Persistence logic
==========
The logic which is written to perform peristence operation is called "Persistence logic".
operations are CRUD/SCUD/CURD.

To write persistence logic injava we have technology and framework
a. technology => JDBC
b. framework => ORM tools like hiberante, jpa,springorm,springdatajpa(hotcake),.....

When we already have JDBC as persitence logic, what is the need to for ORM?
limitations of JDBC
a. 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.
JAVA => WORA
JAVA + JDBC ==> WORA(not supported)

b. 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.

C. 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{

    }
```

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

e. 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.

f. 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".

g. 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.

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

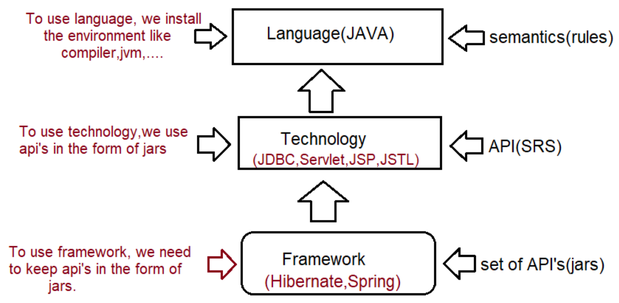
i. JDBC supports only positional parameters,it is difficult for the user to inject the values
It doesnot support namedparamaters.
String sqlInsertQuery = "insert into student(`name`,`email`,`city`,`country`) values(?,?,?,?)";
String sqlInsertQuery = "insert into student(`name`,`email`,`city`,`country`)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

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

h. 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.

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

Solution: To all the problems mentioned above we develpo persistence logic usin "ORM".
ORM tools are hiberante,eclipselink,ibatis,jpa,.....
ORM -> Object Relation Mapping

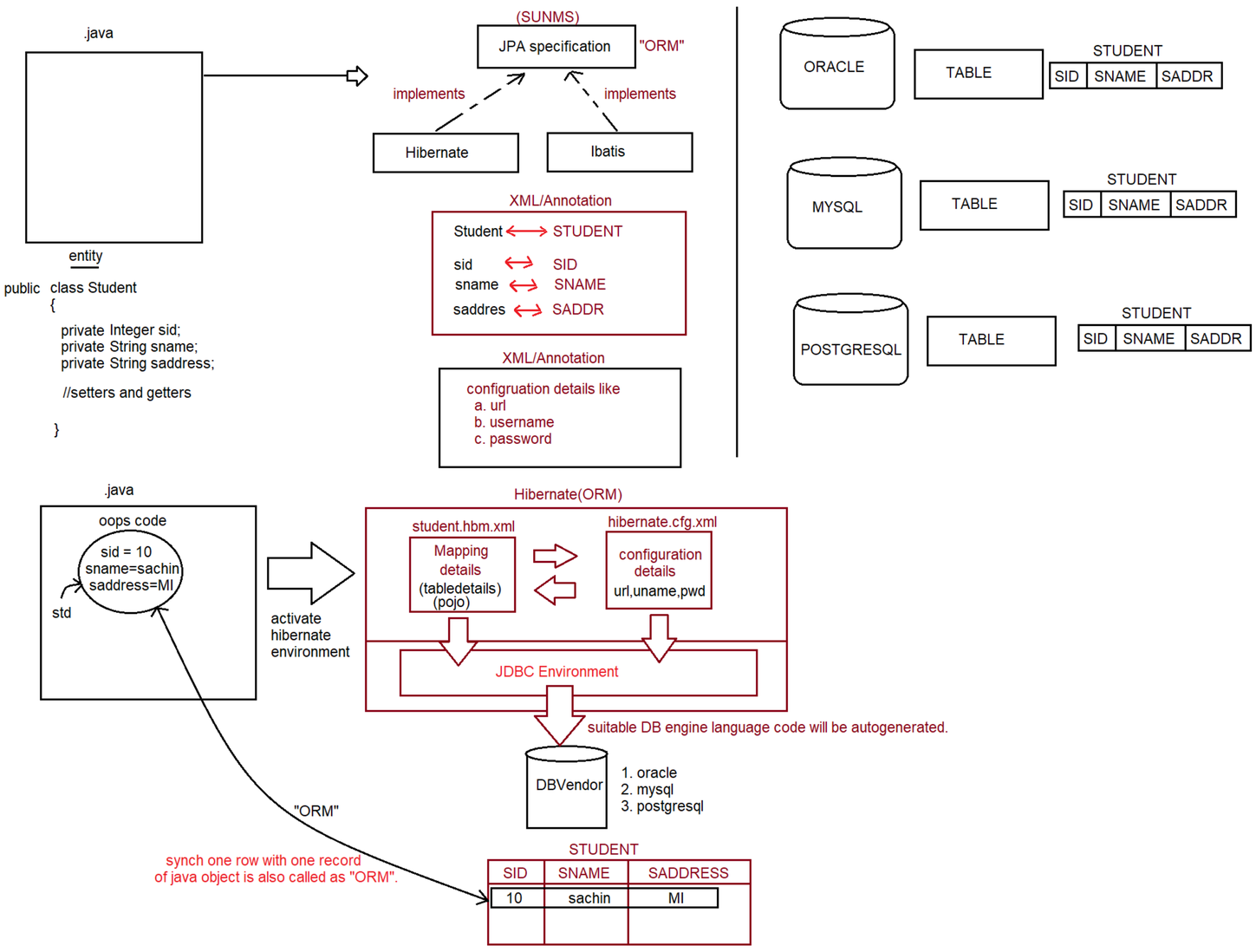

To use language, we install the environment like compiler,jvm,....
Language(JAVA)
semantics(rules)
To use technology,we use api's in the form of jars
Technology
(JDBC,Servlet,JSP,JSTL)
API(SRS)
To use framework, we need to keep api's in the form of jars.
Framework
(Hibernate,Spring)
set of API's(jars)

To overcome the limitations of JDBC, we need to opt for ORM.
1. JDBC code is not portable.
2. JDBC we can't deal with Object injection to the database as the query expects values.

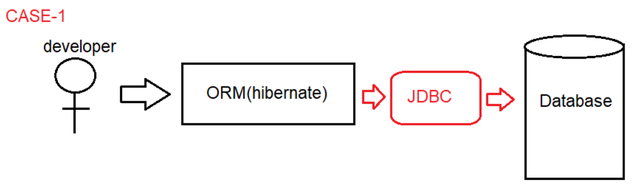
ORM => It stands for Object Relational Mapping
refer: images

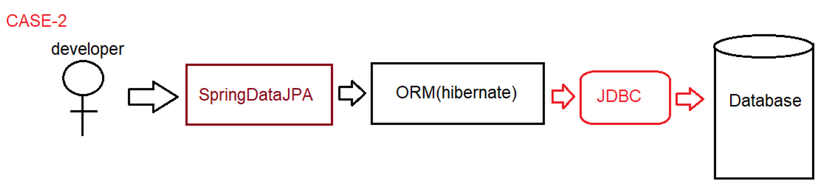
what is ORM?
it stands for Object relational mapping, where the programmer would map the table details with java object details
through xml/annotation.

JPA -> Java Persistence API (set of rules and guidelines to implement ORM)
Hibernate is a tool/framework implemented for JPA specification given by SUNMS.


.java
entity
public class Student
{
private Integer sid;
private String sname;
private String saddress;
//setters and getters
}
(SUNMS)
JPA specification
"ORM"
implements
implements
Hibernate
Ibatis
XML/Annotation
Student ⟷ STUDENT
sid ⟷ SID
sname ⟷ SNAME
saddres ⟷ SADDR
XML/Annotation
configruation details like
a. url
b. username
c. password
ORACLE
TABLE
STUDENT
SID SNAME SADDR
MYSQL
TABLE
STUDENT
SID SNAME SADDR
POSTGRESQL
TABLE
STUDENT
SID SNAME SADDR

.java
oops code
sid = 10
sname=sachin
saddress=MI
std
activate
hibernate
environment
Hibernate(ORM)
student.hbm.xml
Mapping
details
(tabledetails)
(pojo)
hibernate.cfg.xml
configuration
details
url,uname,pwd
JDBC Environment
suitable DB engine language code will be autogenerated.
DBVendor
1. oracle
2. mysql
3. postgresql
"ORM"
synch one row with one record
of java object is also called as "ORM".
STUDENT
SID SNAME SADDRESS
10 sachin MI


CASE-1
developer
ORM(hibernate)
JDBC
Database


CASE-2
developer
SpringDataJPA
ORM(hibernate)
JDBC
Database

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

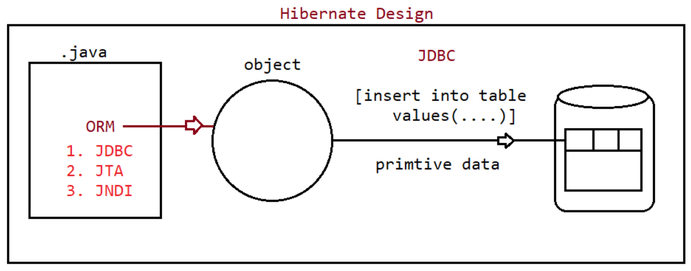
SingleRowOperation(SRO)
==========
a. insert query
            session.save(,)
            session.persist(,)
b. select query
            session.load(,) => if the record is not available it would return "ObjectNotFoundException".
            session.get(,)   => If the record doesnt exists, it would 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.

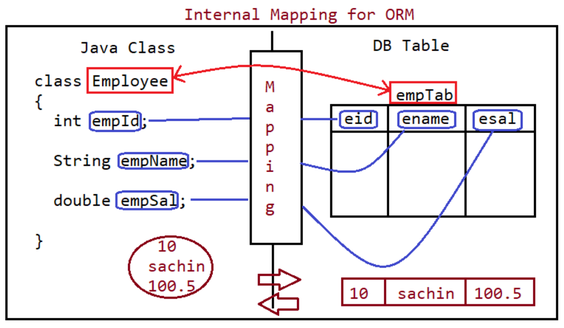
BulkOperation(mulitple rows)
1. HQL/JPQL
2. NativeSQL
3. CriterionAPI/QBC

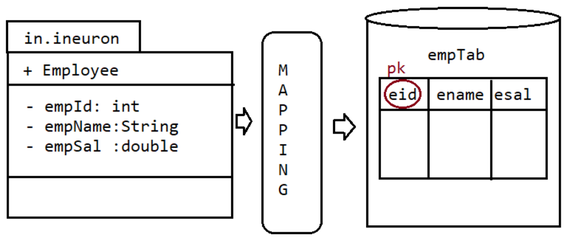
Assignment
==========
1. Using layerd approach perform CRUD operation in console mode(persistencelogic-> hibernate)
2. Using layerd approach perform CRUD operation in webbased mode(using servlets only ,persistencelogic-> hibernate)
2. Using layerd approach perform CRUD operation in webbased mode(using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

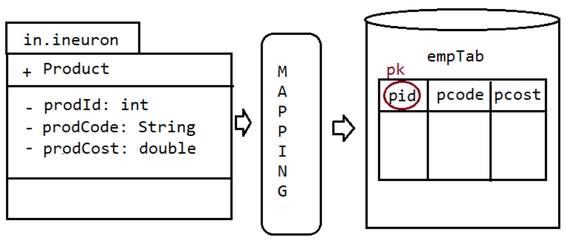
                                persistencelogic(hibernate)

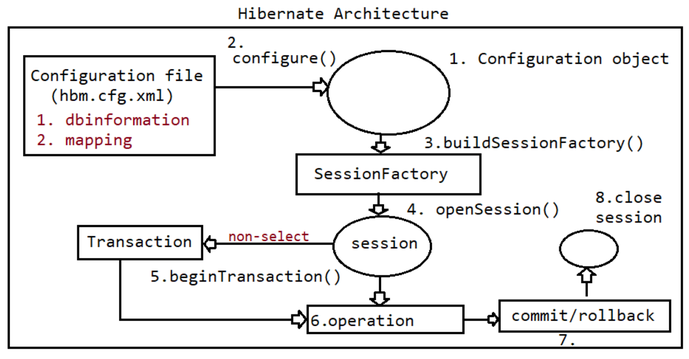
Viewpart(jsp))

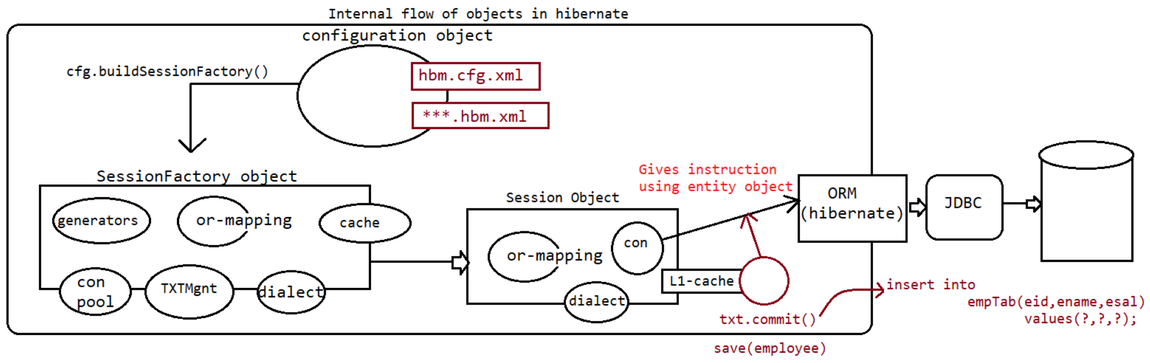

Hibernate Design
.java
ORM
1. JDBC
2. JTA
3. JNDI
object
JDBC
[insert into table
values(....)]
primtive data

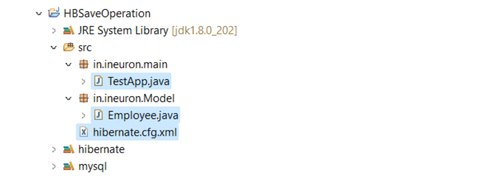

Internal Mapping for ORM
Java Class
DB Table
class Employee
{
int empId;
String empName;
double empSal;
}
Mapping
empTab
eid
ename
esal
10
sachin
100.5
10 | sachin | 100.5

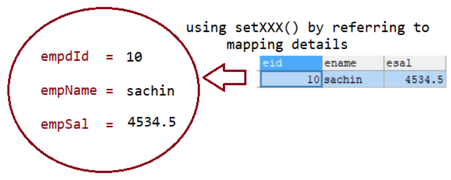

in.ineuron
+ Employee
- empId: int
- empName:String
- empSal :double
MAPPING
empTab
pk
eid | ename | esal

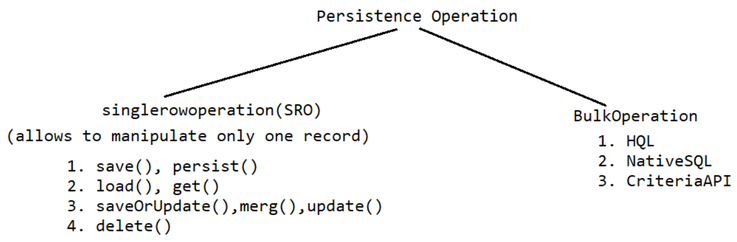

in.ineuron
+ Product
- prodId: int
- prodCode: String
- prodCost: double
MAPPING
empTab
pk
pid | pcode | pcost

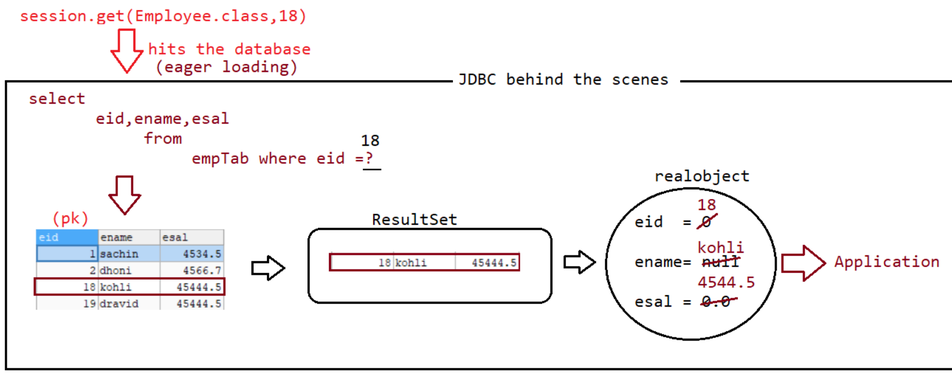

Hibernate Architecture
Configuration file
(hbm.cfg.xml)
1. dbinformation
2. mapping
2.
configure()
1. Configuration object
3.buildSessionFactory()
SessionFactory
4. openSession()
8.close
session
Transaction
non-select
session
5.beginTransaction()
6.operation
commit/rollback
7.

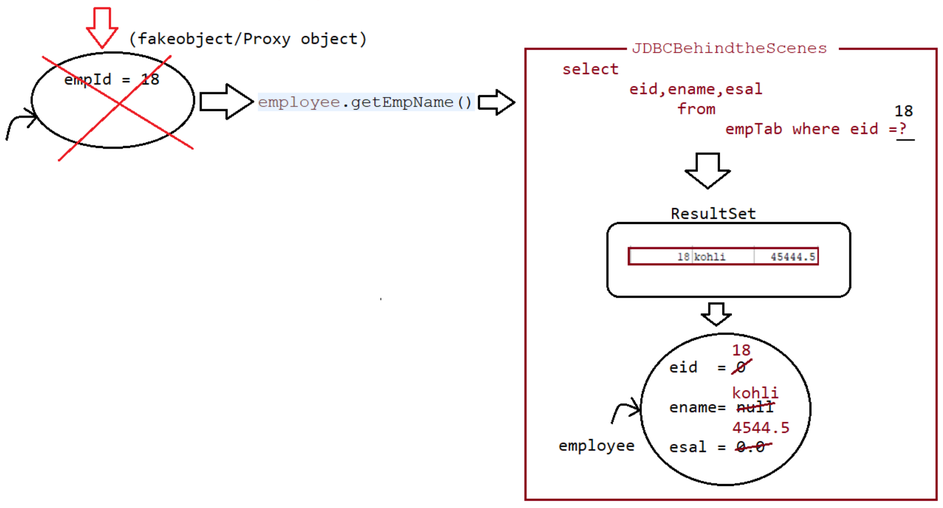

Internal flow of objects in hibernate
configuration object
cfg.buildSessionFactory()
hbm.cfg.xml
***.hbm.xml
SessionFactory object
generators
or-mapping
cache
con
pool
TXTMgnt
dialect
Session Object
or-mapping
con
dialect
L1-cache
Gives instruction
using entity object
ORM
(hibernate)
JDBC
txt.commit()
insert into
empTab(eid,ename,esal)
values(?,?,?);
save(employee)


HBSaveOperation
JRE System Library [jdk1.8.0_202]
src
in.ineuron.main
TestApp.java
in.ineuron.Model
Employee.java
hibernate.cfg.xml
hibernate
mysql


empdId = 10
empName = sachin
empSal = 4534.5
using setXXX() by referring to
mapping details
eid | ename | esal
10 | sachin | 4534.5


Persistence Operation
singlerowoperation(SRO)
(allows to manipulate only one record)
1. save(), persist()
2. load(), get()
3. saveOrUpdate(),merg(),update()
4. delete()
BulkOperation
1. HQL
2. NativeSQL
3. CriteriaAPI


session.get(Employee.class,18)
hits the database
(eager loading)
JDBC behind the scenes
select
eid,ename,esal
from
empTab where eid =?
18
(pk)
eid ename esal
1 sachin 4534.5
2 dhoni 4566.7
18 kohli 45444.5
19 dravid 45444.5
ResultSet
18 kohli 45444.5
realobject
eid = 0 18
ename= null kohli
esal = 0.0 4544.5
Application


session.load(Employee.class,18);
(fakeobject/Proxy object)
empId = 18
employee.getEmpName()
JDBCBehindtheScenes
select
eid,ename,esal
from
empTab where eid =?
18
ResultSet
18 kohli 45444.5
employee
eid = 0 18
ename= null kohli
esal = 0.0 4544.5

```
save()(
 => Serializable .save(Object obj)
=> This method gives instructions to save object and also return the assigned or
generated identity value back to
      the application as the return value.
=> This method is own method of hibernate(not as per specification of JPA).

note: if generators are not configure, then value assigned to id property will be
returned  as identity value.
            eg: increment,sequence,hilo,......

Employee.java
============
@Id
@Column(name = "eid")
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Integer empId;
      As noticed above we have told hibernated to genreate the value of empId, so
the generated value is "AutoIncrement"
      for MySQLDB

persist()
      =>void persist(Object object)
      => return type is void, cannot return the identity value.
      => This method is given by JPA specification and it is implemented by
Hibernate.
      => Gives instruction to hiberante to perform save operation on the object.
      => persist() doesnot allows to work with generators.

code
====
Employee employee = new Employee();
employee.setEmpId(19);
employee.setEmpName("dravid");
employee.setEmpSal(45444.5);

session.persist(employee);
            refer: HBPersistOperation


Performing loading operation
=======================
get()[best suited in standalone application]
      It perform eager loading.(hits the database and gets the record from dbtable
and stores in Entity class object
                                                irrespective of whether we use that
Object/not)
      if we call get(), automatically the hibernate will generate the sqlquery and
hits the database.
      even if the record is not available still its the database,as a result of
which we say get() is costly in realtime applications


load()[best suited  in webapplications]
      It perform lazy loading(hits the database only when we use the object data
other than primaary keyvalue)
      Upong lazy loading,first hibernate creates the proxy object and sets only pk
value to it.
      when we use getter methods on non-primary keyvalue then hibernate will hit
```

the database by executing selectquery.
      if the record found then it will create a new object and injects the value to that object,otherwise it would result in
      "ObjectNotFoundException".

get()
       => It supports eager loading
       => It won't generated proxy object
       => returns null if record not available
      => suitable to check wheter record available or not.
      => Creates only object for Entity class.
      => Best suited for standalone applications(gauranteed that loaded object will be used)
load()
      => It supports lazy loading
      => It generates proxy object
      => It throws ObjectNotFoundException
      => not suitable
      => Creates 2 object(proxy + Entity class)
      => Best suited for webapplications(DAO-> Service-> Controller-> View(jsp using the object is not gauranteed))

Update Operation
===============
1.update()
      i> void update(Object object)
      This method is used to modify the record of the DBTable.
      Set the primary key value and change the other non-primary data for updation.
      To use update(), we should remember wheter record exists or not for the give primary key value.
      otherwise it would result in "HibernateException".
      It would directly generate "update query" without "select query".

      ii>Load the object from database and then modify
                  Here we won't get Exception as the object is available we do modify the Object.

2.saveOrUpdate()
            If the object/record is already available only then it will update the record otherwise it will insert/create a new record

3. merge()
            On the loaded object, if we want to update the data then we need to go for merge()

4. referesh()
            => will be discussed later to demonstrate the synchronziation of dbrecord to java object.

persistence operations
==========
  insert=======> save(obj)/persist(obj)
  read =======> get(obj,pk)/load(obj,pk)
  update======> saveOrUpdate()/update()/merge()
  delete ======> delete(obj)

Deleting object
==========
  => It refers to deleting the record represented by the Entity class Object.
  => Delete the record based on the id value of the given entity object.
  => public void delete(Object obj)

Approach-1
==========
 session.delete(Object obj)
            directly we are trying to delete the object, so not a good approach

Approach-2
==========
  First load the object, if found only then delete the object.
        refer: HBDeleteOperation

How can u show that synchronization would exists b/w EntityObject to DBTable row?
                  refer: HBSynchronizationOperation

Generators in hibernate
==========
 MySQL => primary key value where the generation of these values are made automatic.
While creating a table, we can inform hibernate to create a columns with primary key value using @Id.
It is also possible to set the values to these primary key columns using Generators in hibernate.

There are 3 types of generators in hibernate
      a. Hibernate supplied generators
      b. JPA generators
      c. Custom generators

Hibernate supplied generators
==========
a. assigned
b. increment
c. identity
d. sequence
e. hilo
f. seqhilo
g. native
h. foreign
i. select
j. uuid
h. guid

assigned
==========
   If we use this algorithm then explicitly we need to specify the primary key value to the table.
   assigned => org.hibernate.id.Assigned

It works with all databases as we need to give the primary key value.

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

increment
=========
- It uses max(value) + 1 to generate the primary key value which is of int type
- Works with all Database.
- If dbTable is empty it will generate 1 as the identity value.
- increment => org.hibernate.id.IncrementGenerator

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

identity
======
- Generates the value which are of type int,long,short.
- This generator can be used only in such databases that supports "identity" columns.
- This generator works for MySQL,DB2,SQlServer,...
- it wont work in Oracle,PostgreSQL....
- MySQL=> AutoIncrement feature

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

JPA generators
============
These are given by Sun MS JPA specification
It will work with all ORM Frameworks
4 generators are given
- a. Identity
- b. sequence
- c. table
- d. auto

identity
======
It works with MySQL database.
It is similar to AutoIncrement feature of primary key column.

```
@Id
@Column(name = "eid")
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Integer empId;
```

auto
===
It works with all database

Depending upon the db engine platform, automatically hibernate will use hiberanate_sequence algorithm to generate the primary key value.

```
@Id
@Column(name = "eid")
@GeneratedValue(strategy = GenerationType.AUTO)
private Integer empId;
```

```
pid(pk)
pname
deptno
projId(pk)
projName

@Embeddable
class ProjectInfo
{
      private Integer pid;
      private Integer projId;
}

@Entity
class ProgrammerProjectinfo
{
       @EmbededId
      ProjectInfo info;//HAS-A relationship
      private String pname;
      private Intger deptNo;
      private String projName;
}
```

refer: HBCompositeIDApp

Inserting Date and Time App using hibernate

==========

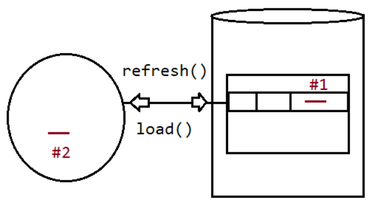
If we are using java.util.* or java.calendar.* then we need to use @Temporal annoatation

If we are using java.time.* then no need to use @Temporal Annoatation.

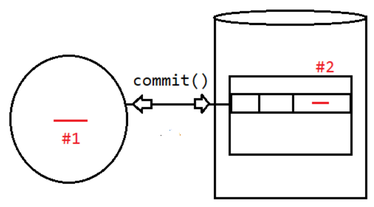
```
eg:
private LocalDate dob;
private LocalDateTime dom;
private LocalTime doj;

person.setDob(LocalDate.of(1973, 4, 24));
person.setDom(LocalDateTime.of(1987, 6, 21, 12, 35));
person.setDoj(LocalTime.of(10, 45));
```

refer: HBDateTimeApp

DBTable to RowObject



Entity Object to DB Row



synchornization is made automatically with
the help of "L1-Cache".

Versioning or ObjectVersioning
==========
It keeps track of how many times object/record is loaded and modified using hibernate.
It generates a special column of type numeric based special number property of Entity class to keep track of the modification.
This special property/col initial value is 0 and it is incrmented by 1 for every modification.
To configure this speical property we need to use one annotation called "@Version".
refer:HBVersioning

TimeStamping
==========
It allows us to keep track of Object is saved(record inserted) and object is lastly updated.
eg: keeping track of when the bank accoutn opened and lastly modified
To do this we use annotations like @CreationTimeStamp,@UpdateTimeStamp
refer:HBTimeStamping

Caching
=======
=>It is a temporoary memory that holds the data for temporary period of time.
=> Cache at client side will hold server data and uses it across the mulitple same requests to reduce the network trip
b/w client and server.
=> Hibernate supports 2 levels of Cache
a. First Level Cache(L-1 cache/session cache/default cache)
b. Second Level Cache(L2- cache/configurable cache)

eg: Stockmarket trading,live game score,weather report,.....

Note:
session.save(obj),session.saveOrUpdate(obj),sesssion.delete(obj) methods keep the object in L1cache unitll
tx.commit() is called.
session.get() will get the object and keep it in L-1 cache and same object will be used across mulitple session.get() method
calls with same entity object id.

Caching
a. evict(Object obj) => it will remove particular object from L1-cache
b. clear() -> it will remove all object present in L1-cache.
c. In L1-cache, duplicate objects won't be available.
refer: HBCachingApp

2nd level cache
==========
This caching is associated with "SessionFactory", so we call it as "Global Cache".
Application will start to search for entity object in the following order
a. L1 cache of current session(if not there)
b. L2 cache of SessionFactory object(if not there)
c. Collect from db and keep in L2 cache and L1 cache then give it to application.
It is a configurable cache and we can enbale or disable it.
hibernae supports L2 cache through "EHCache"

To configure EHCache in our hibernate projects we use
==========
1. Add EHCache jars to the project
2. configure ehcache.xml as shown below

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

Also make changes in hibernate.cfg.xml file as shown below

```
                                    <!-- Configuring EH cache... -->
      <property name="hibernate.cache.use_second_level_cache">true</property>
      <property
name="hibernate.cache.region.factory_class">org.hibernate.cache.ehcache.EhCacheRegi
onFactory</property>
      <property
name="net.sf.ehcache.configurationResourceName">ehcache.xml</property>
```

3. In the model class inform hiberante to use Caching startegy for Read purpose.

```
@Entity
@Cache(usage = CacheConcurrencyStrategy.READ_ONLY)//It specifies caching Strategy
public class InsurancePolicy implements Serializable{}
```

Working with LOB's
==========
To work with LOB in hibernate we use

```
            @Lob
            private byte[] photo;

            @Lob
            private char[] resume;
```

refer: HBLobOperation

Customgenerator
==========

Hibernate and JPA had supplied predefined genearator to create primary key value for almost all databases.

eg: identity,increment,auto,sequence,.....

if we want a primary key value to be generated for our columns as per our application needs then we need to go customgenerators.

To create our own generator we need to implement an interface called "IdentifierGenerator"

It is a functional interface which contains only one method

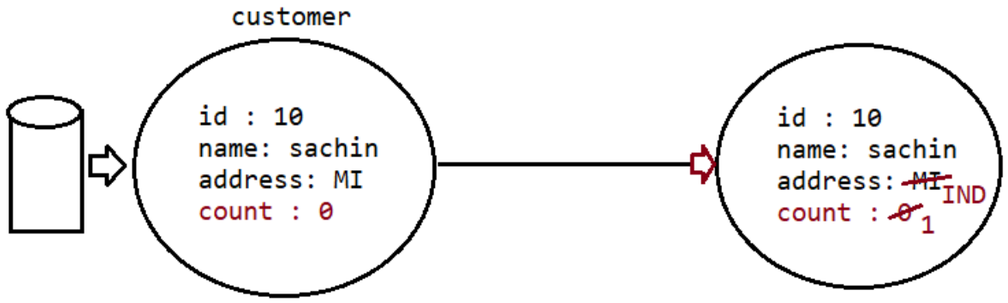
public Serializable generate(SharedSessionContractImplementor session,Object object) throws HBE

```
<id name="empId" type="java.lang.Integer" column="eid" >
                  <generator class="in.ineuron.generator.RandomGenerator"/>
</id>
```

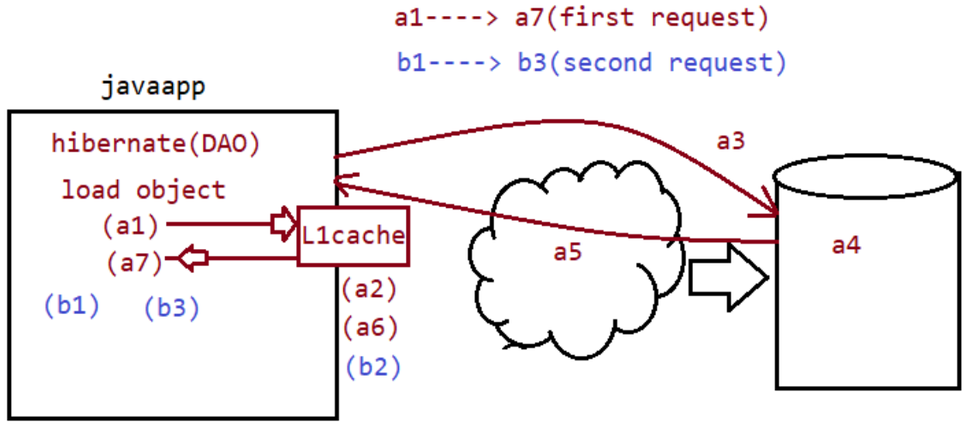
refer: HBCustomGeneratorApp

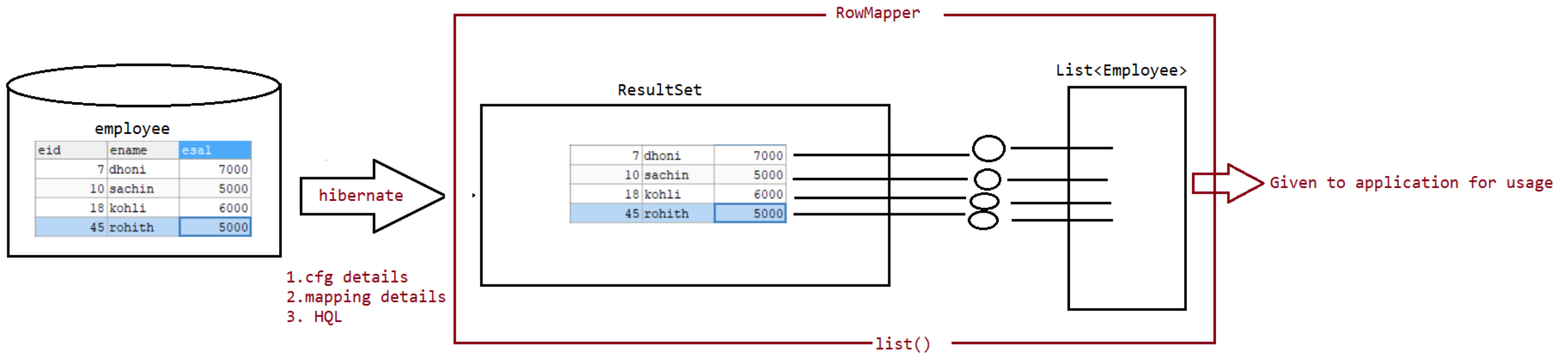
Genearate unique value for student id of iNeuron in the following style
INEURON0101,INEURON00102,INEURON00103,....

ObjectVersioning

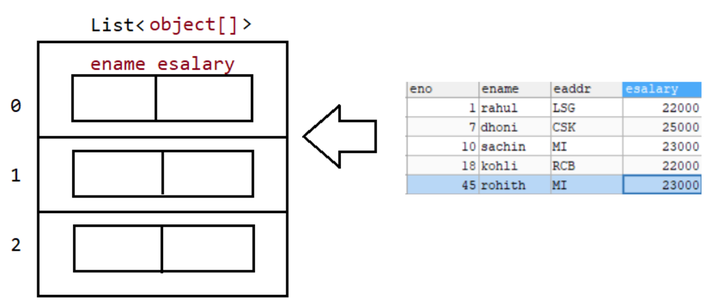


Caching





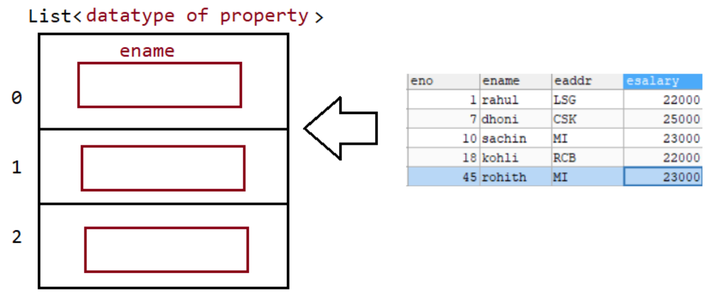
Getting specific columns value from Database



eno	ename	eaddr	esalary
1	rahul	LSG	22000
7	dhoni	CSK	25000
10	sachin	MI	23000
18	kohli	RCB	22000
45	rohith	MI	23000

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

Getting One columns value from Database



eno	ename	eaddr	esalary
1	rahul	LSG	22000
7	dhoni	CSK	25000
10	sachin	MI	23000
18	kohli	RCB	22000
45	rohith	MI	23000

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

Hibernate topics pending list
=======================
1. Bulk Operation
2. Locking
3. Stored Procedure
4. Mapping(1-1,1-*,*-*,*-1)
5. Connection pooling
6. Pagination
7. NamedQuery

Connection pooling
 => SessionFactory object holds jdb connection pool having set of ready made jdbc connection objects and uses them in the
      creation of HB session objects.
=> By default hibernate uses built in jdbc connection pool which is not suitable for production environment because of
     performance issue.
=> To control hibernate build in jdbc connection pool we write the following property in hibernate.cfg.xml
            <property name="hibernate.connection.pool_size">25</property>

Which jdbcconnection pool is best with hibernate integration?
      standalone mode -> Don't use hibernate built in jdbc connection pool
                                    use Third party supplied jdbcconnection pool like
                                    hikaricp(best in market),proxool,viboor,agroal,c3po.....
      webapplication mode-> Don't use 3rd party supplied only, use underlying server provided
                                          connection pool from servers like weblogic,tomcat,wildfly,....

Configuration of hibernate.cfg.xml file for hikaricp production
==
<!-- Hikari cp configuration -->
      <property name="hibernate.connection.provider_class">org.hibernate.hikaricp.internal.HikariCPConnectionProvider</property>

<!-- Maximum waiting time for a connection from the pool (20sec)-->
      <property name="hibernate.hikari.connectionTimeout">20000</property>

<!-- Minimum number of ideal connections in the pool(10 objects) -->
      <property name="hibernate.hikari.minimumIdle">10</property>

<!-- Maximum number of actual connection in the pool(20objects0 -->
      <property name="hibernate.hikari.maximumPoolSize">20</property>

<!-- Maximum time that a connection is allowed to sit ideal in the pool(300secs) -->
      <property name="hibernate.hikari.idleTimeout">30000</property>

                  refer: HibernateBuiltInConnectionPool

Note: DataSource(I)---------> jars provider should implement and give the class.

Why are we not configuring the Datasource class directly in hibernate? why are configuring connection provider class name?

Answer. hibernate f/w is designed to pickup the datasource class based on the connection provider that we have configured.
              by configuring in this style, we can restrict datasource and jdbc connection pool associated with hibernate.
                  hibernate will give support only for few thirdparty vendors like
                              a. hikaricp(best) b. c3po c.proxool d. viboor e. agroal

BulkOperation
============
=> To select or manipulate one or more record/object having our choice criterial value we need to go for "Bulk operation".
      a. HQL.
      b. Native SQL.
      c. Criterian API.

HQL
====
 1. HQL stands for Hibernate Query Language.
 2. It uses Objects based Query Language(these queries will be written based on the entity class names and properties name)
 3. Hibernate dialect internally converts HQL queries to DB specific SQL Queries.
 4. HQL queries are DBIndependent and they supports portability.
 5. HQL supports both select and non-select operation
 6. HQL can also be used to perform SingleRowOperation(SRO) and also for bulk operation having our choice conditions/
      criteria.
 7. HQL supports positional params(?)(supported only in older versions) and also it supports named params(=:name)
 8. HQL keywords are not case sensitive,but entity class names and properties names are case sensitive.
 9. HQL supports relational operators, conditional statements, joins,aggregate functions, projections,....

eg:
SQL> SELECT * FROM EMP WHERE EMPNO>=? AND EMPNO<=?
HQL> FROM in.ineuron.entity.Employee WHERE eno>=? AND eno<=?(positional param)
HQL> FROM in.ineuron.entity.Employee WHERE eno=:firstNum AND eno=:secondNum(named param)

SQL> DELETE FROM EMP WHERE JOB=?
HQL> DELETE FROM in.ineuron.entity.Employee WHERE job=?
HQL> DELETE FROM in.ineuron.entity.Employee WHERE job=:desg

Note: if we are selecting all columns/properties in the HQL Select query then placing select keyword is optional.

HQL select Queries
      a. Entity Queries(Getting all properties values of the record)
                  eg: FROM in.ineuron.entity.Employee(with or without condition)
      b. Scalar Queries(Getting specific column or specific multiple column values)
                  eg: SELECT eno,ename,eaddr FROM in.ineuron.entity.Employee (with or without condition)
                        SELECT eno From in.ineuron.entity.Employee(with or without condition)
                        SELECT count(*) From in.ineuron.entity.Employee

Example to get All the records from the DBTable using hibernate
==========
```
Query<Employee> query = session.createQuery("FROM in.ineuron.Model.Employee");
List<Employee> employees = query.list();
employees.forEach(employee -> System.out.println(employee));
```

Note:
If we use xml approach setter and getter methods are mandatory, but if we use Annotations for mapping setter and getter
methods are not required, hibernate internally uses reflection api and it binds the value from ResultSet to private properties
of the Model.

Note:
In plain jdbc converting ResultSet object to DTO object is a manual process, where as in orm framework like hibernate,
spring jdbc, spring orm and spring datajpa same happens internally using "rowmapper"concept.

list() or getResultList()
==========
1. It internally uses eager loading for bulk operations.
2. It returns the collection directly.
3. Generates only one query to get all the records.
4. suitable for good performance.
5. It won't generate proxy object.
6. It is not deprecated and it is the industry standard approach.

iterate()
======
1. It internally uses lazy loading for bulk operation
2. It returns the iterator pointing to collection object.
3. Generates n+1 query to get all the records.
4. Not suitable, it degrades the performance.
5. It generates Proxy object.
6. It is deprecated becoz of performance issue.

Bulk operation for retreiving the record
---
a. All columns ========> List<EntityType>
b. multiple columns ====> List<Object[]>
c. only one column ====> List<datatype of property>

Note: To avoid null checking in our application,we can use JDK8 supplied api called "Optional".
```
Employee employee = query.uniqueResult();
 if (employee != null)
      System.out.println(employee);
else
      System.out.println("Record not found for the given id ::"+id);

Optional<Employee> optional = query.uniqueResultOptional();
if (optional.isPresent()) {
            Employee employee = optional.get();
            System.out.println(employee);
} else{
            System.out.println("Record not found for the given id :: " + id);
}
```

Note: we should use get()/load() if we are getting record based on primary key value,
           we can use uniqueResult()/uniqueResultOptional() if we are getting record based on non-primary key value.

refer: HibernateBulkOperation

HQLInsert operation

 It is not possible to insert one record to the database directly using insert query,becoz linking generators with HQL insert query
 is not possible. so we use session.save() method to insert a record.
We can use HQL insert query to insert bulk record into one db table by selecting them from another db table.
            eg: insert into .... values (query is not given)
                  insert into ... SELECT FROM ..... (given to perform bulk operation)

insurance policy (table filled with records) ---------->
premium_insurance_policy(new table) where tenure>=25 years

Hibernate pending topics
---
1. Locking
2. Pagination
3. StoredProcedure
4. Mapping(tommo i will discuss)
5. NamedQuery,NativeSQL Query and Criterion Api
6. Project

NamedHQLQuery

  => So far our HQL query is specific to one session object becoz query object is created having hard coded HQl query on session object.
  => To make our HQL query accessible and executable through multiple session objects of Mulitple DAO classes or client a[pps we need to
        go for "NamedHQL" query.
  => We defined NamedHQLQuery in mapping file using <query> tag or in Entity class using "@NamedQuery" havinig logical name and we
       access and execute that HQL query in DAO class.

Code using Annotation
==
```
@Entity
@NamedQuery(name = "HQL_INSERT_TRANSFER_POLICIES",
         query = "INSERT INTO
in.ineuron.model.PremiumInsurancePolicy(policyId,policyName,policyType,company,tenu
re)
      SELECT i.policyId,i.policyName,i.policyType,i.company,i.tenure FROM
in.ineuron.model.InsurancePolicy as i WHERE i.tenure>=:tenure")
```

refer:: HibernateNamedHQLInsertOperation

NativeSQL Query

 => It is given to execute plain SQL queries that are supported by underlying DB S/w.
 => We need to use these operations only when it is not possible through HQL
                 eg: inserting a single record.
 => It supports both select and non select operation
 => These queries performance is bit good compared to HQL because they go to sql directly without any conversion.
 => We write a query using table name and column names.

working with select operation
---
1. working with single record.
2. working with all record.

Working with Nonselect operation
1. using NamedNativeQuery
2. Directly writing the query inside DAOImpl class.

refer: HBNativeSQLQuery

Criterion Api

   SRO ====> hibernate persistence methods
   bulkoperations => we use HQL(query written using classname and properties) to write queries.

=> In case of Criterion api, we can perform both singlerow and bulkoperations without using any queries just like java statements.
=> Criterion api will generate SQL queries based on the given entity classnames and properties name.
=> It doesnot support non-select operation,it supports only select operation.
=> Using Criteria object we can add 3 object
    a. Criterion objects(for where clause condition)
    b. Project objects(for scalar select operation)
    c. Order object(for orderBy operations)

There are 2 modes of writing Criterion api
a. HB QBC(Query by Criteria)====> specific to hibernate only
b. JPA QBC ===> common to all ORM framework
refer: HBQBCApi

Pagination
========
Displaying large volume of records into muliptle pages is called as "pagination".
Hibernate supports pagination through QBC
1. setFirstResult(int pageNo)
2. setMaxResult(int maxNo)

StoredProcedure calls in hibernate
==========================
1. we use ProcedureCall(I) to make a call to storedprocedure
2. we use ParameterMode(Enum) to specify the IN,OUT,INOUT Param

To get all the columns value data in the form of ResultSet

```
ProcedureCall procedureCall =
session.createStoredProcedureCall("GET_POLICIES_BY_TENURE",InsurancePolicy.class);
procedureCall.registerParameter(1, Integer.class,
ParameterMode.IN).bindValue(start);
procedureCall.registerParameter(2, Integer.class, ParameterMode.IN).bindValue(end);
List<InsurancePolicy> list = procedureCall.getResultList();
```

To get specific columns based on the input type

```
ProcedureCall procedureCall =
session.createStoredProcedureCall("GET_POLICY_BY_ID");
procedureCall.registerParameter(1, Integer.class, ParameterMode.IN).bindValue(id);
procedureCall.registerParameter(2, String.class, ParameterMode.OUT);
procedureCall.registerParameter(3, String.class, ParameterMode.OUT);
procedureCall.registerParameter(4, String.class, ParameterMode.OUT);

String policyName = (String) procedureCall.getOutputParameterValue(2);
String companyName = (String) procedureCall.getOutputParameterValue(3);
String policyType = (String) procedureCall.getOutputParameterValue(4);
```

refer: HBStoredProcedure

Locking in hibernate
==========
If multiple apps or threads simultaneoulsy accessing and manipulating the records there is a possibility of getting
concurrency problem.
To Avoid this problem we need to use "Locking " of a record in hibernate.

Hibernate supports 2 levels of Locking
      a. Optimistic Locking
                => It allows secondthread simultaneously to access and modify the record, then first thread notices the
                     modification and throws "Exception".
                => To enable this locking we need to use "@Version" in our application.
                => if we use @Version then automatically optimisitic locking will be achieved.

      b. Pesimistic Locking
                => It will allow FirstThread to Lock the record itself,so if the second thread tries to access and modify the
                     record then it should throw "Exception".
                => To enable this locking we need to use
                              session.get(Class c,Serializable s,LOCKMODE.UPGRADE_NOWAIT) as the third argument value
                                    refer: HBOptimisticLockingApp, HBPesimisticLockingApp

Mapping in hibernate
==========
   In realtime applications, we use mapping to link two classes and these linking at the database side will happen in the form
   of primary-key foreign-key relationship.
   At the java side, we can link 2 classes through "Association".
   At the database side we don't have these linking,but we have something called as "Primary-Foreing" key relationship.
   To support this feature at the hibernate side we have "Mapping".
   There are 4 types of hibernate mapping
      a. One-One Association Mapping
      b. One-Many Association Mapping
      c. Many-One Association Mapping
      d. Many-Many Association Mapping

UniDirectional

One-One Association Mapping

It refers to relationship b/w 2 entities where one instance of one entity should be mapped with exactly one instance of
another entity.
                eg: One Employee has One Account
Annotation used is :: @OneToOne(cascade = CascadeType.ALL)
cascade specifies, if we delete employee table automatically account table also should be deleted

One-Many Association Mapping

It refers to relationship b/w 2 entities where one instance of one entity should be mapped with multiple instances of
another entity.

eg: Single Department has Multiple Employees
Annotation used is :: @OneToMany(cascade = CascadeType.ALL)

Many-One Association Mapping

It refers to relationship b/w 2 entities where mulitple instance of an entity should be mapped with exactly one instance of
another entity.
eg: Multiple Students have joined with Single Branch
Annotation used is :: @ManytoOne(cascade = CascadeType.ALL)

Many-Many Association Mapping

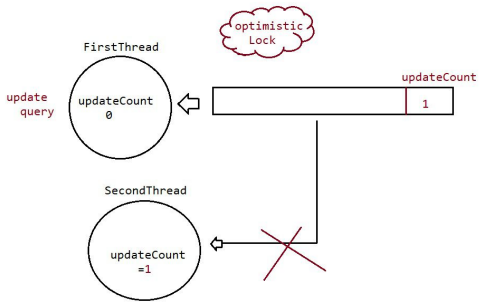
It refers to relationship b/w 2 entities where mulitple instances of an entity should be mapped with multiple instances of
another entity.
eg: Multiple Students have joined with Mulitple Courses.
Annotation used is :: @ManytoMany(cascade = CascadeType.ALL)

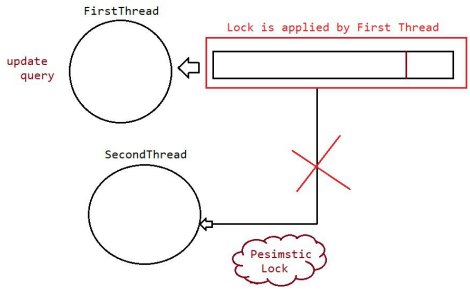
refer::
HB-Many-To-Many-Mapping
HB-Many-To-One-Mapping
HB-One-OneMapping
HB-One-To-Many-Mapping

Learn on ur own
---
1. Hiberante filters
2. Mapping(bi-directional)

Next Week

Spring core would start(SAT-SUN)
SpringBoot
    a. SpringMVC
    b. SpringDataJpa and SpringMongoDB
    c. SpringJDBC/SpringORM
    d. SpringAOP
    e. SpringRest
    f. Spring Mail
    g. SpringSecurity
    h. Microservices and deployment tools


optimistic
Lock
FirstThread
updateCount
update
query
updateCount
0
1
SecondThread
updateCount
=1


FirstThread
Lock is applied by First Thread
update
query
SecondThread
Pesimstic
Lock


CascadeType
ALL
PERSIST
MERGE
REMOVE
REFRESH
DETACH