Terminologies

What is Persistence?
The process of Storing and managing the data for a long time is called "Persistency".
To Perform Persistencey we use secondary devices like HDD,CD,DVD,ThumbDrivers etc
From the application perspective,we store the information inside variables/objects,but these objects would vanish once the application stops its execution.
To achieve persistency we use
1. File Operations(Storing it is HDD)
   java.io.*
2. Database(Storing it in the form of Table)
   java.sql.*

Terminologies associated with Persistency
==========
1. Persistence store
   It is a place/store where data will be saved and managed for a long time.
   eg: Files,DB s/w(MySQL,Oracle,PostgreSQL,.....)
2. Persistence data
   The data of Persistence Store is called 'Persistence Data'.
   eg: File info,DB tables and their records.
3. Persistence operation
   Insert,update,delete,select there are the operations which are performed on "Persistence Data".

4. Persistence Logic
   The logic which is written to perform Persistence operation is called as "Persistence Logic".
   eg: IO Streams logic(Serialization,DeSerialization)
       JDBC code(Technology name is JDBCAPI)
       hibernate code
       Spring JDBC
       Spring ORM(hibernate)
       Spring DataJPA(hot cake)

5. Pesistence technology/Framework
   The technology/Framework through which we write persistence logic is called 'PersistenceTechnology/Framework'.
   eg: JDBC(Technology)
       Hibernate(Framework/tool)
       SpringJDBC/SpringORM/SpringDataJpa(framework)

When we already have JDBC Techology, What is the need to go for framework/tool called ORM?

Limitations of JDBC
---
1. If we use JDBC to develop persistence logic,we need to write sql queries by following the syntax of "Database".
   DBQueries are specific to Database,this makes JDBC not portable across mulitple databases.
2. JDBC technology if we use and write a code, there would be a boiler plate code in our application.
   Boiler plate code => A code which would repeat in multiple parts of the project with no change/small change is called

boiler plate code.

CRUD
=====

1. Load and register the driver(automatic from JDBC4.X)
2. Establish the connection
3. Create PrepareStatement
4. Execute the Query
5. Process the ResultSet
6. Handle the Exception
7. Closing the Resource

Step1,2,3,6,7 boiler plat code becoz it is a common logic.

3. JDBC technology throws only one Exception called "SQLException',but it is a CheckedException which means u should have
handling logic otherwise code would not compile.
a. try{}catch(SQLException e){}
b. public static void main(String... args) throws SQLException{}

4. JDBC technology has only Exception class called "SQLException",so we don't have detailed hierarchy of Exceptions related
to different problems.

5. JDBC ResultSet object is not serializable, so we can't send it over the network, we need to use Bean/POJO to send the data
over the network by writing our own logic.

6. While closing the jdbc connection object, we need to analyze the code allot otherwise it would result in
"NullPointerException".
eg: Connection con = DriverManager.getConnection(url,user,password)
if(con!=null){.....}
closing the connection object should take place in "finally" block only.
To make the usage of AutoCloseable, we need to know the syntax of "try with Resource".

7. Java ====> OOP's based language
Assume we need to send Student object to database, can we write a logic of Database query at Object level if we use JDBC?
No, Not possible becoz DBqueries always expectes the value,but not the object directly.

8. JDBC doen't have good support of Transaction Management
a. local
b. global(no support in JDBC)

9. JDBC supports only positional parameters,it is difficult for the user to inject the values,It doesnot support namedparamaters.
String sqlInsertQuery = "insert into student(`name`,`email`,`city`,`country`) values(?,?,?,?)";
String sqlInsertQuery = "insert into student(`name`,`email`,`city`,`country`) values(:name,:email,:city,:country)";

10. To use JDBC, Strong knowledge of SQL is required.

11. JDBC does not supports versioning ,timestamp as inbuilt features
versioning:: keeping track of how many times record got modified.
timestamp:: keep track of when record was inserted and when lastly it was modified.

12. While developing persitence logic using JDBC, we can't enjoy oops features like
        a. inheritance
        b. polymorphism
        c. composition
    because jdbc does not allow objects as input values in sqlqueries.

                Solution to all these problems is use "ORM".

ORM:- (ORM stands for): Object Relational Mapping.
It is a theory concept used at database programming to perform operations like insert. Update, delete and select in object
format only ie. JDBC converts object to primitive data and SQL Query shuld be written by programmer using primitives,
which is not following OOPs.
        ORM says “Do not convert object data, do operations in OOPs. Format only”.

For this concept programmer should follow mapping rule. Given as
    1. className- Must be mapped with – tableName
    2. VariableName- Must be mapped with – columnName
            ** should be done by programmer using XML/Annotations concept.
            ** Then ORM convert Object ⇄ ROW
            ** Here , ORM only generates SQL Query

What is Framework?
Initially when java was introduced, it has only java api to develop standalone applications.
later group of api's are released under then name jee for developing distrubuted applications.
Developers faced so many problems while creating projects using java and jee api's.
jee is a largest set of api's, it was difficult for developers to remember so many classes and interfaces.
Devleoper needs to write so many boiler plate code to do integration of api's.
To overcome these problem framework was introduced by "thirdparty" vendors.
A framework provides "framework-api" which is an abstraction on top of "java and jee" api's.
Framework is not a new technology, it is an abstraction which is built on top of technology.
With frameworks we have the following facility
    a. Developer burden will be reduced
    b. Project can be deliverd to the client easily
    c. Project maintainence would be easy.

How many types of framework are available?
   There are 2 types of framework
        a. invasive framework
                => Developer has to extend his class from a superclass or interface supplied by the framework-api.
                => The developer class would be tightly coupled, so that class can't be moved to new framework
                        for execution.
                            eg: Servlet, EJB's,Struts

        b. non-invasive framework
                => Developer need not extend his class from a superclass or interface supplied by the framework-api.
                => The developer class would be loosely coupled, so

that the class can be moved to new framework
for execution.
eg: Hibernate(ORM tool) and Spring.

Hibernate(ORM tool/framework)
==========
Language(java)
Technology(JDBC)
Framework(ORM tool)

HibernateArchitecture
==========
refer:******.png

Hibernate Configuration File will provide all the configuration details like driver class name, driver URL,
Database User name , Database password,.... which we required to establish connection with database and
to setup JDBC environment.

Hibernate Mapping file will provide all the mapping details like Bean Class name and Database table name, ID property and
Primary key column , Normal Properties and Normal Columns,....

1) Client Application will perform the following three actions in Hibernate applications mainly.
2) Activating Hibernate Software, in this case, Hibernate Software will take all configuration details from hibernate
configuration file and Hibernate Software will set up the required JDBC Environment to perform database operations.
3) Prepare Persistence Object with the persistence data.
4) Perform Persistence Operation.

When Client Application perform persistence operation, Hibernate Software will perform the following actions.
1) Hibernate Software will take persistence method call and identify persistence Object.
2) Hibernate Software will take all mapping details from hibernate mapping file like database table name and all
column names on the basis of Persistence object.
3) Hibernate Software will prepare database dependent sql query on the basis of table names and column names and
with the persistence object provided data.
4) Hibernate Software will execute the generated database dependent sql query and perform the required persistence
operation.

Steps to prepare Hibernate Application:
1) Prepare Persistence Class or Object.
The main intention of Persistence class or object in Hibernate applications is to manage Persistence data which we want to
store in Database or by using this we want to perform the database operations like select, update, delete.

In Hibernate applications, to prepare Persistence classes we have to use the following Guidelines
1) In Hibernate applications Persistence classes must be POJO classes [Plain Old Java Object], they must not extend or
implement predefined Library.
2) In hibernate Applications Persistence classes must be public, Non abstract and non-final. Where the main intention to
declare persistence classes as public is to bring persistence classes scope to

Hibernate software in order to create objects.
Where the main intention to declare persistence classes as Non abstract is to allow to create Objects for Persistence classes
Where the main intention to declare persistence classes as Non final is to allow to extend one persistence class to another
persistence class as per the requirement.
3) In Persistence classes all Properties must be declared as per database table provided columns, where names are not
required to be matched, but, data types must be compatible.
4) In Persistence classes, all properties must be declared as private in order to improve Encapsulation.
5) In Persistence classes , we must define a separate set of setXXX () and getXXX () methods for each and every property
6) In persistence classes, we must declare all methods are public.
7) In Persistence classes, if we want to provide any constructor then it must be public and 0-arg constructor, because,
while creating object for persistence class Hibernate software will search and execute only public and 0-arg constructor.
8) In Hibernate applications , if we want to provide our own comparison mechanisms while comparing Persistence objects
then it is suggestible to Override equals(--) method.
9) In Hibernate applications, if we want to provide our own hash code values then it is suggestible to Overrid hashCode ()
method.
10) In Hibernate applications, we will use POJO classes, which are not extending and implementing predefined library, but,
it is suggestible to implement java.io.Serializable marker interface in order to make eligible Persistence object for
Serialization and Deserialization.
11) In Persistence classes, we have to declare a property as an ID property, it must represent primary key column in the
respective table.

2) Prepare Mapping File.
The main intention of mapping file in Hibernate applications is to provide mapping between a class,id property and
normal properties from Object Oriented Data Model and a table, primary key column and normal columns from
Relational data model.

In Hibernate applications, mapping file is able to provide the mapping details like Basic OR mapping, Component mapping,
inheritance mapping, Collections mapping, Associations mapping,
To prepare mapping file in Hibernate applications we have to provide mapping file name with the following format.
"POJO_Class_Name.hbm.xml".
The above format is not mandatory, we can use any name but we must provide that intimation to the hibernate software.
In Hibernate applications, we can provide any no of POJO classes, w.r.t each and every POJO class we can define a separate
mapping file.
refer: *****.hbm.xml

3) Prepare Hibernate Configuration File
The main purpose of hibernate configuration file is to provide all configuration details of hibernate application which includes
Jdbc parameters to prepare connection , Transactions configurations,Cache mechanisms configurations, Connection pooling

configurations,.....
In Hibernate applications, the standard name of hibernate configuration file is "hibernate.cfg.xml", it is not fixed,
we can provide any name but that name must be given to Hibernate software.
In Hibernate applications, we are able to provide more than one configuration file , but, for each and every database, that is,
in hibernate applications if we use multiple databases then we are able to prepare multiple configuration files.
To prepare hibernate configuration file with basic configuration details we have to use the following XML tags.

refer: hibernate.cfg.xml

4) Prepare Hibernate Client Application
The main intention of Hibernate Client application is to activate Hibernate Software, creating persistence objects and
performing Persistence operations.
To prepare Client Application in hibernate applications we have to use the following steps.

1. Create Configuration class object
2. Create Session Factory object
3. Create Session Object
4. Create Transaction object if it is required.
5. Perform Persistence operations
6. Close Session Factory and Session objects.

1. Create Configuration class object
In Hibernate, the main intention of Configuration object is to store all the configuration details which we provided in
hibernate configuration file.
To represent Configuration object Hibernate has provided a predefined class in the form of "org.hiberante.cfg.Configuration".
To create Configuration class object we have to use the following constructor from Configuration class.

```
public Configuration()
```

EX: Configuration cfg = new Configuration();
If we use the above instruction in Hibernate applications then we are able to get an empty Configuration object in heap
memory, it will not include any Configuration details.

If we want to store Configuration details from Configuration file we have to use either of the following methods.
1. public Configuration configure()
This method will get configuration details from the configuration file with the name hibernate.cfg.xml.
2. public Configuration configure(String config_file_Name)
This method can be used to get configuration details from hibernate configuration file with an name, it will be used
when we change configuration file name from hibernate.cfg.xml file to some other name .
3. public Configuration configure(File file)
This method can be used to get configuration details from a file which is represented in the form of java.io.File class object.

```
public Configuration configure(URL url)
```

This method can be used to get Configuration details from a file which is available in network represented in the form of
java.net.URL .
EX: Configuration cfg = new Configuration();

```
cfg.configure();
```

When we use configure() method then Hibernate Software will search for

hibernate.cfg.xml file, if it is available then
Hibernate software will load the content of hibernate.cfg.xml file, parse it and read content from configuration file to
Configuration object.

2. Create Session Factory object:
In Hibernate, the main intention of Session Factory object is to manage Connections, Statements, Cache levels, ... and it able
to provide no of Hibernate Session objects.
To represent Session Factory object Hibernate has provided a predefined interface in the form of "org.hibernate.Session Factory".
To get Session Factory object we have to use the following method from Configuration class.

```
            public Session Factory buildSessionFactory()
EX: SessionFactory sf = cfg.buildSessionFactory();
```

In Hibernate applications, if we use multiple Databases then we have to prepare multiple Configuration files,
multiple Configuration Object, w.r.t this, we have to prepare multiple Session Factory objects.
Session Factory object is heavy weight and it is thread safe upto a particular Database, because, it able to allow more than
one thread at a time.

3. Create Session Object:
In Hibernate, for each and every database interaction a separate Session will be created.
In Hibernate, Session is able to provide no of persistence methods in order to perform persistence operations.
To represent Session object, Hibernate has provided a predefined interface in the form of "org.hibernate.Session".
To get Session object, we have to use the following method from Session Factory.

```
            public Session openSession()
EX: Session s =sf.openSession();
```

In Hibernate, Session object is light weight and it is not thread safe, because, for each and every thread a separate Session
object will be created.

Difference b/w Session Object vs SessionFactory object?

SessionFactory

- It is a heavy weight object containing multiple other objects like DataSource,Dialect,TxFactory,generators etc.
- It is a immutable Object.
- ThreadSafe Object is by default.
- Created by using buildSessionFactory() method which is designed based on "builder" design pattern.
- Long Lived Object of the application.
- It maintains Level-L2 cache.

Session

- It is a Light Weigth Object(con++)
- It is a mutable Object.
- Not ThreadSafe by default.
- Created by using openSession() which is designed based on 'factory' design pattern.
- Short lived Object of the application.
- It maintains Level-L1 cache.

4. Create Transaction Object:
Transaction is a unit of work performed by Front End applications on Back end systems.
To represent Transactions, Hibernate has provided a predefined interface in the form of "org.hibernate.Transaction".

To get Transaction object we will use either of the following methods.
1. public Transaction getTransaction()
It will return Transaction object with out begin, where to begin Transaction we have to use the following method.
public void begin()

2. public Transaction beginTransaction()
It will return Transaction and begin Transaction.
In Hibernate applications, after performing persistence operations we must perform either commit or rollback operations
inorder to complete Transactions, for this, we have to use the following methods from Transaction.

5. Perform Persistence Operations:
In Hibernate applications, to perform persistence operations Session has provided the following methods.

Through hibernate we can perform
a. SRO(Single Row Operation)
a. save()/persist() ========> Insert Operation
b. get()/load() ========> Select Operation
c. update()/saveOrUpdate() => Update Operation
d. delete() ==========> Delete Operation
b. Bulk operation(working with mulitple records)
a. HQL/JPQL
b. NativeQuery
c. Criterian API

Language(java)
Technology(JDBC)
Framework (ORMTool-----> Hibernate)

ORM ===> Do operation only in objects
Hibernate(Object<----> ROW)
1.SRO(Single Row Operation)
a. save()/persist()
b. get()/load()
c. update(),saveOrUpdate()
d. delete()
2.Bulk operation(working with more than one Row)
a. HQL/JPQL
b. NativeQuery
c. Criterian API(mostly used by java developers)

Property
---
hibernate.hbm2ddl.auto
ddl=Data definition language(create /alter/drop in SQL)
it has four possible value.
Those are:-
a. validate (default value)
b. create
c. update
d. create-drop

A. validate:- In this case hibernate creates no tables programmer has to create or modify tables manually.
It is only default value.
B. create:- hibernate creates always new tables, if table exist then it will be drop.
C. update:- It creates new table if table not exits else uses same tables.
This attribute is commonly used.
D. create-drop:- This option is used for testing process not in development,
it creates a new table an performs operation and at last table will be drop.
It is just for POC(proof of concept) or for learning purpose.

Hibernate Persistence operation
==========
Objects used in hibernate
a. Configuration
b. SessionFactory(heavy weight)
c. Session
d. Transaction

1.SRO(Single Row Operation)
a. save()/persist()
b. get()/load()
c. update(),saveOrUpdate()
d. delete()

a. save()/persist() => These methods are used to perform insert operation.
What is the difference between save() method and persist() method?
Ans:
In Hibernate applications, save() method can be used to insert a record into the

Database table and it will return
Primary Key value of the inserted record.
This method is from hibernate api.
public Serializable save(Object obj)throws HibernateException

In Hibernate applications, persist() method can be used to insert a record into database table and it will not return any value.
This method is from JPA specification.
public void persist(Object obj)throws HibernateException

Note:
@DynamicInsert(value=true) and @DynamicUpdate(value=true)
=> It is used to generate the dynamic query, based on the fields used in the Entity Object
=> If we want our table name and column name to be same as entity name and field name then no need to use
@Table(name='') and @Column(name='',length='')
=> During the creation of SessionFactory object, by refereing to mapping information hibernate will create
pregenerated sql queries for insert,update,delete,select by using all the fields of the entity class.
=> If we want to avoid that and if we want query to be generated based on the field injection we do on the entity class
then we need to use @DynamicInsert(value='true') or @DynamicUpdate(value='true')

Performing select operation in hibernate
---
b. get()/load()

What are the differences between get(-) method and load(-) method?
Ans:
1. get() method can be used to retrive a record from database table if the record is existed.
If the required record is not existed then get() method will return null value.
public Object get(String class_Name, Serializable pk_Val)
public Object get(Class class_Type, Serializable pk_Val)
2. get() method is able to perform eager or early loading, that is, it will interact with database directly and it will retrive data and return to Hibernate application in the form of Object on the method call.

1. load() method can be used to retreive a record from database table if the record is existed.
If the required record is not existed then load() method will rise an Exception like HibernateException.
public Object load(String class_Name, Serializable pk_Val)
public Object load(Class class_Type, Serializable pk_val)
2.load() method will perform Lazy or late Loading , that is, when we access load() method then a duplicate object will be
created with the primary key value without interacting with database(proxy object).
When we use other properties of the Object then only it will fetch data from database table and return that data to
Java application.

update(),saveOrUpdate()
---
updating a record can be done in 3 ways
a. update total object(not prefered)
remember the object exists with the id and update the entire object.
if the id doesn't exists it would return 'OptimisticLockException'.
b. load the record and update(very useful for partial updation)
c. load and modify the record without using update()[Synrhonization would exist b/w row and object]

saveOrUpdate()
This method call would first perform
a. select operation
if record found then it performs update operation, otherwise it performs insert operation

What is the difference between update() method and saveOrUpdate() method?
Ans:
Where update(-) method will perform updation on a record in database table if the specified record is existed otherwise
it will rise an Exception.
public void update(Object obj)throws HibernateException

Where saveOrUpdate(-) method will insert the specified record in database table if the specified record is not existed .
If the specified record is existed in database table then it will update the record.
public void saveOrUpdate(Object obj)throws HibernateException

delete()

It can be done in 2 ways
a. perform deletion by supplied id value directly
It generates sql select query, if the record exists it would generate delete query, otherwise it won't
generate delete query and it won't throw any Exception.
b. load and delete the record[prefered]

@DynamicUpate(value=true)
It is used to generate the update query only for the fields on which the setter injection is
performed.

Entity object fields are
  sid,sname,sage,saddress

The entity object setter method used is main method is
      student.setSaddress("RCB");

Query generated is
```
update
        Student
    set
        saddress=?
    where
        sid=?
```

Note: Within the transaction, persitence object would be synchronized to a row in the database.

Alternative ways available to configure ORM
---
a. using Programmatic approach(Pure java approach)
b. using Properties file(application.properties)
c. using XML approach(hibernate.cfg.xml)

1) Programmatic Approach
In Programmatic approach, to provide configuration details, first we have to create Configuration class object then we
have to set all hibernate properties to Configuration class object explicitly by using the following method.

```
public void setProperty(String prop_Name, String prop_Val)
```

To add mapping file name and location to Configuration file we have to use the following method.

```
public void addResource(String mapping_File_Name)
```

Note: If we are using annotations in place of mapping file then we have to use the following method to add annotated class.

```
public void addAnnotatedClass(Class class)
```

In Programatic approach, if we want to change configuration details like connection properties or database properties,....
then we have to perform modifications in JAVA code, if we perform modifications in JAVA code then we must
recompile the java application , it is not suggestible in enterprise applications.
To overcome the problem we have to use Declarative approach.

2) Declarative approach:
In Declarative approach, we will provide all configuration details in either properties file or in XML file then we will send
configuration details to Hibernate application.
There are two ways to provide configuration details to Hibernate Applications in Declarative Approach.
1) By using properties file.
2) By Using XML file.

1) Using properties file in Declarative Approach:
In This approach, to provide all hibernate configuration details , we have to

declare a properties file with the default name
"hibernate.properties" under "src" folder . In this context, when we create Configuration class object ,
Hibernate Software will search for the properties file with the name "hibernate.properties" and get all the configuration
details into Configuration class object.
In this approach we must add mapping file explicitly to Configuration File by using the following method.

```
public void addResource(String mapping_File-Name)
```

Note:
If we change name and location of properties file from hibernate.properties to some other "abc.properties" then we have
to give that intimation to Hibernate software, for this, we have to create FileInputStream and Properties class object
then we have to set Properties object to Configuration class object by using the following method.

```
public void setProperties(Properties p)
```

Note:
In hibernate applications, if we use "properties" file then we are able to provide only hibernate connection propeties,
dialect propeties,... , but, we are unable to provide mapping properties,... through properties file, to provide mapping properties
we have to use java methods like addResource(--) or addAnnotatedClass(--) methods from Configuration class.

In Hibernate applications, if we want to provide all configuration details in declarative manner then we have to use XML file approach.

If we want to use xml file to provide all configuration details then we have to use configure() method to get all configuration
details from xml file. Here configure() method will search for xml file with the default name hibernate.cfg.xml under src folder inorder to get configuration details.

If we change the default name and location of hibernate configuration file then we have to pass that name and location to configure(--) method as parameter.

Note:
if we place all the above
approach(hibernate.cfg.xml,hibernate.properties,programmatic approach) then which one will be applied to set up the jdbc environment by Hibernate?
=> cfg.configure() is called in the code after setting the properties then hibernate.cfg.xml file will be used,otherwise programming
logic(cfg.setProperty()) will be used.

Primary Key Generation Algorithms in Hibernate

Primary key is single column or Collection of columns in a table to recognize the records individually.
In Database applications, to perform the database operations like retriving a record, updating a record, deleting a record,....
we need primary key and its value.
In Database applications, we are unable to give option to the users to enter primary key values , because there is no
guarantee for the data entered by the users whether it is unique data or not, but, in Database tables only

unique values are accepted by primary key columns.

To give guarantee for uniqueness in primary key values we have to use Primary key generation algorithms.

Almost all the Persistence mechanisms like Hibernate, Ibatis, Open JPA, Toplink,.... are having their own implementation for primary key generation algorithms.

Hibernate has provided support for the following primary key generation algorithms inorder to generate primary key values.
1. assigned(default)
2. increment
3. sequence
4. identity
5. hilo(removed from HB5.X)
6. native.
7. seq-hilo
8. select
9. UUID
10.GUID
11. foreign

Hibernate has represented all the above primary key generation algorithms in the form of a set of predefined classes provided
in "org.hiberante.id" package.
If we want to use any of the above algorithms in Hibernate applications then we have to configure that algorithms in
hibernate mapping file by using the following tags.

There are 3 types of generators in hibernate
1. Hibernate supplied generators
   a. @GeneratedValue
   b. @GenericGenerator
2. JPA supplied generators
   a. @GeneratedValue
3. Custom generator(** realtime projects)

1. assigned
This algorithm is default primary key generation algorithm in hibernate applications, it will not required configurations in
mapping file.
This algorithm will not have its own mechanism to generate primary key value , it will request to Client Application to
provide primary key value explicitly.
This algorithm is able to support for any type of primary key values like short, int, long, String,.....
This algorithm is supported by almost all the databases like Oracle, MySQL, DB2, ...
This algorithm is not required any input parameter.
This algorithm is represented by Hibernate in the form of a predefined class "org.hibernate.id.Assigned".

```
@Id
@GenericGenerator(name = "gen1", strategy = "assigned")
@GeneratedValue(generator = "gen1")
private Integer eid;
```

2.
increment

This primary key generation algorithm is able to generate primary key value by incrementing max value of the primary key column.
New_Val = max(PK_Column)+1
This alg is able to generate primary key values of the data types like short, int, long,...
This alg is not required any input parameter to generate primary key values.
This alg is supported by almost all the databases which are supporting numeric values as Primary key values.
This alg is representned by Hibernate Software in the form of a short name "increment" and in the form of a predefined class like
"org.hibernate.id.IncrementGenerator".

```
@Id
@GenericGenerator(name = "gen1",strategy = "increment")  // To specify details HB Specific generator
@GeneratedValue(generator = "gen1") // To apply generators on @Id field
private Integer eId;
```

3. sequence
This primary key generation algorithm is able to generate primary key value on the basis of the sequence provided by the underlying Database.
This alg is able to generate primary key values of the data types like short, int, long,...
This alg required "sequence" input parameter with the sequence name as value inorder to generate primary key value.
If we have not provided any sequence name as input parameter then this alg is able to take "hibernate_sequence" as default sequence name,
here developers must create "hibernate_sequence" in database explicitly.
This alg is supported by almost all the databases which are supporting sequences like Oracle, DB2,....
This alg is represented by Hibernate Software in the form of a short name like "sequence" and in the form of a predefined class like
"org.hibernate.id.SequenceGenerator".

1>

```
@Id
@GenericGenerator(name = "gen1", strategy = "sequence",
                            parameters ={
                                                @Parameter(value =
"eid_seq",name="sequence_name")
                                            }
                            )
@GeneratedValue(generator = "gen1")
private Integer eid;
```

2>

```
@Id
@GenericGenerator(name = "gen1", strategy = "sequence")
@GeneratedValue(generator = "gen1")
private Integer eid;
```

4.
identity
This alg is able to generate primary key values on the basis on the underlying database table provided identity column.

Note:

Identity column is a primary key Column with "auto_increment" capability.
This alg is able to provide the primary key values of the data types like short, int, long,....
This alg is not required any input parameter.
This alg is supported by almost all the databases which are supporting Identity column.

EX: MySQL.

To represent this alg, Hibernate has provided "identity" as short name and "org.hibernate.id.IdentityGenerator" as predefined class.

```
@Id
@GenericGenerator(name = "gen1",strategy = "identity")  // To specify details HB Specific generator
@GeneratedValue(generator = "gen1") // To apply generators on @Id field
private Long policyId;
```

5.native
This alg is able to generate primary key value by selecting a particular primary key generation alg depending on the database
which we used.
This alg is not having its own alg to generate primary key value.
It will select "SequenceGenerator" alg if we are using Oracle database, "IdentityGenerator" alg if we are using MySQL database and "TableHiLoGenerator" if we are using some other database which is not supporting SequenceGenerator and IdentityGenerator.
This Primary key generator is able to generate primary key values of the data types like short, int, long,...
This primary key generator is supported by almost all the databases.
To represent this mechanism, Hibernate has provided a short name in the form of "native", Hibernate has not provided any
predefined class.

JPA(Java Persistence API) generators
---
1. These are given by SUNMS JPA specification
2. It will work with all ORM Frameworks
3. We can specify the generators directly using @GeneratedValue(supplied by JPA)
4. It give supports to 4 generators
a. identity b.sequence c.table d.auto

1. IDENTITY:
This value of GenerationType enum will represent IdentityGenerator or "identity" primary key generation algorithm to generate
primary key value on the basis of the underlying database provided identity column.

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Integer eid;
```

Output
```
create table Student (
       sid integer not null auto_increment,
        saddress varchar(255),
        sage integer,
        sname varchar(255),
        primary key (sid)
) engine=InnoDB
```

2. SEQUENCE :
This constant from GenerationType enum is able to represent SequenceGenerator or "sequence" primary key generation algorithm
inorder to gtenerate primary key value on the basis of the sequence which we defined in database.
To configure "sequence" name we have to use "@SequenceGenerator" annotation with the following members.
1. name: It will take logical name of the @SequenceGenerator.
2. sequenceName: it will take "sequence" name provided by underlying database.
To apply @SequenceGenerator to @GeneratedValue annotation we have to use "generator" member in "@GeneratedValue" annotation.

eg#1.
```
@Entity
public class Student {
```

```
      @Id
      @GeneratedValue(strategy = GenerationType.SEQUENCE)
      private Integer sid;
}
Hibernate: create sequence hibernate_sequence start with 1 increment by  1
create table Student (
       sid number(10,0) not null,
        saddress varchar2(255 char),
        sage number(10,0),
        sname varchar2(255 char),
        primary key (sid)
)


eg#2.(sequence already exists and if we want to use that sequence)
@Entity
public class Student {
      @Id
      @SequenceGenerator(name = "gen1", sequenceName = "JPA_SID_SEQ", initialValue
= 5, allocationSize = 5)
      @GeneratedValue(generator = "gen1", strategy = GenerationType.SEQUENCE)
      private Integer sid;
}

Hibernate:
    select
        JPA_SID_SEQ.nextval
    from
        dual

eg#3.(hibernate creates a sequence called SID_SEQ_GEN with initialValue=1,
allocationSize=50)
@Entity
public class Student {

      @Id
      @SequenceGenerator(name = "gen1", sequenceName = "SID_SEQ_GEN")
      @GeneratedValue(generator = "gen1", strategy = GenerationType.SEQUENCE)
      private Integer sid;
}

Hibernate: create sequence SID_SEQ_GEN start with 1 increment by  50
Hibernate:
    select
        SID_SEQ_GEN.nextval
    from
        dual

Note: If we are using MySQL,best suited is  "GenerationType.IDENITY".
        If we are using Oracle,best suited is "GenerationType.SEQUENCE".
        If we are not aware of what databse algorithm supports to generate primary
key then go for "GenerationType.AUTO".

CUSTOM GENERATORS IN HIBERNATE:-
To specify our own format for PrimaryKey use custom Generators concept.
Ex:- Student ID: SAT-85695
Employee ID: EMP-5754
PAN CARD ID: DYBPM1887k
```

All are used as Primary key and they are implemented using Custom Generators

Steps to implement custom Gneratores
1. Create one new public class with any name and any package.
2. Implement above class with interface IdentifierGenerator(org.hibernate.id)
3. Override method generate() which returns PrimaryKey value as
java.io.Serializable
4. In model class use @GenericGenerator and provide strategy as your full class
name.
5. Finally in test class , create model class object and save.

```
@Entity
public class Student {
	@Id
	@GenericGenerator(name = "gen1", strategy =
"in.ineuron.idgenerator.StudentGenerator")
	@GeneratedValue(generator = "gen1")
	private String sid;

}
public class StudentGenerator implements IdentifierGenerator {
	@Override
	public Serializable generate(SharedSessionContractImplementor arg0, Object
arg1) throws HibernateException {
		System.out.println("StudentGenerator.generate()");
		String id = "IN-01";
		return id;
	}

}
Hibernate:
    insert
    into
        Student
        (saddress, sage, sname, sid)
    values
        (?, ?, ?, ?)
Object inserted to  the database with the id :: IN-01
```

Task
Create an application for the customer, where customer id should be
c001,c002,....c009, c010,........c099, c100.....c199,.....
refer:: HB-12-Hibernate-CustomGeneratorsApp

Samplecode

```
import java.io.Serializable;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Random;
import org.hibernate.HibernateException;
import org.hibernate.engine.spi.SessionImplementor;
import org.hibernate.id.IdentifierGenerator;
public class MyGen implements IdentifierGenerator {
	@Override
	public Serializable generate(SessionImplementor session, Object object)
throws HibernateException {
		String date =new SimpleDateFormat("yyyy-mm-dd").format(new Date());
```

```
            int   num=new Random().nextInt(1000);
            String Prefix1 = "Ineuron-";
            String Prefix2 = "HB";
            return Prefix1+date+Prifix2+"-"+num ;//Ineuron-06-03-2023-HB-123
      }
}
```

Composite-id primary key in hibernate
==========
In a database we use a primary key to identify one row uniquely among multiple rows stored in the table.
In most of the cases single column of a table in enough to identify unique row in a table.
In some cases, we need combination of two or more columns is needed to uniquely identify a row, it is called as "composite-key".
To represent composite-key we need to use annotation called "@Embeddable(It is optional),@EmbededId".

Steps : To implement Composite PrimaryKey:-
1. Define one new class used as Primary key data Type, it must implement java.io.Serializable.
2. Move (define) all variables in above class which are involved in composite Primary key creation.
3. On top of this class add @Embeddable annotation.
4. Make HAS-A relation between model Class and DataType class
5. Apply @EmbeddedId Annnotation over HAS-A relation.
            *** use hbm2ddl.auto=create(for first time)
6. Write test class and create object and save to Entity.

refer:: HB-13-HibernateCompositeIdApp

Working with Date values
==========
While dealing with DOB,DOM,DOJ,billDate etc we need to insert and retrieve the value
Hibernate provides abstraction towards inserting the date value,we need not to do multiple conversions as how we did in JDBC.

JDBC Approach
==========
```
Enduser
   |
Stringvalue
   |
SimpleDateFormat.parse()
   |
java.util.Date(C)
   |
java.sql.Date(C)
   |
pstmt.setDate(date)
```

To work with Date and Time values, just take the type of properties from jdk8 api,no need to specify extra annotations.

```
      LocalDate doj;
      LocalDateTime dob;
      LocalTime  dom;
```

refer:: HB-14-HibernateDateOperation

Versioning
==========
=> To keep track of how many times object/record is loaded and modified using hibernate.
=> It generates the special column of type numeric based special number property of Entity class to keep track of modification
=> This special property will be initialized to zero and it will incremented for every updation.
=> To configure this property we need to use @Version annotation.

eg:
```
@Version
private Integer versionCount;
```

refer:: HB-15-HibernateVersioningApp

Tommo and day after tommo(09/03/2023) => holiday
Extra class :: 11/03/2023(saturday) ==> Timings 7.30pm to 11.00pm
==========

Object Time Stamping
==========
=> It allows to keep track of Object is saved and Object is lastly updated.
usecase: Keep track of when the back account is opened and lastly updated.

=> We use 2 annotations like
1. @CreationTimeStamp => To hold object/record creation date and time.
2. @UpdateTimeStamp => To hold when the object is lastly modified /updated.

```
@CreationTimeStamp
LocalDateTime openingDate;

@UpdatetimeStamp
LocalDateTime lastUpdate;
```

=> In annotation driven environment we can apply both Versioning and TimeStamp feature.

refer:: HB-16-HibernateTimeStampingApp

Caching in hibernate
==========
=> It is most important feature/benefit of hibernate.
=> It reduces the no of trips b/w application and database, it improves the performance of the application.
=> Hibernate maintains cache at 2 levels, so it improves the peformance.
=> When an application wants the data from the database, then hibernate looks for the object at the L1-cache,if not then
it looks for the object at L2-cache and if not then it would go for "database".

Methods associated with cache are
a. session evict(object) => To remove particular object from cache.
b. session.clear() => To clean the cache or to remove all objects from cache.
c. session.contains(object) => To check particular object exists in the cache.

L1Cache(inbuilt cache)
This cache is associated with session object, we can't disable it as it is inbuilt present in Session object.

refer:: HB-17-HibernateCachingApp

L2Cache(configurable cache)
It is associated with SessionFactory object,so it is called as "GlobalCache".
It is configurable cache which we can enable/disable.
We have multiple providers supporting L2Cache
a. EH_cache(It support inmemory and diskcaching)
b. swarmcache
c. oscache

L2Cache Concurency strategies
==========
1. read-only : caching will work for read only operation
2. nonstrict-read-write : caching will work for read and write but only at a time.
3. read-write : caching will work for read and write simultaneously.

4. transactional : caching will work for transaction.

      In Annotation driven environment, keep the annotation at the top most class level

```
            @Cache(usage = CacheConcurrencyStrategy.READ_ONLY)
```

hibernate.cfg.xml

```
  <!-- For each cache -->
  <property name="cache.use_second_level_cache">true</property>
  <property name="net.sf.ehcache.configurationResourceName">ehcache.xml</property>
  <property name="hibernate.cache.region.factory_class">org.hibernate.cache.ehcache.EhCacheRegionFactory</property>
  <property name="hibernate.cache.use_query_cache">true</property>
```

ehcache.xml

```
<ehcache>
    <diskStore path="java.io.tmpdir"/>
    <defaultCache
        maxElementsInMemory="100"
        eternal="false"
        timeToIdleSeconds="10"
        timeToLiveSeconds="30"
        overflowToDisk="true"
     />
</ehcache>
```

refer:: HB-18-HibernateCaching-L2

Entity Object Life Cycle diagram/3 states of Entity Object

1. Transient State
2. Persistent State/Attached State
3. Detached State

Transient state

 Here the object of the Entity not bound or linked with Session object.
 Here the object don't have id value and it doesn't hold /represent any Db table data.

```
            eg: Product p = null;
                Product p = new Product();
```

Persistent State

 Here the object is linked with Session.its placed in L1 cache of Session Object.
 Contains id value and it represents the record in the db table having synchornization.
 All persistence operations takes place by bringing Object into this state.
            eg: save(),saveOrUpdate(),load(),get(),persist()

Detached State

 Previously persistent ,but not now.
 Contains id value, but does not represents the record in the db table and it does not maintain synchornization also.

session.close(),session.clear(),session.evict()

```
eg: Product p =new Product();//p-> Transient state
            session.save(p);//p -> persistent state
          transaction.commit()
            session.close()//Detached state
```

What is the difference b/w session.saveOrUpdate() and session.merge()?

saveOrUpdate()

=> First select operation(based on id),if it is succesfull then it performs update operation

otherwise it would perform insert operation.

merge()

1. Version1=> (same as saveOrUpdate)

=> First select operation(based on id), if it exists, then it performs update operation

otherwise it would perform insertion operation.

2. Version2 => (upon we performing the loading explicitly,if we want to perform update operation)

refer:: HB-19-MergeOperationApp

Connection Pooling in Hibernate

=> SessionFactory object holds jdbc connection pool having set of readymade jdbc connection objects and uses them in creation of hibernate objects.

=> By default hibernate app uses hibernate built in jdbc connection pool which is not suitable for production environment,becoz of performance issues.

hibernate.cfg.xml

```
<!-- Connection provider to work with hikaricp -->
<property
name="connection.provider_class">org.hibernate.hikaricp.internal.HikariCPConnection
Provider</property>

 <!-- HikariCP settings -->
<property name="hikari.connectionTimeout">50000</property>
<property name="hibernate.hikari.minimumIdle">10</property>
<property name="hibernate.hikari.maximumPoolSize">20</property>
<property name="hibernate.hikari.idleTimeout">30000</property>
```

refer: HB-20-ConnectionPool

Interaction with more than 1 DBS/w using hibernate
==
   UseCase:: copy one bank customer details to another bank(when banks get merged)
                transfer money from one account to another account(both accounts belongs to different bank)
                                refer: HB-21-InteractionwithMulitpleDB

Working with LOB's
==================
=> Images,audiofile,videofiles are called BinaryLargeObjects(BLOB) because they internally manage the data as binary information.
=> textfiles,rich text files and etc are called CharacterLargeObjects(CLOB) because they internally manage the data in the form of
   character type of data.
=> BLOB (byte[] + @Lob)
=> CLOB (char[] + @Lob)

Code for reading byte data and character data

```
      byte imageContent[]=null;
      char textContent[]=null;
      try(  FileInputStream fis=new FileInputStream("marriage.jpg")){

                  //prepare byte[] from image file
                  imageContent=new byte[fis.available()];
                  fis.read(imageContent);

                  //prepare char[] form text file
                  File file=new File("resume.txt");
                  try(FileReader reader=new FileReader(file)){
                        textContent=new char[(int) file.length()];
                        reader.read(textContent);
                  }//try2
      }//try1
            catch(IOException ioe) {
                  ioe.printStackTrace();
            }catch(Exception e) {
                  e.printStackTrace();
            }
      }
```

Code for Writing byte data and character data to a file

```
//create Dest image file having byte[] imageContent
      fos=new FileOutputStream("store/photo.jpg");
      fos.write(seeker.getPhoto());

//Create a Dest resume.txt file having char[] textContent
      writer=new FileWriter("store/resume.txt");
      writer.write(seeker.getResume());
      fos.flush();
      writer.flush();
```

                        refer:: HB-22-HibernateLobOperation

Locking

If multiple threads/app's simultaneously accessing and manipulating the records then there is a possibility of getting
concurrency problem.
      To avoid this problem, we need to lock the record in hibernate.
      Locking can be done in 2 ways
            a. optimistic locking
                              => Allows second thread/apps simultaneously to access and modify the record, first app notices the modification
                                 and throws Exception
                              => To use this feature we need to enable @Version annotation in entity class.

            b. pesimistic locking
                              => First thread/app locks the record,so if the second thread tries to access and modify the record then it
                                 would result in "Exception".
                              => To use this feature we need to use session.get(,,LockMode.UPGRADE_NOWAIT) as the third argument value.

Optimistic Lock
===============
client1.java
            After getting the record, make the thread to sleep for 30seconds
client2.java
            Get the same record and make some changes.

                        refer:: HB-23-HibernateOptimisticLocking

Pesimistic Lock
===============
client1.java
            After getting the record(use session.get(,,LockMode.UPGRADE_NOWAIT), make the thread to sleep for 30seconds
client2.java
            Get the same record(use session.get(,,LockMode.UPGRADE_NOWAIT) and make some changes.

                        refer:: HB-24-HibernatePesimiticLocking

Bulk operation in hibernate

To select or mainpulate one/more record or object having our choice as criteria value, we need to go for this bulk operation concept
      a. HQL/JPQL
      b. Native SQL
      c. Criteria API

Note: JPQL=> It is a specification given by SUNMS which speaks about the rules to develop object based query Language
        HQL=> It is an implementation of JPQL by hibernate.

1. HQL [Hibernate Query Language]
=> HQL is a pwerfull query language provided by Hibernate inorder to perform manipulations over multiple records.

=> HQL is an object oriented query language, it able to support for the object oriented features like
   encapsulation,polymorphism,...., but, SQL is structered query language.
=> HQL is database independent query language, but, SQL is database dependent query language.
=> In case of HQL, we will prepare queries by using POJO class names and their properties, but, in case of SQL, we will prepare
   queries on the basis of database table names and table columns.
=> HQL queries are prepare by using the syntaxes which are similar to SQL queries syntaxes.
=> HQL is mainly for retrival operations , but, right from Hibernate3.x version we can use HQL to perform insert , update and
   delete operations along with select operations, but, SQL is able to allow any type of database operation.
=> In case of JDBC, in case of SQL, if we execute select sql query then records are retrived from database table and these
   records are stored in the form of ResultSet object, which is not implementing java.io.Serializable , so that, it is not possible
   to transfer in the network,but, in the case of HQL, if we retrive records then that records will be stored in Collection objects,
   which are Serializable bydefault, so that, we are able to carry these objects in the network.
=> HQL is database independent query language, but, SQL is database dependent query language.
=> In case of Hibernate applications, if we process any HQL query then Hibernate Software will convert that HQL Query into database
   dependent SQL Query and Hibernate software will execute that generated SQL query.

Note: HQL is not suitable where we want to execute Database dependent sql queries
EX: PL/SQL procedures and functions are totally database dependent, where we are unable to use HQl queries.

Note:

eg: SQL> SELECT * FROM EMP WHERE EMPNO>? AND EMPNO<?
    HQL> FROM in.ineuron.model.Employee WHERE eno>? AND eno<?

    SQL> DELETE FROM EMP WHERE EMPNO=?
    HQL> DELETE FROM in.ineuron.model.Employee WHERE eno=?

    SQL>SELECT ENO,ENAME FROM EMPLOYEE
    HQL>SELECT eno,ename FROM in.ineuron.model.Employee

    SQL>UPDATE EMPLOYEE SET ENAME=?,ESAL=? WHERE ENO=?
    HQL>UPDATE in.ineuron.model.Employee SET ename=?,esal=? WHERE eno=?

refer:: HB-25-HQLAPP

Bulk operation in hibernate

To select or mainpulate one/more record or object having our choice as criteria value, we need to go for this bulk operation concept

a. HQL/JPQL
b. Native SQL
c. Criteria API

Note: JPQL=> It is a specification given by SUNMS which speaks about the rules to develop object based query Language
HQL=> It is an implementation of JPQL by hibernate.

1. HQL [Hibernate Query Language]
=> HQL is a pwerfull query language provided by Hibernate inorder to perform manipulations over multiple records.
=> HQL is an object oriented query language, it able to support for the object oriented featureslike encapsulation, polymorphism,.... ,
but, SQL is structered query language.
=> HQL is database independent query language, but, SQL is database dependent query language.
=> In case of HQL, we will prepare queries by using POJO class names and their properties, but, in case of SQL , we will prepare queries
on the basis of database table names and table columns.
=> HQL queries are prepare by using the syntaxes which are similar to SQL queries syntaxes.
=> HQL is mainly for retrival operations , but, right from Hibernate3.x version we can use HQL to perform insert , update and delete operations
along with select operations, but, SQL is able to allow any type of database operation.
=> In case of JDBC, in case of SQL, if we execute select sql query then records are retrived from database table and these records are stored in
the form of ResultSet object, which is not implementing java.io.Serializable , so that, it is not possible to transfer in the network,
but, in the case of HQL, if we retrive records then that records will be stored in Collection objects, which are Serializable bydefault, so
that, we are able to carry these objects in the network.
=> HQL is database independent query language, but, SQL is database dependent query language.
=> In case of Hibernate applications, if we process any HQL query then Hibernate Software will convert that HQL Query into database
dependent SQL Query and Hibernate software will execute that generated SQL query.

Note: HQL is not suitable where we want to execute Database dependent sql queries
EX: PL/SQL procedures and functions are totally database dependent, where we are unable to use HQl queries.

Note:

eg: SQL> SELECT * FROM EMP WHERE EMPNO>? AND EMPNO<?
HQL> FROM in.ineuron.model.Employee where eno>? and eno<?

SQL> DELETE FROM EMP WHERE EMPNO=?
HQL> DELETE FROM in.ineuron.model.Employee where eno=?

SQL>SELECT ENO,ENAME FROM EMPLOYEE
HQL>SELECT eno,ename from in.ineuron.model.Employee

```
    SQL>UPDATE EMPLOYEE SET ENAME=?,ESAL=? WHERE ENO=?
    HQL>UPDATE in.ineuron.model.Employee SET ename=?,esal=? Where eno=?
```

HQL SELECT: we can use select queries to fetch data DB tables (Multiple rows).
Final output is given by Hibernate is java.util.List(no.of rows=no.of objects ----->stored in List Collection only)

FULL LOADING :-select all columns using Query(HQL/SQL) is known as Full loading (One Full row = One Complete Model class Object).
So final output will be List<T>, T=Type/Model-Class-Name

PARTIAL LOADING : selecting one column (=1 column) or more then one columns (>1 column) is known as Partial loading.
Final outoput is given as:=
      1 column : List<DT DT=DataType of varivales/colum
      >1 column : List<Object[]>

Selecting all records

```
---------------------
Session session = sessionFactory.openSession();
Query<Employee> query = session.createQuery("from in.ineuron.Employee");
System.out.println("Using list() method");
System.out.println("-------------------------");
List<Employee> list = query.list();
System.out.println("ENO\tENAME\tESAL\tEADDR");
System.out.println("------------------------------");
for(Employee e: list) {
    System.out.print(e.getEno()+"\t");
    System.out.print(e.getEname()+"\t");
    System.out.print(e.getEsal()+"\t");
    System.out.println(e.getEaddr());
}
```

Selecting only one record

```
-------------------------
try(Session ses=HibernateUtil.getSf().openSession()){
      String hql="select ename from in.ineuron.model.Employee";
      Query q=ses.createQuery(hql);
      List<String>list=q.list();
      for(String s:list){
            System.out.println(s);
      }
}catch(Exception ex){}
```

Selecting multiple record

```
-------------------------
try(Session ses=HibernateUtil.getSf().openSession()) {
      String hql="select ename,eaddr from in.ineuron.model.Employee";
      Query q=ses.createQuery(hql);
      List<Object[]>list=q.list();
      for(Object[]ob:list){
            System.out.println(ob[0]+","+ob[1]);
      }
}catch(Exception ex){}
```

```
                    or
      list.forEach(row-> {
                                  for(Object obj: row)
                                        System.out.print(obj+":");
                              }
                                  System.out.println();
                )
```

Named Parameters:- it is used to provide a name in place of ? symbole, to indicate data comes at runtime.

* This is new concept in Hibernate, not exist in JDBC
* Name should be unique in HQL (duplicates not allowed)

**** We can use variableName as parameter name also.

- Name never gets changed if query is changed.
- To pass data at runtime code is : setParameter(name,data)
- Syntax is : name (colon name)

Usage of Named Parameter

```
------------------------
try(Session ses=HibernateUtil.getSf().openSession()) {
            String hql="from in.ineuron.model.Employee where eid=:id or
ename=:name";
            Query q=ses.createQuery(hql);

            q.setParameter("id",10);
            q.setParameter("name","sachin");
            List<Employee>list=q.list();
            list.forEach(System.out::println);
}catch(Exception ex){}
```

In clause:- To work with random rows in DB table use in-clause.

Syntax:

```
                Select ....from ... Where column in (values);
```

- To handle this in Hibernate
- Used named parameters
- Create values collection
- Call setParameterList method

```
============code=========
Test class:// cfg,sf,ses
String hql=”from in.ineuron.model.Employee where empId in (:id)”;
Query q=ses.createQuery(hql);
List<Integer> al=Arrays.asList(10,12,14,8);
q.setParameteList(“id”,al);
List<Employee> e=q.list();
e.forEach(System.out::println);
```

uniqueResult():-

This method is used for select HQL operation.

If query returns one row data then choose this method instead of query.list() method.

- if will save memory, by avoiding list object for one row data.

Program to demonstrate uniqueResult()

```
-------------------------------------
try(Session ses=HibernateUtil.getSf().openSession()) {
            String hql="from in.ineuron.model.Employee where eid=:id";
            Query q=ses.createQuery(hql);
            q.setParameter("id",10);
            Employee employee=(Employee)query.uniqueResult();
            if(employee!=null)
                  System.out.println(employee);
            else
                  System.out.println("Record not found for the given id :: "+id);
}catch(Exception ex){}

Not recomended from JDK8.0,because we have a new API called "Optional(I)".
      Optional(C)-> This api is very useful to hold the object, where it would
check the availablity of the object without explicitly performing
                   NullPointerException.


Program to demonstrate uniqueResultOptional()
-----------------------------------------
try(Session ses=HibernateUtil.getSf().openSession()) {
            String hql="from in.ineuron.model.Employee where eid=:id";
            Query q=ses.createQuery(hql);
            q.setParameter("id",10);
            Optional<Employee> opt=(Employee)query.uniqueResultOptional();
            if(opt.isPresent())
                  System.out.println(opt.get());
            else
                  System.out.println("Record not found for the given id :: "+id);
}catch(Exception ex){}

                                    refer:: HB-25-HQLSelectAPP

Executing HQLNon Select Queries
-------------------------------
HQL NON=SELECT OPERATION:- HQL supports non-select operations like
a. Update multiple rows
b. Delete multiple rows
c. Insert operation[Copy rows from one table to another table (backup data)]
            • Use method executeUpdate():int return no.of rows effected
            • It supports named parameters.

Note: HQL insert query is not given directly, because linking generators with HQL
insert query is not possible.
       To link with generators we need to use session.save() method only.

Code for update operation
-------------------------
Transaction tx = null;
try (Session ses = HibernateUtil.getSf().openSession()) {
            tx=ses.beginTransaction();
            Employee emp=new Employee();
            String hql="update in.ineuron.model.Employee set ename=:name,esal=:sal
where eid=:id";

            Query q=ses.createQuery(hql);
            q.setParameter("name", "sachin");
            q.setParameter("sal", 3345);
            q.setParameter("id", 10);
```

```
            int count=q.executeUpdate();
            tx.commit();
            System.out.println(count);
} catch (Exception ex) {
      ex.printStackTrace();
}
```

Code for Delete opearation

```
Transaction tx = null;
try (Session ses = HibernateUtil.getSf().openSession()) {
            tx=ses.beginTransaction();
            Employee emp=new Employee();
            String hql="delete in.ineuron.model.Employee where eid=:id";

            Query q=ses.createQuery(hql);
            q.setParameter("id", 10);
            int count= q.executeUpdate();
            tx.commit();
            System.out.println(count);
} catch (Exception ex) {
            ex.printStackTrace();
}
```

refer:: HB-26-HQLNonSelectApp

Code for Insert operation

```
Query query = session.createQuery("insert into
in.ineuron.model.PremiumInsurancePolicy(policyId,policyName,company,policyType,tenu
re)
                                                select
i.policyId,i.policyName,i.company,i.policyType,i.tenure from InsurancePolicy as i
where
      i.tenure>=:min");
Transaction tx = session.beginTransaction();
query.setParameter("min",5);
int rowCount = query.executeUpdate();
tx.commit();
```

NamedHQL Queries

```
=> So far Our HQL Query is specific to one Session Object becoz Query object
created having hard coded HQL Query on session object.
=> To make our HQL query accessible and executable through multiple session objects
of multiple DAO classes or client apps we need to go
   for "NamedHQL".
=> we defined HQL query in Entity class using
@NamedQuery(name="HQL_INSERT_QUERY",query="....")

@Entity
@NamedQuery(name="HQL_TRANSFER_POLICIES",query="insert into


in.ineuron.model.PremiumInsurancePolicy(policyId,policyName,company,policyType,tenu
re)                                                                    select
i.policyId,i.policyName,i.company,i.policyType,i.tenure from InsurancePolicy
                                                                    as i where
```

```
i.tenure>=:min"")
public class PremiumInsurancePolicy implements Serializable
{
      private Long pid;
      private String policyName;
      private String policyType;
      private String company;
      private Integer tenure;
}

Query query= session.getNamedQuery("HQL_TRANSFER_POLICIES");
query.setParameter("min",10);
int rowCount = query.executeUpdate();
```

refer:: HB-27-HQLInsertQuery

Hibernate
   Don't write SQL Query(DB specific)

Hibernate
      a. SRO(Single Row Operation)
      b. Bulk Operation
                a. HQL(Hibernate Query Language)=> Write query using Entityname and properties.
                b. By writing NativeSQL query   => Write query using Tablename and column names.
                c. Using Criterian API(Pure java code).

NativeSQL

=> This is another bulk operation technique to perform both select and non select operation.
=> With this native sql, we can execute sql commands on database for bulk operations.
=> The two reasons to give NativeSQL in hibernate is
      a. For programmer convinence
      b. To migrate jdbc application as hibernate application easily.
=> The only problem with nativesql is it makes hibernate application "Database dependent".
=> To run sql command we need to first create SQLQuery object, we can create this by using createSQLQuery().
=> For select operation we call list()/getResultList() and for non-select operation we call executeUpdate().
=> These queries performance is good as they go to database directly for execution.
=> These queries will be written specifically by using db tablename and db column names.

select query without using Entity object

```
NativeQuery<Object[]> query = session
                                    .createSQLQuery("select * from employee where eid>=:id1 and eid<=:id2");
query.setParameter("id1", 1);
query.setParameter("id2", 7);
List<Object[]> emps = query.list();
emps.forEach(row -> {
                        for (Object obj : row) {
                                    System.out.print(obj + "\t");
                              }
                        System.out.println();
               });
```

select query using Entity object

```
NativeQuery<Employee> query = session
                                    .createSQLQuery("select * from employee where eid>=:id1 and eid<=:id2");
query.setParameter("id1", 1);
query.setParameter("id2", 7);
query.addEntity(Employee.class);
List<Employee> emps = query.list();
emps.forEach(System.out::println);
```

Select particular column from a table

```
-------------------------------------
NativeQuery<Object[]> query = session
                                    .createSQLQuery("select ename,esalary from
employee where eid>=:id1 and eid<=:id2");
query.setParameter("id1", 1);
query.setParameter("id2", 7);
query.addScalar("ename", StandardBasicTypes.STRING);
query.addScalar("esalary", StandardBasicTypes.INTEGER);
List<Object[]> emps = query.list();
      emps.forEach(row -> {
                        for (Object obj : row) {
                                    System.out.print(obj + "\t");
                              }
                        System.out.println();
                  });

Note: It is a good practise to bind EntityQuery result with Entity class and
Scalary query results with Hibernate data types.
            Query<T>(I)
               |
               | extends
               |
         NativeQuery<T>(I)



Performing insert query(need transaction object)
-----------------------------------------------
NativeQuery query = session.createSQLQuery(
                                    "insert into
employee(`ename`,`eage`,`eaddress`,`esalary`)values(:name,:age,:addr,:sal)");

query.setParameter("name", "nitin");
query.setParameter("age", 30);
query.setParameter("addr", "RCB");
query.setParameter("sal", 1500);
rowCount = query.executeUpdate();


NamedNativeQuery
---------------
@Entity
@NamedNativeQuery(name = "SQL_INSERT_QUERY",
                  query = "insert into
employee(`ename`,`eage`,`eaddress`,`esalary`)values(:name,:age,:addr,:sal)")
public class Employee implements Serializable {
                  ;;;;;
                  ;;;;;
                  ;;;;;
}

NativeQuery query = session.getNamedNativeQuery("SQL_INSERT_QUERY");
query.setParameter("name", "hyder");
query.setParameter("age", 28);
query.setParameter("addr", "RCB");
query.setParameter("sal", 2000);
rowCount = query.executeUpdate();

Working with StoredProcedure
```

```
----------------------------
=> To make persistence logic/buisness logic reusable across multiple modules of the
project by keeping logic in db software, then we need to
   go for StoredProcedure.
=> The syntax of stored procedure varies from database to database
            a. Oracle(PL/SQL)
            b. MySQL
      When we work with StoredProcedure we use 3 types of params like
            a. IN(default mode)
            b. OUT
            c. INOUT

Getting the complete resultList
==============================
CREATE DEFINER=`root`@`localhost` PROCEDURE `P_GET_PRODUCT_BY_NAME`(IN name1
VARCHAR(20), IN name2 VARCHAR(20))
BEGIN
            SELECT pid,pname,price,qty FROM products WHERE pname IN (name1,name2);
      END$$

DELIMITER ;

To call a procedure we use
      CALL `P_GET_PRODUCT_BY_NAME`("fossil","tissot")

code
----
ProcedureCall procedureCall =
session.createStoredProcedureCall("P_GET_PRODUCT_BY_NAME", Product.class);
      String name1 = "fossil";
      String name2 = "tissot";
      procedureCall.registerParameter(1, String.class,
ParameterMode.IN).bindValue(name1);
      procedureCall.registerParameter(2, String.class,
ParameterMode.IN).bindValue(name2);

      List<Product> products = procedureCall.getResultList();
      products.forEach(System.out::println);


Getting the record on the basis of id
====================================
CREATE DEFINER=`root`@`localhost` PROCEDURE `P_GET_PRODUCT_DETAILS_BY_ID`(IN id
INT,OUT NAME VARCHAR(20),
                  OUT rate INT, OUT qnt INT)
BEGIN
            SELECT pname,price,qty INTO NAME,rate,qnt FROM products WHERE pid = id;
      END$$

DELIMITER ;

To call a procedure we use
      CALL `P_GET_PRODUCT_DETAILS_BY_ID`(1,@name,@rate,@qnt)
      SELECT @name,@rate,@qnt


Code
----
ProcedureCall procedureCall =
```

```
session.createStoredProcedureCall("P_GET_PRODUCT_DETAILS_BY_ID");
Integer id = 1;

procedureCall.registerParameter(1, Integer.class, ParameterMode.IN).bindValue(id);
procedureCall.registerParameter(2, String.class, ParameterMode.OUT);
procedureCall.registerParameter(3, Integer.class, ParameterMode.OUT);
procedureCall.registerParameter(4, Integer.class, ParameterMode.OUT);

String pname = (String) procedureCall.getOutputParameterValue(2);
Integer price = (Integer) procedureCall.getOutputParameterValue(3);
Integer qty = (Integer) procedureCall.getOutputParameterValue(4);

System.out.println("PID\tPNAME\tPRICE\tQTY");
System.out.println(id+"\t"+pname+"\t"+price+"\t"+qty);
```

Criterian api

It is another technique to perform bulk operations.
Using Criterian api we can perform only select operations.
Even by using HQL we can perform select operation by writing select query.
If we directly write HQL select query, then we need to tune the query for better performance.
we can read the same data from database by constructing query in multiple ways, but performance is important.
Performance: reading the data with small amount of time.
In Hibernate applications, to avoid totally query langugaes like SQL and HQL,....and to provide the complete Persistence logic in the form of JAVA code we must use "Criterion API".
Criterian api does not supports StoredProcedure.
Criterian api does not supports NamedQuery,NamedNativeQuery.

There are 2 modes of Writing Criterian API
- a. HB QBC/Criteria API(specific to hibernate only and we can use to work only for select operation)
- b. JPA QBC/Criteria API(common to all ORM frameworks,we can perform both select and non-select operation)

1. Create Criteria Object:
Criteria object is a central object in Critera API, it able to manage HQl query repersentation internally and it has provided predefined methods to defined query logic.
To create Criteria object we have to use the following method from Session.

```
public Criteria createCriteria(Class cls);
```

EX: `Criteria c = session.createCriteria(Employee.class);`
Note: It is equalent to the HQL query internally "from Employee".

2. Prepare Criterion objects and add that Criterion objects to Criteria object:
Criterion is an object , it able to manage a single Conditional expression in database logic.
To create Criterion object we have to use the following methods from "org.hibernate.Restrictions" class.

```
public static Criterion isEmpty(String property)
public static Criterion isNotEmpty(String property)
public static Criterion isNull(String property)
public static Criterion isNotNull(String property)
```

```
      public static Criterion in(String property, Object[] obj)
      public static Criterion in(String property, Collection c)
      public static Criterion between(String property, Object min_Val, Object
max_Val)
      public static Criterion between(String property, Object[] obj)
      public static Criterion eq(String property, Object val)
      public static Criterion ne(String property, Object val)
      public static Criterion lt(String property, Object val)
      public static Criterion le(String property, Object val)
      public static Criterion gt(String property, Object val )
      public static Criterion ge(String property, Object val)
      -----
      -----
```

To add a particular Criterion object to Criteria object we have to use the following method.

```
                  public void add(Criterion c)
```

EX:

```
Criterion c1 = Restrictions.ge("esal", 60000);
Criterion c2 = Restrictions.le("esal", 90000);
            c.add(c1);
            c.add(c2);
```

Note: With the above steps, Critera object is able to prepare the query like "from Employee where esal>=6000 and esal<=9000".

3. Create Projection objects , add Projection objects to ProjectionList and add ProjectionList to Criteria object:

The main intention of Projection object is to represent a single POJO class property.

To get Projection object with a particular Property name we have to use the following method from "Projections" class.

```
                  public static Projection projection(String pro_Name)
```

To create ProjectionList object we have to use the following method from Projections class.

```
                  public static ProjectionList projectionList()
```

To add Projection object to ProjectionList we have to use the following method from ProjectionList class.

```
                  public void add(Projection p)
```

To set ProjectionList to Criteria object we have to use the following method.

```
                  public void setProjection(ProjectionList pl)
```

Ex:

```
ProjectionList pl = Projections.projectionList();
pl.add(Projections.property("eno"));
pl.add(Projections.property("ename"));
pl.add(Projections.property("esal"));
pl.add(Projections.property("eaddr"));
c.setProjection(pl);
```

EX: With the above , Criteria object is able to prepare HQL query internally like below.

```
                        "select eno, ename, esal, eaddr from Employee where esal
>=6000 and esal<=90000".
```

4) Provide a particular Order to the query:

To represent a particular Order over the results, we have to use either asc(-) or desc(-) methods from "Order" class.

```
            public static Order asc(String prop_Name)
            public static Order desc(String prop_Name)
To add Order object to Criteria object we have to use the following method.
            public void addOrder(Order o)
EX:
      Order o = Order.desc("ename");
      c.addOrder(o);

Note: With this, Criteria object is able to create HQl query like
       "select eno, ename, esal, eaddr from Employee where esal >=6000 and
esal<=90000 order by desc(ename)"


Scalar Projection
-----------------
Criteria criteria = session.createCriteria(Employee.class);

Criterion c1 = Restrictions.le("salary", 65000);
Criterion c2 = Restrictions.gt("salary", 6000);
criteria.add(c1);
criteria.add(c2);

ProjectionList projectionList = Projections.projectionList();
      projectionList.add(Projections.property("eid"));
      projectionList.add(Projections.property("ename"));
      projectionList.add(Projections.property("eage"));

      criteria.setProjection(projectionList);

      Order order = Order.desc("ename");
      criteria.addOrder(order);

      List<Object[]>emps = criteria.list();
      emps.forEach(row->{
                        for (Object obj : row) {
                              System.out.print(obj+"\t");
                        }
                        System.out.println();
                  });



Adding multiple condition with or,AND
-------------------------------------
Criteria criteria = session.createCriteria(Employee.class);

Criterion cond1 = Restrictions.between("salary",6000,65000);
Criterion cond2 = Restrictions.in("ename", "sachin","kohli");
Criterion cond3 = Restrictions.ilike("eaddress","R%");
Criterion finalCond = Restrictions.or(Restrictions.and(cond1,cond2),cond3);
      criteria.add(finalCond);

List<Employee> list = criteria.list();
list.forEach(System.out::println);


Adding only one column
----------------------
Criteria criteria = session.createCriteria(Employee.class);
```

```
Criterion c1 = Restrictions.eq("eaddress", "RCB");
criteria.add(c1);

PropertyProjection property = Projections.property("ename");
criteria.setProjection(property);

List<String> emps = criteria.list();
emps.forEach(System.out::println);


HQL/JPQL
--------
1. Persistence logic is DB independent
2. Query will be written using Entity classname and property name.
3. Supports only named params.
4. Supports both select and non-select queries.
5. It is not suitable for calling storedprocedure.

NativeSQL
--------
1. It is dependent
2. Query will be written using DBTablenames and column names.
3. Supports only named params.
4. Supports both select and non-select queries.
5. It is  suitable for calling storedprocedure.


CriteriaAPI
----------
1. It is database independent
2. Query will be written using Entity classname and property name.
3. It won't support params
4. Only select operation using HB-QBC
5. It is not suitable for calling storedprocedures.


Remaining topics of hibernate
-----------------------------
Filters,SoftDeletion
Pagination
ORMapping(uni-directional,BiDirectional using joins)
      a. OnetoOne
      b. OnetoMany
      c. ManytoOne
      d. ManytoMany
```

HBFilters

 It is a named,parameterized global condition that can be enabled or disabled on session object.
 This concept is very useful to execute SQLQuery without implicit condition.
 use cases :The blocked,closed account should not participate in any Query execution.
     note: In real time application, blocked/closed account would not be deleted from the table rather status would be marked as closed/blocked.
             This is called as "SoftDeletion".

BankAccount.java

```
@Entity
@Table(name = "bankaccount")
@FilterDef(name = "FILTER_BANKACCOUNT_STATUS", parameters = {
                        @ParamDef(type = "string", name = "accType1"),
                        @ParamDef(type = "string", name = "accType2")
                    }
            )
@Filter(name = "FILTER_BANKACCOUNT_STATUS", condition = "STATUS NOT IN(:accType1,:accType2)")
public class BankAccount implements Serializable {

}
```

SelectApp.java

```
Filter filter = session.enableFilter("FILTER_BANKACCOUNT_STATUS");
      filter.setParameter("accType1", "blocked");
      filter.setParameter("accType2", "closed");

Query<BankAccount> query = session.createQuery("FROM in.ineuron.entity.BankAccount WHERE balance>=:amt");
      query.setParameter("amt", 2000.0f);

List<BankAccount> resultList = query.getResultList();
      resultList.forEach(System.out::println);

      System.out.println();

session.disableFilter("FILTER_BANKACCOUNT_STATUS");
Query<BankAccount> query1 = session.createQuery("FROM in.ineuron.entity.BankAccount WHERE balance>=:amt");
            query1.setParameter("amt", 2000.0f);

List<BankAccount> result = query1.getResultList();
      result.forEach(System.out::println);
```

Soft Deletion in hibernate

It is not about deleting the record from the table, it is all about marking the record from DB table as deleted.
When we close the account in the bank, they would not actually delete the record, rather the record would be marked as "blocked".
When we perform soft deletion, for a call of session.delete() 'delete query' should not be executed rather 'update query' should be executed

To do so we need to configure as shown below

a.
```
@Entity
@Table(name = "bankaccount")
@SQLDelete(sql = "UPDATE bankaccount SET status='closed' WHERE accno=?")
@Where(clause = "STATUS NOT IN ('blocked','closed')")
public class BankAccount implements Serializable {


}
```

b.
```
session = HibernateUtil.getSession();
transaction = session.beginTransaction();

BankAccount account = new BankAccount();
account.setAccno(6);
session.delete(account);====> update query will be generated..
```

c.
```
Query<BankAccount> query = session.createQuery("From
in.ineuron.entity.BankAccount");
List<BankAccount> accounts = query.getResultList();
accounts.forEach(System.out::println);
```

Output
======
```
select
        bankaccoun0_.accno as accno1_0_,
        bankaccoun0_.balance as balance2_0_,
        bankaccoun0_.holderName as holderna3_0_,
        bankaccoun0_.status as status4_0_
    from
        bankaccount bankaccoun0_
    where
        (
            bankaccoun0_.STATUS NOT IN (
                'blocked','closed'
            )
        )
BankAccount [accno=7, holderName=dhoni, balance=44000.0, status=active]
```

d.
```
Query query = session.createQuery("UPDATE in.ineuron.model.BankAccount set
status='closed' where accno=:no");
query.setParameter("no", 1234);
rowCount = query.executeUpdate();
```

In the above case, if we keep delete query,then by referring to entity update query won't be executed.
so if we use HQL,NativeSQL,QBC logics then we need to explicity write "update sql query" for soft deletion.

Note: while working with @SQLXXXX annotations/custom queries execution takes place only for single row operation,

Pagination

=> The process of displaying huge no of records page by page is calle pagination.
=> It would avoid loading of all records,load only those records/objects that are required to display in the current page to
   improve the performance.
                eg: Gmail inbox,report generation, google search results ......

Using HQL/NativeSQL/QBC we can achieve pagination as shown below
==========

```
      Query<InsurancePolicy> query = session.createQuery("from
in.ineuron.model.InsurancePolicy");

                  //pagination settings
                  query.setFirstResult(0);
                  query.setMaxResults(3);

      List<InsurancePolicy> insurance = query.list();
      insurance.forEach(System.out::println);
```

UserInput => pageSize(3)=> No of records to be displayed in each page

parameter => pageNo([1],[2],[3],....)=> Service layer would generate
                pageCount(In how many pages the records should be displayed)=>
Service layer would generate

Sequence flow
----
                 scope(session)
Controller =======pageSize=========> Service ====pageSize=======> DAO =========>
Database
   |
   |scope(request)
   | pageNo from link
   | pagesCount
   | listDTO
   |
 report.jsp

code for generating hyperlinks

```
<c:if test="${pageNo>1 }">
      <b><a href='./controller?pageNo=${pageNo-1}&s1=link'>previous</a>
  </b>
</c:if>

<c:forEach var='i' begin='1' end='${pagesCount}' step="1">
      <b><a href='./controller?pageNo=${i}&s1=link'>[${i}]</a>   </b>
</c:forEach>

<c:if test="${pageNo<pagesCount }">
      <b> <a href='./controller?pageNo=${pageNo+1}&s1=link'>next</a>
  </b>
</c:if>
```


Hibernate ReverseEngineering

      Entity class <====== ORM <====== Table already available

To do ReverseEngineering we use Hiberante tools from jbosss.

To install plugin we need to do the following steps
      a. Go to help menu
      b. Go to eclipse market place
      c. In the search box, type as jboss plugin.
      d. install jboss tools 4.2.6 plugin

Steps to perform ReverseEnginnering in eclipse
----
1. Change the perspective from java environment to hibernate
            Open perspective, choose hibernate and click on open

2. Select the project
            Click on file, new option
                  a. choose hibernate.cfg.xml file, provide suitable information
                  b. choose hibernate console configuration and provide suitable
information.

c. choose hiberante reverse engineering file to choose the table for which model should be should generated

3. Now click on run configuration of hibernate
    a. choose hibernate configuration and supply details like
        1. choose the console configuration name
        2. choose the output directory(project name till src folder)
        3. click on check box (Reverse engineer from jdbc connection)
        4. write the package name where the model should be generated(in.ineuron.model)
        5. choose revenge.xml file which is generated in the previous steps.
        6. click on run option.

Advanced ORMapping

There are multiple mismatches b/w Java and RDBMS like, we can keep entity classes in composition or inheritance, but we can't keep table names in inheritance or composition,but we need to map entity classes of composition or inheritance with 1 or more db table by taking the support of "OR" mapping
    a. Component mapping[Entity classes are in composition]
    b. Inheritance mapping[Entity classes are in inheritance]
    c. Collection mapping[Entity classes are having collection with mapping]
    d. Association mapping[Entity classes are in relation like 1---*,*---1,1----1,*----*]

Component Mapping
---
1. The property that is pointing to single column in db table is called "Simple property".
2. The property that is pointing to multiple columns of the db table having sub properties is called "Component property".
3. Component Mapping in java side is represented as "HAS-A" relationship.

eg#1.
```
package in.ineuron.model;

import javax.persistence.Embedded;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "student")
public class StudentInfo {

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private Integer sid;
	private String sname;
	private Integer sage;
```

```
      @Embedded
      private Address address;

      //setters,getters,toString
}

Address.java
------------
package in.ineuron.model;
import javax.persistence.Embeddable;

@Embeddable
public class Address {
      private String countryName;
      private String stateName;
      private String cityName;

      //setters,getters and toString
}
output
create table student (
       sid integer not null auto_increment,
        cityName varchar(255),
        countryName varchar(255),
        stateName varchar(255),
        sage integer,
        sname varchar(255),
        primary key (sid)
    ) engine=InnoDB
```

OR-Mapping
      a. Composition Mapping(HAS-A) ======> @Embeded,@Embedable
      b. Inheritance Mapping(IS-A)
      c. Collection Mapping(List,Set,Map)
      d. Association Mapping
                  a. 1...*   b. *.....1
                  c. 1...1   d. *......*

Inheritance Mapping(IS-A)
==========================
1.@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
2.@Inheritance(strategy = InheritanceType.JOINED)
3.@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
4.@DiscriminatorColumn
5.@DisciriminatorValue
6.@PrimaryKeyJoinColumn

Table per Class(single table)

Inheritance Mapping
----
1. TABLE PER CLASS HIERARCHY:
This design provides single table for all model classes. It will considers all classes variables as column names and takes one extra column “Discriminator” which provides information like “Row related to which model class”.
=> Discriminator column can be int type or String/char type.

For IS-A Relation Mapping enum and Annotation are given as:
enum: InheritanceType
Annotation: @Inheritance

For TABLE PER CLASS HIERATCHY DESIGN CODE:
@Inheritance(strategy= InheritanceType.SINGLE_TYPE)
For single table design, we should also provide extra column ‘name,type and value’ using code:
Annotation.: @DiscriminatorColumn
Enum : DiscriminatorType

Code look like:
@DiscriminatorColumn(name=”obType”,discriminatorType= DiscriminatorType.STRING)

For object value code is:
@DiscriminatorValue(“-----------“)

Code

```
@Entity
@Table(name = "commonTab")
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "objType", discriminatorType =
DiscriminatorType.STRING)
@DiscriminatorValue(value = "STD")
public class Student {

}
```

```
@Entity
@DiscriminatorValue(value = "ADD")
public class Address extends Student {

}

@Entity
@DiscriminatorValue(value = "CLS")
public class Classes extends Student{

}
```

TABLE PER SUB CLASS:-
=> In this case hibernate creates table for every child case along with parent class.
=> On saving data, parent data stored at parent table and child data stored at child table.
=> Parent table and child table is connected using PK-FK column. FK column also called as key column.

Code

Payment.java
============

```
@Entity
@DiscriminatorColumn(name = "paymentmode", length = 10)
@Inheritance(strategy = InheritanceType.JOINED)
public abstract class Payment implements Serializable {
	@Id
	@GeneratedValue
	private Integer pid;
}
```

CardPayment
===========

```
@Entity
@PrimaryKeyJoinColumn(name = "payment_id",referencedColumnName = "pid")
@DiscriminatorValue("CARD")
public class CardPayment extends Payment {
	private Long cardNo;
	private String cardType;
	private String paymentGatewa
}
```

ChequePayment
=============

```
@Entity
@DiscriminatorValue("cheque")
@PrimaryKeyJoinColumn(name = "payment_id",referencedColumnName = "pid")
public class ChequePayment extends Payment {
	private Integer chequeNo;
	private String chequeType;
	private LocalDate expiryDate;

}
```

TestApp.java
============

```
CardPayment payment = new CardPayment();
      payment.setAmt(45000.0f);
      payment.setCardNo(24567L);
      payment.setCardType("credit");
      payment.setPaymentGateway("VISA");

ChequePayment payment2 = new ChequePayment();
      payment2.setAmt(9000f);
      payment2.setChequeNo(15744);
      payment2.setChequeType("self");
      payment2.setExpiryDate(LocalDate.of(2021, 10, 21));

      session.save(payment);
      session.save(payment2);
```

TABLE PER CONCRETE CLASS
This design creates independent tables for every class in IS-A relation including parent class variable.

```
@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public abstract class Payment implements Serializable {

      private static final long serialVersionUID = 1L;

      @Id
      @GeneratedValue
      private Integer pid;
      private float amt;

}

@Entity
public class ChequePayment extends Payment {
      private static final long serialVersionUID = 1L;

      private Integer chequeNo;
      private String chequeType;
      private LocalDate expiryDate;

}

@Entity
public class CardPayment extends Payment {
      private static final long serialVersionUID = 1L;

      private Long cardNo;
      private String cardType;
      private String paymentGateway;

}
```

TestApp.java
============

```
CardPayment payment = new CardPayment();
      payment.setAmt(45000.0f);
      payment.setCardNo(24567L);
      payment.setCardType("credit");
      payment.setPaymentGateway("VISA");
```

```
ChequePayment payment2 = new ChequePayment();
      payment2.setAmt(9000f);
      payment2.setChequeNo(15744);
      payment2.setChequeType("self");
      payment2.setExpiryDate(LocalDate.of(2021, 10, 21));

      session.save(payment);
      session.save(payment2);

Output
create table CardPayment (
       pid integer not null,
        amt float not null,
        cardNo bigint,
        cardType varchar(255),
        paymentGateway varchar(255),
        primary key (pid)
    ) engine=InnoDB

create table ChequePayment (
       pid integer not null,
        amt float not null,
        chequeNo integer,
        chequeType varchar(255),
        expiryDate date,
        primary key (pid)
    ) engine=InnoDB

Note:since the Payment class is abstract and we are not doing any operation,db
table won't be created for Payment class.
     since new table is created for every class,we don't need any discriminator
value.

Collection Mapping in Hibernate:
=> It is all about taking collection type properties with Strings/Wrappers in
Entity class.
=> For every collection property one seperate child table will be created having FK
pointing PK Col of Entity class db table(parent table).
=> The child table for Collection property(List/Set/Map) will have min 2
columns(Set) and max 3 columns(List/Map).
=> Set Collection child table contains
           a. element column  b. foreign key column(no index column)
=> List/Map  collection child table contains
           a. element column  b. foreign key column  c. index column.


Note: Collection are of two types
           a. Indexed Collection     => List/Map
           b. Non-Indexed Collection => Set

On every collection property we must add
      a. @ElementCollection=> Specifying element column name.
      b. @CollectionTable  => Specify child table name and FK column name.
      c. @OrderColumn      => To specify the list index column name.
      d. @MapKeyColumn     => To specify the map index column name.
```