JPA with Hibernate 3.0

Association and Mapping

Lesson Objectives

After completing this lesson, participants will be able to understand:

- What is entity association?
- Different types of entity associations
- What are class inheritance mappings?
- Implementing associations and mapping using JPA

What is Entity Association?

Association represents relationship between entities.

A Java class can contain an object of another class or a set of objects of another class.

There is no directionality involved in relational world, its just a matter of writing a query. But there is notion of directionality which is possible in java.

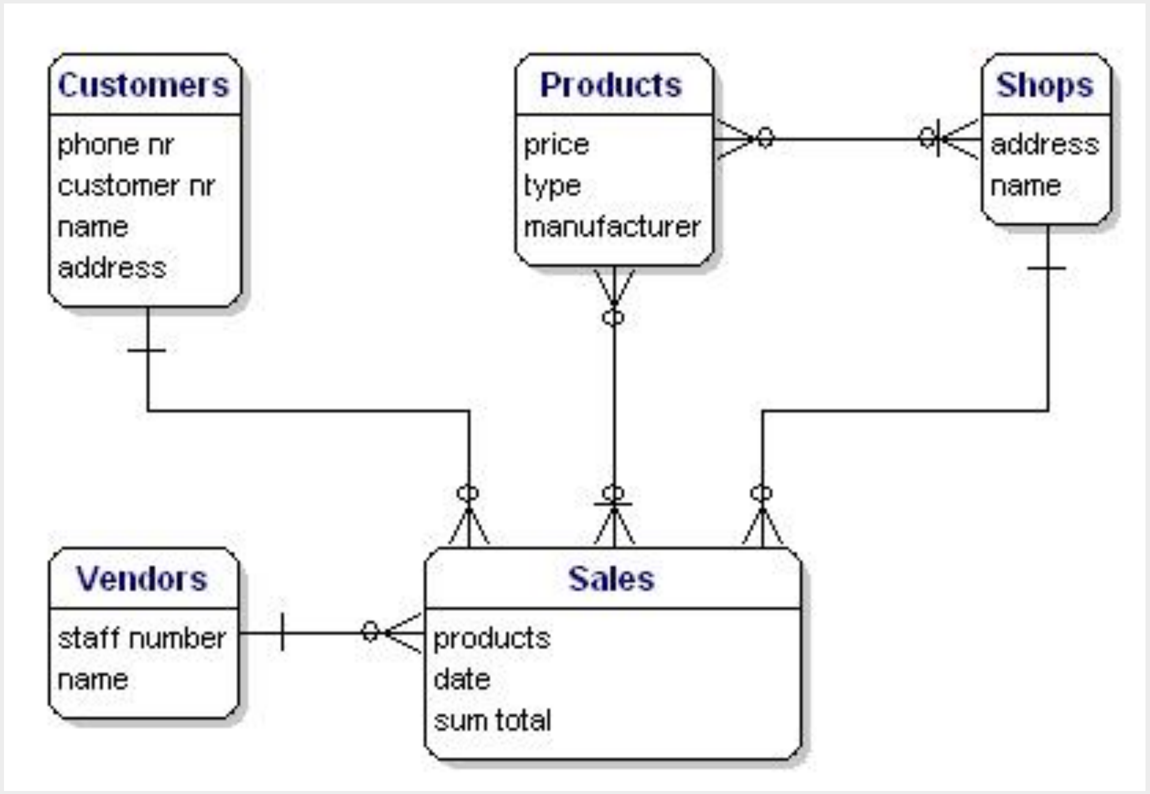
Hence associations are classified as

- unidirectional
- bidirectional.

Different types of associations

Unidirectional
- One to One
- One to Many
- Many to Many

Bidirectional
- One to One
- One to Many/Many to One
  - Without Join Table
  - With Join Table



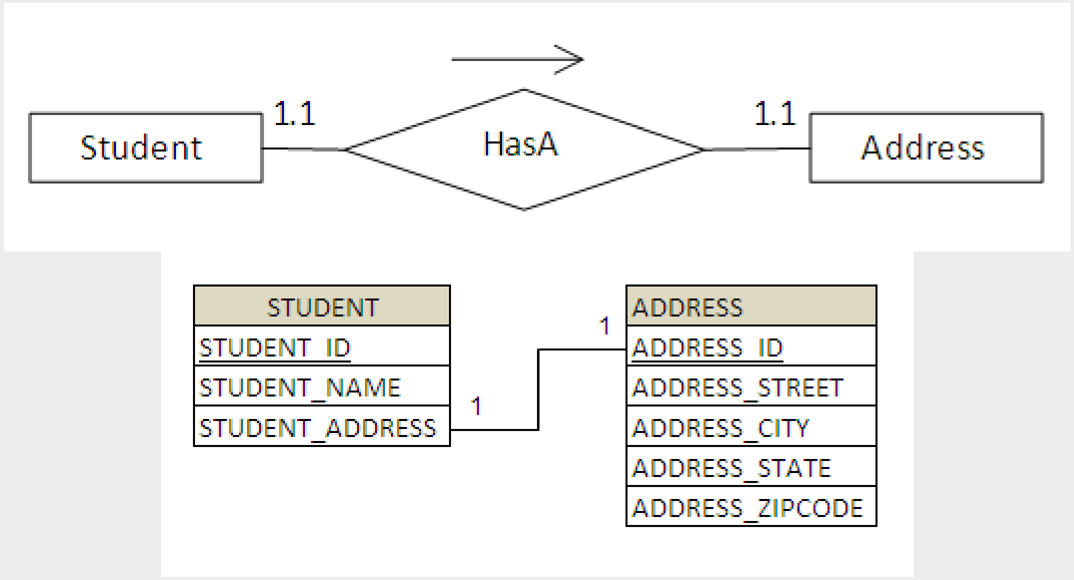
Associations

Unidirectional one to one

Consider the relationship between Student and his/her permanent address

According to the relationship each student should have a unique permanent address.

To create this relationship you need to have a STUDENT and ADDRESS table. The relational model is shown below.

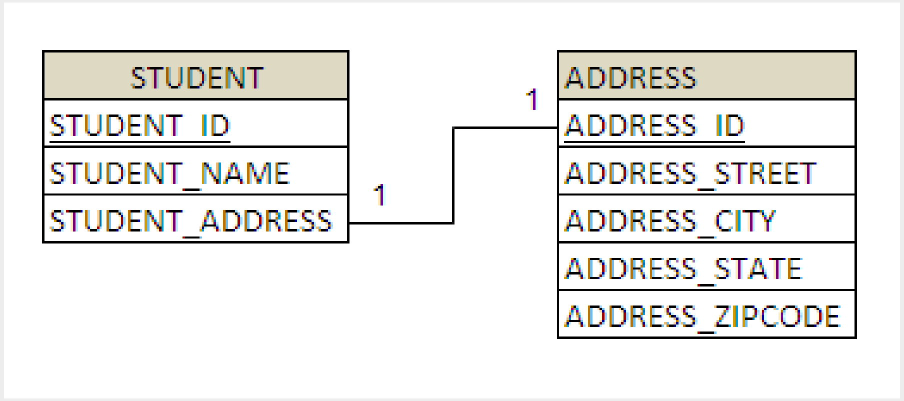


Associations

Unidirectional one to one

```
@Entity
public class Student ..... {
        @Id
        private int studentId;
        private String name;
        @OneToOne
        private Address
address;
```

```
@Entity
public class Address ..... {
        @Id
        private int addressId;
        private String street;
        private String city;
        private String state;
        private String zipcode;
```



Associations

Cascading associated Entities

Cascade attribute is mandatory, whenever we apply relationship between objects, cascade attribute transfers operations done on one object onto its related child objects.

This attribute indicates JPA operations on associated entity along with owner of association. It may take one of the value represented by CascadeType enumeration.

- PERSIST
- MERGE
- REMOVE
- ALL

Associations

Demo

JPAOneToOneUni

Bidirectional one to one

In this example, one employee can have one address and one address belongs to one employee only. Here, we are using bidirectional association.

In such case, a foreign key is created in the primary table.

Consider the following classes:

```
@Entity
public class Student ….. {
        @Id
        private int studentId;
        private String name;
        @OneToOne
        private Address
address;
```

```
@Entity
public class Address ….. {
        @Id
        private int addressId;
        private String street;
        private String zipcode;
        @OneToOne(mappedBy="addr
ess")
        private Student student;
```

Associations

Demo

JPAOneToOneBI

Associations

Bidirectional one to many

In a one to many/many to one association, a one class contains a collection of other class object and the second class has an object of the first.

Consider following classes:

```
@Entity
public class Employee ….. {
      @Id
      private int id;
      private String name;
      @ManyToOne
      @JoinColumn(name="dept_
no")
      private Department
department ;
```

```
@Entity
public class Department ….. {
      @Id
      private int id;
      private String name;
      @OneToMany(mappedBy="departme
nt")
      private Set<Employee> employees;
```

Demo

JPAOneToManyBI

Associations

Bidirectional Many to many using Join Table

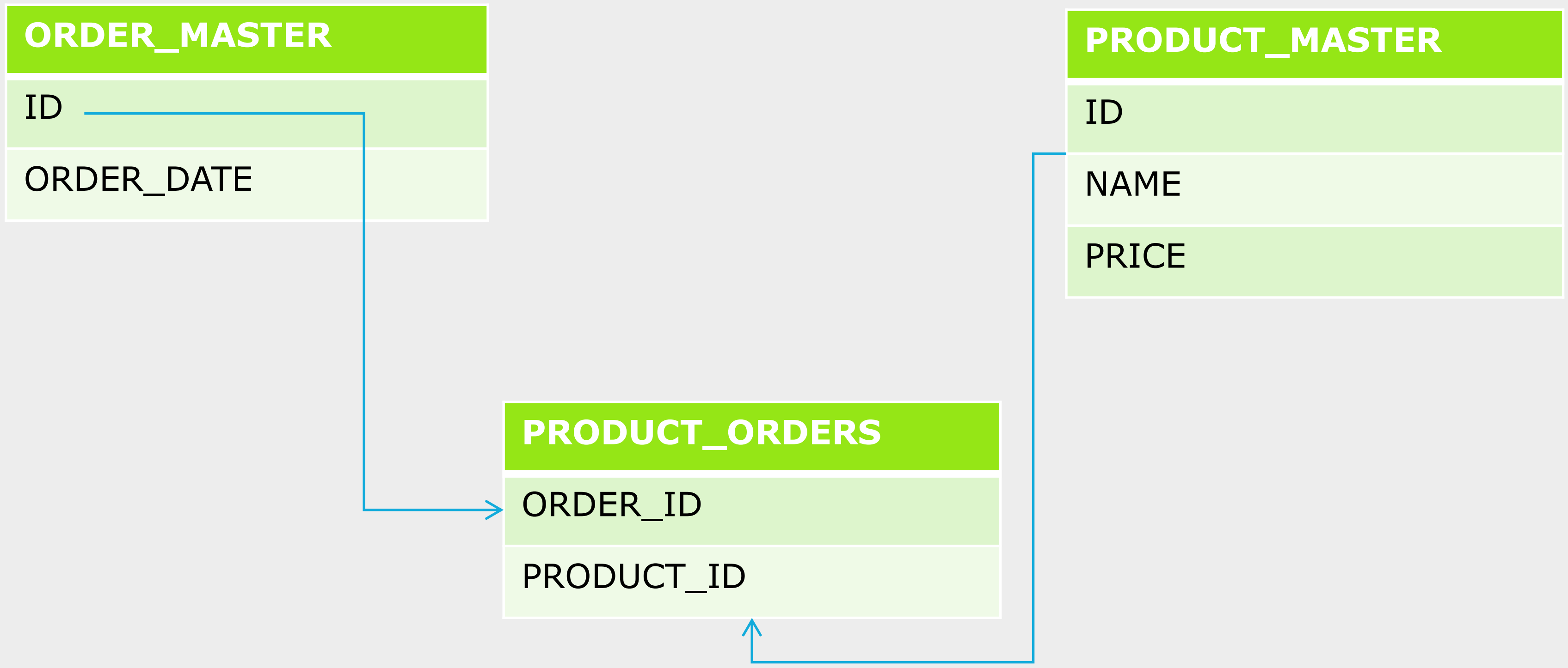
In the below example, an order can have any number of products and also product can be part of multiple orders.

```
@Entity
public class Order ….. {
        @Id
        private int id;
        private Date purchaseDate;
        @ManyToMany
        private Set<Product> products ;
```

```
@Entity
public class Product ….. {
        @Id
        private int id;
        private String name;
        @ManyToMany(mappedBy="products")
        private Set<Order> orders;
```

Bidirectional Many to many using Join Table

In database data of products and orders can be stored using Join table.



Associations

Demo

JPAManyToManyBI

Mapping Inheritance

Java classes are may related to each other in form of inheritance

However, there is no inheritance between database tables

JPA allows hierarchical classes to be mapped with tables with below listed strategies:

- Single Table per class hierarchy
- Table per class
- Joined subclass

Inheritance

Single Table per Class Hierarchy

In this strategy, a single table is created for all classes in the inheritance hierarchy.

It uses additional column called discriminator to distinguish the object of child classes

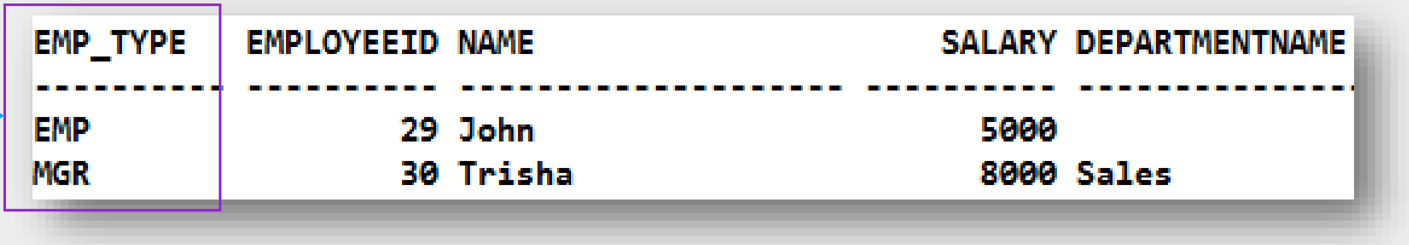
The value in discriminator column is used to identify rows belonging to subclasses

Single Table per Class Hierarchy

```
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
public class Employee ….. {
        @Id
        private int id;
        private String name;
        private double salary;
```

```
@Entity
public class Manager extends Employee {
        private String departmentName;
```

Discriminator Column



```
EMP_TYPE   EMPLOYEEID NAME                     SALARY DEPARTMENTNAME
---------- ---------- -------------------- ---------- ---------------
EMP                29 John                       5000
MGR                30 Trisha                     8000 Sales
```

Inheritance

Demo

JPASTInheritance

Inheritance

Table per Concrete Class

In this strategy, a table is created for each class in the hierarchy

Each table will have columns for all properties in parent and child class

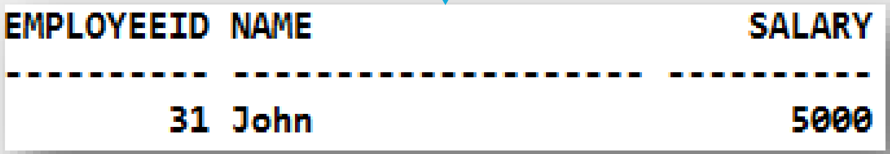
Support for this strategy is optional, and may not be supported by all Java Persistence API providers.

Inheritance

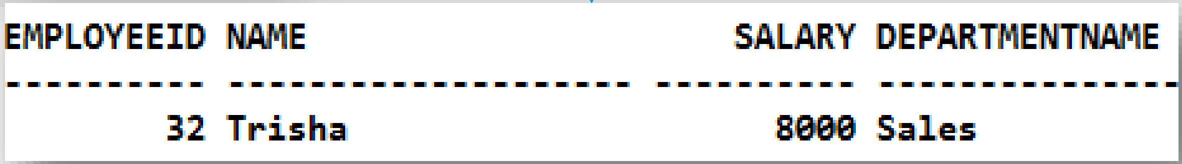
Table per Concrete Class

```
@Entity
@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)
public class Employee ..... {
	@Id
	private int id;
	private String name;
	private double salary;
```

```
@Entity
public class Manager extends Employee {
	private String departmentName;
```



```
EMPLOYEEID NAME                      SALARY
---------- -------------------- ----------
        31 John                       5000
```



```
EMPLOYEEID NAME                      SALARY DEPARTMENTNAME
---------- -------------------- ---------- ---------------
        32 Trisha                     8000 Sales
```

Inheritance

Demo

JPATPCInheritance

Inheritance

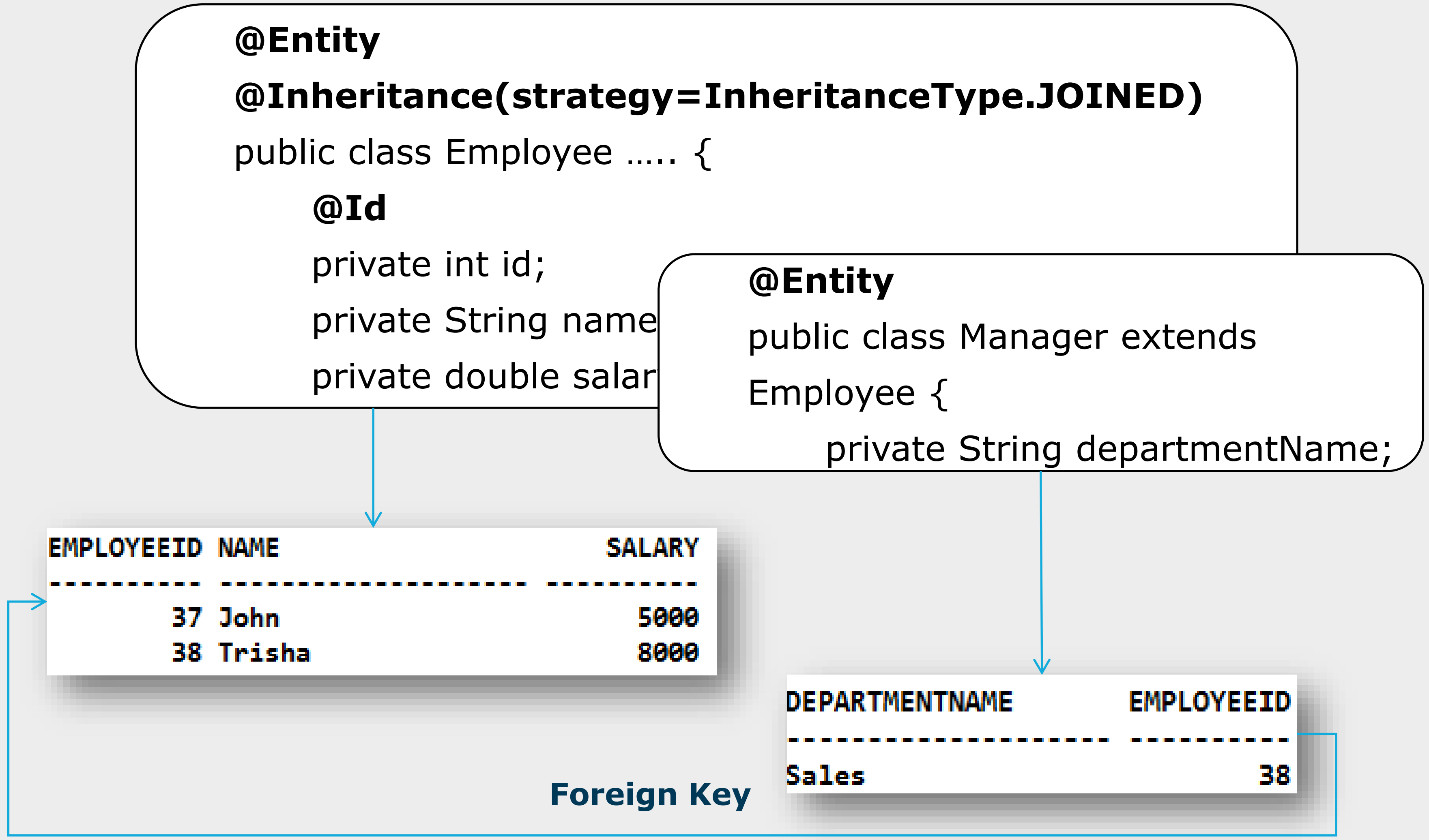
Joined Subclass Strategy

In this strategy, a separate table is created for each class in the inheritance hierarchy.

However, columns inherited from the parent class are not repeated in subclass.

The parent table primary key is used as foreign key for child tables.

Joined Subclass Strategy



```
@Entity
@Inheritance(strategy=InheritanceType.JOINED)
public class Employee ….. {
      @Id
      private int id;
      private String name
      private double salar
```

```
@Entity
public class Manager extends
Employee {
      private String departmentName;
```

```
EMPLOYEEID NAME                     SALARY
---------- -------------------- ----------
        37 John                       5000
        38 Trisha                     8000
```

```
DEPARTMENTNAME       EMPLOYEEID
-------------------- ----------
Sales                        38
```

Inheritance

Demo

JPAJSInheritance

Lab

Associations and Mapping

Summary

In this lesson, you have learnt:

- Entity associations and inheritance mapping
- Implementing associations and inheritance using JPA

Review Question

Question 1: Which one of the following inheritance mapping type is suitable if you want to create single table for all classes in hierarchy?

- InheritanceType.TABLE_PER_CLASS
- InheritanceType.SINGLE_TABLE
- InheritanceType.JOINED

Question 2: In many-to-many bidirectional relationships, either side may be the owning side.

- True/False