Hibernate Framework

(1) The extension of JDBC is generally referred as Hibernate framework.

(2) Disadvantages of JDBC —

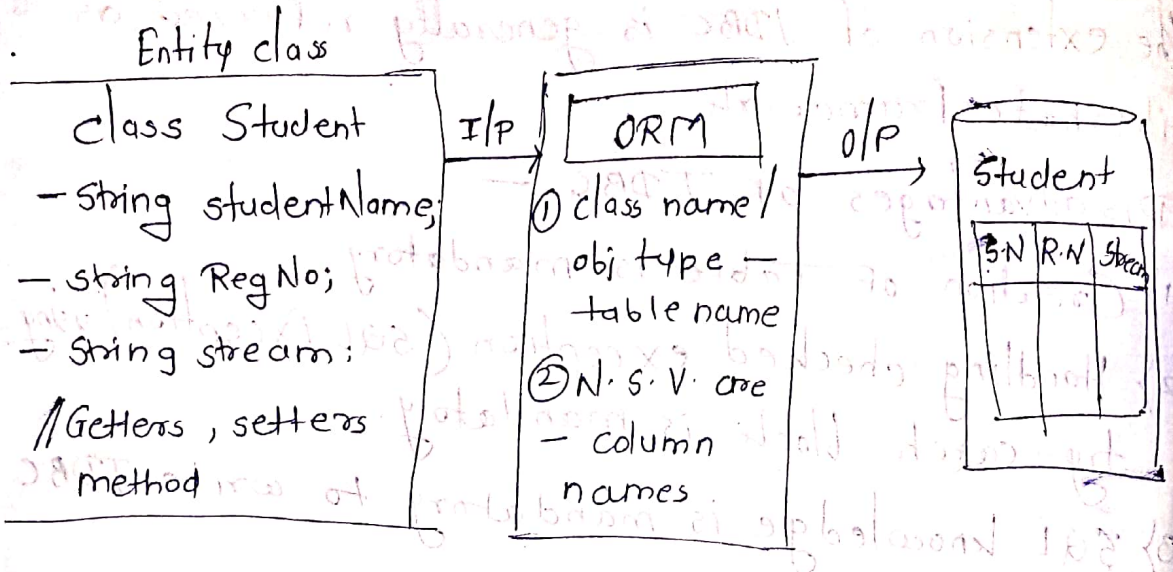
1) Creation of Table is mandatory.
2) Handling checked exception (SQL Exception) using try-catch block is mandatory.
3) SQL knowledge is mandatory to write JDBC programs.
4) All the database operation in JDBC program is performed by using queries.

(3) To overcome the above drawback of JDBC programmers prefers hibernate frame work.

(4) Advatages of Hibernate —

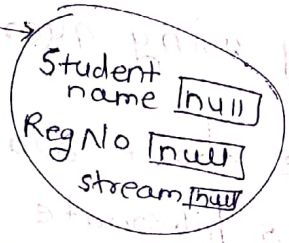
1) Creating the tables is optional
2) Handling the exception is optional b coz all exceptions are unchecked exceptions.
3) No SQL knowledge is necessary to write hibernate program.
4) To perform any database operations we prefer "Objects" [Insert, update, delete, select]

ORM [Object Relational Mapping]



Hibernate framework

Student s1 = new Student()



→ The process of creating the tables from Entity classes is technically Refered Object Relational Mappings.

→ ORM Follows two important rules -

① Entity classnames will be considered as Table names.

② Variable names will be considered as Column names.

③ The class which helps the programmer in creating the table is technically referred as "Entity classes".

Note - If ORM encounter same object type for the second & consecutive times, ORM will modify the existing table in the database.

Steps involved in writing Hibrnate program

1) Create JPA project and add connection
2) Copy and paste JPA jar files into the project & perform JAVA BUILD Path.
3) Create a package under src folder & create entity classes (JPA entity class).
4) provide the url of the specific database in Persistance.XML file → (Connection option → Transactiontype default → Populate from Connection)
5) Right click on the project & select JPA tools option & click on "Generate tables from entities option" which results in table creation in database.
6) Create a main class & write JPA template which includes 2 interfaces.
   ① EntityManagerfactory
   ② EntityManagerInterface.
7) Run the main class.

Ctrl+N → JPA project → project name → Target runtime (jre1.8.0_271) → next → next → platform → eclipselink 2.5.x → Disable Library Configuration → Add Connection → Connection (MySQL) → next → → new driver → Name |Type (MySQL 5.1) → JAR List → properties → change url name db → password.

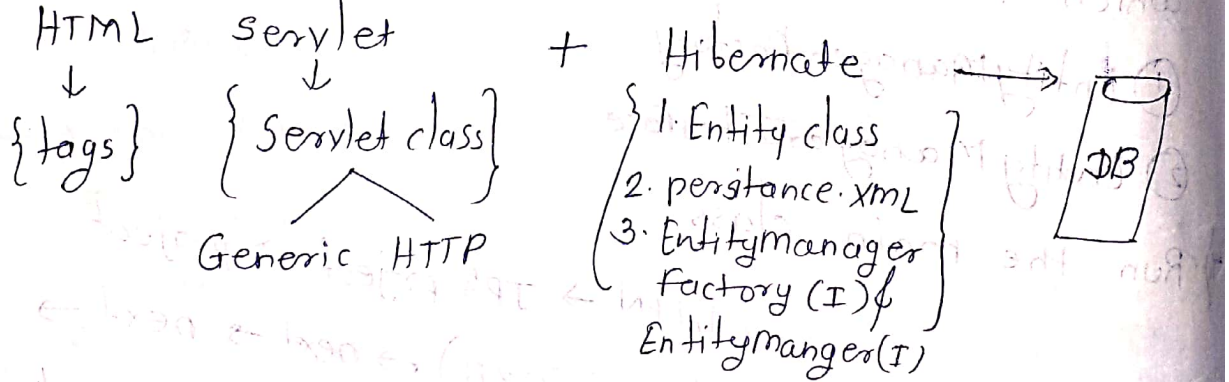
Combining hibernate with Servlet

→ Programmers will ~~ease~~ create Dynamic web page & make suitable changes so that Servlet and Hibernate coding can be combined.

→ By Default Dynamic web project does not contain persistance.xml file. In order to get persistance.xml file in Dynamic web project, programmer have to follow below steps.

Note: Right click on project & click on properties, Select Project Facets option & check JPA option click on further Configuration available option & followed by selecting connection click on OK, Apply and close option.

→ Into lib Folder of Dynamic web project, paste the JPA jar files & servlet API jar file.



Revese Engineering Process (R-EP)

Alt+shift+s

→ REP process helps the programmer in retriving the data in the form of objects [Tables → objects]

→ To perform REP, we have to make use of Find()
which takes 2 arguments

① Entity class name corresponds to the table from which we have to retrive the data.

② Value corresponding to primary key column.

Note: If given primary key value is not existing in the table then find method returns NULL.

→ If primary key column value is existing in the table then find method returns the object which is corresponding to the value passed.

Entity class

```
@Entity
+ class Emp
{
  @Id
  int empId;
  String empName;
  String job;
  //Getters Setters
```

EntityManagerFactory (I)

EntityManger (I)

```
begin()
Emp e1 = manager.find(Emp.class, 11);
Sopln(e1);
commit();
```

Emp

EmpId	Name	Job
10	abc	dev
11	xyz	dev

o/p as obj

hibranate file

ORM

REP

EmpId 0
name null
job null

Emp type

```
select *
from emp
where EmpId=11;
```

Syntax -

EntityClassName refvar = Manager.find (EntityclassName.class, primarykey / columnName)

Car c1 = manager.find (Car.class, "T50IAX5736");

→ Where Entity className refers to the corresponding table from which data has to be retrived.

Update Operation

→ To perform update operation, we have to perform reinitialization in java.

→ Before updating the data we have to use "find method" to target the specific record which has to be updated. find() will convert that specific record into an object.

→ After converting into the object, we can reinitialize the object by using Setters method.

```
cr: setCarColour("Black");
cr: setCarPrice("45678");
```

Delete Operation

→ To perform delete operation we have to make use of remove().

→ Firstly we have to use find() to target the specific record which has to be deleted.

→ find() will convert that specific recent into an object.

→ later, we pass that object as an argument for remove()

```
Manager.remove(cr)
```

Disadvantage -

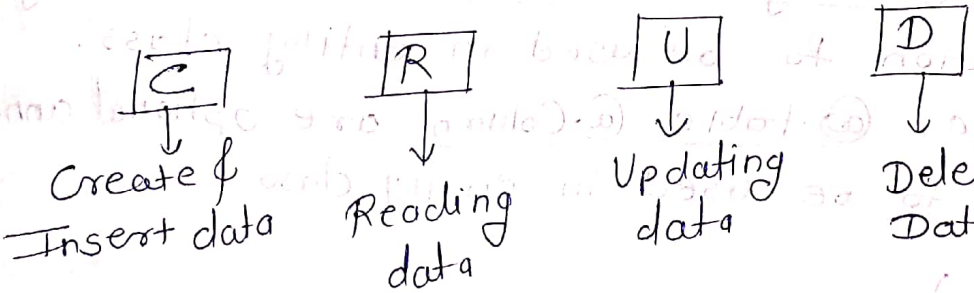
→ All the above methods which perform DB operations [Insert, update, Delete] will effect "only one record" beoz of "primary key column value".

To overcome from this drawback we have to make use of JPQL [Java persistance Query language]

CRUD Operation

→ CRUD stands for the database operations which can be performed by the programmers.

→ The abbrievation of CRUD is —



→ The CRUD operation is performed in JDBC by using queries. Insert, Select, Update & Delete query are used for this process.

→ The CRUD operations is performed in Hibernate by using objects along with inbuilt methods

Inbuilt Methods which are preferred are:

- persist(object)
- find(tableName.class, PRCV)
- setters()
- remove(object).

JPQL (Java persistance Query Language)

① @Table annotation is used to provide programmer defined table names. using this programmer can explicitly specify the table name which has to be created. [can edit table name]

② @Column is used to provide programmer defined column names [can edit columnnames] using this programmer can specify the column name which has to be created.

Note - @entity and @Id are mandatory annotation to be used in entity class. Where as @table @Column are optional annotation to be used in entity class.

Case(I)

@Entity -

```
public class Student {
@Id
private int studentId;
private String stuName;
private String stream;

//setters method
```

⟹

Id	Name	Stream

Case II

@Entity
@Table (name = Employee)
public class student
{
@Id
@Column (name = "EmpId");
private int stuId;
@ column (name = "EmpName");
private String stuName;
@ Column (name = "stream");
private String stream;
}

→ JPQL queries are the extension of SQL queries.
→ In SQL queries, we make use of table names and column names
Where as in JPQL queries, we make use of entity class names & variable names.
→ To use JPQL queries in the program we have to follow 3 steps.

① Define JPQL query

string jpql = "update Employee set empName='mno'
where empId in (12, 15);

② Inform hibernate framework above working with JPQL queries.

Query query = manager. createQuery (jpql);

③ Execute JPQL Queries.

query. executeUpdate();

To inform hibernate framework about JPQL queries working we have to call createQuery() as an argument we pass jpql query.

SQL	JPQL
Select * from emp where empId = 12;	update emp1 set empName = "mno" where employee Id = 15;
Select * from emp	from Student s where s.rollNumber = 101;
Delete from emp	delete from employee

Note - //step2 [Informing hibernate that we are working for JPQL]

```
Query query = manager.createQuery(jpql);
```

Placeholder →

?1 ?2

rollNo = :em

SQL	JPQL
1) SQL queries is given with table names & column names.	1) JPQL queries is given with entity class Names & variable name.
2) SQL queries can take runtime values & only by using place holders.	2) JPQL queries consist of two ways to pass run-time values.
3) In SQL queries we have specific setters method for each data type.	3) In JPQL queries we have one common method by name setparameters() to pass any type of data.
4) In SQL select query writing select clause is mandatory	4) In JPQL select query writing the select clause is optional.
5) In SQL select query giving the aliasing name for the table is optional	5) In JPQL select query, providing ref variable name (aliasing) for the entity class name & mandatory.
6) In SQL select query *token is valid	6) In JPQL select query providing * token is invalid

→ For executing JPQL select query we have to make use of getResultList().

It performs 4 functionalities internally.

(1) move the cursor from one position to another position in the table [BFR next()]

② Convert each record into an object.

③ add the object into an Arraylist collection & upcasted into List type.

④ return that list type to the programmers

Example:

✓ Select e1 from employee e1 → Valid - get all records.

✗ Select * from employee e1 → Invalid.

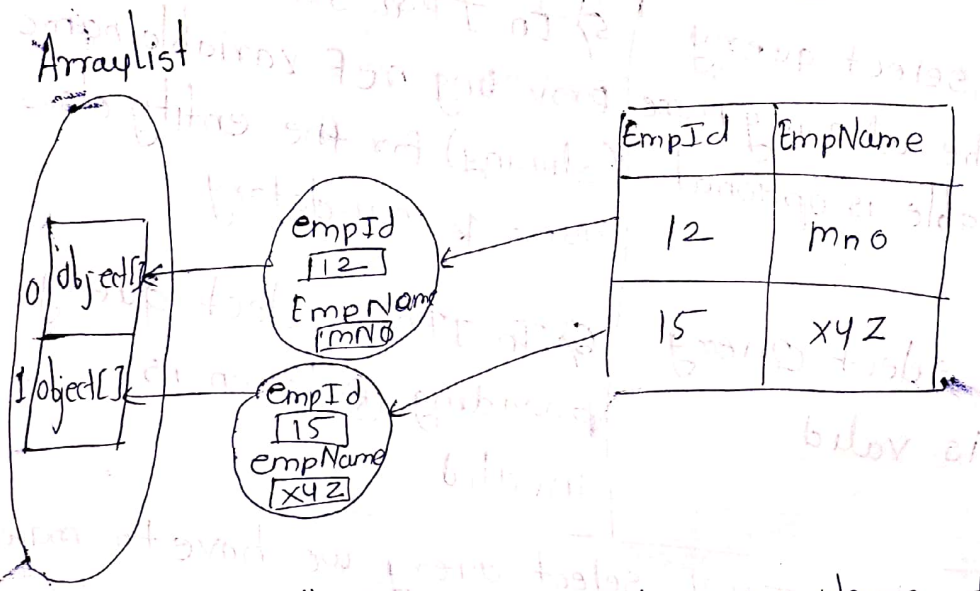
from employee e1 → Valid - get all records

Select e1 * from employee e1 → Invalid

✗ Select e1 from employee → r.v. not there

from employee → Invalid (r.v.)

Special Case of JPQL (Select Query)



String jpql = "select e.empId, e.empName from employee e;"

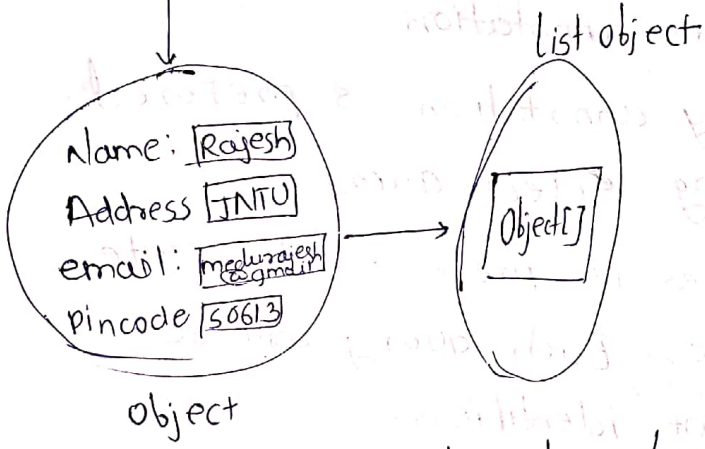
Query query = manager.createQuery(jpql);

List <Object[]> list = query.getResultList();

```
for (object[] obj : list) {
    for (object o1 : obj) {
        sopln(o1);
    }
}
```

userInfo → Name
→ Address
→ Email
→ Pincode
→ PhoneNumber

Name	Address	Email	Pincode	PhoneNumber
Rajesh	JNTUH	medurajeshlo @ gmail.com	506132	9515393252
Muradi	HYD	murali @ gmail.com	500092	9876543210



→ The two major disadvantages of JPQL queries is.

1) JPQL queries has the influence of SQL syntax.

2) If JPQL queries has to be accessed from one class to another class then we draw to create object. More no. of classes retrives more no. of object, which occupies more memory & Results in low process.

→ To overcome above drawbacks we used Named Queries.

Named Queries –

→ The main advantage of writing named queries is, we can avoid multiple object creation in order to access the queries in another class.

→ All the named queries must be declared only above methods "entity classes". so that it can be accessed in all other classes without creating 'object'.

→ The anotations used for working with Named queries are

1) Named-Query annotation.

2) @ Names queries annotation.

3) @ named query annotation is prefered only for writing Select query.

4) @ Names-queries is preferred to write multiple queries. Each query will be having different identifiers.

5) @ Named Query consist of two attributes

① Name attribute - Takes identifier to be used while execution.

② Query attribute - Takes the actual query which has to be executed.

JPQL	Named Queries
1) JPQL queries are return with in the main class.	1) Named Queries are written above the entity class.
2) To assess JPQL queries in other classes. It is necessary that we have to create an object.	2) To access named queries in another classes we need identifiers only
3) The method used for JPQL queries is createQuery();	3) For named queries the method used is created NamedQuery();

```
@NamedQueries({
    @NamedQuery(name="", query="");
    @NamedQuery(name="", query="");
})
```

JPQL → Disadv sql object class perfor)	NamedQuery → Disadv SQL syntax
1) In JPQL SQL syntax is necessary	1) In Named query SQL syntax is necessary
2) Query acts as platform	2) Query acts as platform

Criteria JPQL

→ 1) The advantage of using criteria is to avoid the influence of SQL syntax in the hibernate program.

→ Few interfaces involved in criteria JPQL

1. CriteriaBuilder (I) → Inform hibernate that working on criteria.
2. CriteriaQuery (I)
3. CriteriaUpdate (I)
4. CriteriaDelete (I)

- The helper methods involved for the above interfaces are -

1. getCriteriaBuilder()
2. createQuery()
3. createCriteriaUpdate()
4. createCriteriaDelete()

→ To inform hibernate above criteria working we have to make use of CriteriaBuilder (I) & getCriteriaBuilder()

→ Step 1 [inform hibernate about criteria]

```
CriteriaBuilder builder = manager.getCriteriaBuilder
```

→ Step 2

To perform select operation, we have to make use of criteriaQuery. the helper method for this interface is createQuery()

```
CriteriaQuery<GenericType> select = builder.
                                    createQuery(Car.class)
```

→ To perform update operation, we have to make use of CriteriaUpdate. The helper method for this interface is createCriteriaUpdate()

```
CriteriaUpdate<Car> update = builder.createCriteriaUpdate(Car.class)
```

→ To perform delete operation CriteriaDelete ① & helper method createCriteriaDelete()

```
CriteriaDelete<Car> delete = builder.createCriteriaDelete(Car.class)
```

JPQL

```
Select c  from Car c
   ⇓          ⇓
createQuery   Root
   ⇓          ⇓
createQuery   from()
```

→ after specifying the type of operation, we have to specify from which entity class data has to be retrived.

→ step: 3 [Specifying criteria class Name from we have to read date]

```
Root<Car> from = select from (Car.class)
```

→ after creating criteria we have to pass for createQuery() followed by its execution

→ the steps involved in writing criteria code is

step: 1 Inform hibernate about criteria working

step: 2 Specify the type of operation to be perform (select, update, delete)

Step3: Specify the entity class name on which operation has to be performed.

Step4: Specify the ~~entity class rel~~ condition on the entity class. [keeping where condition].

Note - Programmers prefer Criteria builder interface for two important reasons.

1. It is used to inform hibernate about criteria concept execution.
2. It also provides predefined methods which corresponds to sql operators.

→ Where clause of SQL is representated in java program by using "where()".

→ which must be called by using ref. variable name of type criteria query, criteria update & criteria delete.

→ In order to work with set clause in the criteria, we have to make use of set() which must be called by using ref. variable name of type criteriaUpdate (I)

→ In order to use variable names of entity class with in criteria, we have to make use of "get()", which must be called by using ref. variable name of Root type.

// Criteria for Reading Emp data whose salary is > 100 and also must be into empId of 10 to 20.

// Criteria for Reading the data whose name starts with 'A'. change their salary of those emp to 5000.

// Criteria for removing the records whose design is tester and also salary of that emp is 710000.

→ Select e from Emp where sal >1000 and EmpId b/w 10 and 20;

→ Update e set sal = 8000 where ename Like A%.

→ Delete e from emp where sal > 10,000 and Job = "tester";

JPQL

→ select e from Emp e where CId between 12 and 20) and like.("%.s");

Named Query

→ ⓐ Namedquery (name = "Identifiers" query="")

CreateQuery - where (builder. and (parameter.));

both equal in Natin

Association (HAS-A-Relationship)

→ Combining more than one class together is called as has-a-relationship. the minimum requirement to perform Has-A relationship we need more than one class which consi of primary key.

→ we have diff types of Has-A-Relationship

① One to One

② One to many

→ one → one association means one object will be associated with another object of other class.

→ One → many association means one object of one class is associated with many object of an other class.

→ Foreign key in the table can be create by using association concept of Java.

Persistence -

To store state of objects in database.

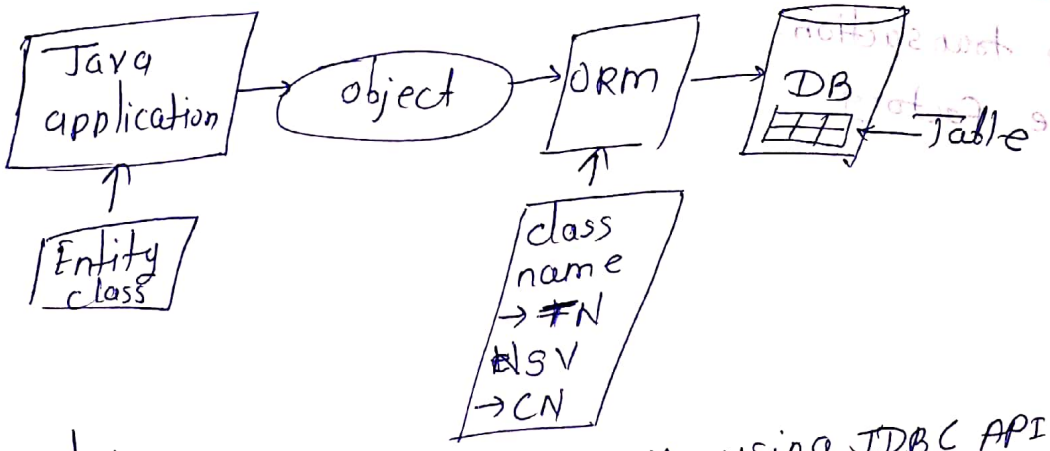
Hibernate → data persistence → relational databases.

Hibernate → implementation of Java Persistence API (JPA) specification. For easy usage in Java SE & Java EE

Hibernate → used to interact with DB.

ORM → simplifies data creation, data manipulation & data access.

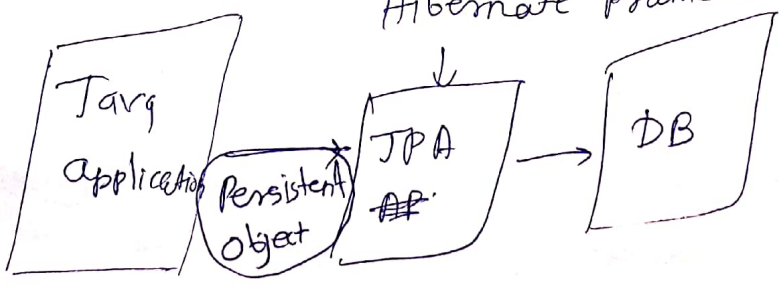
maps object → to store data → DB.



Note → ORM tool internally using JDBC API to interact with the database.

'javax.persistence ⟶ Java Persistence API

Hibernate Framework.



EntityManagerfactory - (I)

Persistence - (HC)

createEntitymanagerFactory("ProjectName");

encapsulate entity in I (I)

used to read, delete & write entity.

object referenced by an entity is managed by entity manager.

1. Creating an entitymanagerfactory.
2. obtaining an entity manager from factory
3. Intilizing an entity manager.
4. Persist used to store data in DB.
5. close transaction.
6. close factory.