Java as Product

* Java Standard Edition [ Java SE] - JDK // JAR
  + Console Application ---
  + Java Libraries
  + Desktop [ GUI ] Application [ Java Swing , Java FX ]
* Java Enterprise Edition [ Java EE] - Java EE SDK
  + Web Application / WAR
  + Enterprise Application //EAR
* Java Micro Edition [ Java ME ]
* Java Card - Store information in smart card chip.

**Java EE 7 Specification**

* **Donated to Eclipse for further enhancement.**

Jakarta EE

1. Web Profile
   1. Servlet
   2. JSP
   3. EJB Lite
   4. JAX-RS
2. Full Profile [ includes Web Profile]
   1. JAX-WS
   2. Messaging API
   3. Java Mail API

Java EE compatible Application Server will supply Java EE SDK

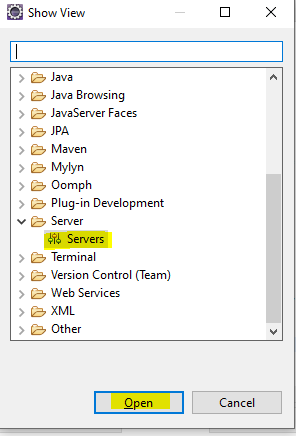
* Oracle WebLogic Server
* Redhat JBoss Server
* IBM Websphere
* Tom EE+ Application Server
* Tomcat Sever

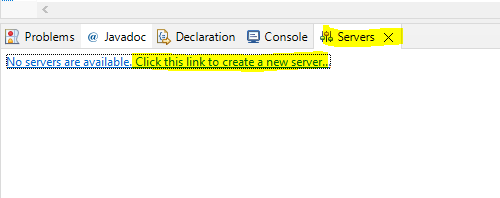
<https://www.apache.org/dyn/closer.cgi/tomee/tomee-8.0.15/apache-tomee-8.0.15-plus.zip>

* Download zip file
* Then extract it to a folder

In eclipse

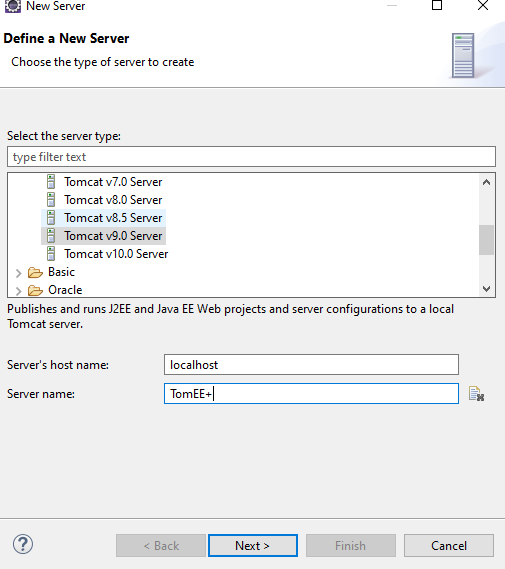
* Window – show view – Other – Server – Open



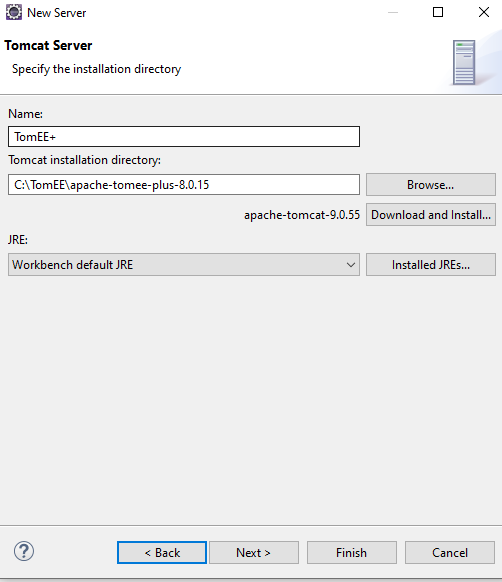
\

Click the link to create a new Server.

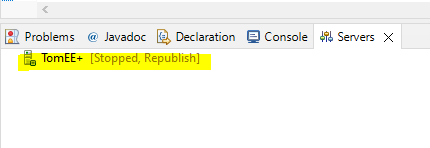
Select Apache Tomcat Sever V 9



Click next



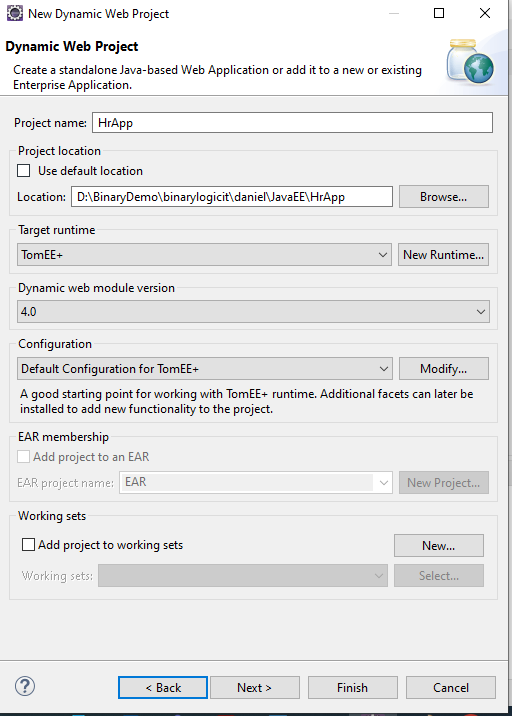
Next – Finish



Right click on TomEE+ and start

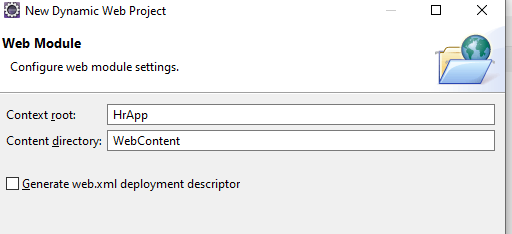
**Creating your First Web Application using Java EE**

* File – New – Other – Web – Dynamic Web Project

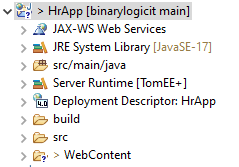


Next and Next Again

Content Director – Change to “WebContent”



Click Finish



M

OTHER JAVA CLASSES

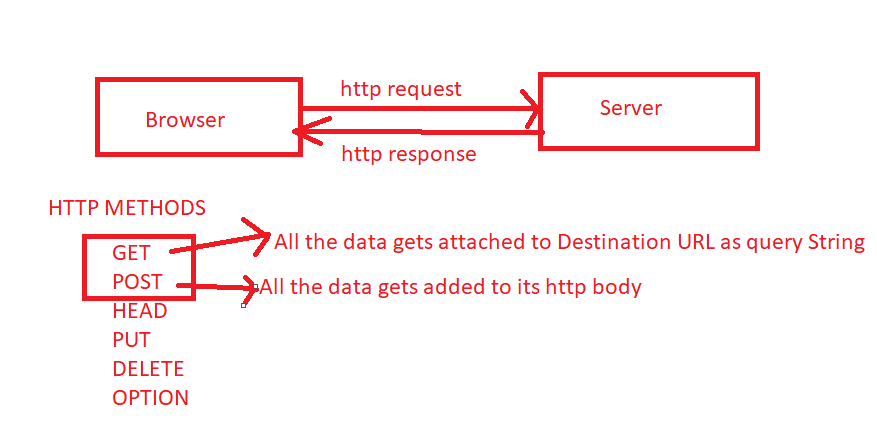
V : View

HTML, JSP

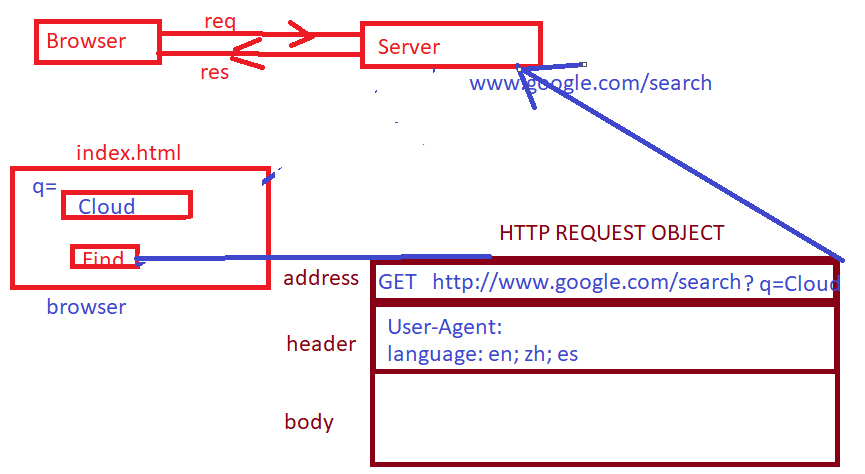
SERVLET

C : CONTROLLER

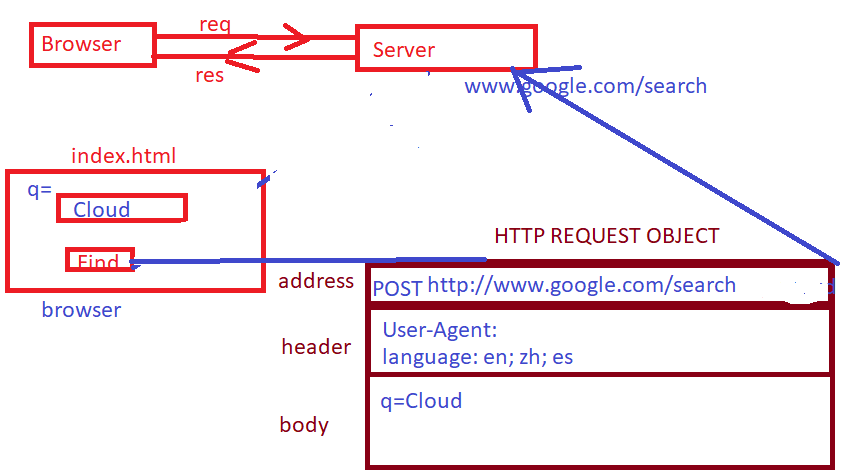
SERVLET

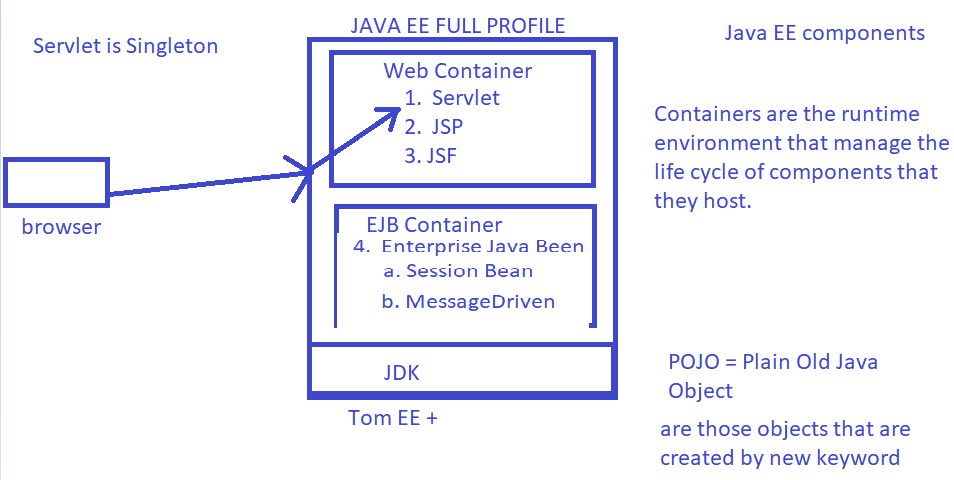


GET REQUEST



**POST**





Servlet Life Cycle

* Instantiation of Servlet is managed by Web Container.
* Only one instance / JVM is created – This means Servlet is Singleton.
* Instance if servlet is created at time when it was requested first.

//invoke only once after instantiation

public void init() { }

// invoke everytime it receives a request

service(ServletRequest req, ServletResponse res){

//doGet(HttpServletRequest req, HttpServletResponse res)

//doPost

}

// only once before servlet gets destroyed

destroy() {

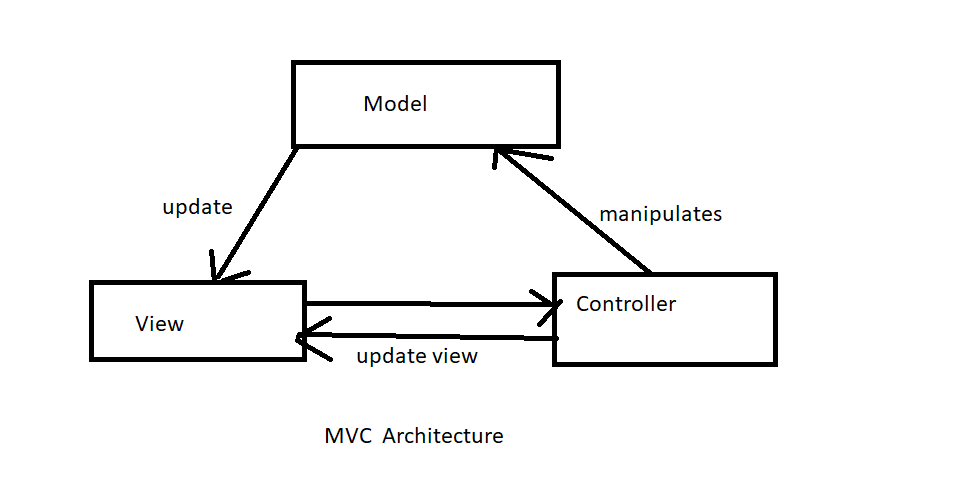
// it is invoked at the time servlet destruction.

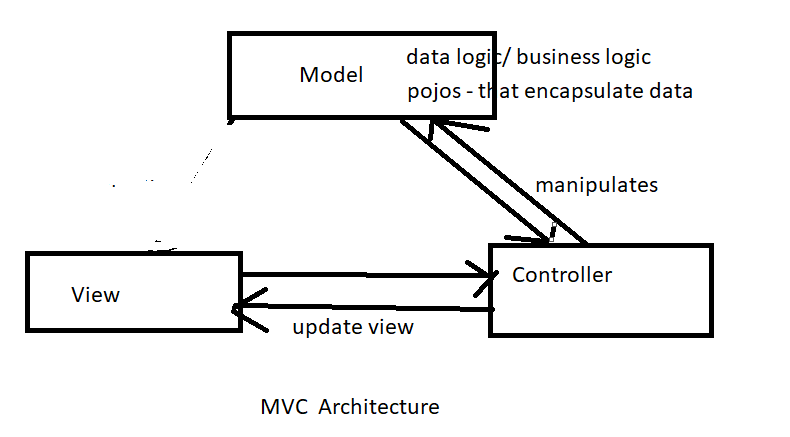
}

SDLC : Software Development Life Cycle

* Requirement Analysis.
* Design
* Coding
  + Unit Testing
* Testing [ QA ]
* Deployment

**Model – View – Controller Architecture**





**Model :** Handle data/business logic

**Controller :** It controls data flow into model object and update view whenever data change. It is responsible for handle request n response.

**View :** It displays the information from the model to the user.

**Java EE Web Application**

Model :

DAO, Pojos

Controller:

Servlet

View :

Servlet

JSP [ popular ]

**RequestDispatcher is used to redirect the call to other components[ view ] from controller.**

RequestDispatcher reqDispatcher=request.getRequestDispatcher("EmployeeUI");

reqDispatcher.forward(request, response);

**How the controller handover the data to View**

It is going to take the help of memory scoping to share the data.

* Request Scope : Explicit to single request – response cycle. Each new request will have its own copy of request & response. Data stored in 1st request will not visible to 2nd request.
* Session Scope : Explicit to User/Client Session – A data maintain in session will be visible to all request – response cycles for same client.
* Application Scope : Explicit to Application - A data maintain in application scope is visible to all users/clients.

**Java Server Page : JSP**

JSP is used to represent view / presentation logic for web application. This is server – side component and this can easily work with Java Objects.

* It is easier to maintain than servlet with respect to complexities that derived by containing embedded HTML
* It supports Expression Language and that helps a lot to simplify JSP page development.

JSP is the extension to servlet. In fact, it is servlet.

**Life Cycle of JSP page**

When use access JSP page first time.

<http://localhost:8080/HrApp/index.jsp>

1. Translation of JSP Page to Index\_Servlet.java by web container.
2. Compilation of .Java
3. ClassLoader loads the class [ .class ]
4. Instantiate an Object of this class // [ only one instance ]
5. init()
6. service()
   1. doGet
   2. doPost
7. destroy()

**Second Time onwards – all the request for JSP Page will start from Step 6**

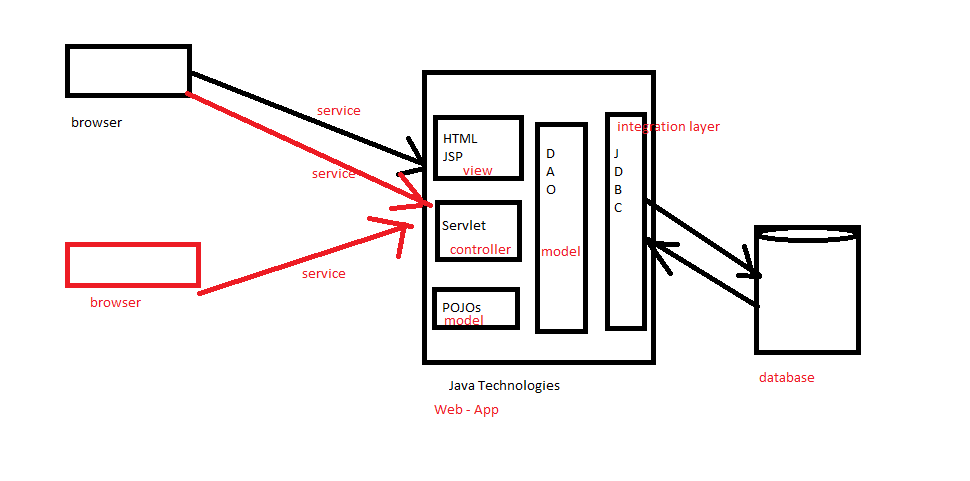
**JSP provides implicit Objects**

1. **request**
2. **response**
3. **session**
4. **out**

**Enterprise Java Bean**

1. **Session Bean**
2. **MessageDriven [ Java Messaging Service ]**

**Session Bean : It is managed by EJB Container. And It is used heavily for Business Logic. It comes with the concept of EJB pooling.**

** Dependency Injection using @EJB for EJB Component**

**Servlet**

**@EJB**

**EmployeeBL dao;**

**@EJB**

**EmployeeDao dao;**

**Web Services**

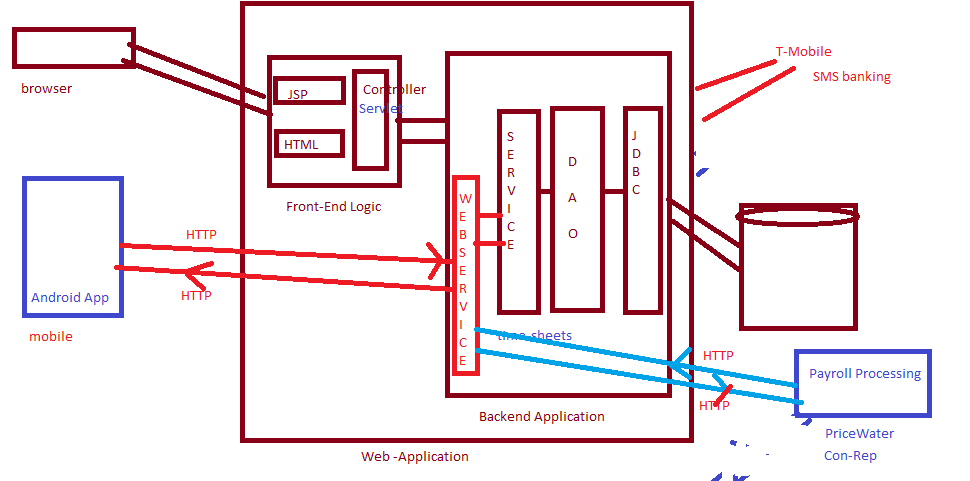
A Service that exposes its functionality / or Business Operation to make client – server interaction interoperable, is known as a web service.

Client Application & Backend Application are needed to be in same languages. As a result , one can develop multiple UJ Client connecting to Backend in the language that they are comfortable with.

JAX-RS : RESTful Service // it uses HTTP protocol for communication layer and therefore it is popular in industry as most of the application & backend server are accessible over the web - infrastructure

JAX – WS : SOAP Web Service // Support other protocols as well. It can be useful for enterprise level of communication.

Also, other organization – business service can invoke the backend operation as a part of their B2B Service agreement.



RESTful Service : **RE**presentational **ST**ateful Service // REST Service

In Java EE

JAX-RS Specification to developer REST Service

* REST Service is to expose business functionalities for interoperable service
* ws-i.org is responsible to ensure the rules for interoperability being followed by different technology vendor
* JAX-RS implementation is supplied by Application Server
* XML or JSON are data that can be used to transport.

Java EE provides JAX-B API to transform its Java Object to XML / JSON

JAX-B : Java API for XML Binding

**Testing Rest Service**

POSTMAN

**SOAP WEB SERVICE/ XML Web Service // JAX-WS : Java API for XML Web Service**

SOAP : Simple Object Access Protocol

It acts as envelop and it hold the data and information about method/object. It encapsulate data in XML form.

Protocol- they hold the destination address.

JAX-B

WSDL

XML – XSD

WSDL : Web Service Descriptor Language

<http://localhost:8080/HrApp/webservices/EmployeeWS?wsdl>

**Hibernate: An ORM Software**

It is a framework that simplify JDBC operation. It is opensource , lightweighted and popular ORM tools.

ORM stands for **O**bject **R**elational **M**apping. It is the implementation of **Java Persistence API** for data logic operation.

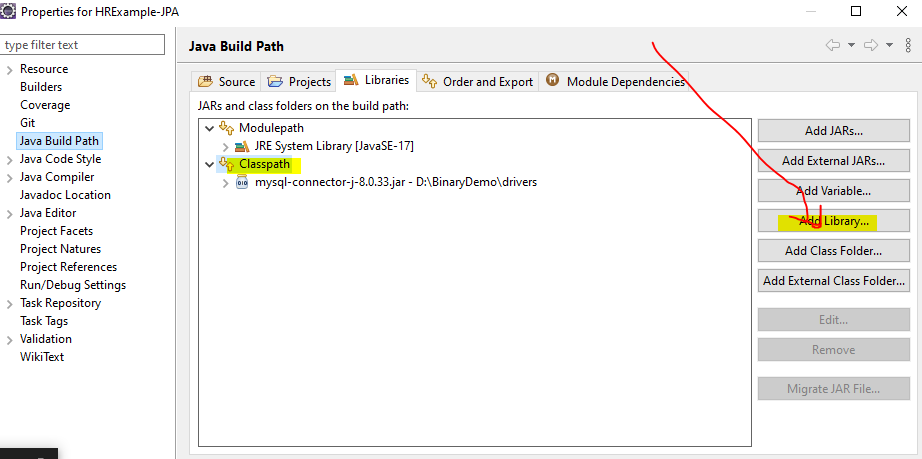
JPA: Java Persistence API – It is the part of Java EE specification and can be used independently with Java SE as well.

There are popular implementation available for JPA Specification

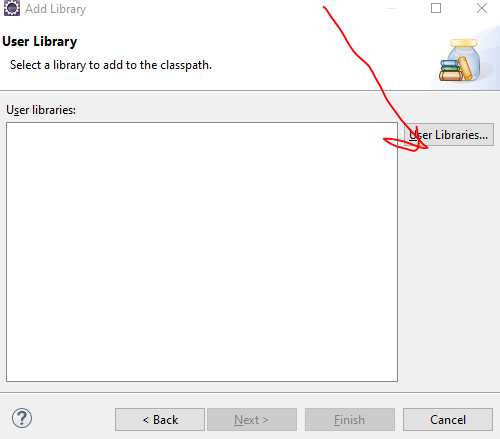
* Toplink
* EclipseLink
* **Hibernate**
* OpenJPA

Setting up Hibernate in your Java SE Project

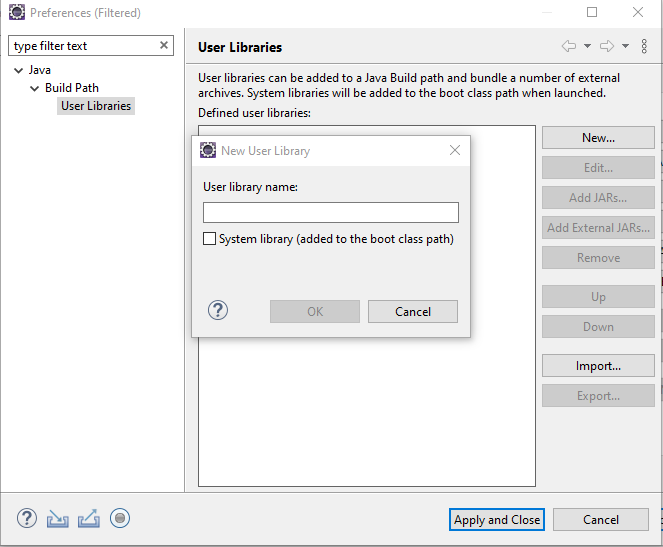
* Download the Hibernate from <http://github/srinit16/birnarylogicit>
* Unzip it
* Right Click On Project – Click on Build Path
* Click on Add Library as shown below



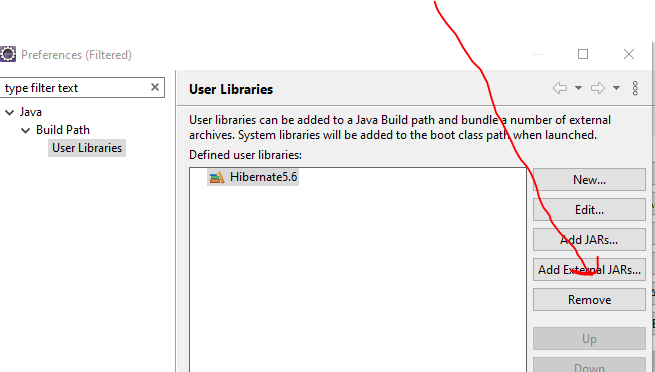
Add Library – User Library – Next and click on User Library again



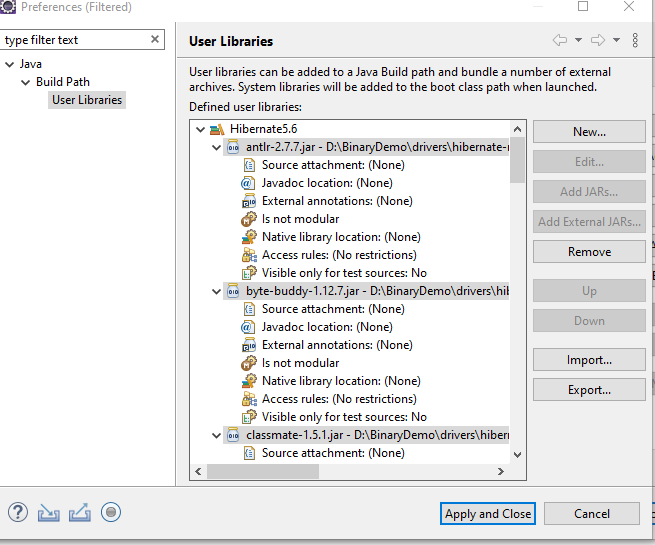
Click on New and Give Name to Library and ok



Click on Add External JAR and add all the jars file from hibernate – lib – required folder.



Click on Apply and back to project



**Developing Data Operation Logic using Hibernate**

* + - 1. Entity – Table Mapping
      2. Update EmployeeBL logic

ORM Mapping can be done by using

* Annotation
* Descriptor File

**Entity Class / Persistence Class**

**Using Annotation :**

@Entity @Table(name="EMPLOYEE")

**public** **class** Employee {

@Id @Column(name="EMP\_ID")

**private** **int** id;

@Column(name="EMP\_NAME")

**private** String name;

@Column(name="SALARY")

**private** **double** salary;

@Column(name="DEPT\_ID")

**private** **int** deptId;

}

**Main**

SessionFactory sessionFactory=**new** Configuration()

.configure("com/example/config/hibernate.cfg.xml")

.buildSessionFactory();

Session s=sessionFactory.openSession();

Q1. Difference between load n get

**CRUD Operation**

CREATE / INSERT

Employee emp=**new** Employee(6,"Peter Parker",1200.00,102);

**try** {

s.getTransaction().begin();

s.save(emp);

s.getTransaction().commit();

System.***out***.println("Record inserted");

}**catch**(Exception ex) {

s.getTransaction().rollback();

ex.printStackTrace();

}

READING

Employee e=s.get(Employee.class, 1); // select emp\_id, emp\_name, salary, dept\_id from employee where emp\_\_id=1

Employee e=s.load(Employee.**class**, 1);

Update

Employee emp=s.get(Employee.**class**, 5);

**if**(emp!=**null**) {

emp.setSalary(emp.getSalary()\*1.10);

**try** {

s.getTransaction().begin();

// s.save(emp);

s.saveOrUpdate(emp);

s.getTransaction().commit();

System.***out***.println("Record inserted");

}**catch**(Exception ex) {

s.getTransaction().rollback();

ex.printStackTrace();

}

emp.print();

**Delete**

Employee emp=s.get(Employee.**class**, 5);

**if**(emp!=**null**) {

//emp.setSalary(emp.getSalary()\*1.10);

**try** {

s.getTransaction().begin();

// s.save(emp);

//s.saveOrUpdate(emp);

s.remove(emp);

s.getTransaction().commit();

System.***out***.println("Record deleted");

}**catch**(Exception ex) {

s.getTransaction().rollback();

ex.printStackTrace();

}

// emp.print();

}

Hibernate offers HQL / Hibernate Query Language / Java Persistence Query Language

Select e from Employee e;

**findAll**

Query<Employee> q=s.createQuery("select e from Employee e", Employee.**class**);

List<Employee> l=q.getResultList();

l.forEach(e-> e.print());

Query<Employee> q=s.createQuery("select e from Employee e where e.deptId=?1", Employee.**class**);

q.setParameter(1, 101);

List<Employee> l=q.getResultList();

l.forEach(e-> e.print());

// HQL for custom queries with bind variable

Query<Employee> q=s.createQuery("select e from Employee e where e.deptId=:did", Employee.**class**);

q.setParameter("did", 101);

List<Employee> l=q.getResultList();

l.forEach(e-> e.print());

**NamedQuery**

Query<Employee> q=s.createNamedQuery("Employee.findAll", Employee.**class**);

List<Employee> l=q.getResultList();

l.forEach(e-> e.print());

@Entity @Table(name="EMPLOYEE")

@NamedQuery(name="Employee.findAll", query="select e from Employee e")

**public** **class** Employee {

**//everything else as above in employee**

**}**