* Hibernate *

* Hibernate

- Hibernate is a Framework, developed by gavin king in 2001
- hibernate is an ORM tool

What is ORM tool ?

- ORM stands for object relational mapping.
- in ORM tool, object mapped with table, hence we can perform crud operation easily
- Object can be mapped in xml & annotation way also
- due to ORM tool, app becomes independent from db
- in market there are many ORM tool available, EJB EJB2, Struct, Struct2, iBatis, Toplinks, Hibernate etc.

eg.

```
class Student
{
    private int id;
    private String name;
    private String city;
    private double percentage;
    // setter & getter
    // to string
}
```

Student

id name city percentage

```
save()
delete()
update()
get()
```

Features of hibernate

1. Automatic table creation
2. Query Status
3. HQL & HCQL
4. table relation
5. complex join
6. Fetching technique
7. caching technique

```
package com.model;
public class Student {
    private int id;
    private String name;
    private String city;
    private double percentage;
    // setter getter
    // to string
```

```
<hibernate-mapping>
  <class name = "com.model.Student">
    <id name = "id" type = "int">
      <generator class = "identity"></generator>
    </id>
    <property name = "name" type = "java.lang.String">
    </property>
    <property name = "city" type = "java.lang.String">
    </property>
    <property name = "percentage" type = "double"></property>
  </class>
</hibernate-mapping>
```

4

What is hbm.xml?

- in hibernate, hbm.xml stands for hibernate mapping xml file
- hbm.xml can be used to mapped object by using xml
- in hbm.xml, <hibernate-mapping>, <id>, <generators> <property> etc tag can be used

eg.

```
<hibernate-mapping>
    <class name = "com.model.Employee">
        <id name = "id" type = "int">
            <generator class = "identity"></generator>
        </id>
        <property name = "name" type = "java.lang.String">
        </property>
        <property name = "designation" type = "java.lang.String">
                                                    </property>
        <property name = "company" type = "java.lang.String">
                                                    </property>
        <property name = "salary" type = "double"></property>
    </class>
</hibernate-mapping>
```

- For every individual object, we have to create separate hbm.xml file
- this file can be located in src/main/java or class path.

what is cfg.xml

- in hibernate, cfg.xml stands for hibernate configuration xml file
- in cfg.xml, configuration of hibernate will be done in xml way
- in cfg.xml, <hibernate-configuration>, <session-factory>, <property>, <mapping> tag can be used.

eg.

```
<Hibernate-Configuration>
    <session-factory>
        <property name = "hibernate.connection.driver_class">
com.mysql.cj.jdbc.Driver</property>
        <property name = "hibernate.connection.url">jdbc:
mysql://localhost:3306/jap77</property>
        <property name = "hibernate.connection.username">root
                                                    </property>
        <property name = "hibernate.connection.password">root
                                                    </property>
        <property name = "hibernate.dialect">org.hibernate.
                dialect.mysql8Dialect</property>
        <property name = "hibernate.hbm2ddl.auto">update
                                                    </property>
        <property name = "hibernate.format-sql">true
                                                    </property>
        <mapping resource = "Employee.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

- in application, there can be only one cfg.xml file.
- default name of this file can be set as a hibernate.cfg.xml file

-this file can be load by using configure() method of configuration class.
- this file can be located in src/main/java or classpath.

Configuration
- configuration is a class, which is located in org.hibernate.cfg package
- by using configuration class, we can load configuration of hibernate.
eg.

```
Configuration con = new Configuration().configure();
```

SessionFactory
- SessionFactory is an interface, which is located in org.hibernate package
- instance of SessionFactory can be achieve by using buildSessionFactory() method of Configuration class
- Session factory can be used to hold database connection
- sessionFactory is thread-safe or synchronized
- sessionFactory is an immutable object
eg.

```
Configuration con = new Configuration().configure();
SessionFactory sf = con.buildSessionFactory();
```

Session
- Session is an interface, which is located in org.hibernate package.
- instance of session can be achieve by using openSession() & getCurrentSession() method of SessionFactory interface.
- openSession() method returns new instance of session
- getCurrentSession() method returns existing session.
- by using session, we can perform CRUD operation
- session is a thread-unsafe or non-synchronized
- session is a mutable object.

e.g.

```
Configuratn con = new Configuration().configure();
SessionFactory SF = con.buildSessionFactory();
Session session = SF.openSession()
```

Transaction
- Transaction is an interface, which is located in org.hibernate package
- instance of transaction can be achieved by using getTransaction() & beginTransaction() method of Session interface.
- getTransaction() method returns existing Transactn
- beginTransaction() method returns new instance of Transaction
- Transaction can be used to commit number of changes made by session
- e.g.

```
Configuration con = new Configuratn().configure();
SessionFactory SF = con.buildSessionFactory();
Session session = SF.openSession();
```

```
Transaction tx = session.beginTransaction();

Employee e = new Employee();
e.setName("rajiv parthale");
e.setDesignation("ui developer");
e.setCompany("cts");
e.setSalary(12000);

session.save(e);
tx.commit();
```

* State of Objects in hibernate

```
Employee a = new Employee()
e.setName("ABC");
e. setDesignatn("java developer");
e.setCompany("TCS");
e. setSalary(12000);
```

Transient

session.save(e);

new processing old

persistence

tx.commit();

Detached

① Transient state
- in hibernate, object considered in transient state if doesn't associated with session yet.
- transient object also considered as a new object

② persistence state
- in hibernate, object considered in persistence state if it associated with session.
- persistence object also considered as a processing object

③ detached state
- in hibernate, object considered in detached state if it associated with database not session anymore.
- detached object also considered as an old object.

✓ Methods of session

① save()
- save() method can be used to insert a record
- save() method returns serializable id
- save() method required transient object.

② delete()
- delete method can be used to delete a record
- delete() method requires detached id

③ get()
- get() method can be used to find record by id
- get() method returns object or null

④ load()
- load() method can be used to find record by id.
- load() method returns object or exception
- load() method can be used to if we assure about the object.

⑤ update()
- update() method can be used to update record
- update() method requires detached object.

⑥ saveOrUpdate()
- saveOrUpdate() method can be used to save or update record.
- saveOrUpdate() insert a record if mentioned object is transient
- saveOrUpdate() update a record if mentioned object is detached.

* JPA API
- JPA stands for javax persistance api
- JPA is a set of annotations, which can be used to mapped object java based.
- due to JPA API, we don't have to create separate hbm.xml for each
- list of annotations

① @Entity
- @Entity is an annotatn, which is located in javax. persistence package
- this annotation can be used to denotes a class as an entity
- this annotatn can be used over class name.

② @Id
- @Id is an annotation, which is located in javax. persistance package.
- this annotation can be used to denotes property as an id
- this annotation can be used over property name.

③ @GeneratedValue
- @GeneratedValue is an annotation, which is located in javax. persistence package.
- this annotation can be used to denotes generation type of id
- this annotation can be used over property name.

④ @Table
- @Table is an annotation, which is located in javax. persistence package.
- this annotation can be denotes user-define name of table
- this annotation can be used over table name

⑤ @Column
- @column is an annotation, which is located in javax. persistence package.
- this annotation can be used to denotes user-defined column
- this annotation can be used over property name.

⑥ @Transient
@Transient is an annotation, which is located in javax. persistence package.
- this annotation can be used to denotes property as a transient property
- in hibernate transient property will not be a part of session anymore
- this annotation can be used over property name.

① @Embedded

- @Embedded is an annotation, which is located in javax.persistence package.
- this annotation can be used to denotes property as an embedded property
- property of embeddable class considered as an embedded property
- this annotation can be used over property name

② @Embeddable

- @Embeddable is an annotation, which is located in javax.persistence pkg.
- this annotation can be used to denotes a class as an embeddable class.
- in hibernate embeddable class will not create a separate table, it becomes a part of a table
- this annotation can be used over class name.

```
package com.model

@Entity
public class product {
  @Id
  @GeneratedValue (Strategy = GenerationType.IDentity)
  private int id;
  private String name;
  private string type;
  @Column (columnDefinition = "double (10,2)")
  private double price;
  // setter & getter
  // to String
  @Override
```

```
package com.util
import java.util.properties;
public class HBUtils {
  public static SessionFactory SF = new Configuration()
      .setProperty (P1())
      .addAnnotatedClass (product.class)
      .buildSessionFactory();

private static properties P1()
 {
  properties p = new properties();
p.setProperty ("hibernate.connection.driver_class", "com.mysql.
                                    cj.jdbc.Driver");
p.setProperty ("hibernate.connection.url", "jdbc:mysql://
localhost:3306/jap77");
p.setProperty ("hibernate.connection.username", "root");
p.setProperty ("hibernate.connection.password", "");
p.setProperty ("hibernate.dialect", "org.hibernate.dialect.
                    MySQL5Dialect");
p.setProperty ("hibernate.hbm2ddl.auto", "update");
p.setProperty ("hibernate.show-sql", "true");
p.setProperty ("hibernate.format-sql", "true");
  return p;
 }
}

public class App
 {
  public static void main (String [] args)
  {
   Session session = HBUtils.SF.openSession();
  Transaction tx = session.beginTransaction();
```

```
product p = new product();
p.setName("oven");
p.setType("home appliance");
p.setPrice(1200);
session.save(p);
tx.commit();
```

App.java

```
Session session = HButils.sf.openSession();
Transaction tx = session.beginTransaction();
String[] types = {};
Random r = new Random();
for(int i=1; i<=10000; i++)
{
    product p = new product();
    StringBuilder sb = new StringBuilder();
    for(int j=1; j<=5; j++)
        sb.append((char)(r.nextInt(26)+65));
    p.setName(sb.toString());
    p.setType(types[r.nextInt(types.length)]);
    p.setPrice(r.nextDouble()*100000);
    session.save();
}
tx.commit();
```

★ HQL

- HQL stands for Hibernate query language
- HQL is similar like SQL & easier than SQL

HQL - delete from product where id=40

SQL - -11-

SQL works on table & column

HQL works on class & property

- Hence for java developer HQL Becomes easy to learn and program
- HQL considered as a database independent query
- HQL works on class & property, hence it is considered as an object oriented query
- As per mention clarket, HQL works on database differnt hence it is also known as a polymorphic query.
- by using HQL we can perform, delete, update & display query.

- To work with HQL, we can use createQuery method of session interface
- createQuery() method returns instance of query interface
- Methods of Query

① list()

- list() method can used to execute DQL query
- list() method returns instance of list interface
- list() method can be used if multiple objects can be return

② UniqueResult()

- uniqueResult() method can be used to execute DQL query
- uniqueResult() method returns object or null
- uniqueResult() method can be used if single object can be return

③ executeUpdate()

- executeUpdate() method can be used to execute DML query.
- executeUpdate() method returns int value

④ setParameter()

- setParameter() method can be used to pass parameter into query.
- setParameter() method returns instance of query

⑤ setMaxResults()

- setMaxResults() method can be used to set limit on records
- setMaxResults() method returns instance of query

⑤ setFirstResult()

- setFirstResult() method can be used to skip number of records.
- setFirstResult() method returns instance of query

⑦ setCacheable()

Display all product

eg

```
Query<Product> q = session.createQuery("from Product
product.class");
List<Product> list = q.list();
for(Product p:list)
    System.out.println(p);
```

OR

```
List<Product> list = session.createQuery("from Product"
,Product.class)
.list();
for(Product p:list)
    System.out.println(p);
```

Display product by id

eg

```
Query<Product> q = session.createQuery("from Product
where id =:a", Product.class);
q.setParameter("a", 50);
Product p = q.uniqueResult();
System.out.println(p);
```

OR

```
Product p = session.createQuery("from Product where id =:a",
        Product.class)
        .setParameter("a", 50)
        .uniqueResult();
System.out.println(p);
```

Find product by type

e.g.

```
Query <product> q = session.createQuery("from product where type = :a", product.class);
q.setParameter("a", "camera");
List <product> list = q.list();
for(product p : list)
    System.out.println(p);
```

Find product By id & type

e.g.

```
Query <product> q = session.createQuery("from product where id = :a & type = :b", product.class);
q.setParameter("a", 79);
q.setParameter("b", "tablet");
product p = q.uniqueResult();
S.O.p(P)
```

Find product by id or type

e.g.

```
query <product> q = session.createQuery("from product where id = :a or type = :b", product.class);
q.setParameter("a", 79);
q.setParameter("b", "tablet");
List <product> list = q.list();
for(product p : list)
    S.O.p(P);
```

Find product by price less than

e.g.

```
Query <product> q = session.createQuery("from product where price < :a", product.class);
q.setParameter("a", 10000.0);
List <product> list = q.list();
for(product p : list)
    S.O.p(P);
```

Find product by price betn //not betn

e.g.

```
Query <product> q = session.createQuery("from product where price betn :a and :b", product.class);
q.setParameter("a", 1000.0);
q.setParameter("b", 3000.0);
List <product> list = q.list();
for(product p : list)
    S.O.p(P);
```

Find product by price not betn

e.g.

```
Query <product> q = session.createQuery("from product where price not betn :a & :b", product.class);
```

Find product by name like

e.g.

```
Query<Product> q = session.createQuery("from Product where name like :a", Product.class);
q.setParameter("a", "%pp%");
List<Product> list = q.list();
for(Product p: list)
    s.o.p(p);
```

display all names

e.g.

```
Query<String> q = session.createQuery("select name from Product", String.class);
List<String> list = q.list();
for(String s: list)
    s.o.p(s);
```

Display first 10 records

e.g.

```
Query<Product> q = session.createQuery("from product", product.class);
q.setMaxResult(10);
List<Product> list = q.list();
for(Product p:list)
    s.o.p(p);
```

Skip first 10 & display next 10 records

e.g.

```
Query<Product> q = session.createQuery("from product", product.class);
q.setFirstResult(10);
q.setMaxResult(10);
List<Product> list = q.list();
for(Product p:list)
    s.o.p(p);
```

pegination - 1

```
public static void pagination(Session session)
{
    int page = 0;
    int size = 50;
    while(true)
    {
    Query<Product> q = session.createQuery("from product", product.class);
    q.setMaxResults(size);
    q.setFirstResult(page * size);
    List<Product> list = q.list();
    if(!list.isEmpty())
    {
    System.out.println("page no: " + (++page));
    for(Product p:list)
        s.o.p(p);
    System.out.println("- - - - - - - - - -");
    }
    else
        break;
    }
```

pagination 2

```
public static List <product> pagination2 (session session,
int page)
{
    int size = 20;
    return session.createQuery ("From product", product.class)
        .setFirstResult ((page-1) * size)
        .setMaxResult (size)
        .list ();
}
```

HCQL

- HCQL stands for hibernate criteria query language
- by using HCQL we can perform complex validations.
- HCQL can be used for display purpose only, Hence we don't have to write any query
- by using HCQL, we can perform complex validations easily

e.g.

```
Criteria c = session.createCriteria(product.class);
List<product> list = c.list();
for (product p : list)
    System.out.println(p);
```

- to work with HCQL, we can use createCriteria() method of Session interface
- createCriteria() method returns instance of criteria interface
- methods of criteria interface

① list()

- list() method returns instance of list interface
- list() method can be used if multiple objects can be returned

② uniqueResult

- uniqueResult() method returns object or null
- uniqueResult() method can be used if single object can be returned.

③ add()

- add() method can be used to add restrictions into query
- to add restriction, we can use Restrictions class
- Restrictions is a class, which is located in org.hibernet.criteri-on package
- Restrictions class contains static method like eq(), ne(), lt() le(), gt(), ge(), betn(), in(), like(), and(), or(), not()

- add() method returns instance of Criteria interface

④ addOrder()

- addOrder() method can be used to add order by clause
- to add order by clause, we can use order class
- order is a class, which is located in org.hibernate.criterion pkg.
- order class static method like asc() & desc()
- return instance of criteria interface

⑤ setProjection()

- setProjection() method can be used for SQL aggregate func
- to use aggregate func, we can use projection class
- projections is a class, which is located in org.hibernate.criterion pkg.
- projections is a class contains sum(), avg(), min(), max(), count(), property() etc.
- return instance of criteria interface

⑥ setMaxResults()

- setMaxResults() method can be used set limit on records
- setMaxResults() method returns instance of criteria interface

⑦ setFirstResult()

- setFirstResult() method can be used to skip records
- setFirstResult() method returns instance of criteria interface.

⑧ setCacheable()

Find all product

e.g.

```
Criteria c = session.createCriteria(product.class);
List<Product> list = c.list();
For (product P:list)
    System.out.println(P);
```

OR

```
List<product> list = session.createCriteria(product.class)
.list();
for (product P: list)
    System.out.println(P);
```

Find product by id

e.g.

```
Criteria c = session.createCriteria(product.class);
c.add(Restrictions.eq("id", 21));
product P = (product) c.uniqueResult();
System.out.println(P);
```

OR

```
Product P = (product) session.createCriteria(product.class)
.add(Restrictions.eq("id", 21))
.uniqueResult();
System.out.println(P);
```

Find product by process less than

e.g.

```
Criteria c = session.createCriteria(product.class);
c.add(Restrictions.lt("price", 1000.0));
List<product> list = c.list();
for (product P: list)
    System.out.println(P);
```

OR

```
List<product> list = session.createCriteria(product.class)
.add(Restrictions.lt("price", 1000.0))
.list();
for(product p: list)
    System.out.println(p);
```

Find product by price between

e.g.

```
Criteria c = session.createCriteria(product.class);
c.add(Restrictions.between("price", 2000.0, 3000.0));
List<product> list = c.list();
for(product p: list)
    System.out.println(p);
```

OR

```
List<product> list = session.createCriteria(product.class)
.add(Restrictions.btn("price", 2000.0, 3000.0))
.list();
for(product p: list)
    System.out.println(p);
```

Find product by price not between

e.g.

```
Criteria c = session.createCriteria(product.class);
c.add(Restrictions.not(Restrictions.between("price", 1000.0,
                                            999000.0)));
List<product> list = c.list();
for(product p: list)
    System.out.println(p);
```

OR

```
List<product> list = session.createCriteria(product.class)
.add(Restrictions.not(Restrictions.between("price", 1000.0, 999000.0)))
.list();
for(product p: list)
    System.out.println(p);
```

Find product by id & type

eg.

```
Product p = (Product) session.createCriteria(product.class)
.add(Restrictions.and(Restrictions.eq("id", 19)).add(Restrictions.eq
("type", "smart watches")))
    .uniqueResult();
System.out.println(p);
```

OR

```
product p = (product) session.createCriteria(product.class)
.add(Restrictions.eq("id", 19))
.add(Restrictions.eq("type", "smart watches"))
.uniqueResult();
System.out.println(p);
```

Find product by id or type

eg.

```
Criteria c = session.createCriteria(product.class);
c.add(Restrictions.or(Restrictions.eq("id", 9)).add(Restrictions.
.eq("type", "smart watches")));
List<product> list = c.list();
for(product p: list)
    System.out.println(p);
```

or

```
List <product> list = session.createCriteria (product.class)
.add (Restrictions.or(Restrictions.eq ("id", 19))
.add (Restrictions.eq("type", "smart watches")))
.list ();
for (product p: list)
    System.out.println (p);
```

Find product by id in

eg.

```
List <Integer> id = new ArrayList();
id.add (10);
id.add (20);
id.add (30);
id.add (40);
id.add (50);
Criteria c = session.createCriteria (product.class);
if (!id.isEmpty())
    c.add (Restrictions.in ("id", id));
List <product> list = c.list();
for (product p: list)
    System.out.println (p);
```

Find product by name like

eg.

```
Criteria c = session.createCriteria (product.class);
c.add (Restrictions.like ("name", "%ABC%"));
List <product> list = c.list();
for (product p: list)
    System.out.println (p);
```

or

```
List <product> list = session.createCriteria( product.class)
.add (Restrictions.like ("name", "%mm%"))
.list ();
for (product p:list)
    System.out.println (p);
```

Display product order by price desc

eg.

```
List <product> list = session.createCriteria (product.class)
.addOrder (order.desc ("price"))
.list ();
for (product p: list)
    System.out.println (p);
```

Display Highest price of product

eg.

```
double price = (double) session.createCriteria (product.class)
.setProjection (projections.max ("price"))
.uniqueResult ();
System.out.println (price);
```

Display all names

eg.

```
List <String> names = session.createCriteria (product.class)
.setProjection (projections.property ("name"))
.list ();
for (String s: names)
    System.out.println (s);
```

Highest price product.

e.g.

Select * from product where price = (select max (price) from product);

```
product p = (product) session.createCriteria (product.class)
.add (Restrictions.eq ("price", session.createCriteria (product.class)
.setProjection (projections.max ("price"))
.uniqueResult ()))
.uniqueResult ();
System.out.println (p);
```

Find second highest price product-

e.g.

Select * from product where price = (select max (price) from product where price < (select max (price) from product));

```
product p = (product) session.createCriteria (product.class)
.add (Restrictions.eq ("price", session.createCriteria (product.class)
.add (Restrictions.lt ("price", session.createCriteria (product.class)
.setProjection (projections.max ("price"))
.uniqueResult ()))
.setProjection (projections.max ("price"))
.uniqueResult ()))
.uniqueResult ();
System.out.println (p);
```

* Fetching Technique.

- Fetching Technique is concept where we can fetch record as per requirement
- by using fetching technique we can enhance performance of appn
-

- Types of fetching Technique

① Lazy fetching Technique

- @onetoone (feten = FetchType.Lazy)
- in lazy fetching Technique relational records will not be loaded from begining

② Eaget fetching Technique

- @onetoone (fetch = FetchType.Eaget)
- in Eaget fetching Technique relational records will be loaded from the begining.

* Catching Technique

- Catching Technique is a concept when frequent search record stored in cache memory, if we request for the same hibernate will not execute query for same
- by using Catching Technique, we can enhance performance of appn

Types of catching Technique

① first level catch

- In hibernate first level catch is unable for session by default
- here session means one user

② 2nd level catch

- In hibernate 2nd level cache is disable by default
- 2nd level cache can be enable for sessionFactory

- here sessionFactory means throughout use
- in market many 2nd level cache available like JBoss, Oracle, SwarmCache, Ehcache

Hibernate 2nd level cache

step - 1 : Add dependencies
ehcache 2.10.6
hibernate ehcache 5.4.10

step - 2 : set properties
hibernate.cache.use_second_level_cache', "true"
hibernate.cache.region.factory_class = org.hibernate.cache.ehcache.internal.EhcacheRegionFactory
,

Step - 3 : add annotation
@Cache(usage = CacheConcurrencyStrategy.READ_ONLY)
class student
{
}

- NOTE: Second level cache doesn't work on HQL or HCQL

how to enable 2nd level cache with HQL or HCQL
Step - 1 : set property
hibernate.cache.use_query_cache = true

Step - 2 : add annotation
@Cacheable
class student
{
}

step - 3
use setCacheable() method.

Spring framework

* Spring framework
- Spring framework developed by Rod Johnson in 2003
- Spring framework can be used to make appn loosely couple.
- if applicatn is loosely couple it's become easy to build & test.

1) tightly couple appln
- tightly couple applⁿ is dependent of each object
- such applⁿ required alot of statement to make changes

e.g.

```
Package com.sim
public class AirtelSim {
  public void startAirtelSim(){
    System.out.println("welcome to airtel network");
  }
}
```

```
package com.sim
public class RelianceSim{
  public void startRelianceSim()
  {
    System.out.println("welcome to reliance network");
  }
}
```

```
package com.sim
public class IdeaSim{
public void startIdeaSim()
 {
   System.out.println("welcome to Idea network");
 }
}
```

```
Package com.mobile

import com.sim.Ideasim
public class Relaincemobile {
    private Ideasim r = new Ideasim();

    Public Reliancemobile()
    {
        StartMobile();
        r.startIdeasim();
    }
    Private void startMobile()
    {
        System.out.println("welcome to relaince mobile");
    }
}
```

ii) half couple appln

- by using interface we can achieve half couple appln

e.g.

```
Package com.sim;
public interface Sim{
    void startSim();
}
```

Then whose sim implements with Sim interface and change method that will be StartSim();

Reliance mobile

only change

```
public RelianceMobile {
    Private Sim r = new Reliance Sim();
```

You can only change in that of Sim.

iii) loosely couple appln

- to achieve loosely couple appln we can use spring framework

e.g.

bean xml

```
<beans>
    <bean id = "obj" class = "com.sim.Ideasim"> </beans>
</beans>
```

Reallace mobile

```
public class Reliancemobile{
    ApplicationContext context = new classpathxmlApplnContext("bean.xml");
    private Sim r = context.getBean(Sim.class);
```