-RS
11. HTTP Methods,
12. Get🡺 to get information from Server
13. Post🡺 to send or save information in the server
14. Put🡺 is use to update data of that URL
15. Delete🡺 to delete data of that URL
16. Metadata🡺 along with req and res we also have header information which is Meta data
17. HTTP Status Code,
18. 2xx🡺 success
19. 1xx🡺informational
20. 3xx🡺redirection
21. 4xx🡺Client error
22. 5xx🡺Server error
23. We can pass the input parameters to operation In 2 ways for a rest service
24. Request params
25. Path parameter
26. Syntax for Path parameter🡺

Baseurl/100🡺 where 100 is employee id this is sent as method input

1. Syntax for Request parameter or Query Parameter,

baseURL**?id**=1

Note🡺 Query parameter is not like a path parameter its additional parameter that gets added to http header

1. As discussed early along with req and res we have header information🡺 note we have req header and response header
2. List of annotations used
3. @Path🡺 both at class level and method level
4. @Produces🡺is used to, what we want to return from WS 🡺 ms is method level @
5. @Consumes🡺 is used to, what we want to send to WS🡺 ms is method level @
6. @PathParam
7. @Get
8. @Post
9. @Put
10. @Delete
11. JSON will store the value in the form of key and value pair
12. JSONLint is used to validate JSON.
13. @Produces(MediaType.APPLICATION\_JSON)
14. Example
15. web.xml🡺 root tag for web.xml is <web-app>

<servlet>

------

<servlet-class>org.glassfish.jersey.servlet.ServletContainer</ servlet-class>

<initiparam>

<param-name>jersey.config.server.provider.packages</param-name>

<param-value>com.capgemini.services</param-value>

</initiparam>

<load-on-startup>1</ load-on-startup>

</servlet>

<init-param>🡺 in our app we might have 10 packages among this 10 packages only one package will have all service classes thus we need to tell JAX\_RS from where or which package it should start searching for WS classes

1. Java file

@Path(“/Capgemini”)

Public class NotifyNondlineRestService

{

@GET

@Path(“{a}/{b}”)

Public int add (@PathParam(“a”) int a, @PathParam(“b”) int b)

{

Return a+b;

}

}

1. Output🡺

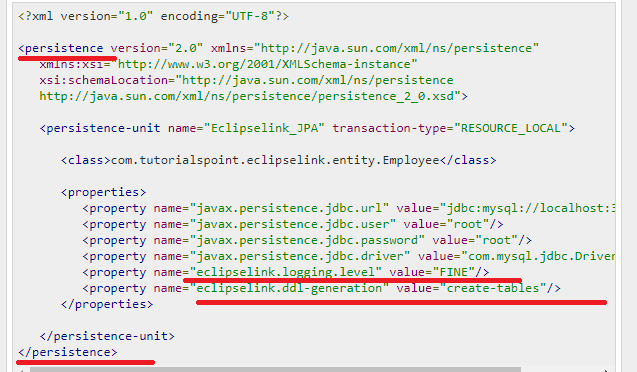
http:..loclahost:8080/applicationContext/url in web.xml file/url as in @path annotation**/10/20**

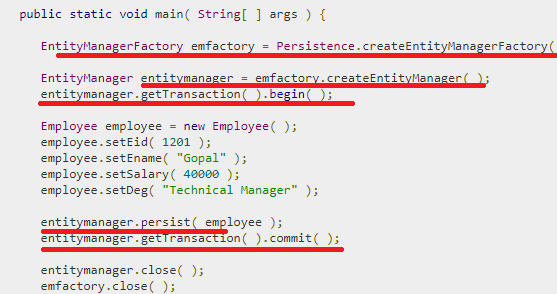
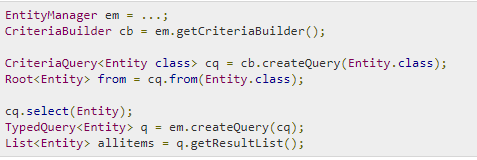
JPA

1. JPA is an api(classes and interface) to persistently store the vast amount of data into DB
2. JPA is by Oracle
3. While Introducing EJB 3.0 the persistence layer was separated and specified as JPA 1.0
4. JPA 2.1 in 2013
5. Has EMF, EM ,E, ETransaction, Query, Persistence
6. Persistence🡺 this class contains static method to obtain EMF
7. EM and ET has one to one relationship
8. ORM takes care of Converting data from Object type to relational type and Vice versa
9. Mapping.xml🡺



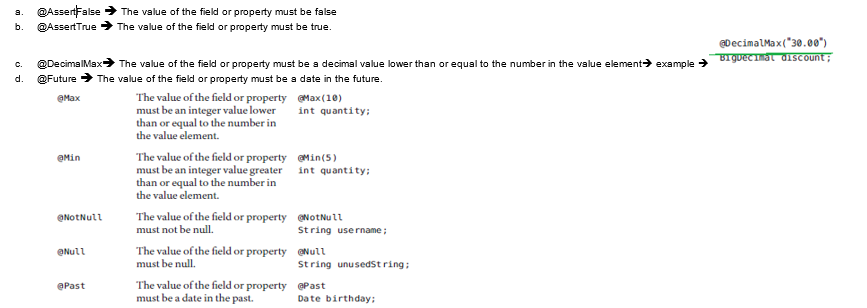
1. List of Annotations
2. @SequenceGenerator
3. @TableGenerator
4. @UniqueConstraint
5. Bean Conversion 🡺should gave getters for non-Boolean Property and must have isEmpty() for boolean properties
6. Persistence.xml



1. EntityManagerFactory emfactory = Persistence.createEntityManagerFactory( "Eclipselink\_JPA" );
2. CRUD methods 🡺 ms
3. Persist
4. Find
5. Remove
6. Update
7. 
8. CQ is Type-safe
9. 

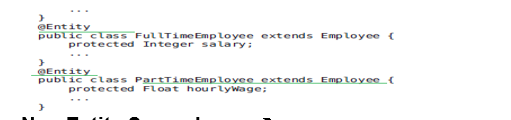
JPA Documentation

1. **Needed an Example on Arrays storage**
2. **Map Was not covered in Collections Section**
3. Bean Validation

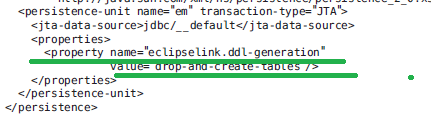


1. @Pattern and @Size🡺
2. **Example on primary key as Date field**
3. Caccade\_type.All/Detach/Merge/persist/refresh/remove
4. Abstract entities🡺 an abstract class annotated with @Entity. Abstract entities are like concrete entities but cannot be instantiated.
5. Mapped superclass🡺 Entities may inherit from super classes that contain persistent state and mapping information but are not entities. That is, the superclass is not decorated with the @Entity annotation and is not mapped as an entity by the Java Persistence provider.

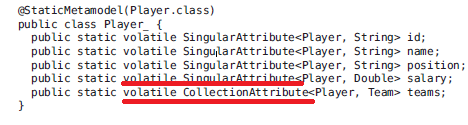
Example🡺 Parent 

Child🡺

1. @SecondaryTable🡺
2. MetaModel🡺 MS🡺CriteriaQuery uses metamodel to create query
3. **Metamodel classes model an entity’s attributes and are used by Criteria queries to navigate to an entity’s attributes**
4. Automatic table generation inJPA



1. An Equivalent Meta model class for every entity gets generated at ,**compile time🡺 in maven I think during maven install on JPA project**
2. Syntax🡺



1. The **EntityManager.createQuery and EntityManager.createNamedQuery** methods are used to query the data store by using Java Persistence query language queries
2. Syntax🡺 same as in hibernate, in case of createQuery(“Sql Query is sent as input)

And in case of createNamedQuery(Name of the Query is sent as input);

1. Same as in hibernate JPA aslo has Positional parameter prefixed with “ ?Position number” and Named parameter which is prefixed with “**:**parameter\_name”
2. Needed more information on
3. MEMBER OF
4. EXISTS
5. NOT EXISTS🡺 Scenarios
6. Set of Functions🡺 see later
7. **NEEDED MORE INFORMATION ON CASE EXAMPLE(AND ALSO IN PLSQL)**
8. **Constructor expressions allow you to return Java instances that store a query result element instead of an Object [].**

**Example🡺**

****

NOTE 🡺 ms 🡺**CriteriaQuery has methods like, select, from, where, having, groupBy and orderBy**

1. **Metamodel classes are typically generated by annotation processors either at development time or at runtime**
2. One of the programmatic way of getting metaModel class is🡺

object\_OF\_Root.getModel:

we can also get from em as shown below

**EntityType<Employee>Employee\_=em.getMetamodel().entity(Employee.class)**

1. For more information On CQ and CB methods First see the Notes
2. Ms🡺CriteriaBuilder methods is returning Predicate
3. **NEEDED MORE INFORMATION ON PREDICATE AND EXPRESSION IN JPA**
4. Methods of CriteriaBuilder,
   1. and🡺 return type is Predicate
   2. asc🡺Order
   3. des🡺Order
   4. avg🡺Expression
   5. sum🡺
   6. Count🡺Expression
   7. concat🡺 Expression
   8. between🡺Predicate
   9. equal🡺Predicate
   10. equals🡺Boolean object
   11. exists🡺P
   12. ge()🡺P
   13. greaterThan🡺P
   14. greaterThanOrEqualTo🡺p
   15. greatest🡺P
   16. gt()🡺P, lt
   17. in
   18. isEmpty()🡺P
   19. isnotEmpty
   20. isFalse🡺P
   21. isMember🡺P
   22. isNotMember
   23. like
   24. min
   25. max
   26. not
   27. notEqual
   28. notLike
   29. length
5. Methods of the Root<T> is🡺 ms actually root refers to “from” part of the SQL query
   1. get(field\_name)
   2. fetch(field\_name)
   3. in()
   4. isNull
   5. isNotNull
   6. join()
   7. joinCollection
   8. joinList
   9. joinMap
   10. joinSet

Enum

1. Example🡺

Public enum Week implements Serializable{

SUNDAY(“sun”, “First Day of the Week),

MONDAY(“mon”, “second Day of the Week),

TUESDAY(“tues”, “third Day of the Week),

WEDNESDAY(“wed”, “fourth Day of the Week),

THURSDAY(“thurs”, “fifth Day of the Week),

FRIDAY(“fri”, “sixth Day of the Week),

SATURDAY(“sat”, “seventh Day of the Week),

Private String code;

Private String description;

Private Week(String cd, String des){

this. code= cd;

this. description= des;

}

Public getcode(){

return code;

}

Public getdescription(){

return description;

}

}

Out the class calling an enum syntax is🡺

String codeFromEnum=Week.SUNDAY.getCode();

String descriptionFromEnum=Week.SUNDAY. getdescription();