To write one application we need to use 4 files as shown below

- a. Model class
- b. Mapping code(xml/annotation)
- c. Configuration file(xml)
- d. Test class

Modle class

It represent Model data, it can be also called as Entity/Pojo
It is a class which follows rules given by Hibernate Framework

- a. class must have package statement
- b. class must be a public type
- c. No of tables = No of classes
- d. Class can have variables,must be private
  [No of columns = No of variables]
- e. class should have zero argument constructor and setter-getter methods.
- f. class can override toString(),hashcode(),equals() from Object class.
- g. class can have annotations given by JPA and also core library annotations.
- h. class can inherit(IS-A)[extends/implements] only hibernate api.

Mapping Code
============
1. Annotations
2. XML

Using Annotations we can map java class to DBTables as shown below
===

1. @Entity
   It maps models class with DBTable and Variables with Column Names.
2. @Id
   It indicates primary key, Every table must contain primary key column.
3. @Table(optional)
   It indicates the tableName which is been mapped with Model class.
4. @Column(optional)
   It indicates the columName of table which is been mapped with variableName of Model class.

Note:
if @Table,@Column are not provided then by default className is TableName,variableName is
ColumnName(taken by hibernate)

eg#1
```
@Entity
@Table(name="empTab")
public class Employee
{
      @Id
      @Column(name="eid")
      private int empId;

      @Column(name="ename")
      private String empName;

      @Column(name="esal")
      private double empSal;
}

eg#2.
@Entity
@Table(name="prodTab")
public class Product
{
      @Id
      @Column(name="pid")
      private int prodId;

      @Column(name="pcode")
      private String prodCode;

      @Column(name="pcost")
      private double prodCost;
}
```

Maping w.r.t XML
==============

```
                        Employee.hbm.xml
                        ================
eg#1.
<hibernate-mapping>
   <class name="in.ineuron.model.Employee" table="empTab">
      <id name="empId"  column="eid"/>
      <property name="empName"  column="ename" />
      <property name="empSal"  column="esal" />
   </class>
</hibernate-mapping>

eg#2
            Product.hbm.xml
            ==============
<hibernate-mapping>
   <class name="in.ineuron.model.Product" table="prodTab">
      <id name="prodId"  column="pid"/>
      <property name="prodCode"  column="pcode" />
      <property name="prodCost"  column="pcost" />
   </class>
</hibernate-mapping>
```

3. Configuration file

      For one application, one configuration file should be given
      It is XML format.
      ****configuration = Property + mapping class
      Property => It represents key-value pair data.

```
      hibernate.cfg.xml
      ==============
<hibernate-configuration>
      <session-factory>
            <!-- Database connection settings -->
            <property
name="connection.driver_class">com.mysql.jdbc.Driver</property>
            <property
name="connection.url">jdbc:mysql:///enterprisejavabatch</property>
            <property name="connection.username">root</property>
            <property name="connection.password">root123</property>
            <!-- JDBC connection pool (use the built-in) -->
            <property name="connection.pool_size">1</property>

            <!-- SQL dialect -->
            <property name="dialect">org.hibernate.dialect.MySQLDialect</property>

            <!-- Echo all executed SQL to stdout -->
            <property name="show_sql">true</property>

            <!-- Format SQLOuput to stdOut--->
            <property name ="format_sql">true</property>

            <!--Mapping information-->
            <mapping resource="Employee.hbm.xml"/>
            <mapping class = "in.ineuron.Model.Employee"/>

      </session-factory>
</hibernate-configuration>
```

```
<!-- SQL dialect -->
<property name="dialect">org.hibernate.dialect.MySQLDialect</property>
```

dialect => It is a class available inside package called org.hibernate.dialect,it will generate the SQLQuery when the programmer performs operation.
For every database dialect is different.

```
        Oracle          => nature of query
        MySQL         => nature of query
        PostgreSQL => nature of query
```

```
<!-- Echo all executed SQL to stdout -->
<property name="show_sql">true</property>
```

This property is used to see the Query generated by the dialect based on the datbase environment on the console.

```
<!-- Echo all executed SQL to stdout -->
<property name="format_sql">true</property>
```

This property is used to fromat the Query generated by the dialect based on the datbase environment on the console.

```
<property name =
'hibernate.hbm2ddl.auto">[validate/create/update/create-drop]</property>
```

validate =>hibernate creates no table, programmer should create or modify tables manually.
this is considered as default value.
create = > hibernate always creates new table,if table exists it will drop the table.
update => hibernate creates new table,if table doesnot exists, otherwise it will reuse the same table.
create-drop=>This option is used for testing purpose not in development
creates a new table and perform operation,at last it will drop the table.

4. Test class
   To perform any operation in hibernate we must write Test class.
   It is used to perform operation like select/nonselect.
   "Transaction" object is requried if we perform non-select operation
   "Transaction" object is not required if we perform select operation.

```
Test class coding and its execution flow
================================
1. Create a configuration object
2. Load .cfg.xml file into configuration object using configure().
3. Build SessionFactory object using cfg which handles
            a. Loading driver class
            b. Creating connection
            c. Prepare statement objects.
                        ;;;;;;
4. use SessionFactory and get Session object to perform Persistence operation.
5. Begin Transaction, if the operation in Non-Select.
6. Now perform operation using Session object.
7. Commit or rollback if transaction has started.
8. close the session at the end.

Note: To specify the configuration details and mapping details we need to write xml
file.
          if the filename is hibernate.cfg.xml then it promotes automatic loading,
otherwise
          we need to read those data from "FileInputStream".

1. Using hibernate persistence operations can be peformed using methods as shown
below

SingleRowOperation(SRO)
======================
a. insert query
            session.save(,)
            session.persist(,)
b. select query
            session.load(,) => if the record is not available it would return
"ObjectNotFoundException".
            session.get(,)   => If the record doesnt exists, it would return null.
c. updateQuery
            session.update(,)
            session.saveOrUpdate(,)=> first performed selection,record found, so
latest values it updated using update query.
                                                => first performed seelction,record
not found, so perform insert operation.
d. deleteQuery
            session.delete(,)=> Check whether record exists,only if it exists
perform deletion.

BulkOperation(mulitple rows)
1. HQL/JPQL
2. NativeSQL
3. CriterionAPI/QBC

Assignment
===========
1. Using layerd approach perform CRUD operation in console mode(persistencelogic->
hibernate)
2. Using layerd approach perform CRUD operation in webbased mode(using servlets
only ,persistencelogic-> hibernate)
2. Using layerd approach perform CRUD operation in webbased mode(using
servlets(controller) ,

                              persistencelogic(hibernate)
Viewpart(jsp))
```