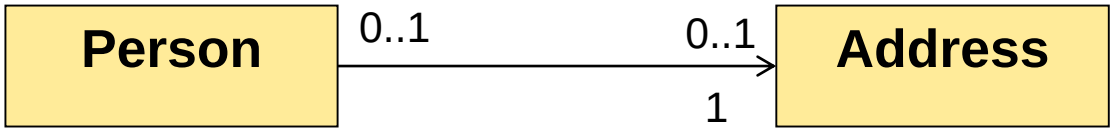
Java Persistence API

- **Entity Manager**
- **Relations**
- **Samples (with focus on owner / inverse)**
  - OneToOne unidirectional
  - OneToOne bidirectional
  - OneToMany bidirectional
  - ManyToMany bidirectional

OneToOne unidirectional



- One may specify in the database scheme that the foreign key must not be null
  - @OneToOne(optional = true) 0..1 [Default]
  - @OneToOne(optional = false) 1

```
CREATE MEMORY TABLE ADDRESS(
   ID INTEGER NOT NULL PRIMARY KEY,
   CITY VARCHAR(255), STREET VARCHAR(255))
CREATE MEMORY TABLE PERSON(
   ID INTEGER NOT NULL PRIMARY KEY,
   ADDRESS_ID INTEGER NOT NULL,
   CONSTRAINT FK203A7330FF0EDE FOREIGN KEY(ADDRESS_ID)
                               REFERENCES ADDRESS(ID))
```

OneToOne unidirectional

```
@Entity
public class Person {
   @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
   private int id;
   private String name;
   @OneToOne // (optional=false)
   private Address address;

   protected Person() { }
   public Person(String name) { this.name = name; }

   public int getId() { return id; }

   public Address getAddress() { return address; }
   public void setAddress(Address address) {
      this.address = address; }
   ...
}
```

OneToOne unidirectional

```
@Entity
public class Address {
   @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
   private int id;
   private String street, city;
   private int zip;

   protected Address() { }
   public Address(String street, int zip, String city) {
      this.street = street; this.city = city; this.zip = zip;
   }

   public int getId() { return id; }

   ...
}
```

OneToOne unidirectional

```
em.getTransaction().begin();
Person p = new Person("Dominik");
Address a = new Address("Bahnhofstrasse 6", 5210, "Windisch");

p.setAddress(a);

em.persist(p);
em.persist(a);   // (1)
em.getTransaction().commit();
```

- **(1)**
  - Order of persist operations is not relevant, data is written upon commit
  - Only necessary if PERSIST-cascading is not defined
    - => see next slide

OneToOne unidirectional

```
@Entity
public class Person {
   @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
   private int id;
   private String name;
   @OneToOne(cascade = { CascadeType.PERSIST, CascadeType.REMOVE })
   private Address address;

   private Person() { }
   public Person(String name) { this.name = name; }

   public int getId() { return id; }

   public Address getAddress() { return address; }
   public void setAddress(Address address) { this.address=address; }
   ...
}
```

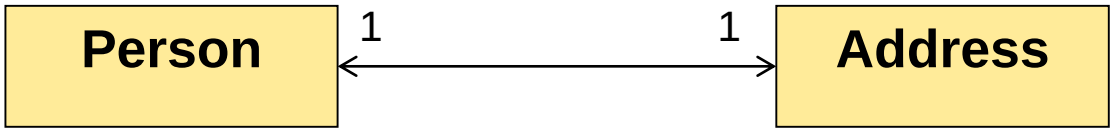
Address no longer has to be persisted separately

Implies that the address is removed if the person is removed.

Java Persistence API

- **Entity Manager**
- **Relations**
- **Samples (with focus on owner / inverse)**
  - OneToOne unidirectional
  - OneToOne bidirectional
  - OneToMany bidirectional
  - ManyToMany bidirectional

OneToOne bidirectional



- **Owner**
  - For bidirectional associations one side is the owner of the association
    - The state of the owner defines what is stored in the database
    - The state of the inverse side is ignored upon saving the entity
  - The non-owner side is named inverse side
    - In JPA the inverse side is marked with a "`mappedBy`" annotation.
    - Optional attribute (`optional = true`) on inverse side is ignored
- **Example Person <--> Address**
  - Let us define the Person entity to be the owner of the Person-Address association

OneToOne bidirectional

```
@Entity
public class Person {
   @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
   private int id;
   private String name;
   @OneToOne(cascade = {CascadeType.PERSIST, CascadeType.REMOVE}
             optional = false)
   private Address address;

   protected Person() { }
   public Person(String name) { this.name = name; }

   public int getId() { return id; }

   public Address getAddress() { return address; }
   public void setAddress(Address address) { this.address=address; }
   ...
}
```

OneToOne bidirectional

```
@Entity
public class Address {
   @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
   private int id;
   private String street, city;
   private int zip;
   @OneToOne(mappedBy="address")
   private Person person;

   protected Address() { }
   public Address(String street, int zip, String city) {
      this.street = street; this.city = city; this.zip = zip;
   }

   public Person getPerson() { return person; }
   public void setPerson(Person person) { this.person = person; }
}
```

OneToOne bidirectional

- **Person**

```
CREATE MEMORY TABLE PERSON(
   ID INTEGER NOT NULL PRIMARY KEY,
   ADDRESS_ID INTEGER NOT NULL,
   CONSTRAINT FK203A7330FF0EDE FOREIGN KEY(ADDRESS_ID)
                                          REFERENCES ADDRESS(ID))
```

  - `NOT NULL` only if `optional=false`

- **Address**

```
CREATE MEMORY TABLE ADDRESS(
   ID INTEGER NOT NULL PRIMARY KEY,
   CITY VARCHAR(255), STREET VARCHAR(255))
```

=> exactly the same schema as for the unidirectional case!

OneToOne bidirectional

```
em.getTransaction().begin();
Person p = new Person("Dominik");
Address a = new Address("Bahnhofstrasse 6", 5210, "Windisch");
p.setAddress(a);
em.persist(p);

// p = em.find(Person.class, 1);
System.out.println(p.getAddress().getStreet());
System.out.println(p.getAddress().getPerson().getName());

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

- Person -> Address: cascade contains PERSIST
- What is printed on the console?

OneToOne bidirectional

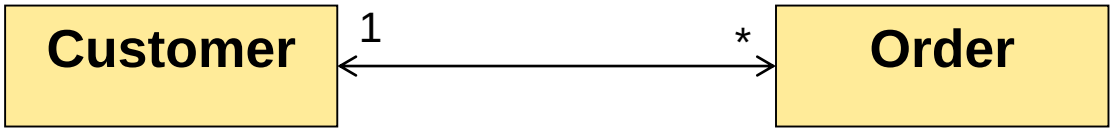
```
em.getTransaction().begin();
Person p = new Person("Dominik");
Address a1 = new Address("Bahnhofstrasse 6", 5210, "Windisch");
Address a2 = new Address("Steinackerstrasse 5", 5210, "Windisch");
p.setAddress(a1);
a2.setPerson(p);
em.persist(p);
em.persist(a2);
em.getTransaction().commit();
```

- What is saved in the data base?

Java Persistence API

- **Entity Manager**
- **Relations**
- **Samples (with focus on owner / inverse)**
  - OneToOne unidirectional
  - OneToOne bidirectional
  - OneToMany bidirectional
  - ManyToMany bidirectional

OneToMany bidirectional



- For OneToMany bidirectional associations, the many side has to be the owner of the association (=> Order)
  - This is not always the natural choice!
  - This may change for further versions of JPA (but not in JPA 2.2/3.0)

- Spec Section 2.9

> • The many side of one-to-many / many-to-one bidirectional relationships must be the owning side, hence the *mappedBy* element cannot be specified on the *ManyToOne* annotation.

OneToMany bidirectional

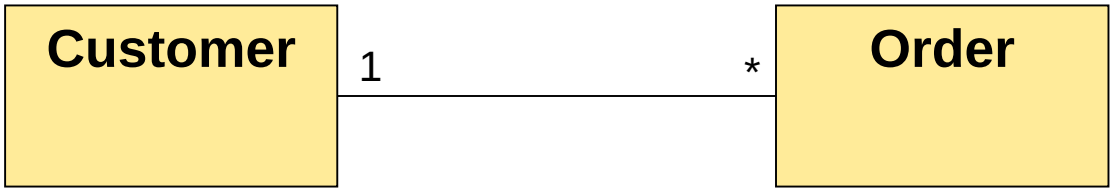
```
@Entity
@Table(name="CUSTOMERS")
public class Customer {
   @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
   private int id;

   // this is the inverse side of the relationship
   @OneToMany(mappedBy="customer", cascade=CascadeType.ALL)
   private Collection<Order> orders;

   public Collection<Order> getOrders() {
      return orders;
   }
   public void setOrders(Collection<Order> orders) {
      this.orders = orders;
   }
}
```

OneToMany bidirectional

```
@Entity
@Table(name="ORDERS")
public class Order {
   @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
   private int id;
   @ManyToOne                              // Order is the owner
   private Customer customer;              // of the relationship

   public Order() { }

   public Customer getCustomer() { return customer; }
   public void setCustomer(Customer customer) {
      this.customer = customer;
   }

   public int getId() { return id; }
}
```

OneToMany bidirectional

- **Representation in database**




```
create table customer (
        ID              integer,
        FIRSTNAME       varchar(32),
        LASTNAME        varchar(32),
);
```

```
create table address (
        ID              integer,
        CUSTOMER_ID     integer,
        STATUS          varchar(4),
        PRICE           decimal(10,2)
);
```

OneToMany bidirectional

- **Bidirectional 1:n relation**
  - n-side is always the owner of a bidirectional 1:n relation
  - Only references from order to customer are persisted!

```
em.getTransaction().begin();
Customer c = new Customer();
Order o1 = new Order();
Order o2 = new Order();

c.setOrders(List.of(o1, o2));

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

=> the two orders are stored in the DB (due to the cascade=PERSIST)

OneToMany bidirectional

- **Bidirectional 1:n relation**
  - => the associations are NOT persisted!!!

SELECT * FROM CUSTOMERS;

ID
1

(1 row, 2 ms)

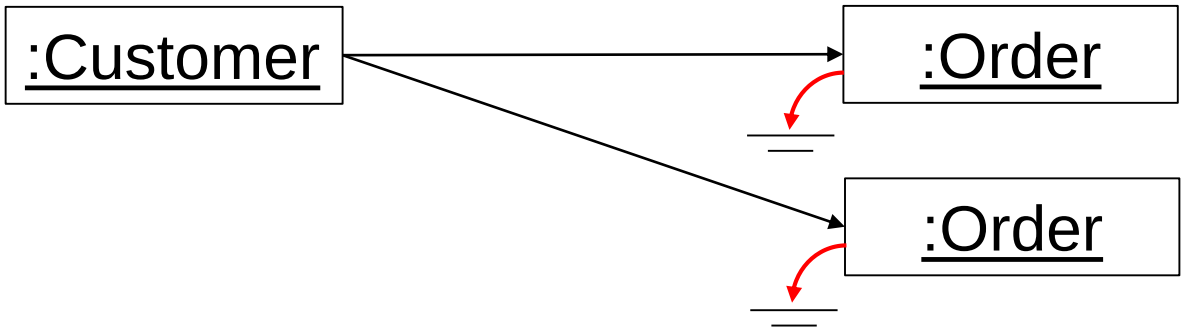
Edit

SELECT * FROM ORDERS;

ID	CUSTOMER_ID
2	*null*
3	*null*

(2 rows, 11 ms)

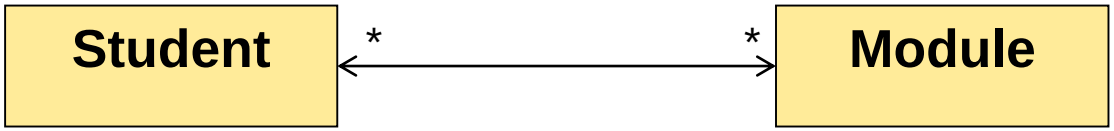
Edit



Java Persistence API

- **Entity Manager**
- **Relations**
- **Samples (with focus on owner / inverse)**
  - OneToOne unidirectional
  - OneToOne bidirectional
  - OneToMany bidirectional
  - ManyToMany bidirectional

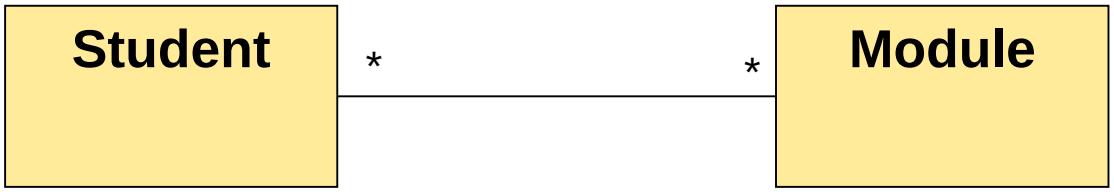
ManyToMany bidirectional



- Either side may be the owner of the association
- Realized using a mapping table
  - Additional columns in mapping table cannot be added
    - No UML association classes
  - Name of mapping table can be changed with the annotation `@JoinTable`
    - On owner side only, ignored on inverse side
  - Name of the columns in the join table can be changed with `@JoinTable` annotation attributes
    - `joinColumns = @JoinColumn(name="students_id")`
    - `inverseJoinColumns=@JoinColumn(name="modules_id")`

ManyToMany bidirectional

- **Representation in Database**




```
create table student (
        id              integer,
        name            varchar(32),
        prename         varchar(32)
);
```

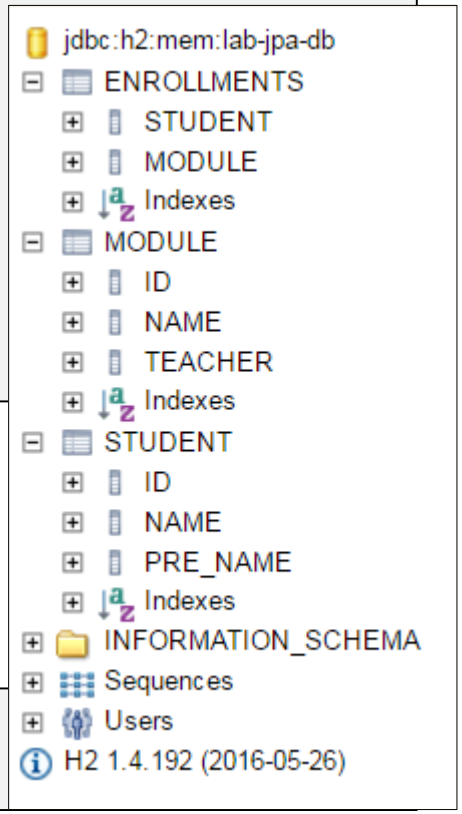
```
create table module (
        id              integer,
        name            varchar(32)
        teacher         varchar(32)
);
```

```
create table student_module (
        students_id     integer,
        modules_id      integer
);
```

ManyToMany bidirectional

- **Student: Configuration of the mapping table**

```
@JoinTable(
    name = "ENROLLMENTS",
    joinColumns = @JoinColumn(name = "student"),
    inverseJoinColumns =
                    @JoinColumn(name = "module")
)
@ManyToMany
private List<Module> modules = new LinkedList<>();
```



  - Configuration of the mapping table is defined on the owner side, e.g. on class `Student` in this example

- **Module: inverse side**

```
@ManyToMany(mappedBy="modules")
private List<Student> students = new LinkedList<>();
```

ManyToMany bidirectional: Test

```
em.getTransaction().begin();
Module m1 = new Module("webfr", "Luthiger");
Module m2 = new Module("conpr", "Kröni");
Student s1 = new Student("Meier");
Student s2 = new Student("Müller");
em.persist(m1);  // all entities are persisted
em.persist(m2);
em.persist(s1);
em.persist(s2);

m1.setStudents(List.of(s1, s2));
s1.setModules(List.of(m1, m2));
m2.setStudents(List.of(s2));

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

- **What is stored in the DB? => socrative.com**

Java Persistence API

- **Entity Manager**
- **Relations**
  - Relation Types
  - Cascade Types
  - Fetch Types
- **Samples (with focus on owner / inverse)**
  - OneToOne unidirectional
  - OneToOne bidirectional
  - OneToMany bidirectional
  - ManyToMany bidirectional