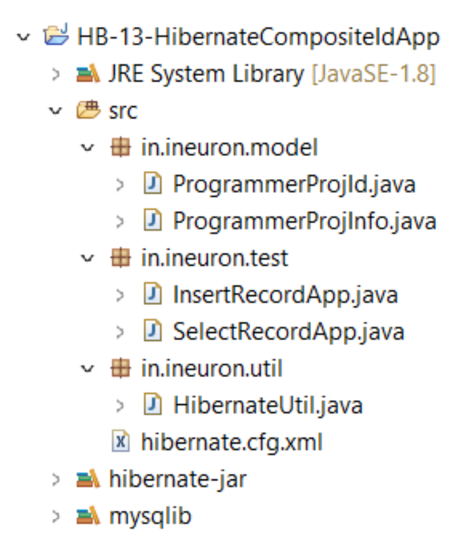
(pid,projId)

compositekey ⇦ Take 2 columns inside seperate class and inject that object in main class.

ProgrammerId   ProgrammerProjectInfo

ProgrammerProjectInfo

pid	pname	deptno	projId	projName
100	sachin	10	501	IPL
100	sachin	10	502	IND
101	kohli	19	501	IPL
101	kohli	19	502	IND



```
HB-13-HibernateCompositeIdApp
  JRE System Library [JavaSE-1.8]
  src
    in.ineuron.model
      ProgrammerProjId.java
      ProgrammerProjInfo.java
    in.ineuron.test
      InsertRecordApp.java
      SelectRecordApp.java
    in.ineuron.util
      HibernateUtil.java
    hibernate.cfg.xml
  hibernate-jar
  mysqlib
```

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