Association Mapping

CS544: Enterprise Architecture

Association Mapping

- In Java associations are made with object references

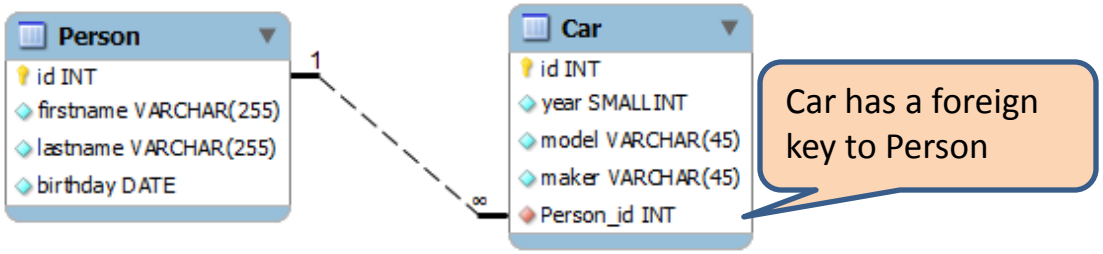
Person has a cars collection of references

```
public class Person {
  private int id;
  private String firstname;
  private String lastname;
  private List<Car> cars
    = new ArrayList();

  ...
```

```
public class Car {
  private int id;
  private short year;
  private String model;
  private String maker;
  private Person owner;

  ...
```

Car also has an owner reference back to its owner

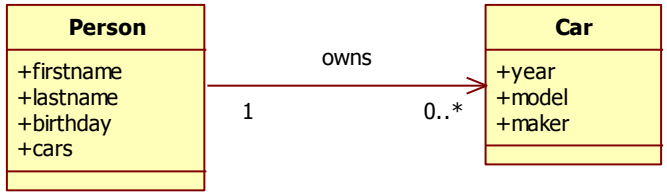
- In a relational schema associations are made with Foreign keys



- O/R Mapping translates references into foreign keys and visa versa.

OO Association Directionality

- Uni-directional association



Can only be traversed from person to car

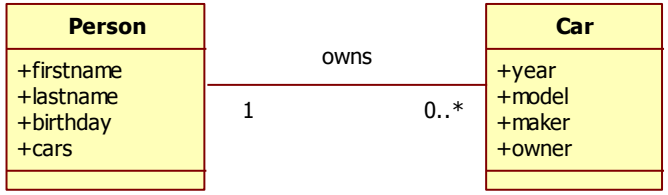
Person has a collection of references to Car objects

```
public class Person {
  private int id;
  private String firstname;
  private String lastname;
  private List<Car> cars
    = new ArrayList();

  ...
```

```
public class Car {
  private int id;
  private short year;
  private String model;
  private String maker;

  ...
```

Car does not have a reference back to person

- Bi-directional association



Association Can be traversed in both directions

Person has a collection of references to Car objects

```
public class Person {
  private int id;
  private String firstname;
  private String lastname;
  private List<Car> cars
    = new ArrayList();

  ...
```

```
public class Car {
  private int id;
  private short year;
  private String model;
  private String maker;
  private Person owner;

  ...
```

Car also has a reference back to person

Seven Types of Associations

- There are seven types of associations
  - 4 Uni-directional
  - 3 Bi-directional

Multiplicity	Uni-Directional	Bi-directional
One To One	Uni-Directional →	Bi-Directional
Many To One	Uni-Directional	Bi-Directional
One To Many	Uni-Directional	
Many To Many	Uni-Directional →	Bi-Directional

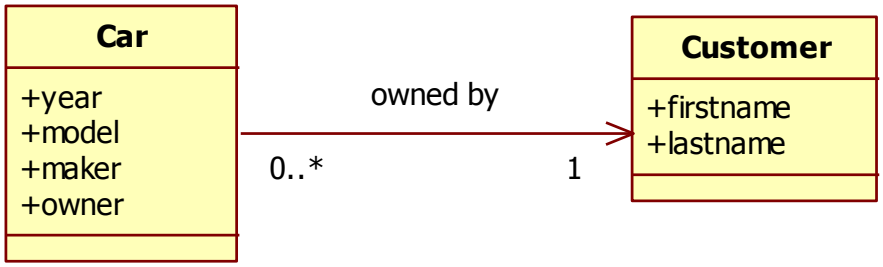
- One to Many and Many to One are different sides of the same bi-directional association

Association Mapping

MANY TO ONE ASSOCIATIONS

Many to One Uni-Directional

- Objects



A Car has one Customer and a Customer can be the owner of more than one Car

- Database

CAR table

ID	MAKER	MODEL	YEAR	CUSTOMER_ID
1	Honda	Acord	1996	1
2	Volvo	S80	1999	1

The CAR table has a foreign key of the customer

The foreign key column is always at the many site.

CUSTOMER table

ID	FIRSTNAME	LASTNAME
1	Frank	Brown

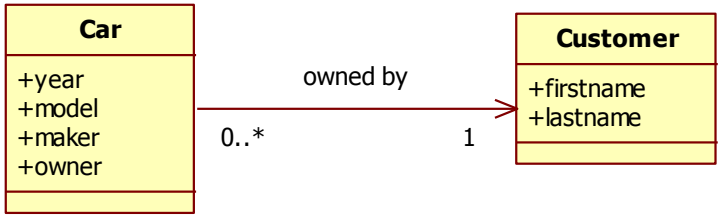
Many to One Uni-Directional

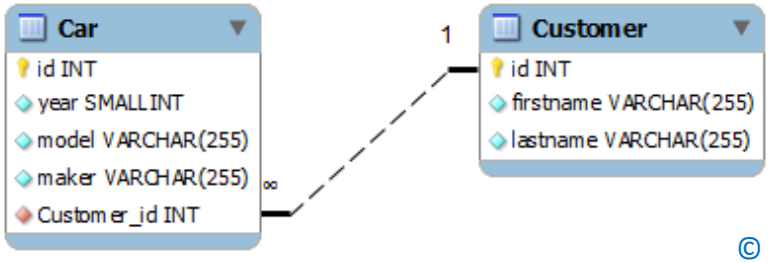
```java
@Entity
public class Car {
  @Id
  @GeneratedValue
  private int id;
  private short year;
  private String model;
  private String maker;
  @ManyToOne
  @JoinColumn(name="customer_id")
  private Customer customer;

  ...
```

@ManyToOne

Optional @JoinColumn to specify the FK column name

```java
@Entity
public class Customer {
  @Id
  @GeneratedValue
  private int id;
  private String firstname;
  private String lastname;

  ...
```





CAR table

ID	MAKER	MODEL	YEAR	CUSTOMER_ID
1	Honda	Acord	1996	1
2	Volvo	S80	1999	1

CUSTOMER table

ID	FIRSTNAME	LASTNAME
1	Frank	Brown

Uni-directional Many to One XML

```xml
<hibernate-mapping package="manyToOne_uni">
  <class name="Car" >
    <id name="id">
      <generator class="native"/>
    </id>
    <property name="year" />
    <property name="model" />
    <property name="maker" />
    <many-to-one name="customer" class="Customer"
          column="customer_id" />
  </class>
</hibernate-mapping>
```

```
public class Car {
  private int id;
  private short year;
  private String model;
  private String maker;
  private Customer customer;

  ...
```

<many-to-one> maps a FK column in the Car table

As before the column attribute is optional

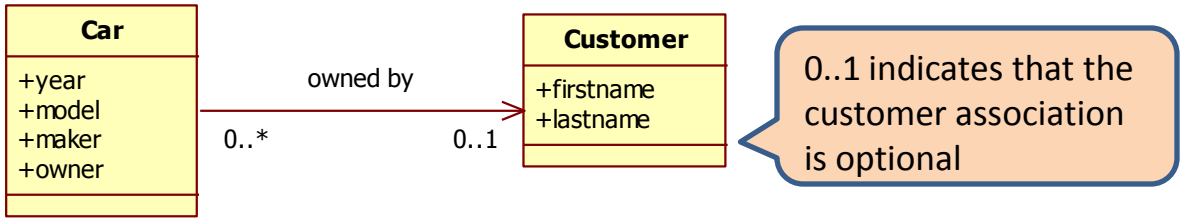
```xml
<hibernate-mapping package="manyToOne_uni">
  <class name="Customer" >
    <id name="id">
      <generator class="native"/>
    </id>
    <property name="firstname" />
    <property name="lastname" />
  </class>
</hibernate-mapping>
```

```
public class Customer {
  private int id;
  private String firstname;
  private String lastname;

  ...
```

Normal Customer class

Optional Associations

- Optional associations are associations that may not exist
  - A Car can exist without a Customer



CAR table

ID	MAKER	MODEL	YEAR	CUSTOMER_ID
1	Honda	Acord	1996	1
2	Volvo	S80	1999	

CUSTOMER table

ID	FIRSTNAME	LASTNAME
1	Frank	Brown

To facilitate this CUSTOMER_ID would have to be nullable

Avoid Nullable FK Columns

- Nullable columns are generally frowned upon since they break normalization
- To avoid nullable foreign key columns on optional associations you can use a join table for an optional many to one associations :

CAR table

ID	MAKER	MODEL	YEAR
1	Honda	Acord	1996
2	Volvo	S80	1999

CAR_CUSTOMER table

CUSTOMER_ID	ID
1	1

CUSTOMER table

ID	FIRSTNAME	LASTNAME
1	Frank	Brown

Join table to store the Car-Customer association

Optional Many to One (join table)

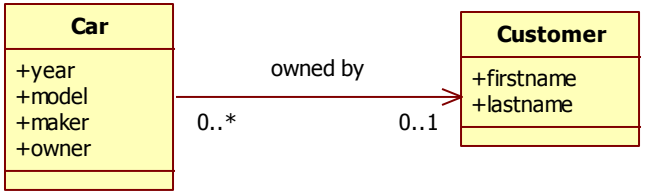
```java
@Entity
public class Car {
  @Id
  @GeneratedValue
  private int id;
  private short year;
  private String model;
  private String maker;
  @ManyToOne
  @JoinTable(name="car_customer")
  private Customer customer;

  ...
```

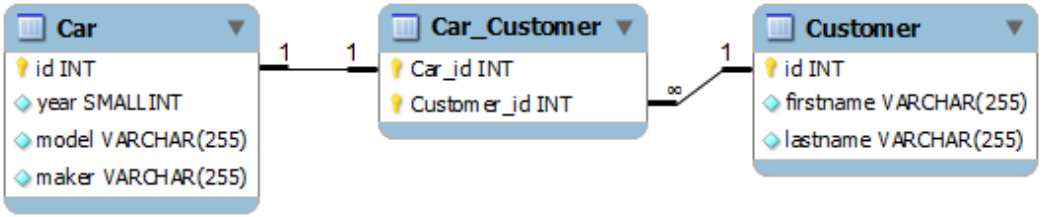
@ManyToOne

@JoinTable and the join table name are required

Normally mapped Customer class

```java
@Entity
public class Customer {
  @Id
  @GeneratedValue
  private int id;
  private String firstname;
  private String lastname;

  ...
```





Optional Many to One XML

```
<hibernate-mapping package="manyToOne_uni">
  <class name="Car" >
    <id name="id">
      <generator class="native"/>
    </id>
    <property name="year" />
    <property name="model" />
    <property name="maker" />
    <join table="car_customer" optional="true">
      <key column="car_id" />
      <many-to-one name="customer" class="Customer"
          column="customer_id" />
    </join>
  </class>
</hibernate-mapping>
```

<join> is used to specify the join table

```
public class Car {
  private int id;
  private short year;
  private String model;
  private String maker;
  private Customer customer;

  ...
```

<many-to-one> now maps a FK column in the join table

```
<hibernate-mapping package="manyToOne_uni">
  <class name="Customer" >
    <id name="id">
      <generator class="native"/>
    </id>
    <property name="firstname" />
    <property name="lastname" />
  </class>
</hibernate-mapping>
```

```
public class Customer {
  private int id;
  private String firstname;
  private String lastname;

  ...
```

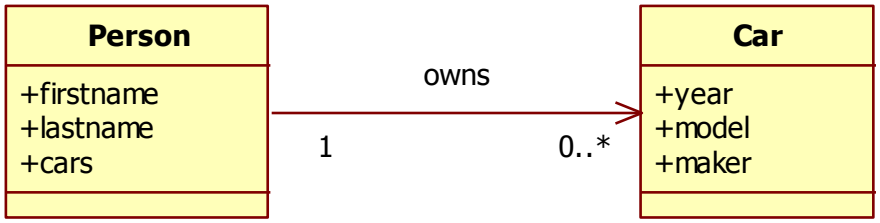
Normal Customer class

Association Mapping

ONE TO MANY ASSOCIATIONS

One to Many Uni-directional

- Objects



- Database (2 options)

Default

PERSON table

ID	FIRSTNAME	LASTNAME
1	Frank	Brown

CAR table

FK column on the many side

ID	MAKER	MODEL	YEAR	PERSON_ID
1	Honda	Acord	1996	1
2	Volvo	S80	1999	1

PERSON table

ID	FIRSTNAME	LASTNAME
1	Frank	Brown

PERSON_CAR table

PERSON_ID	CAR_ID
1	1
1	2

Join table

CAR table

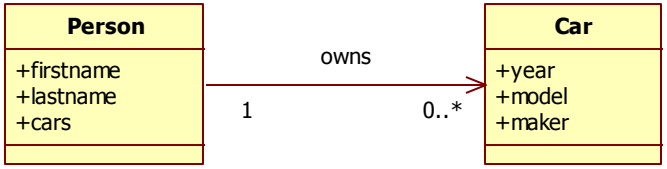
ID	MAKER	MODEL	YEAR
1	Honda	Acord	1996
2	Volvo	S80	1999

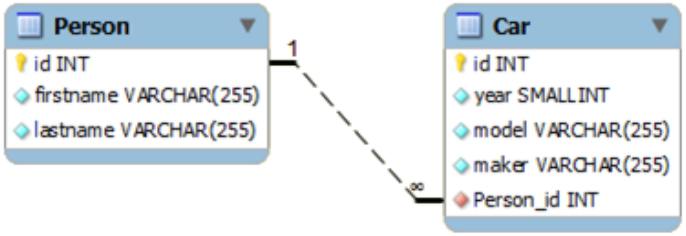
One to Many Uni-Directional FK

```
@Entity
public class Person {
  @Id
  @GeneratedValue
  private int id;
  private String firstname;
  private String lastname;
  @OneToMany
  @JoinColumn(name="person_id")
  private List<Car> cars = new ArrayList();

  ...
```

@OneToMany

Collection of car references





Using a FK on the many side

Normal mapping of Car class

```
@Entity
public class Car {
  @Id
  @GeneratedValue
  private int id;
  private short year;
  private String model;
  private String maker;

  ...
```

Uni–directional One to Many FK XML

```xml
<hibernate-mapping package="oneToMany_uniFK">
  <class name="Person" >
    <id name="id">
      <generator class="native"/>
    </id>
    <property name="firstname" />
    <property name="lastname" />
    <bag name="cars">
      <key column="person_id" />
      <one-to-many class="Car"/>
    </bag>
  </class>
</hibernate-mapping>
```

A <key> tag to specify a foreign key join column in Car

<one-to-many> tag to map the FK relation

```java
public class Person {
  private int id;
  private String firstname;
  private String lastname;
  private List<Car> cars =
            new ArrayList();

  ...
```

```xml
<hibernate-mapping package="oneToMany_uniFK">
  <class name="Car" >
    <id name="id">
      <generator class="native"/>
    </id>
    <property name="year" />
    <property name="model" />
    <property name="maker" />
  </class>
</hibernate-mapping>
```

Regular mapping of the Car class

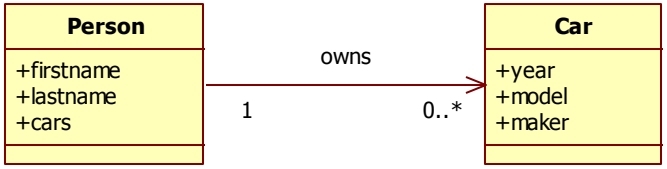
```java
public class Car {
  private int id;
  private short year;
  private String model;
  private String maker;

  ...
```

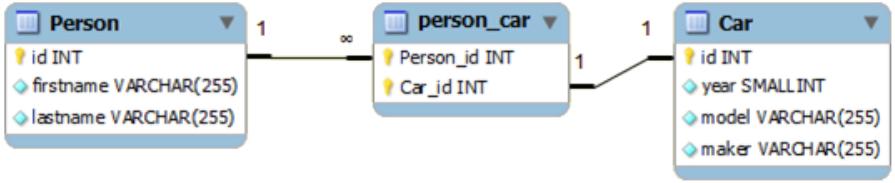
One to Many Uni-directional

```
@Entity
public class Person {
  @Id
  @GeneratedValue
  private int id;
  private String firstname;
  private String lastname;
  @OneToMany
  @JoinTable(name = "person_car",
      joinColumns = { @JoinColumn(name = "Person_id") },
      inverseJoinColumns = { @JoinColumn(name = "Car_id") }
  )
  private List<Car> cars = new ArrayList<Car>();

  ...
```

@OneToMany

Optional @JoinTable to specify the join table name and columns





Normal mapping of Car class

```
@Entity
public class Car {
  @Id
  @GeneratedValue
  private int id;
  private short year;
  private String model;
  private String maker;

  ...
```

Uni-direction One to Many XML

```
<hibernate-mapping package="oneToMany_uni">
  <class name="Person" >
    <id name="id">
      <generator class="native"/>
    </id>
    <property name="firstname" />
    <property name="lastname" />
    <bag name="cars" table="person_car">
      <key column="person_id" />
      <many-to-many column="car_id" unique="true" class="Car"/>
    </bag>
  </class>
</hibernate-mapping>
```

<bag> maps the collection and can specify the join table name

```
public class Person {
private int id;
private String firstname;
private String lastname;
private List<Car> cars =
            new ArrayList();

  ...
```

<many-to-many> used for join table mappings 'unique' constrains it to <one-to-many> functionality

```
<hibernate-mapping package="oneToMany_uni">
  <class name="Car" >
    <id name="id">
      <generator class="native"/>
    </id>
    <property name="year" />
    <property name="model" />
    <property name="maker" />
  </class>
</hibernate-mapping>
```

Normal mapping of the Car class

```
public class Car {
  private int id;
  private short year;
  private String model;
  private String maker;

  ...
```

Many to One / One to Many (Bi)

```java
@Entity
public class Person {
  @Id
  @GeneratedValue
  private int id;
  private String firstname;
  private String lastname;
  @OneToMany
  @JoinColumn(name="person_id")
  private List<Car> cars =
            new ArrayList();

  ...
```

This OneToMany association is stored in the foreign key column with name 'person_id' in the CAR table

```java
@Entity
public class Car {
  @Id
  @GeneratedValue
  private int id;
  private short year;
  private String model;
  private String maker;
  @ManyToOne
  @JoinColumn(name="owner_id")
  private Person owner;

  ...
```

This ManyToOne association is stored in the foreign key column with name 'owner_id' in the CAR table

PERSON table

ID	FIRSTNAME	LASTNAME
1	Frank	Brown

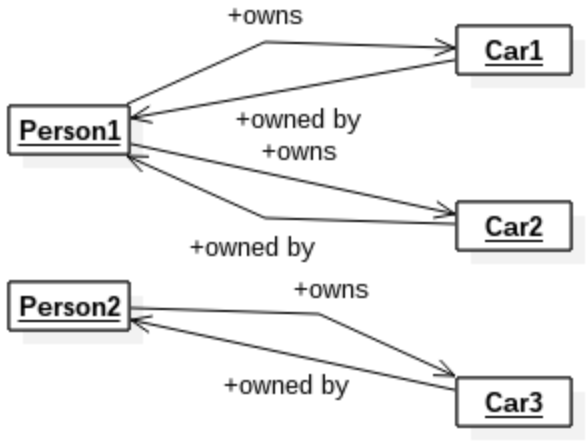
Hibernate sees this bi-directional association as 2 independent associations

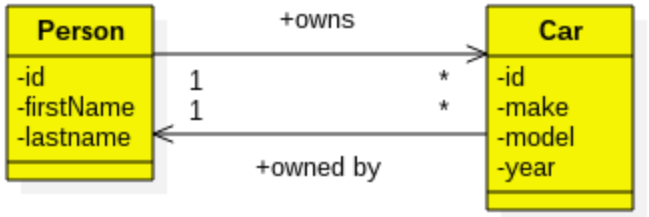
CAR table

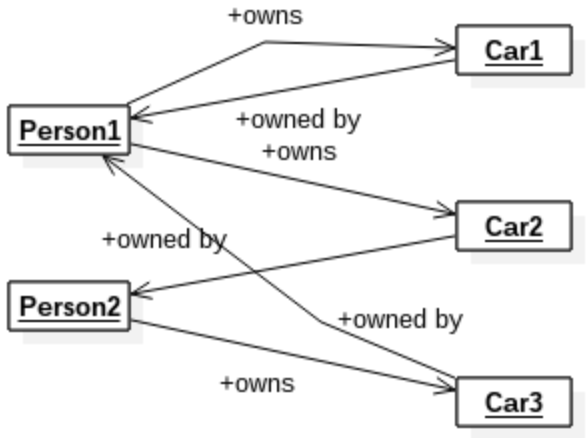
ID	MAKER	MODEL	YEAR	OWNER_ID	PERSON_ID
1	Honda	Acord	1996	1	1
2	Volvo	S80	1999	1	1

Both FK column contain the same information

Bi-directional VS 2 Uni-directional







mappedBy

```
@Entity
public class Person {
  @Id
  @GeneratedValue
  private int id;
  private String firstname;
  private String lastname;
  @OneToMany(mappedBy="owner")
  private List<Car> cars =
            new ArrayList();

  ...
```

mappedby indicates that the FK is on the other side

```
@Entity
public class Car {
  @Id
  @GeneratedValue
  private int id;
  private short year;
  private String model;
  private String maker;
  @ManyToOne
  @JoinColumn(name="owner_id")
  private Person owner;

  ...
```

Optional @JoinColumn to specify FK

The bi-directional association is stored in one FK colunm

PERSON table

ID	FIRSTNAME	LASTNAME
1	Frank	Brown

CAR table

ID	MAKER	MODEL	YEAR	OWNER_ID
1	Honda	Acord	1996	1
2	Volvo	S80	1999	1

Many to One / One to Many XML

```xml
<hibernate-mapping package="oneToMany_uni">
  <class name="Car" >
    <id name="id">
      <generator class="native"/>
    </id>
    <property name="year" />
    <property name="model" />
    <property name="maker" />
    <many-to-one name="owner" column="owner_id" class="Person" />
  </class>
</hibernate-mapping>
```

<many-to-one> creates a FK column to the Person table

```java
public class Car {
  private int id;
  private short year;
  private String model;
  private String maker;
  private Person owner;

  ...
```

```xml
<hibernate-mapping package="oneToMany_uni">
  <class name="Person" >
    <id name="id">
      <generator class="native"/>
    </id>
    <property name="firstname" />
    <property name="lastname" />
    <bag name="cars" inverse="true">
      <key column="owner_id" />
      <one-to-many class="Car"/>
    </bag>
  </class>
</hibernate-mapping>
```

inverse specifies that the FK is on the other side

<key> specifies the name of the FK column

```java
public class Person {
  private int id;
  private String firstname;
  private String lastname;
  private List<Car> cars =
            new ArrayList();

  ...
```

Bi-directional One to Many Join Table

- Although generally not needed, it is certainly possible to map a Bi-directional One-To-Many with a Join Table in between

```
@Entity
public class Person {
  @Id
  @GeneratedValue
  private int id;
  private String firstname;
  private String lastname;
  @OneToMany(mappedBy="owner")
  private List<Car> cars =
            new ArrayList();

  ...
```

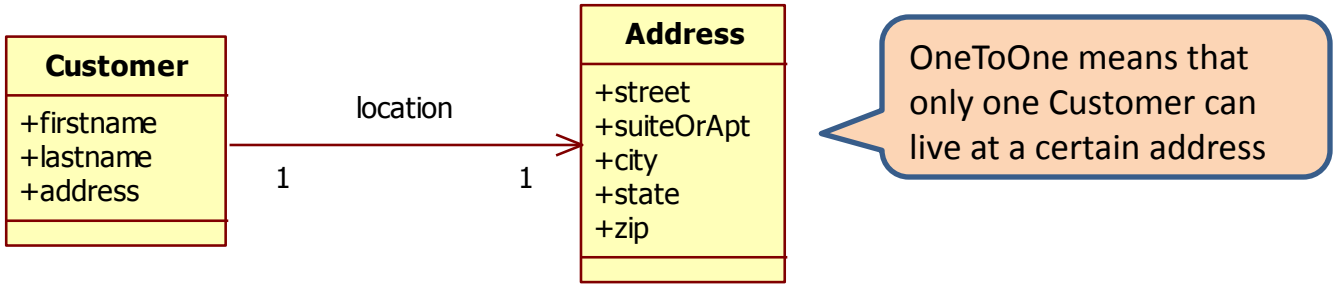
```
@Entity
public class Car {
  @Id
  @GeneratedValue
  private int id;
  private short year;
  private String model;
  private String maker;
  @ManyToOne
  @JoinTable(name="Car_Person")
  private Person owner;

  ...
```

Association Mapping

ONE TO ONE ASSOCIATIONS

OneToOne Uni-Directional

- Objects



- Database

CUSTOMER table

ID	FIRSTNAME	LASTNAME
1	John	Smith
2	Frank	Brown
3	Jane	Doe

ADDRESS table

ID	CITY	STATE	STREET	SUITEORAPT	ZIP
1	city1	state1	street1	suite1	zip1
3	city3	state3	street3	suite3	zip3

Shared primary key

OneToOne and ManyToOne

- OneToOne

Only one Customer can live at a certain address

CUSTOMER table

ID	FIRSTNAME	LASTNAME
1	John	Smith
2	Frank	Brown
3	Jane	Doe

ADDRESS table

ID	CITY	STATE	STREET	SUITEORAPT	ZIP
1	city1	state1	street1	suite1	zip1
3	city3	state3	street3	suite3	zip3

Shared primary key

- ManyToOne

More than one Customer can live at the same address

CUSTOMER table

ID	FIRSTNAME	LASTNAME	ADDRESS_ID
1	John	Smith	1
2	Frank	Brown	
3	Jane	Doe	2

ADDRESS table

ID	CITY	STATE	STREET	SUITEORAPT	ZIP
1	city1	state1	street1	suite1	zip1
3	city3	state3	street3	suite3	zip3

Foreign key

With a Unique constriant this is still basically a one-to-one

Uni-directional OneToOne

- JPA does not support a shared PK OneToOne

```
@Entity
public class Customer {
  @Id
  @GeneratedValue
  private int id;
  private String firstname;
  private String lastname;
  @OneToOne
  private Address address;

  ...
```

@OneToOne

```
@Entity
public class Address {
  @Id
  @GeneratedValue
  private int id;
  private String street;
  private String suiteOrApt;
  private String city;
  private String state;
  private String zip;

  ...
```

- This mapping results in a ManyToOne with a Unique Constraint (Therefore OneToOne)

CUSTOMER table

ID	FIRSTNAME	LASTNAME	ADDRESS_ID
1	John	Smith	1
2	Frank	Brown	
3	Jane	Doe	2

ADDRESS table

ID	CITY	STATE	STREET	SUITEORAPT	ZIP
1	city1	state1	street1	suite1	zip1
3	city3	state3	street3	suite3	zip3

Unique Constriant XML

```
<hibernate-mapping package="oneToOne_uni">
  <class name="Customer" >
    <id name="id">
      <generator class="native"/>
    </id>
    <property name="firstname" />
    <property name="lastname" />
    <many-to-one name="address" class="Address" unique="true"/>
  </class>
</hibernate-mapping>
```

```
public class Customer {
  private int id;
  private String firstname;
  private String lastname;
  private Address address;

  ...
```

many-to-one with a foreign key that is unique

Unique=true enforces one to one behavior

```
<hibernate-mapping package="oneToOne_uni">
  <class name="Address" >
    <id name="id">
      <generator class="native"/>
    </id>
    <property name="street" />
    <property name="suiteOrApt" />
    <property name="city" />
    <property name="state" />
    <property name="zip" />
  </class>
</hibernate-mapping>
```

```
public class Address {
  private int id;
  private String street;
  private String suiteOrApt;
  private String city;
  private String state;
  private String zip;

  ...
```

Normal Address class

One to One Bi-directional

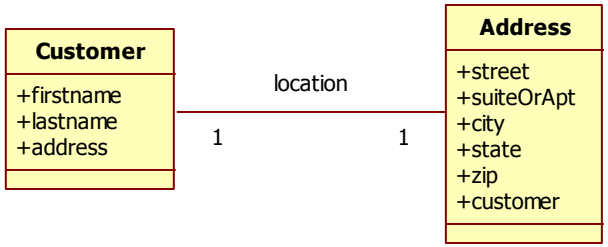
```
@Entity
public class Customer {
  @Id
  @GeneratedValue
  private int id;
  private String firstname;
  private String lastname;
  @OneToOne
  @JoinColumn(name="address_id")
  private Address address;

  ...
```

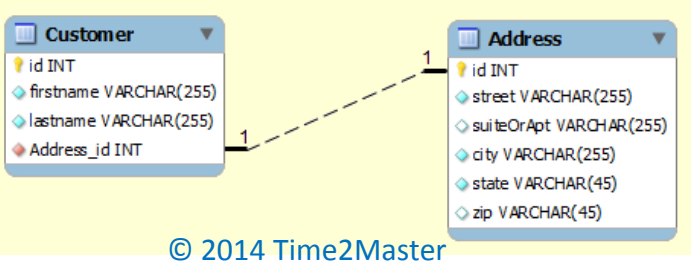
Optional @JoinColumn to specify FK name

```
@Entity
public class Address {
  @Id
  @GeneratedValue
  private int id;
  private String street;
  private String suiteOrApt;
  private String city;
  private String state;
  private String zip;
  @OneToOne(mappedBy="address")
  private Customer customer;

  ...
```

Other side also uses @OneToOne and specifies that the FK is mappedBy address

mappedBy thus specifies that this side is ***not*** the owning side





One to One bi-directional in XML

```xml
<hibernate-mapping package="oneToOne_bi">
  <class name="Customer" >
    <id name="id">
      <generator class="native"/>
    </id>
    <property name="firstname" />
    <property name="lastname" />
    <many-to-one name="address" class="Address" unique="true"/>
  </class>
</hibernate-mapping>
```

```xml
public class Customer {
  private int id;
  private String firstname;
  private String lastname;
  private Address address;

  ...
```

<many-to-one> creates a FK column to the Address table

Unique=true constrains the many-to-one into one-to-one behavior

```xml
<hibernate-mapping package="oneToOne_bi">
  <class name="Address" >
    <id name="id">
      <generator class="native"/>
    </id>
    <property name="street" />
    <property name="suiteOrApt" />
    <property name="city" />
    <property name="state" />
    <property name="zip" />
    <one-to-one name="customer" class="Customer"
        property-ref="address"/>
  </class>
</hibernate-mapping>
```

```xml
public class Address {
  private int id;
  private String street;
  private String suiteOrApt;
  private String city;
  private String state;
  private String zip;
  private Customer customer;

  ...
```

<one-to-one> specifies that the FK is mapped by address in Customer, and thereby that this side is ***not*** the owning side

Shared PK Workaround: @PrimaryKeyJoinColumn

```
@Entity
public class Customer {
  @Id
  @GeneratedValue
  private int id;
  private String firstname;
  private String lastname;
  @OneToOne
  @PrimaryKeyJoinColumn
  private Address address;

  ...
```

@PrimaryKeyJoinColumn
Join on PK value

Primary key value not generated

```
@Entity
public class Address {
  @Id
  private int id;
  private String street;
  private String suiteOrApt;
  private String city;
  private String state;
  private String zip;

  ...
```

Id has to be set manually

CUSTOMER table

ID	FIRSTNAME	LASTNAME
1	John	Smith
2	Frank	Brown
3	Jane	Doe

ADDRESS table

ID	CITY	STATE	STREET	SUITEORAPT	ZIP
1	city1	state1	street1	suite1	zip1
3	city3	state3	street3	suite3	zip3

Shared primary key

One to One Shared PK XML

```xml
<hibernate-mapping package="oneToOne_uni_PKJoin">
  <class name="Customer" >
    <id name="id">
      <generator class="native"/>
    </id>
    <property name="firstname" />
    <property name="lastname" />
    <one-to-one name="address" class="Address" />
  </class>
</hibernate-mapping>
```

```java
public class Customer {
private int id;
  private String firstname;
  private String lastname;
  private Address address;

  ...
```

One-to-one indicates that this reference is mapped without FK

```xml
<hibernate-mapping package="oneToOne_uni_PKJoin">
  <class name="Address" >
    <id name="id" />
    <property name="street" />
    <property name="suiteOrApt" />
    <property name="city" />
    <property name="state" />
    <property name="zip" />
  </class>
</hibernate-mapping>
```

Again no PK generation, Id has to be set manually

```java
public class Address {
  private int id;
  private String street;
  private String suiteOrApt;
  private String city;
  private String state;
  private String zip;

  ...
```

One to One Shared PK Bi-directional

```
@Entity
public class Customer {
  @Id
  @GeneratedValue
  private int id;
  private String firstname;
  private String lastname;
  @OneToOne
  @PrimaryKeyJoinColumn
  private Address address;

  ...
```

AUTO generated

Specify PK Join

```
@Entity
public class Address {
  @Id
  @GeneratedValue(generator="myGenerator")
  @org.hibernate.annotations.GenericGenerator(
      name="myGenerator",
      strategy="foreign",
      parameters=@Parameter(name="property",
                            value="customer")
  )
  private int id;
  private String street;
  private String suiteOrApt;
  private String city;
  private String state;
  private String zip;
  @OneToOne
  @PrimaryKeyJoinColumn
  private Customer customer;

  ...
```

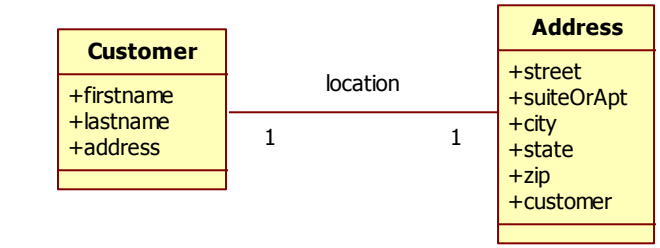
Custom generator

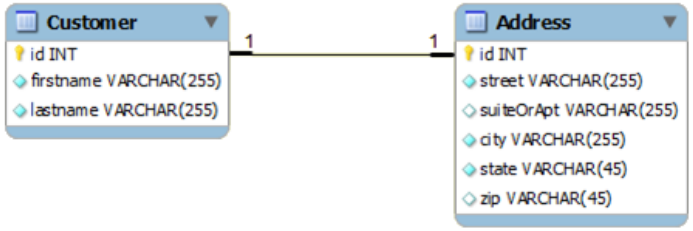
Hibernate extension

'foreign ' strategy

Select value from customer PK

Also Specify PK Join on this side





Shared PK XML Bi-directional

```
<hibernate-mapping package="oneToOne_bi_PKJoin">
  <class name="Customer" >
    <id name="id">
      <generator class="native"/>
    </id>
    <property name="firstname" />
    <property name="lastname" />
    <one-to-one name="address" class="Address" />
  </class>
</hibernate-mapping>
```

```
public class Customer {
  private int id;
  private String firstname;
  private String lastname;
  private Address address;

  ...
```

<one-to-one> specifies no additional FK

```
<hibernate-mapping package="oneToOne_bi_PKJoin">
  <class name="Address" >
    <id name="id">
      <generator class="foreign">
        <param name="property">customer</param>
      </generator>
    </id>
    <property name="street" />
    <property name="suiteOrApt" />
    <property name="city" />
    <property name="state" />
    <property name="zip" />
    <one-to-one name="customer" class="Customer"
                constrained="true" />
  </class>
</hibernate-mapping>
```

```
public class Address {
  private int id;
  private String street;
  private String suiteOrApt;
  private String city;
  private String state;
  private String zip;
  private Customer customer;

  ...
```

Foreign strategy selects pk from customer

<one-to-one> specifies no additional FK

constrained attribute adds FK constraint to PK

Association Mapping

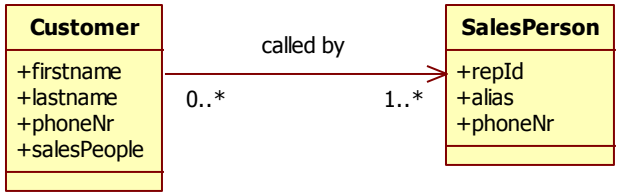
MANY TO MANY ASSOCIATIONS

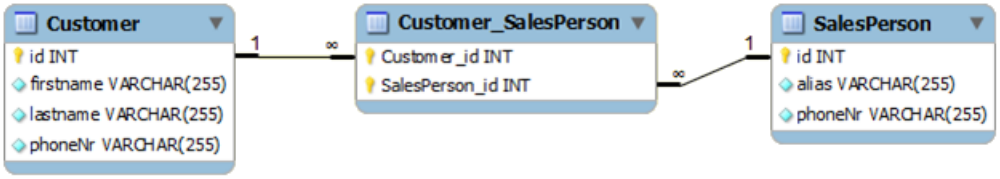
Many to Many Uni-directional

```
@Entity
public class Customer {
  @Id
  @GeneratedValue
  private int id;
  private String firstname;
  private String lastname;
  private String phoneNr;
  @ManyToMany
  @JoinTable(name = "Customer_SalesPerson",
      joinColumns = { @JoinColumn(name = "Customer_id") },
      inverseJoinColumns = { @JoinColumn(name = "SalesPerson_id") }
  )
  private List<SalesPerson> salesPeople = new ArrayList();
```

@ManyToMany

Optional @JoinTable






```
@Entity
public class SalesPerson {
  @Id
  @GeneratedValue
  private int id;
  private String alias;
  private String phoneNr;

  ...
```

Normal mapping of SalesPerson class

Uni-directional Many to Many XML

```
<hibernate-mapping package="manyToMany_uni">
  <class name="Customer" >
    <id name="id">
      <generator class="native"/>
    </id>
    <property name="firstname" />
    <property name="lastname" />
    <property name="phoneNr" />
    <bag name="salesPeople" table="customer_salesperson">
      <key column="customer_id" not-null="true" />
      <many-to-many column="salesperson_id"  class="SalesPerson" />
    </bag>
  </class>
</hibernate-mapping>
```

```
public class Customer {
private int id;
  private String firstname;
  private String lastname;
  private List<SalesPerson> salesPeople
        = new ArrayList();

  ...
```

Table attribute specifies the join table name

<many-to-many> tag to map the join table based relation

```
<hibernate-mapping package="manyToMany_uni">
  <class name="SalesPerson" >
    <id name="id">
      <generator class="native"/>
    </id>
    <property name="alias" />
    <property name="phoneNr" />
  </class>
</hibernate-mapping>
```

```
public class SalesPerson {
  private int id;
  private String alias;
  private String phoneNr;

  ...
```

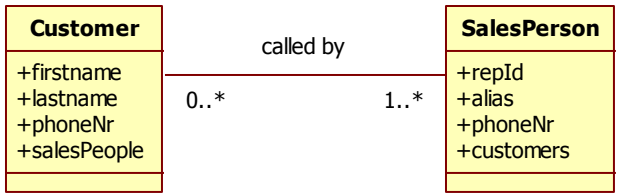
Regular mapping of the SalesPerson class

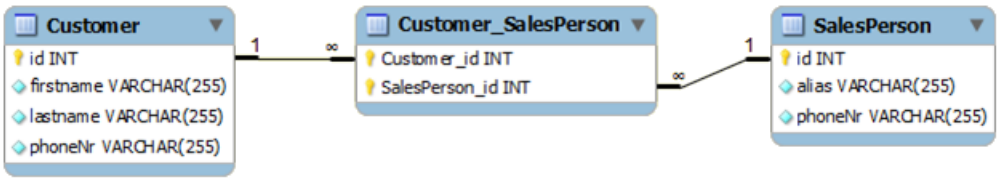
Many to Many Bi-directional

```
@Entity
public class Customer {
  @Id
  @GeneratedValue
  private int id;
  private String firstname;
  private String lastname;
  private String phoneNr;
  @ManyToMany
  @JoinTable(name = "Customer_SalesPerson",
    joinColumns = { @JoinColumn(name = "Customer_id") },
    inverseJoinColumns = { @JoinColumn(name = "SalesPerson_id") }
  )
  private List<SalesPerson> salesPeople = new ArrayList();

  ...
```

@ManyToMany

@JoinTable is optional






```
@Entity
public class SalesPerson {
  @Id
  @GeneratedValue
  private int id;
  private String alias;
  private String phoneNr;
  @ManyToMany(mappedBy="salesPeople")
  private List<Customer> customers =
          new ArrayList();

  ...
```

mappedBy specifies that the other side is the owning side

Bi-Directional Many to Many XML

```
<hibernate-mapping package="manyToMany_bi">
  <class name="Customer" >
    <id name="id">
      <generator class="native"/>
    </id>
    <property name="firstname" />
    <property name="lastname" />
    <property name="phoneNr" />
    <bag name="salesPeople" table="customer_salesperson">
      <key column="customer_id" />
      <many-to-many column="salesperson_id" class="SalesPerson" />
    </bag>
  </class>
</hibernate-mapping>
```

```
public class Customer {
  private int id;
  private String firstname;
  private String lastname;
  private String phoneNr;
  private List<SalesPerson> salesPeople
        = new ArrayList();

  ...
```

<bag> maps collection and specifies table name

Think of <many-to-many> as the join table mapping tag

```
<hibernate-mapping package="manyToMany_bi">
  <class name="SalesPerson" >
    <id name="id">
      <generator class="native"/>
    </id>
    <property name="alias" />
    <property name="phoneNr" />
    <bag name="customers" table="customer_salesperson" inverse="true">
      <key column="salesperson_id" />
      <many-to-many column="customer_id" class="Customer" />
    </bag>
  </class>
</hibernate-mapping>
```

```
public class SalesPerson {
private int id;
  private String alias;
  private String phoneNr;
  private List<Customer> customers =
          new ArrayList();

  ...
```

Inverse specifies that the other side is the owning side

Opposite mapping is similar, but adds inverse

Association Mapping

ASSOCIATION CASCADES

Association Cascades

```
@Entity
public class Person {
  @Id
  @GeneratedValue
  private int id;
  private String firstname;
  private String lastname;
  @OneToMany(mappedBy="owner")
  private List<Car> cars =
            new ArrayList();

  ...
```

```
@Entity
public class Car {
  @Id
  @GeneratedValue
  private int id;
  private short year;
  private String model;
  private String maker;
  @ManyToOne
  @JoinColumn(name="owner_id")
  private Person owner;

  ...
```

- By default hibernate does not cascade
  - During a session.persist(person) its car(s) will not be persisted
  - During a session.update(person) its car(s) will not be updated
  - During a session.delete(person) its car(s) will not be deleted

Specifying Cascades

- Each association tag has a cascade attribute

```
@Entity
public class Person {
  @Id
  @GeneratedValue
  private int id;
  private String firstname;
  private String lastname;
  @OneToMany(mappedBy="owner", cascade=CascadeType.PERSIST)
  private List<Car> cars = new ArrayList();

  ...
```

Association will cascade on Persist operations

When a person is persisted its cars will also be persisted

- Specify an array of cascade types:

```
@Entity
public class Person {
  @Id
  @GeneratedValue
  private int id;
  private String firstname;
  private String lastname;
  @OneToMany(mappedBy="owner", cascade={CascadeType.PERSIST, CascadeType.MERGE})
  private List<Car> cars = new ArrayList();

  ...
```

Cascade on persist and Merge

Cascade Types

Hibernate	JPA	Description
all	ALL	Cascade on all operations
persist	PERSIST	Cascade on persist operations
merge	MERGE	Cascade on merge operations
remove	REMOVE	Cascade on remove operations
refresh	REFRESH	Cascade on refresh operations
save-update	-	Cascade on save or update operations or on flush
delete	-	Cascade on delete or remove operations
lock	-	Cascade on lock operations
replicate	-	Cascade on replicate operations
evict	-	Cascade on evict operations

Hibernate Cascade Annotation

- Hibernate annotation extensions can be used to specify Hibernate specific cascade types

```
@Entity
public class Person {
  @Id
  @GeneratedValue
  private int id;
  private String firstname;
  private String lastname;
  @OneToMany(mappedBy="owner", cascade={CascadeType.PERSIST, CascadeType.MERGE})
  @org.hibernate.annotations.Cascade(
      org.hibernate.annotations.CascadeType.SAVE_UPDATE
  )
  private List<Car> cars = new ArrayList();

  ...
```

Cascade on persist and Merge

Cascade on save or update

XML Cascades

```
<hibernate-mapping package="cascade">
  <class name="Car" >
    <id name="id">
      <generator class="native"/>
    </id>
    <property name="year" />
    <property name="model" />
    <property name="maker" />
    <many-to-one name="owner" column="owner_id" class="Person"
        cascade="persist, merge, save-update" />
  </class>
</hibernate-mapping>
```

Cascade attribute on *-to-one> tags

Any of the Hibernate cascade types

```
<hibernate-mapping package="cascade">
  <class name="Person" >
    <id name="id">
      <generator class="native"/>
    </id>
    <property name="firstname" />
    <property name="lastname" />
    <bag name="cars" inverse="true" cascade="persist, merge, save-update">
      <key column="owner_id" />
      <one-to-many class="Car"/>
    </bag>
  </class>
</hibernate-mapping>
```

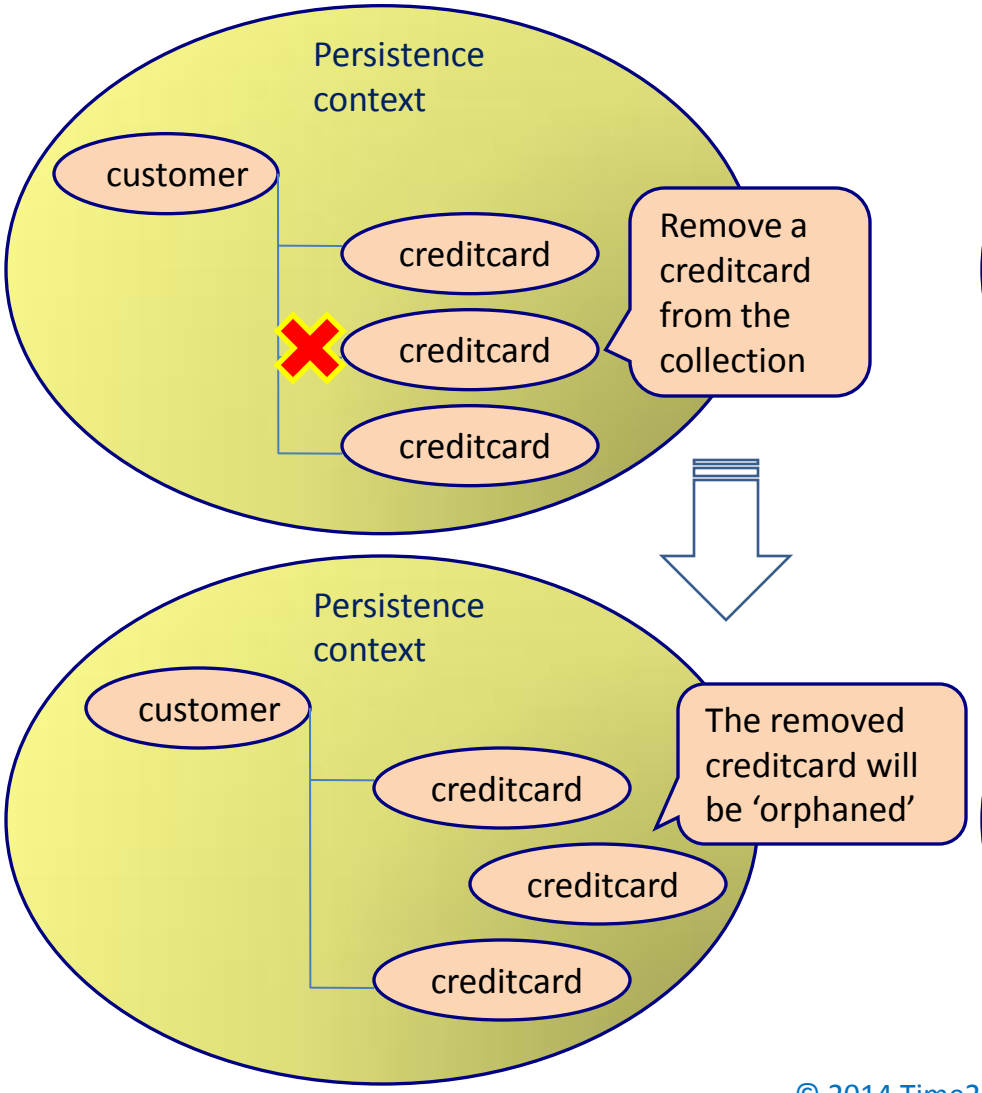
Cascade attribute on collection tags

Any of the Hibernate cascade types

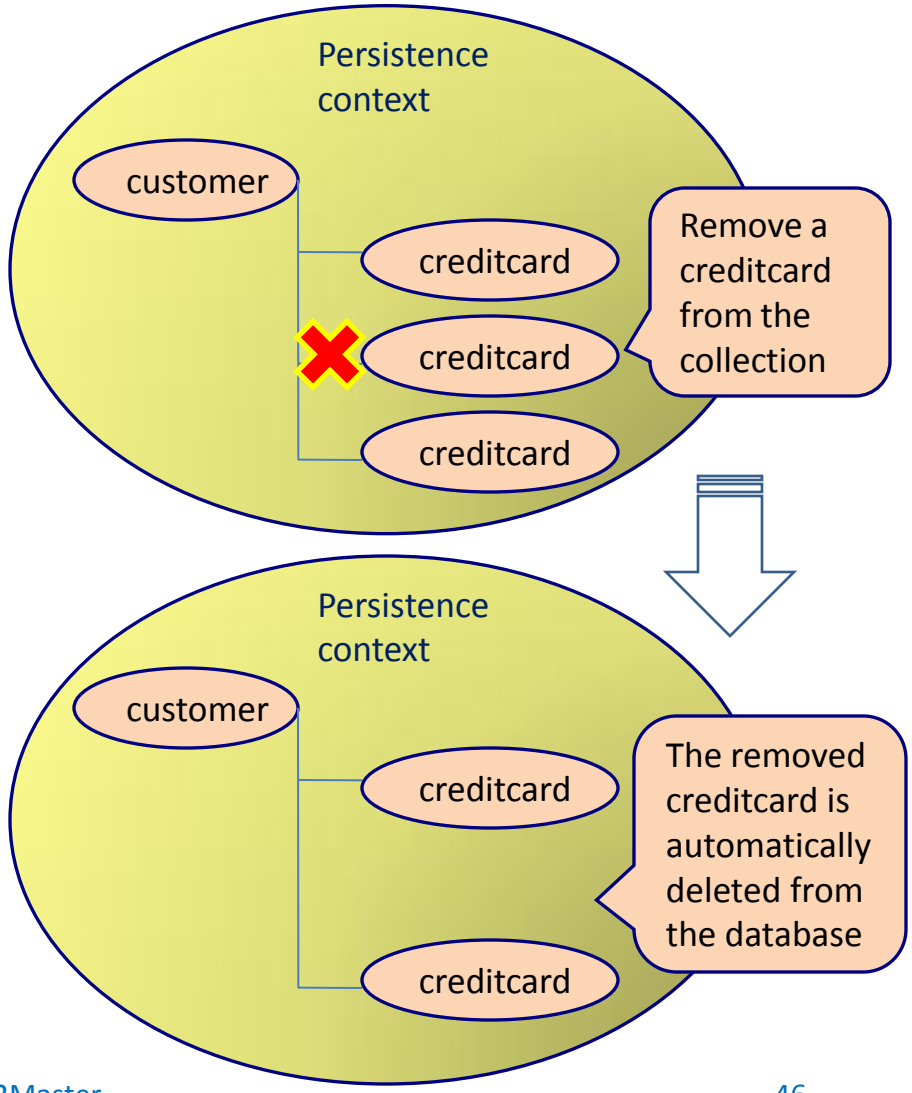
Orphan Removal

Available on uni and bi directional @OneToMany and @OneToOne

One to Many without Orphan Removal



One to Many using Orphan Removal



Orphan Removal

- Add ‘orphanRemoval=true’ to an @OneToOne or @OneToMany (uni or bi directional)

```
@Entity
public class Customer{
  @Id
  @GeneratedValue
  private int id;
  private String firstname;
  private String lastname;
  @OneToMany(mappedBy="customer", orphanRemoval=true)
  private List<CreditCard> cards = new ArrayList<>();

  ...
```

Cascade on persist and Merge

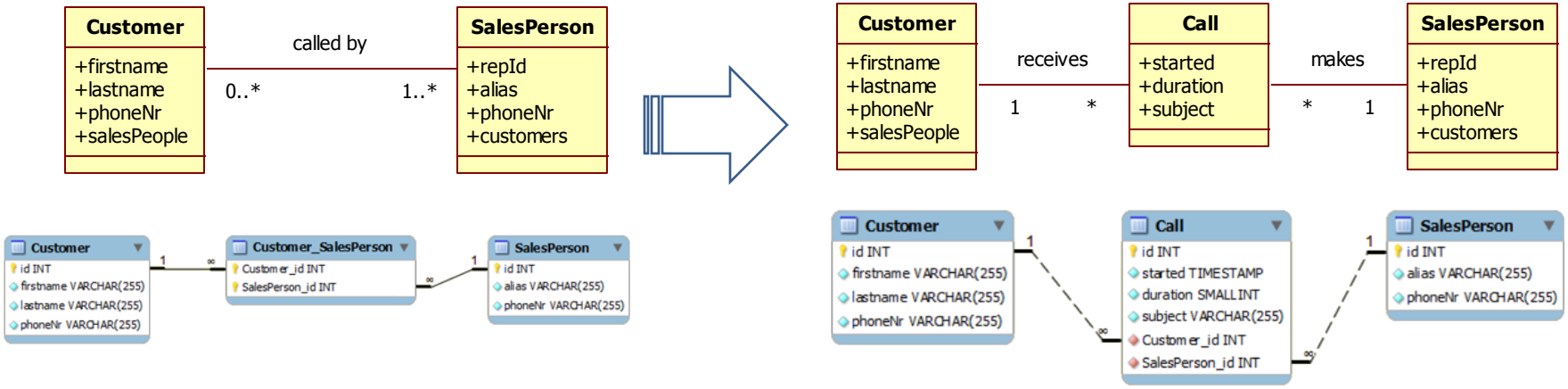
Cascade on save or update

Association Mapping

WRAPPING UP

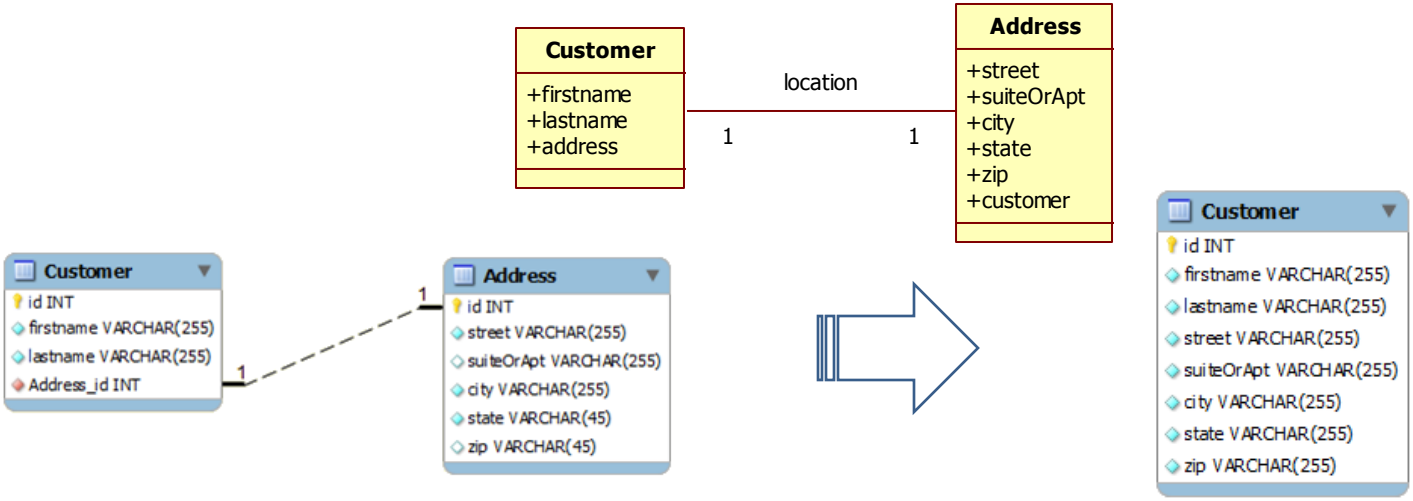
Mapping Tips

- Reconsider many to many relationships
  - Often people want to store additional data related to many-to-many association
  - If not now then perhaps in the near future
  - May be better to map the join table as an entity



Mapping Tips

- One to One relationships are often very tight
  - E.g. a Customer always has an Address
  - You may want to to include address in the customer table
  - You can do so using embedded classes which we will explain later in the course



Convenience Methods

- To create or remove a bi-directional association two references have to be created or removed
- It can become a bit tedious to set both sides
- Create convenience methods that set both sides in one go

Normal Getter / Setter for the collection of car references

```
public class Person {

  ...
  public List<Car> getCars() { return cars; }
  public void setCars(List<Car> cars) { this.cars = cars; }

  public boolean addCar(Car car) {
    car.setOwner(this);
    return cars.add(car);
  }
  public boolean removeCar(Car car) {
    car.setOwner(null);
    return cars.remove(car);
  }
```

Additional convenience methods to set references on both sides

Enforced 'Convenience'

- Don't allow the collection to be modified except through convenience methods
- This Ensures that the association is persisted
- Price: Hibernate field access for the collection

```
public class Person {

  ...
  public List<Car> getCars() { return Collections.unmodifiableList(cars); }
  public void setCars(List<Car> cars) { this.cars = cars; }

  public boolean addCar(Car car) {
    car.setOwner(this);
    return cars.add(car);
  }
  public boolean removeCar(Car car) {
    car.setOwner(null);
    return cars.remove(car);
  }
```

Getter returns unmodifiable collection

Can also remove / omit setter

Have to use convenience methods, the association is always persisted

Active Learning

- What are the seven different types of associations?

- Why would anyone want to use a join table in a uni-directional **one-to-many** association?

Module Summary

- In this module we discussed how to map the seven different types of associations
  - Both uni-directional and bi-directional associations
  - ManyToOne, OneToMany, OneToOne, and ManyToMany
- Most can be mapped in more than one way
  - With or without a join table / joining on FKs or PKs
- Association can also specify which actions should cascade down to the related entities
  - Persist, merge, remove, refresh, all

Main Point

- The seven different types of OO associations can often be mapped in more than one way. Generally though there is one normal way to map, and one or more exceptional ways to map an association depending on your DB or OO Domain needs.
- *Science of Consciousness*: Transcendental Meditation settles the mind, allowing one to select the right tool for the specific situation at hand, allowing you to do less and accomplish more.