

# Object Relational Mapping Introduction

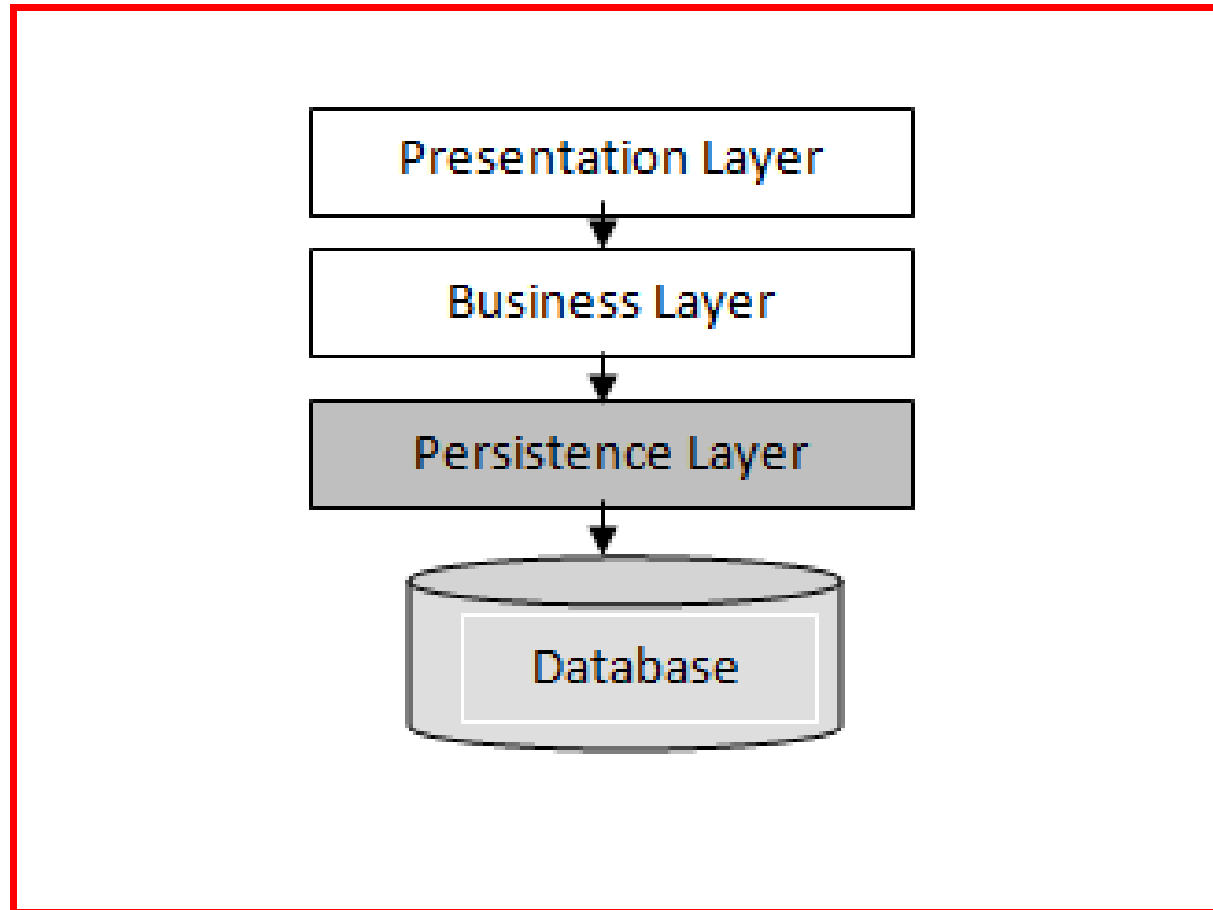
Arvo Lipitsäinen

HAAGA-HELIA University of Applied Sciences

DBTech EXT project

University of Malaga , 6.4.2010

# Layered software architecture



Source: Bauer, C. & King, G. 2007. Java Persistence with Hibernate.  
Manning Publication Co.

# Object oriented paradigm

- OO paradigm in the development of information system:
  - Object oriented analysis
  - Object oriented programming
    - OOP languages: C++, Java, ...
  - Object oriented design

# Object oriented persistence

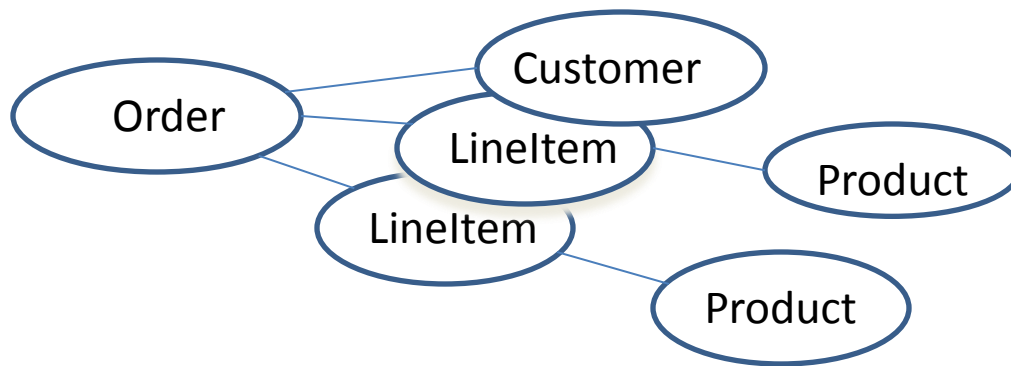
- State of object can be stored to disk and the same object can be later re-created
- The class must implement Serializable

```
public class Customer implements Serializable
{
    ...
}
Customer customer = new Customer( ... );

objectOutputStream.write(customer);
...
objectInputStream.read(customer);
```

# Object oriented persistence

- Many objects interconnected with each other can be stored disk and later re-created



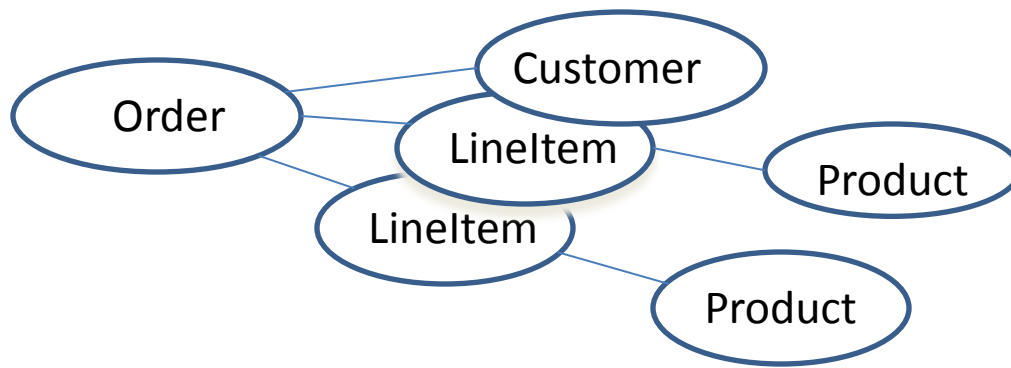
```
objectOutputStream.write(order) ;
```

```
...
```

```
objectInputStream.read(order) ;
```

# Object oriented persistence

- Navigating from object to another is easy



```
objectOutputStream.write(order) ;
```

```
...
```

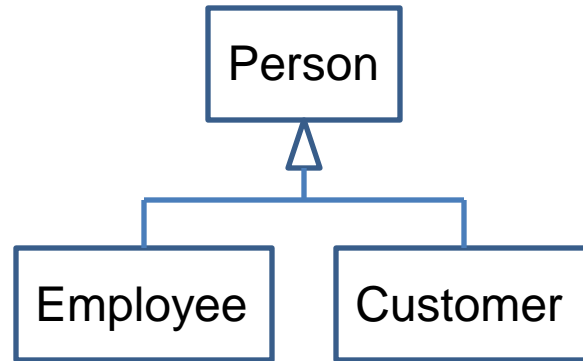
```
objectInputStream.read(order) ;
```

```
String productName =
```

```
    order.getLineItem().getProduct().getName() ;
```

# Polymorphics

- OO paradigm supports inheritance and polymorphics through inheritance:



```
Person personx = Employee e;  
Person persony = Customer c;  
personx.getName();  
persony.gerName();
```

# Relational databases

- Predominant technology for management of organization-wide persistent data
- Relational model: all data is represented as mathematical n-ary relations
- Programming interface of the RDBM systems is SQL (Sequential Query Language):
  - Relational database table
  - Row, column
  - CRUD: create, read, update, delete

# Object model / Relational model

- The persistent data is stored in the relational databases
- Applications are developed and run in the object oriented environment (Java)
- Object model and relational model are different
- JDBC (Java Database Connectivity) is SQL based interface to the RDBM systems

# ORM mismatches

- **Problem of granularity**
  - **Problem of subtypes**
  - **Problem of identity**
  - **Problems relating to associations**
  - **Problem of data navigation**
- 
- => Object Relational mapping is needed

# Solutions of ORM mismatch problems

- Trials of Object Databases, but ...
- EJB 2 tried solution, but failed
- ORM tools: Hibernate, Toplink
- JPA (Java Persistence API) – Specification
  - JPA 2.0, released in November 2009
- EJB 3 – Specification
- JBoss application server (open source)
  - EJB 3
  - based on Hibernate

# JPA – Java Persistence API

- Persistent domain objects - entities - are POJOs (Pure Old Java Objects) with `@Entity` annotation
- Typically an entity class represent a table in a relational database
- Entity instance corresponds to a row in that table.
- Every entity must have a primary key
- The `@Id` annotation is used to denote a simple primary key

# Entity - annotations

```
@Entity
@Table(name = "CUSTOMER_DBTECH")
public class Customer
{
    @Id
    private int id;
    private String name;
    . . .
    public int getId(){
        return id;
    }
    public void setId(int id){
        this.id = id;
    }
    public String getName(){
        return name
    }
    . . .
}
```

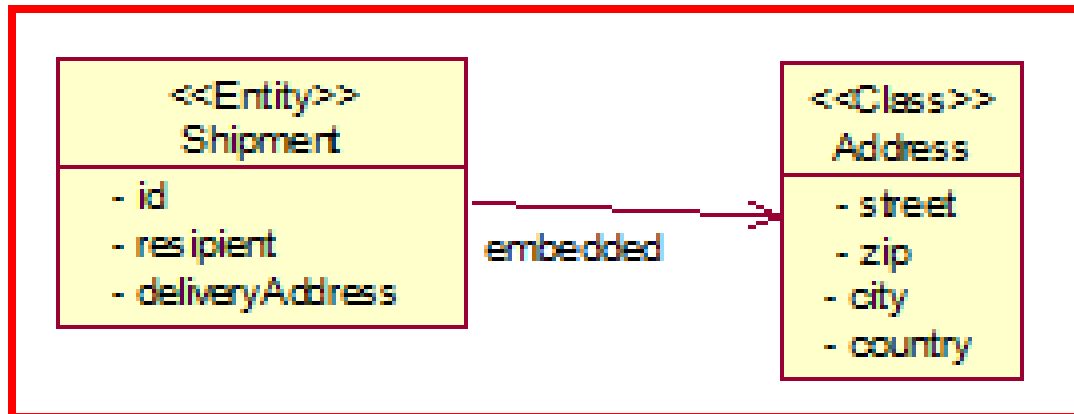
# Entity - annotations

```
@Entity
@Table(name = "CUSTOMER_DBTECH")
public class Customer
{
    private int id;
    private String name;
    . . .
    @Id
    @Column(name = "ID_DBTECH")
    public int getId(){
        return id;
    }

    @Column(name = "NAME_DBTECH")
    public String getName(){
        return name
    }
    . . .
}
```

# Embedded classes

- An entity can use other fine-grained classes, embedded classes, to present entity state
- Embedded objects belong strictly to their owning entity



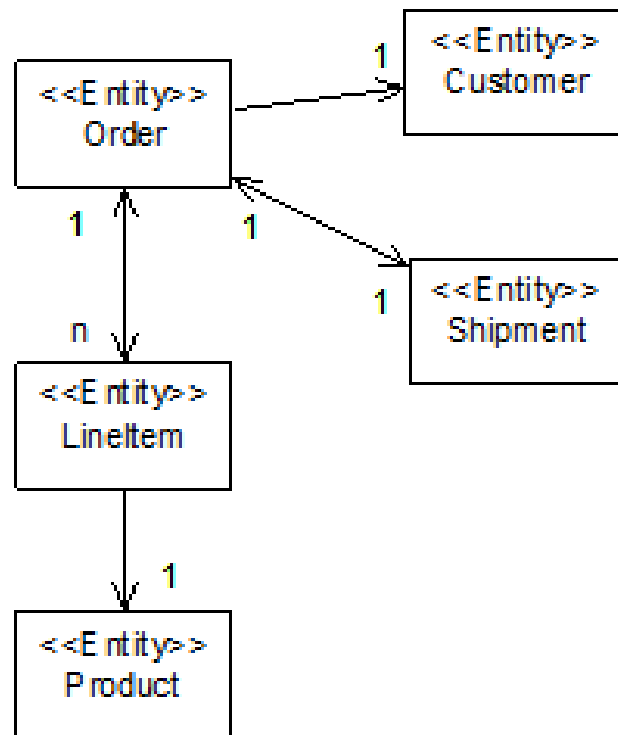
# Embedded classes

```
@Entity
public class Shipment
{
    private int id;
    private String date;
    @Embedded
    private Address deliveryaddress;
    .
    .
}
```

```
@Embeddable
public class Address
{
    private String street;
    private String city;
    private String zip;
    private String country;
    . . .
}
```

# Entity - relationships

- Entities can have relationships to other entities



# Entity - relationships

- OneToOne / OneToMany / ManyToOne / ManyToMany
- Uni-directional / bi-directional relationship
- Relationship must have an owning side
  - Unidirectional relationship has only an owning side
  - Bidirectional relationship has both an owning side and an inverse (non-owning) side

# Entity – relationship fields

```
@Entity
public class Order {
    ...
    private Customer customer;
    private Shipment shipment;
    private Collection<LineItem> lineItems;
    ...
    @OneToOne
    public Customer getCustomer() {
        return customer;
    }
    @OneToOne
    public Shipment getShipment() {
        return shipment;
    }
    @OneToMany(mappedBy="order")
    public Collection<LineItem> getLineItems() {
        return lineItems;
    }
    ...
}
```

# Entity – relationship fields

```
@Entity
public class Customer {
    @Id
    private int id;
    private String name;
    private String address;
    // no relationship field
}

@Entity
public class Shipment {
    @Id
    private int id;
    private Order order;
    ...
    @OneToOne(mappedBy="shipment")
    public Order getOrder() {
        return order;
    }
    ... }
```

# Entity – relationship fields

```
@Entity
public class LineItem {
    private int id;
    ...
    private Order order;
    private Product product;
    ..
    @ManyToOne
    public Order getOrder() {
        return order;
    }
    @OneToOne
    public Product getProduct() {
        return product;
    }
    ...
}
```

# Mapping of relationship field

- Relationship field is mapped to the owning side's table as a foreign key



**ORDER table**

ID	AMOUNT	ORDERDATE	CUSTOMER_ID
1	0	16.2.2010 10:45:3...	1001
2	0	16.2.2010 10:45:3...	1002
3	0	16.2.2010 10:45:3...	1001

**CUSTOMER table**

ID	ADDRESS	NAME
1001	Ramblas 5, Barcelona	Company X
1002	Corso 6, Roma	Company Y
1003	Broadway 118, New York	Company Z

# Mapping of relationship field

**ORDER table**

ID	AMOUNT	ORDERDATE	CUSTOMER_ID	SHIPMENT_ID
1	0	16.2.2010 16:1...	1001	1
2	0	16.2.2010 16:1...	1002	
3	0	16.2.2010 16:1...	1001	2

**SHIPMENT table**

ID	DATE	DESTINATION
1	16.2.2010 16:18...	Ramblas 5, Barcelona
2	16.2.2010 16:18...	Ramblas 5, Barcelona

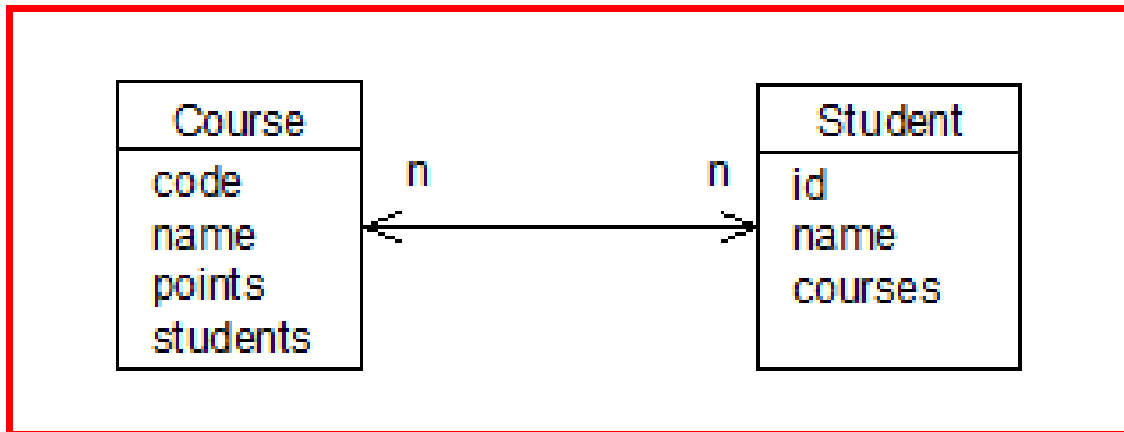
**ORDER table**

ID	AMOUNT	ORDERDATE	CUSTOMER_ID
1	151	7.4.2010 17:52:17 0...	1001
2	115,8	7.4.2010 17:52:17 1...	1002

**LINEITEM table**

ID	AMOUNT	PRICE	PRODUCT	ORDER_ID
1	1	151	Hugo Boss trousers	1
2	3	7,85	Estola Reserva 2004	2
3	1	92,25	Vega Sicilia 2003	2

# Mapping of many-to-many relationship



# Mapping of many-to-many relationship

```
@Entity
public class Course {

    @Id
    private String code;
    ...
    private Collection<Student> students;

    ..
    @ManyToMany(mappedBy="courses")
    public Collection<Student> getStudents() {
        return students;
    }

    ...

}
```

# Mapping of many-to-many relationship

```
@Entity
public class Student {
    @Id
    private String id;
    ...
    private Collection<Course> courses;

    ...
    @ManyToMany
    public Collection<Course> getCourses() {
        return courses;
    }
    ...
}
```

# Mapping of many-to-many relationship

- **Course** and **Student** are mapped ..
- The relationship is mapped to a join table **STUDENT\_COURSE** (owner name first)



COURSE table

CODE	NAME	POINTS
D019	Data Mining	6
B251	Business Management	5
P103	Foreign Affairs of EU	2

STUDENT table

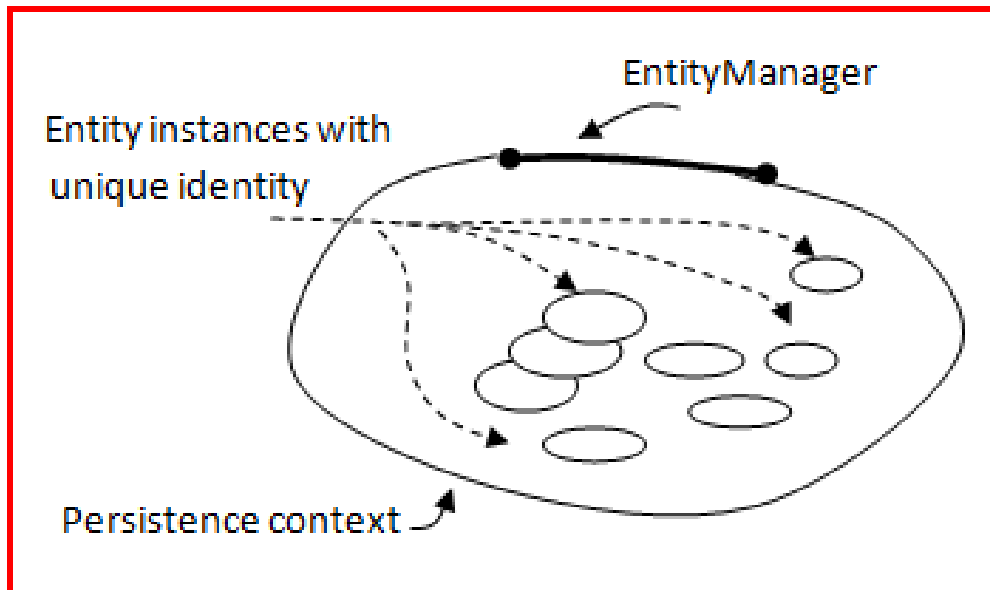
ID	NAME
08100	Jose Manuel Barroso
08101	Eva Paradise
08102	Angela Merkel
08103	Catherine Ashton
09100	Herman Van Rompuy
09101	Tim Berner
09102	Silvio Berlusconi

STUDENT\_COURSE table

STUDENTS_ID	COURSES_CODE
08103	P103
09102	P103
08100	P103
08102	D019
09102	D019
09100	D019
09101	B251

# Persistence context

- A persistence context is a set of entity instances with unique entity identity.
- An **EntityManager** instance is associated with a persistence context.



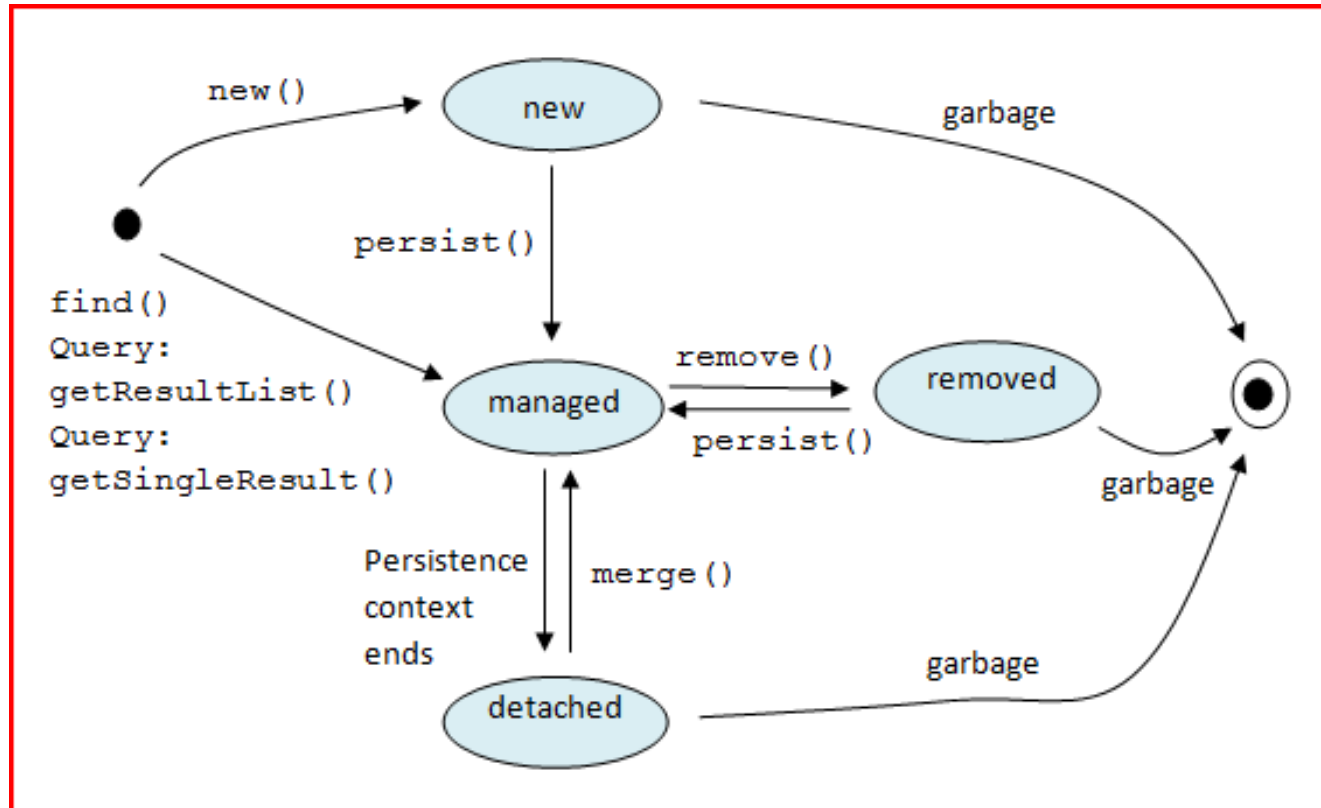
# Entity Manager

- **EntityManager** manages the entity instances and their life cycle within the persistent context
- The scope of the persistence context can either be the transaction (default), or can extend beyond lifetime of a single transaction.

# Entity Manager

```
public interface EntityManager
{
    void close();
    Query createNamedQuery(String name);
    Query createNativeQuery(String sqlString);
    Query createQuery(String qlString);
    <T> T find(Class<T> entityClass, Object primaryKey);
    void flush();
    <T> T merge(T entity);
    void persist(Object entity);
    void refresh(Object entity);
    void remove(Object entity);
}
```

# States of an entity instance



# Java Persistence query language

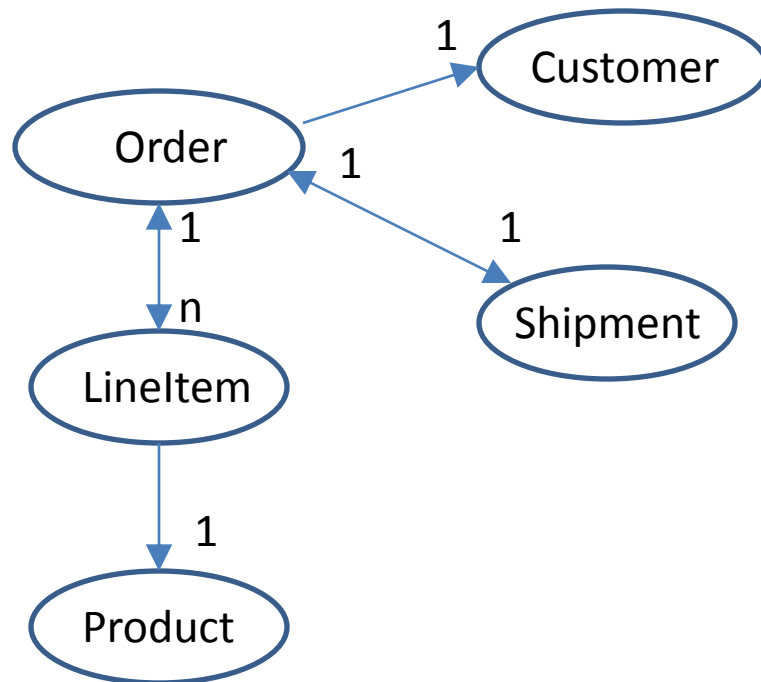
- for defining queries over entities and their persistent state
- truly platform independent:
  - entities and their fields are referenced by their names
  - no reference to the table or column names of the underlying relational database

# Query language syntax

```
SELECT clause1  
FROM clause2  
[WHERE clause3]  
[GROUP BY clause4]  
[HAVING clause5]  
[ORDER BY clause6]
```

# Abstract persistence schema

- The query language uses the abstract persistence schema of entities and relationships for its data model



# SELECT statements

```
SELECT p  
FROM Product p
```

The result of the query is all products

```
SELECT p  
FROM Product p  
WHERE p.productGroup = 'Wines'
```

The result of this query is a list of products, whose product group is Wines

# SELECT statements

```
SELECT p  
FROM Product p  
WHERE p.price >= 0.0 AND p.price < 500.00
```

The result of the query is a list of products, whose price is equal or more than 0.0 and less than 500.00.

# SELECT statement – input parameters

- The query can have input parameters
- A **named parameter** is denoted by an identifier that is prefixed by the ":" symbol

```
SELECT p  
FROM Product p  
WHERE p.productGroup = :pgroup
```




- The value of the input parameter is given by calling **setParameter()** method of **Query** object

# SELECT statement – input parameters

- Another type of input parameter, a **positional parameter**, is designated by the question mark (?) prefix followed by an integer

```
SELECT p
FROM Product p
WHERE p.price >= ?1 AND p.price < ?2
```



# Navigating in the query

- Two entities can be joined with **JOIN** keyword for navigating from an entity to another via a one-to-one relationship field.

```
SELECT o  
FROM Order o JOIN o.customer c  
WHERE c.name = 'HAAGA-HELIA'
```

# Navigating in the query

- In one-to-many relationship, the multiple side consists of a collection of entities.
- A collection member declaration is declared by using **IN** operator.

```
SELECT o  
FROM Order o, IN (o.lineItems) l JOIN l.product p  
WHERE p.code = 'W0995'
```

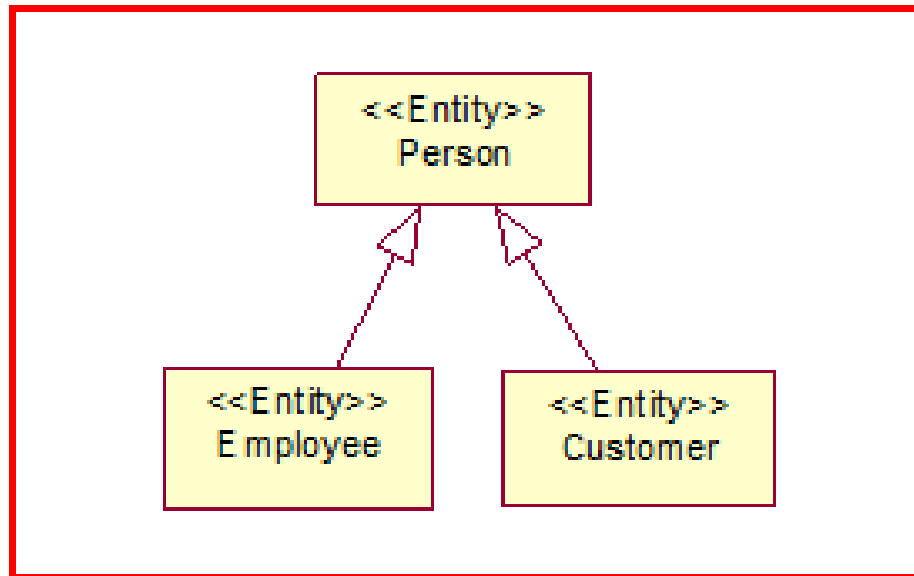
# Navigating in the query

- The **SELECT** query can return also persistent fields of the entity.

```
SELECT p.name  
FROM Order o, IN (o.lineItems) l JOIN  
    l.product p JOIN o.customer c  
WHERE c.name = 'Pedro Cavador'
```

# Mapping of class hierarchies

- How to map a class hierarchy to the relational database?



# Mapping of class hierarchies

- There are three basic strategies for mapping a class or class hierarchy to a relational database:
  - a **single table** per class hierarchy (default strategy)
  - a **joined** subclass strategy, in
    - fields that are specific to a subclass are mapped to a separate table than the fields that are common to the parent class
  - a **table per** concrete entity **class**

# Single table per class hierarchy

- All the classes in a hierarchy are mapped to a single table.
- “discriminator column”, DTYPE, identifies the specific subclass

```
SELECT * FROM PERSON
```

DTYPE	CODE	ADDRESS	DEGREE	GENDER	NAME	DEPARTMENT	SALARY	TITLE	COMMENT	PREFERENCE	TYPE
Employee	1234	Bruessels	Doctor	M	Herman Van Rompuy	Administration	24 322,59	President of EU	[null]	[null]	[null]
Person	2201	Switzerland	???	M	Wilhelm Tell	[null]		[null]	[null]	[null]	[null]
Customer	9834	Munich	Doctor	M	Hasso Plattner	[null]		[null]	Very good person	Beer	Super

# Mapping of class hierarchies

- See the laboratory worksop afternoon

# ORM – Object Relational Mapping

THANK YOU!