Hibernate is a very good ORM.

Hibernate acts as a layer between java and database.

Hibernate uses JDBC internally and provides us easier way to work with databases.

We can create a POJO class and mark with annotation @Entity to make this class as a table in the database.

@Id annotation on a variable will make it as a primary key column in the table.

Automatically creates table from a class, with all variable names as column name and class name as table name.

Hibernate.cfg.xml file under the src/main/resources folder will have the details regarding database url, username, password and all class/Entities which are linked with database tables.



Listed the below steps to use hibernate in our class.

//step 1 -> create and configure the config object

Configuration config = new Configuration();

config.configure();

// step 2 -> session factory

SessionFactory factory = config.buildSessionFactory();

//step 3 -> creating session from factory

Session session = factory.openSession();

//step 4 -> create transaction object

Transaction tr = session.beginTransaction();

//step 5 -> create objects to insert in the database

Investor i1 = new Investor();

i1.setName("Yasin");

i1.setAmount(45000);

//step 6 -> save object into session

session.save(i1);

//step 7 -> commit transaction

tr.commit();

Query<Investor> query = session.createQuery("select i from Investor i");

List<Investor> list = query.list();

Iterator<Investor> iter = list.iterator();

while(iter.hasNext())

{

System.out.println(iter.next());

}

//step 8 -> close session

session.close();

Annotations:

@Entity(name="alien\_table") -> changes entity name and table name too in the DB

@Entity \n @Table(name="alien\_table") -> changes db table name alone and not Entity name in java

@Column(name="alien\_column") -> yeah you know now..

@Transient -> prevents a column not to be stored in DB

@OneToOne, @OneToMany, @ManyToOne, @ManyToMany

-> these are used to map relations between two tables

-> this is a very interesting handson

@OneToOne(mappedBy="column-name-of-tableB")

by default, the fetching type is LAZY, there is one more called EAGER.

-> this LAZY fetch value only when we need it

@oneToMany(mappedBy="alien", fetch=FetchType.EAGER)

-> set the fetch type to eager, and it loads in advance

getCurrentSession() -> default session of the hibernate context, we can use it and don’t have to close the session, it will get closed when session factory is closed.

openSession()

* we open a new session, then we have to close the same.
* Create new session for each thread in a multi-threaded project.

get() load()

* Loads actual data from DB -> returns a proxy object at the beginning
* Eager loading -> lazy loading

Hibernate – first level cache

* Enabled by default, and no option to disable.
* Improves performance by reducing number of queries executed in DB.
* Is associated with session object.
* Chache is lost of session is closed.

Hibernate – second level cache

* Is disabled by default, we can enable it through config.
* Example : EHCache