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