OneToOne:-
=========

--at table level,we can maintain OTO relation by taking FK as Unique.

unidirectional:-

--Assume one department has only one employee and one employee belongs from only one dept.

we can take other example also

ex:-

Emp --> Address

Person --> DL

--here we need to use @OneToOne annotation

```
@Entity
public class Department {

        @OneToOne(cascade = CascadeType.ALL)
        private Employee emp;

}
```

Main class:-

```
                Department d1=new Department();
                d1.setDname("Sales");
```

```
          d1.setLocation("kolkata");



          Employee emp = new Employee();
          emp.setEname("Ram");
          emp.setSalary(8500);

          d1.setEmp(emp);

          em.getTransaction().begin();

          em.persist(d1);

          em.getTransaction().commit();
```

--here 2 table will be created

1.employee (empid,name,salary)

2.department(did,dname,dlocation, emp_empid) (this emp_empid will be the FK)

--if we want to change this auto generated FK column name then we need to apply @JoinColumn anno

ex:-

```
@OneToOne
@JoinColumn(name="eid")
private Employee emp;
```

bidirectional:-
----

onetoone bidirectional :-
----

here on both side define opposit class variables:-

ex:-

```
Department:-
        @OnetoOne
        private Employee emp

Employee:-
        @OneToOne
        private Department dept
```

--while inserting the record we need to associate both object with each other.
ex:-

```
Department d1=new Department();
d1.setDname("Sales");
d1.setLocation("kolkata");


Employee emp = new Employee();
emp.setEname("Ram");
emp.setSalary(8500);

d1.setEmp(emp);
emp.setDept(d1);


em.getTransaction().begin();
em.persist(d1);
em.getTransaction().commit();
```

```
                    System.out.println("done..");
```

--in this case 2 table will be created both will containes the id of each other as FK as an extra column.

department:-(emp_empid as FK)

employee:- (dept_did as FK)

--if we want that only one table should maintains the FK col then we use mappedBy on any side.

ex:-

Department:-

```
  @OneToOne(mappedBy = "dept")
      private Employee emp;
```

--here Employee class maintains the FK id by name dept_did

--if we want to change this FK column name then

```
        @OneToOne
        @JoinColumn(name = "did_FK")
        private Department dept;
```

ex:-

Navigating from dept to emp:-

```
         Department d= em.find(Employee.class, 2).getDept();
```

```
                    System.out.println(d);
```

Inheritence Mapping:-
=====================

--In DataAccessLayer,between persistent class IS-A relationship is posibly exist.

--but in DB we don't have IS-A relationship between corresponding tables.

--to solve this problem we use inheritence mapping in HB.

JPA supports inheritence mapping with 3 strategy:-

1.one table for entire hirarchy/Single table.

2.table per sub-classes/Joined Table

3.Table per concreate class/ Table Per class:-

1.one table for entire hirarchy/Single table:
--

--this strategy is the default strategy in HB to perform the inheritance mapping

here we will take a single table with the all the columns, corresponding to generalized properties of super class and specialized properties for all the sub classes and one extra discriminator column.

--with the help of this descriminator value DB table maintains which Entity class of the inheritence hirarcy inserting the record.

Example:

Employee.java:-

```
@Entity
@Inheritence(strategy=InheritanceType.SINGLE_TABLE) // this line is optional, it is the
default strategy
public class Employee {

        @Id
        @GeneratedValue(strategy=GenerationType.AUTO)
        private int eid;
        private String name;

//getters and setters


}
```

ContractualEmployee.java:-

```
@Entity
public class ContractualEmployee extends Employee{

        private int noOfWorkingDays;
        private int costPerDay;

//setters and getters

}
```

SalaryEmployee.java:-

```
@Entity
public class SalaryEmployee extends Employee{

        private int salary;
//setters and getters

}
```

Demo.java:-

```
		Employee emp=new Employee();
		emp.setName("Ram");


		SalariedEmployee semp=new SalariedEmployee();
		semp.setName("Mohan");
		semp.setSalary(85000);

		ContractualEmployee cemp=new ContractualEmployee();
		cemp.setName("Hari");
		cemp.setCostPerDay(3000);
		cemp.setNoOfWorkingDays(10);

		em.getTransaction().begin();
		em.persist(emp);
		em.persist(semp);
		em.persist(cemp);
		em.getTransaction().commit();

		System.out.println("done");
```

--here one single table is created with all columns (for all the properties of super class Entity and all the proeperties of all the sub class Entities) plus one extra column DTYPE, which represents which class has made the entry.

--we can change this DTYPE column name and its corresponding value as follows:

```
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="emptype",discriminatorType=DiscriminatorType.STRING)
@DiscriminatorValue(value="emp")
public class Employee {
	int id;
	String name;

//setters and getters

}
```

ContractualEmployee.java:-

```
@Entity
@DiscriminatorValue(value="contEmp")
public class ContractualEmployee extends Employee{


	private int noOfWorkingDays;
	private int costPerDay;

//setters and getters

}
```

SalariedEmployee.java:-

```
@Entity
@DiscriminatorValue(value="salEmp")
public class SalariedEmployee extends Employee{

	private int sal;

//setters and getters

}
```

Note:-the limitation of the above strategy is :-

1.designing a table with huge number of column is not recomended,against the rule of normalizations.

2.with the above strategy,we can not apply not null to the coulmns

2.Table per sub-classes strategy/Joined Table:-
=====================================

--in this,every Entity class of inheritence hirarchy will have its own table and these table will participate in relationship,that means every record of child table will represent one record of parent table.

--in this mode of inheritence mapping,each child record of child table maintains association with a record of parent table .

--inside all the child tables we should have a FK column that reffers Pk column of parent table.

--while saving data by using child class obj,the common properies data will be saved to parent table and child class properties will be saved in child table.

Adv of table per subclasses strategy:-

1.DB tables can be designed by satisfying normalization forms/rules.

2.no need to take any discriminator value.

3. not null constraint can be applied.

@Inheritence(strategy=InheritenceType.JOINED) to mention table per child class

@PrimaryKeyJoinColumn(name="PKid") to modify the FK coulmn name in the child class

Example:

Employee.java:-

```
@Entity
@Inheritance(strategy=InheritanceType.JOINED)
public class Employee {


        @Id
        @GeneratedValue(strategy = GenerationType.AUTO)
        private int empId;


        private String name;
```

ContractualEmployee.java:-

```
@Entity
@PrimaryKeyJoinColumn(name="eid")
public class ContractualEmployee extends Employee {

        private int noOfWorkingDays;
        private int costPerDay;
}
```

SalariedEmployee.java

```
@Entity
@PrimaryKeyJoinColumn(name="eid")
public class SalariedEmployee extends Employee{

        private int salary;

}
```

3.Table per concreate class/ Table Per class:-
==================================

--in this strategy,every Entity class of inheritence hirarcy will have its own DB table these tables need not stay in relationship.

--in this strategy all the child class corresponding tables has all the column of its super class coresponding columns also.
for ex:-

class Employee(id,name)--->employee(id,name);

class SalaryEmployee extends Employee(salary)------->semployee(id,name,salary);

class ContractualEmployee extends Employee(noOfWorkingDays,costPerDay)---
------->cemployee(id,name,noOfWorkingDays,costPerDays);

--due to this,same column of parent table will be repeated inside all the child table.

Example:

Employee.java:-

```
@Entity
@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)
public class Employee {

	@Id
	@GeneratedValue
	private int empid;

	private String name;


}
```

ContractualEmployee.java:-

```
@Entity
public class ContractualEmployee extends Employee{

	private int noOfWorkingDays;
	private int costPerDay;

}
```

SalariedEmployee.java:-

```
@Entity
public class SalariedEmployee extends Employee{

	private int salary;

}
```

There is another category called MappedSuperclass:-

--Using the MappedSuperclass strategy, we can save only child class object, (here all the data of the child Entity and inherited data of the parent class will be persisted).

--in this strategy parent class will not be an Entity, it will be a normal java class.(can be an abstract class also)

Example

Employee.java:-

```
@MappedSuperclass
public abstract class Employee {


	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	private int empId;


	private String name;
}
```

Demo.java:-

```
		//This emp is not an Entity so it can't be persisted
		//Employee emp=new Employee();
		//emp.setName("Ram");


		SalariedEmployee semp=new SalariedEmployee();
		semp.setName("Mohan");
		semp.setSalary(85000);

		ContractualEmployee cemp=new ContractualEmployee();
		cemp.setName("Hari");
		cemp.setCostPerDay(3000);
		cemp.setNoOfWorkingDays(10);
```

```
                    em.getTransaction().begin();
                    //em.persist(emp);
                    em.persist(semp);
                    em.persist(cemp);
                    em.getTransaction().commit();

                    System.out.println("done");
```

Data Access Layer
JDBC :
JPA with Hibernate
Spring Data JPA : Hibernate

Designing Service Layer/ Business Logic layer using Spring:
==

J2EE architecture:
==============

J2SE :- it is a specification from Oracle, and the implementation of this specification is JDK s/w.

J2EE : Java Enterprise Edition (it is also an open specification and the implementation of this specification is "ApplicationServer" s/w)

JDBC : it is a specificaiton, the implementation is jdbc driver s/w.

JPA : it is also a api specification, the implementation is ORM s/w (Hibernate)

Business Application: 2 types:

1.small scale

2.large scale : Enterprise

Enterprise: large scale business organization, which provides/run their services in a large scale are/ entire world.

ex: Flipkart, Amazon, ICICI bank, LIC, travel company, insurance, IRCTC.

--Enterprises also required computer application to computerized their services.

--the application we develop to computerized the services of an Enterprise
is called as Enterprise Application.

--Persistence logic, Business logic, presentation logic is common for
the enterprise level computer application also.

Challanges to develop EA:
=====================

--An Enterprise application is bydefault a distributed application,
because EA is divided into seperate modules, and each module can be a independent
application, here each module will collaborate with each other and provide services
as whole.

1. data security : the data exchange should be encrypted and decripted, authentication
and authorization is also required.

2. client can accces the application in a platform and language independent form.
i.e by making our presentation logic web-enabled. so client can access our application
through the web-browser.

Presentation could be in GUI : that can be accesed by the browser (webapplication)
they are not reusable.

Presentation could be in the form raw data (JSON or xml format) (webservices)
they are the reusable

```
GoAir :-----> webapplication  ---------->GUI ---> book the ticket (it is used by end-user).
    |
    |
----------> it is having a webservices which generates a REST api and raw data (in the form
of JSON or XML) (it can be reused by some other application)
   |
Yatra : (webapplication)----->GUI---->book the ticket
```

3. transaction management : (either everything or nothig should happen)

4. logging

5.messaging

6.mailing

etc..

--to overcome these challanges 2 company comes with f/w software:

1. microsoft : .net f/w

2.sun-microsystem : J2EE f/w

--these both f/w comes in the form of specification.

.net :- it is a proprietery specication

J2EE : is a open specification

--The implementation of .net specification is "IIS server s/w".

--the implementation of J2EE specification is "ApplicationServer s/w"

Some of the application server s/w are:

Websphere server : from IBM

Weblogic server : from Oracle

Glassfish server : from Oracle

JBOSS server/ Wildfly server: RedHat

etc..

Presentation Layer