Associations
Managing the impedance mismatch

ORM Impedance Mismatch

- 2 Different Technologies – 2 different ways to operate
- **EXAMPLE**
  - **OO traverse objects through relationships**
    - Category category = product.getCategory();
  - **RDB join the data rows of tables**
    - SELECT c.* FROM product p,category c where p.category_id = c.id;

- **OTHERS:**

Many-to-many relationships
Inheritance
Collections
Identity [Primary Key .vs. a.equals(b)]
Foreign Keys
Bidirectional ["Set both sides"]
Granularity [# of Tables .vs. # of Classes]

Associations

- Java associations are made with **references**

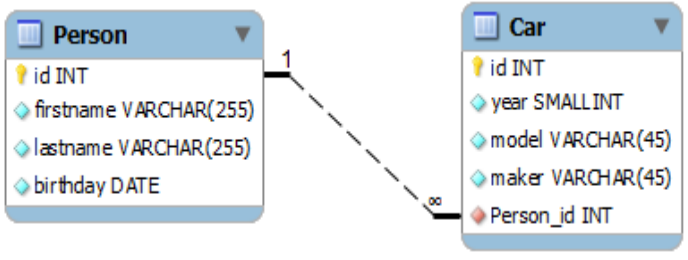
Person has cars collection of references

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

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

Car has an owner reference back to Person

- Relational association are made with **foreign keys**



Car has an Foreign Key to Person

- ORM maps refs to FKs (and FKs to refs)

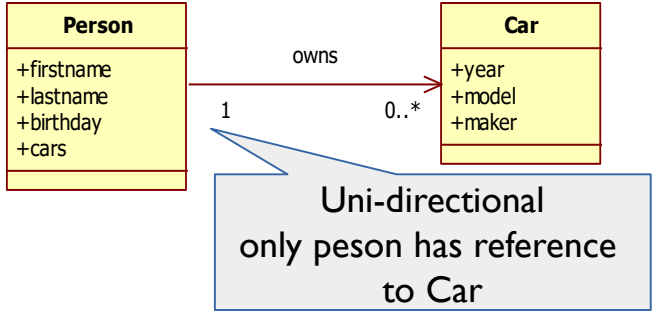
Directionality

- OO has **uni-directional** and **bi-directional**
  - Relational is always bi-directional (can emulate?)

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

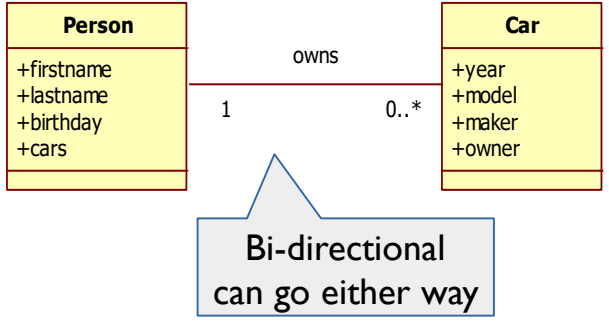
```
public class Car {
        private Long id;
        private short year;
        private String model;
        private String maker;
```

No ref to Person




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

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



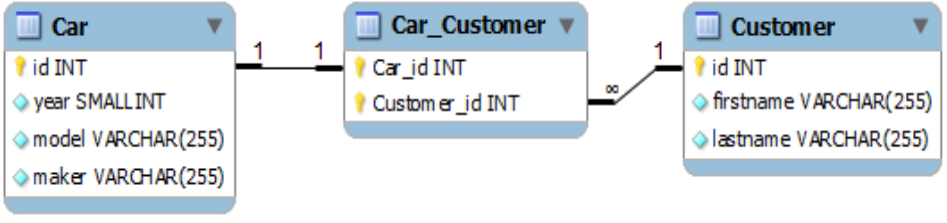
Types of Relationships

- **7 types** of relationships: 4 uni, 3 bi-directional
  - ManyToOne and OneToMany are different sides of the same bi-directional relationship

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

Join Table

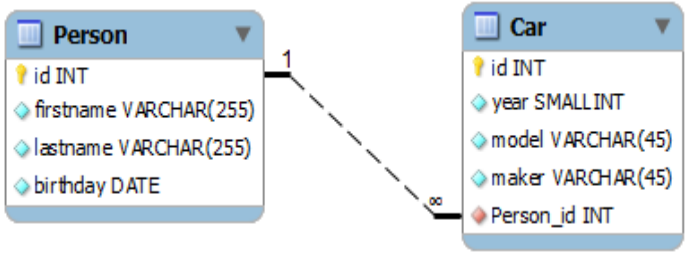
- Relational can use a table to hold foreign keys
  - Required to make a many-to-many relationship
  - Can also be used for **any relationship**



- This concept has many names:
  - Junction table, association table, link table, ...

Mapping Bi-directional

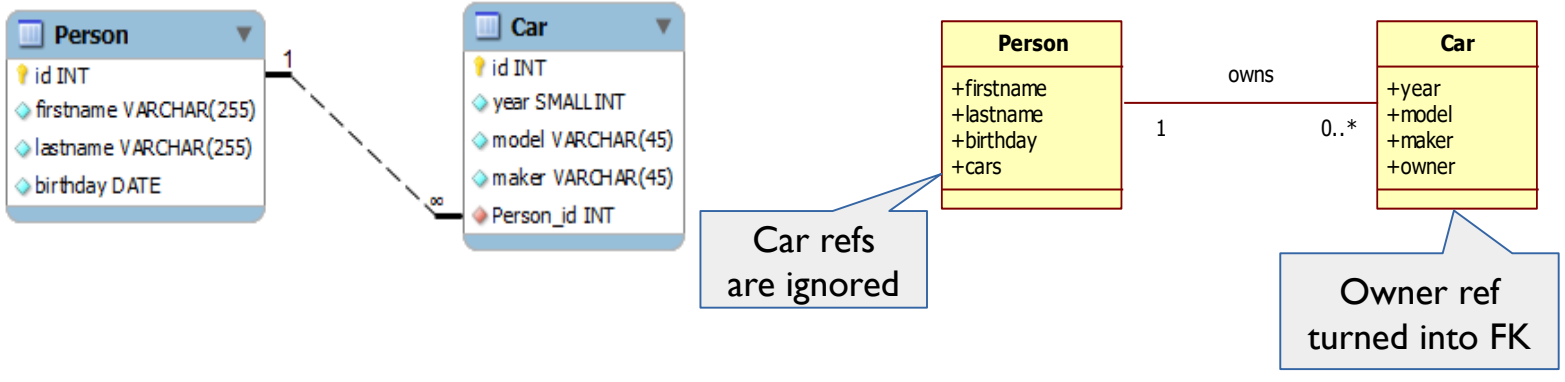
- Relational has one FK for bi-directional
  - Can be joined either direction



- OO has two sides that both need reference(s)
  - One side will become the 'owning side'

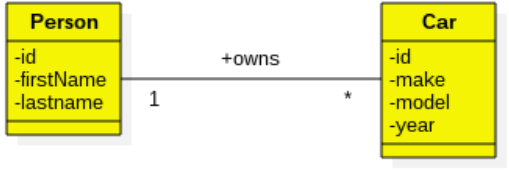
Owning Side

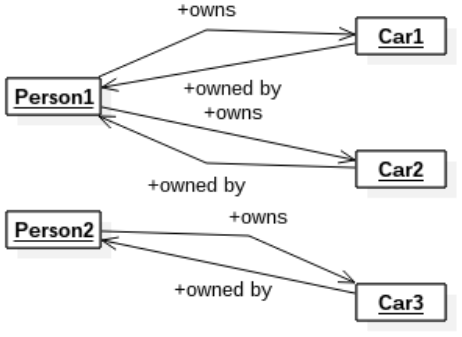
- The owning side in a bi-directional association
  - These references are turned into FK values
  - **Other side** references are **ignored** when persisting
  - In ManyToOne the many side is **natural owner**

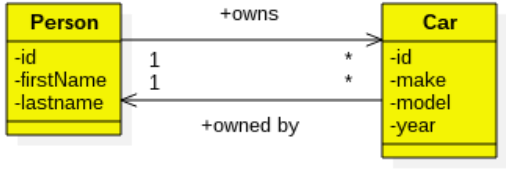


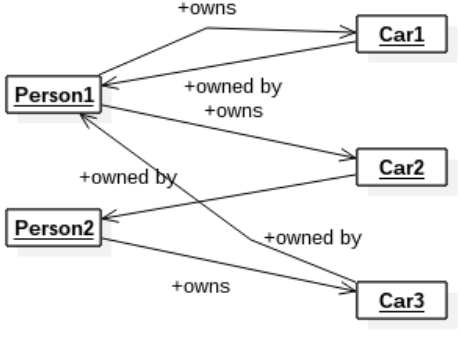
Bi-Directional VS 2 Uni-Directional

- If you **do not specify** an owning side
  - You get **two uni-directional** references!









Bi-Directional Convenience

- Create convenience methods
  - Properly **maintain bi-directional** association in Java

```
@Entity
public class Person {

        ...

        public boolean addCar(Car car) {
                if (cars.add(car)) {
                        car.setOwner(this);
                        return true;
                }
                return false;
        }

        public boolean removeCar(Car car) {
                if (cars.remove(car)) {
                        car.setOwner(null);
                        return true;
                }
                return false;
        }
```

Set both references

Unset both references

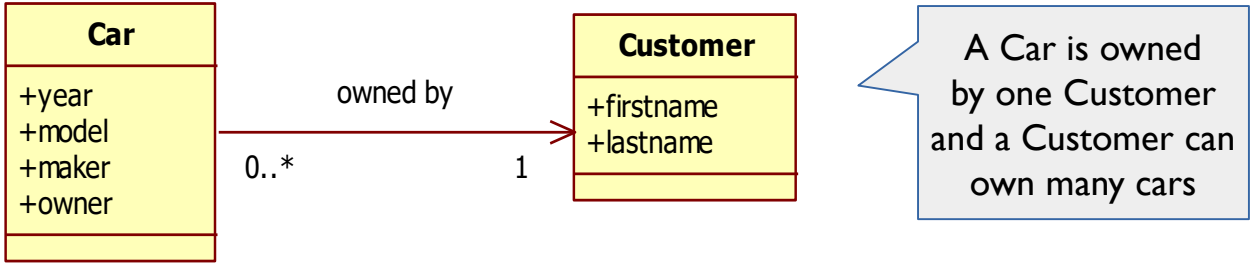
CS544 EA

Hibernate

Association: ManyToOne

ManyToOne uni-directional

- OO:



- Relational:

FK is always on the many side

CAR table

Car table has a Customer FK

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

CUSTOMER table

ID	FIRSTNAME	LASTNAME
1	Frank	Brown

Code

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

        ...
```

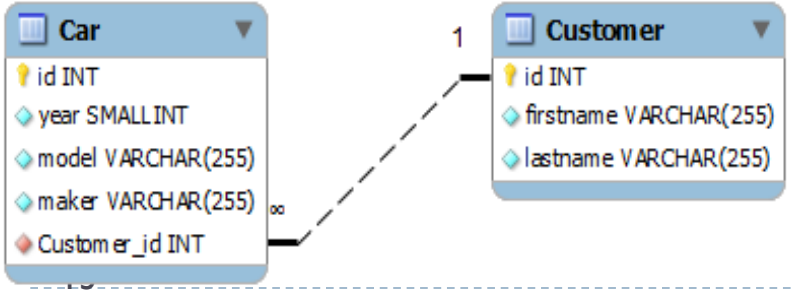
@ManyToOne

Optional @JoinColumn to name the column in DB

Default name: owner_id

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

        ...
```

Normally mapped Customer Entity



CAR table

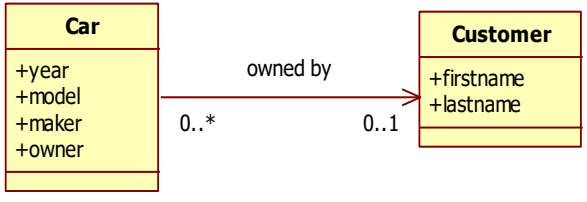
ID	MAKER	MODEL	YEAR	CUSTOMER_ID
1	Honda	Acord	1996	1
2	Volvo	S80	1999	1

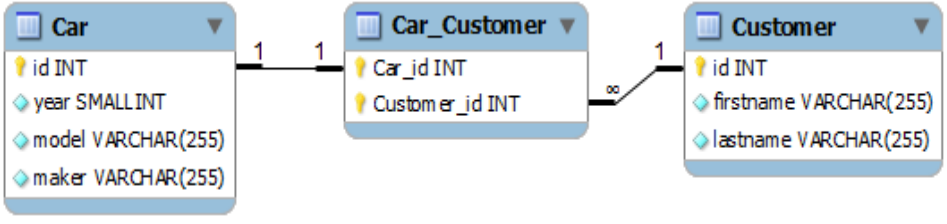
CUSTOMER table

ID	FIRSTNAME	LASTNAME
1	Frank	Brown

Join Table

- ManyToOne can be mapped with a JoinTable
  - Useful for optional (0..1) associations
  - Optional would require the FK to be nullable
    - Normalization does not like nullable columns





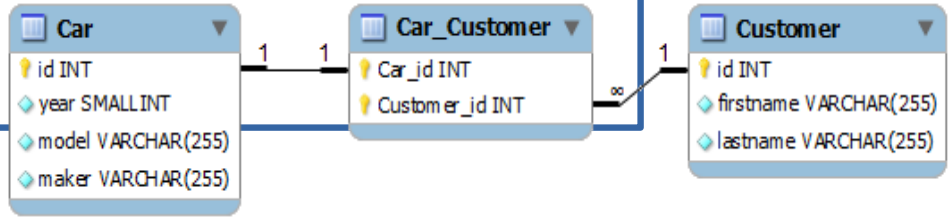
Code

```
@Entity
public class Car {
        @Id
        @GeneratedValue
        private Long id;
        private short year;
        private String model;
        private String maker;
        @ManyToOne
        @JoinTable(name="car_customer"
         joinColumns = { @JoinColumn(name = "customer_id") },
         inverseJoinColumns = { @JoinColumn(name = "cars_id") }
        )
        private Customer owner;

        ...
```

@JoinTable and its name are required

Column names are optional

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

        ...
```



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

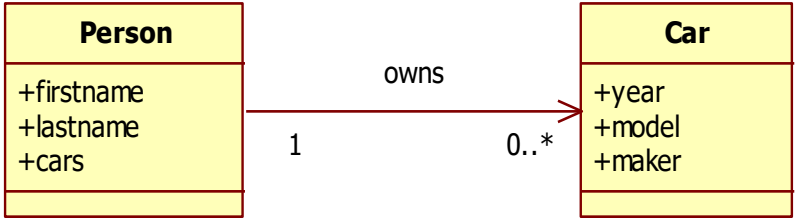
CS544 EA

Hibernate

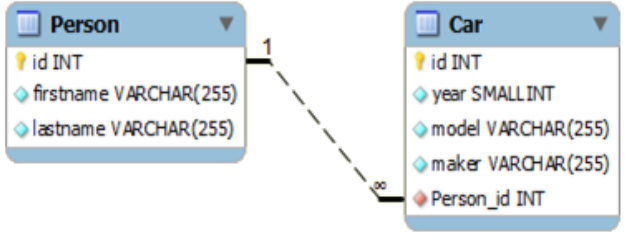
Association: OneToMany

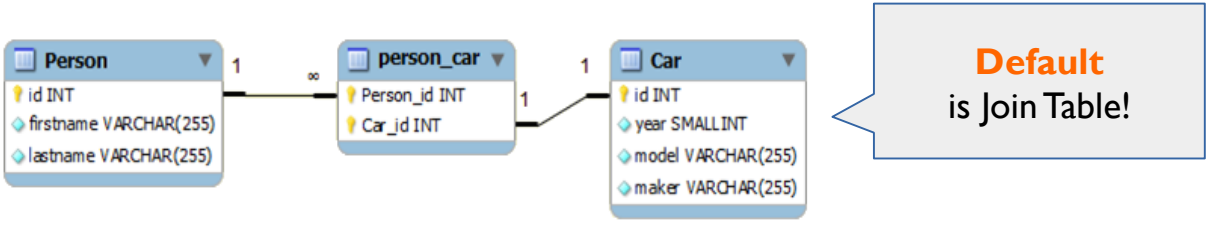
OneToMany

- OO:



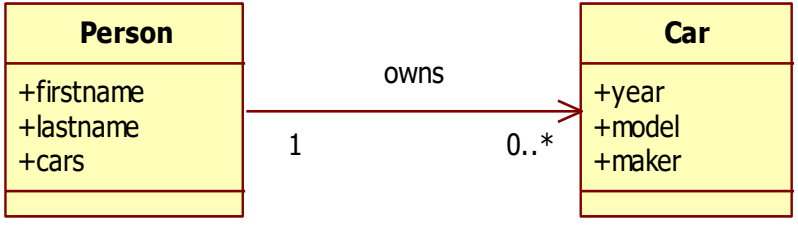
- Relational
  - Again 2 options:

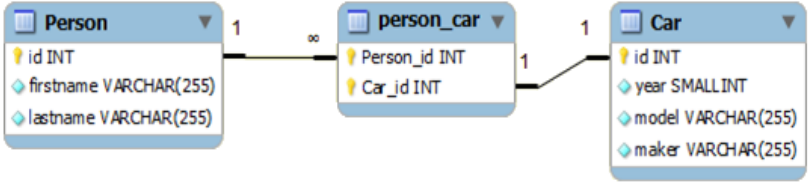




Why Default Join Table?

- **Uni-direct** OneToMany defaults to join table
  - OO: The many side **should not** have a reference
  - Relational: FK (like a ref) is **on many side**
    - Is it possible to have FK on other side?
    - Join Table only solution to this problem!





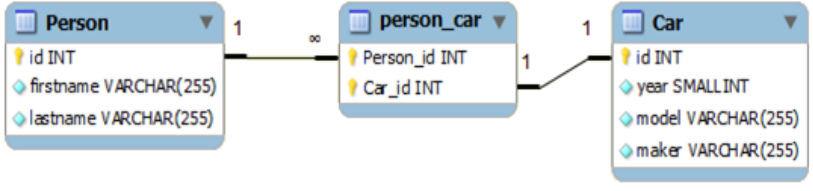
Uni-direct OneToMany

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

        ...
```

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

        ...
```

Makes Join Table!

@JoinTable can be added to change table and column names



PERSON table

ID	FIRSTNAME	LASTNAME
1	Frank	Brown

PERSON_CAR table

PERSON_ID	CAR_ID
1	1
1	2

CAR table

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

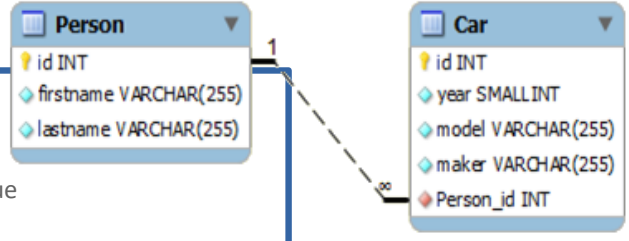
JoinColumn

- Does not match the spirit of Uni-Directional
  - Does work when specified

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

@JoinColum
name defaults to:
cars_id

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

        ...
```



PERSON table

ID	FIRSTNAME	LASTNAME
1	Frank	Brown

CAR table

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

CS544 EA

Hibernate

Association: Bi-Directional

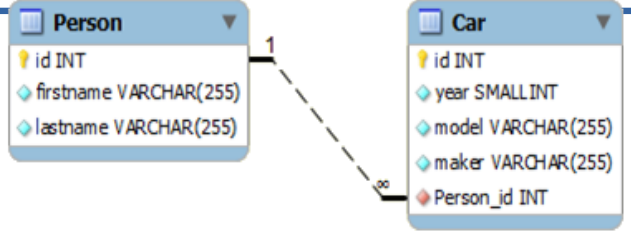
ManyToOne / OneToMany

- Bi-directional OneToMany == ManyToOne
  - Needs owning side → **mappedBy()**

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

        ...
```

Name of property on other side that has the mapping

```
@Entity
public class Car {
        @Id
        @GeneratedValue
        private Long id;
        private short year;
        private String model;
        private String maker;
        @ManyToOne
        private Person person;

        ...
```



Which Side?

- mappedBy() says other side mapped this association
  - Gives up control of the association
  - Says that the **other side is the owning side**

- For Bi-Directional OneToMany / ManyToOne:
  - Only @OneToMany has mappedBy() option

@ManyToOne
cannot say mappedBy()
Even if it wanted to!

Side with the FK,
Natural owner
of the association

Join Table

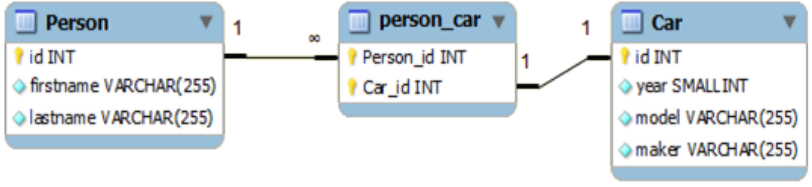
- You can use a Join Table
  - Annotation has to be on @ManyToOne

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

        ...
```

Because of mappedBy() extra mapping annotations throws AnnotationException

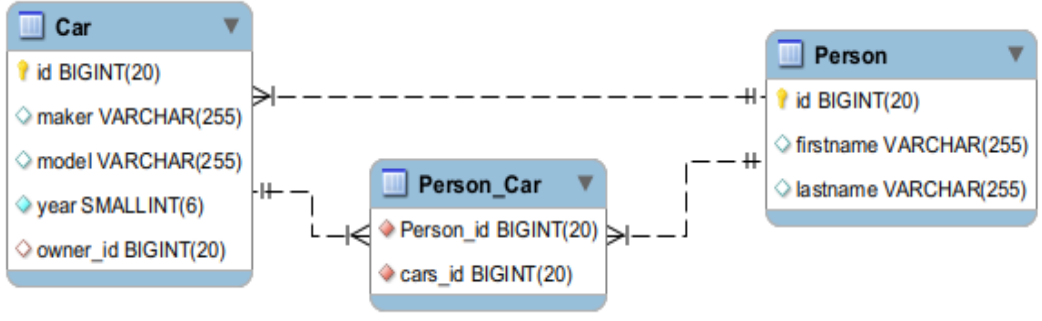
Only on this side do extra mapping annotations work

```
@Entity
public class Car {
        @Id
        @GeneratedValue
        private Long id;
        private short year;
        private String model;
        private String maker;
        @ManyToOne
        @JoinTable(name = "person_car",
                joinColumns = {
                        @JoinColumn(name = "car_id")
                }, inverseJoinColumns = {
                        @JoinColumn(name = "person_id")
                })
        private Person person;

        ...
```



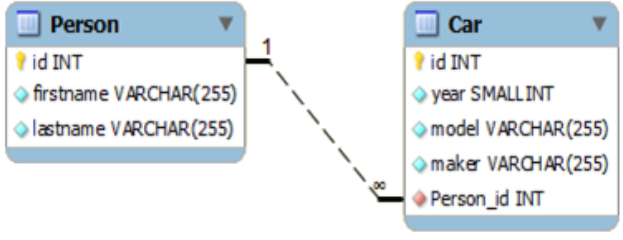
No mappedBy()

- What happens if you forget mappedBy()?
  - 2 uni-directional associations
  - Uni-directional @ManyToOne uses FK
  - Uni-directional @OneToMany uses Join Table



JoinColumn

- What if you forget mappedBy()
  - But specify @JoinColumn on @OneToMany
  - No join table, schema looks okay
  - By default, two FKs is generated



- Both sides will update the one FK
  - Creating a **race condition** (not sure which one wins)
  - Bad programming!

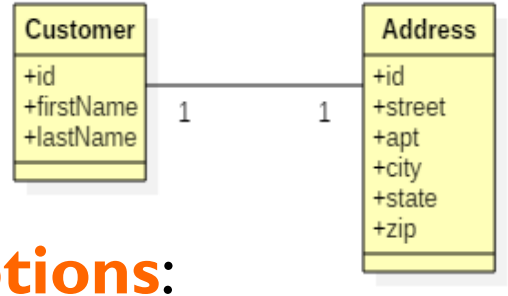
CS544 EA

Hibernate

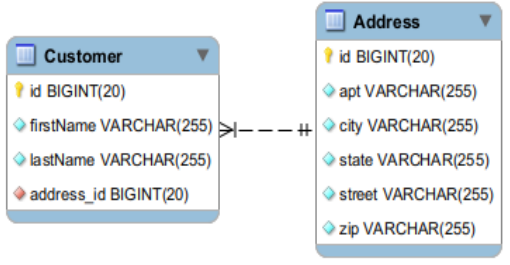
Association: OneToOne

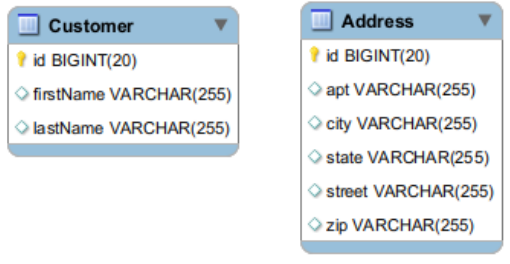
OneToOne

- OO: Customer and Address (if bi-directional) have a reference to each other



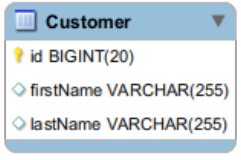
- Relational, **two options**:
  - FK (on one side) with unique constraint
  - Shared Primary Keys

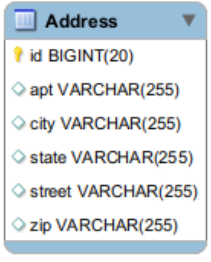




Shared Primary Key

- Shared Primary Key uses the Primary Key as Foreign Key
  - By having the **same value** rows connect





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

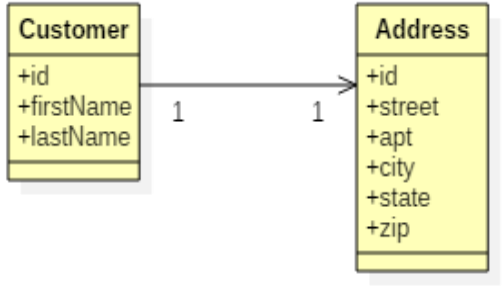
Uni-Directional FK

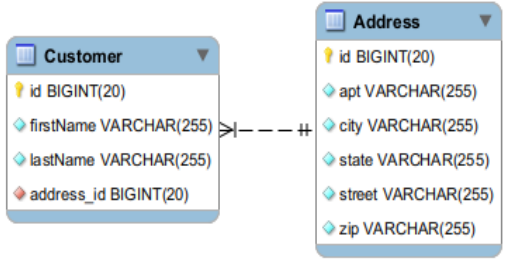
- Uni-directional use a FK
  - On the side that has the reference
  - Best match for spirit of uni-direct

```
@Entity
public class Customer {
        @Id
        @GeneratedValue
        private Long id;
        private String firstName;
        private String lastName;
        @OneToOne
        private Address address;

        ...
```

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

        ...
```

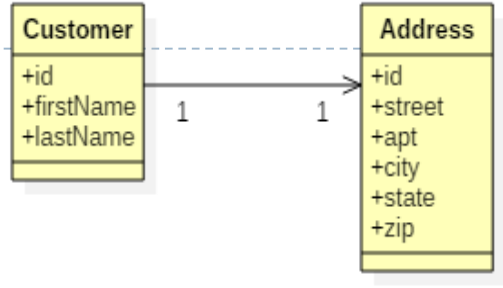
Simply place
@OneToOne
on the association

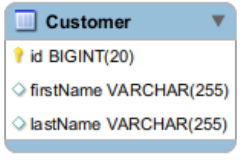


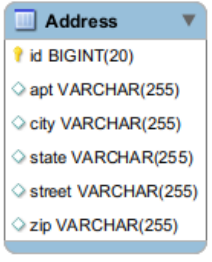


Uni-Directional Shared PK

- Not as 'in the spirit'
  - Works properly if you specify it
  - Remember to **assign the ID** for address!








```
@Entity
public class Customer {
        @Id
        @GeneratedValue
        private Long id;
        private String firstName;
        private String lastName;
        @OneToOne
        @PrimaryKeyJoinColumn
        private Address address;

        ...
```

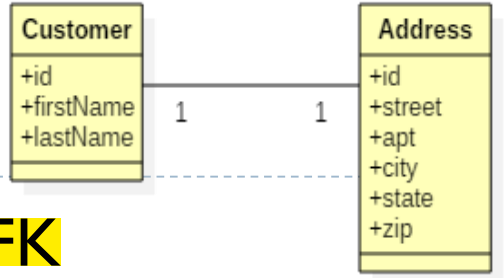
Add **@PrimaryKeyJoinColumn** to the association

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

        ...
```

Cannot generate @Id

The value has to be same as ID value of Customer
Programmer has to set it!

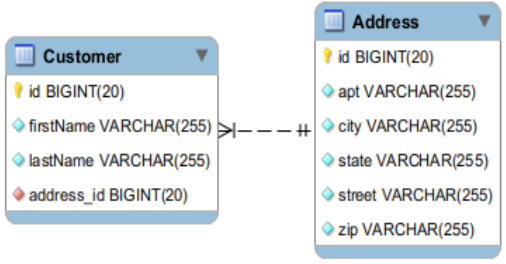
Bi-Directional FK



- A bi-directional associations based on a FK
  - Uses **@OneToOne** on both sides
  - One side has to give up control with **mappedBy()**

```
@Entity
public class Customer {
        @Id
        @GeneratedValue
        private Long id;
        private String firstName;
        private String lastName;
        @OneToOne
        private Address address;

        ...
```

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

        ...
```



From a business perspective Address is less important therefore it gives up ownership (says mappedBy)

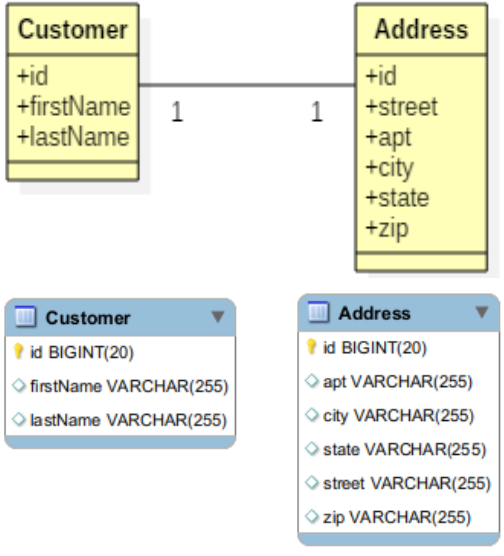
Bi-Directional Shared PK

- The 'owning side' generates the ID
  - Programmer **manually sets value** on the other side

```
@Entity
public class Customer {
        @Id
        @GeneratedValue
        private Long id;
        private String firstName;
        private String lastName;
        @OneToOne
        @PrimaryKeyJoinColumn
        private Address address;
```

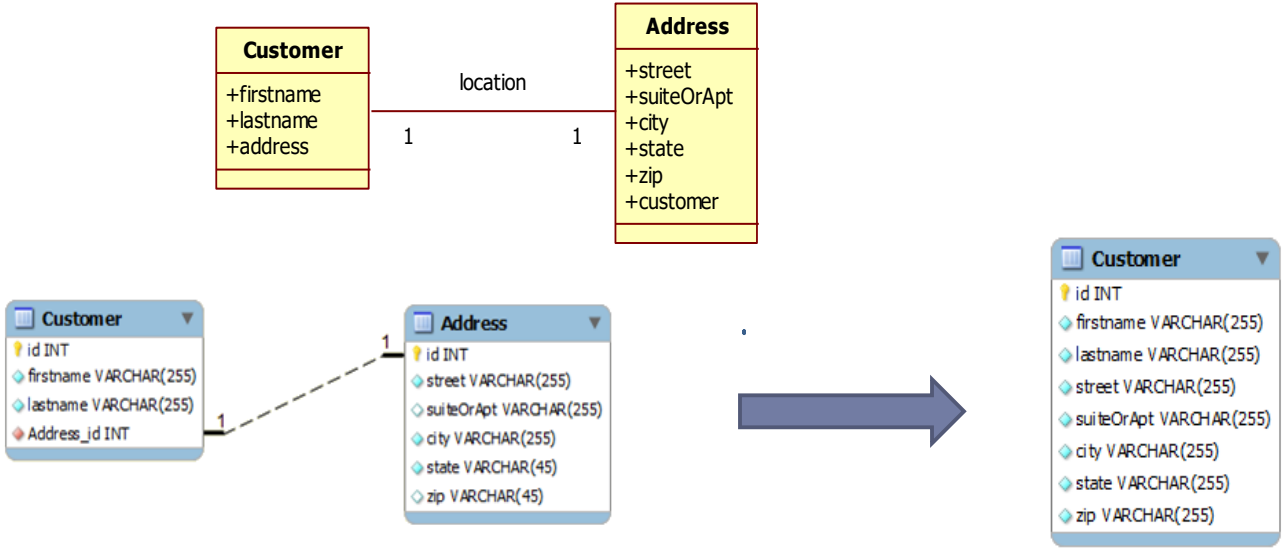
```
@Entity
public class Address {
        @Id
        private Long id;
        private String street;
        private String apt;
        private String city;
        private String state;
        private String zip;
        @OneToOne
        @PrimaryKeyJoinColumn
        private Customer customer;

        ...
```

Both sides specify
@PrimaryKeyJoinColumn
No need for mappedBy



Embedded Classes

- During analysis Consider changing a **@OneToOne to** be an **embedded class**
  - We will discuss embedded in an upcoming lecture



CS544 EA

Hibernate

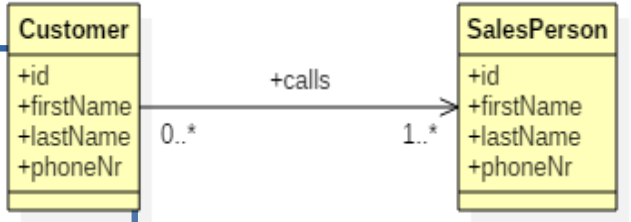
Association: ManyToMany

Uni Directional ManyToMany

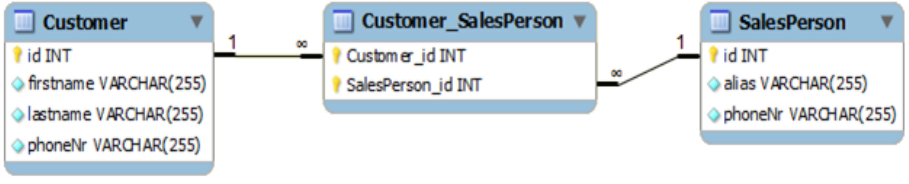
- **@ManyToMany** associations can only be implemented with a JoinTable

```
@Entity
public class Customer {
        @Id
        @GeneratedValue
        private Long id;
        private String firstName;
        private String lastName;
        @ManyToMany
        private List<SalesPerson> salesPeople
                = new ArrayList<>();

        ...
```

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

        ...
```



Optionally add @JoinTable to specify name of table and / or column names

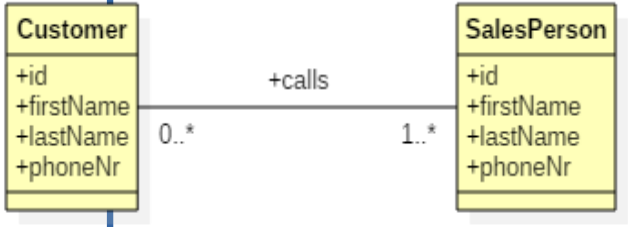


Bi-Directional ManyToMany

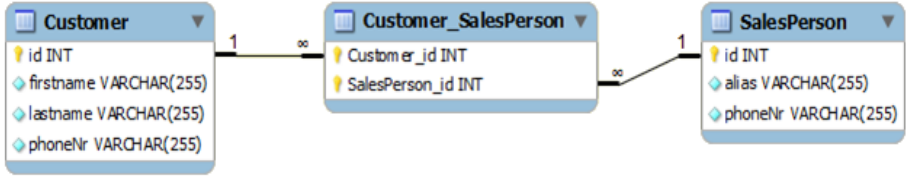
- **Choose** which side specifies **mappedBy**
  - Business may find one side more important (owner)

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

        ...
```

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

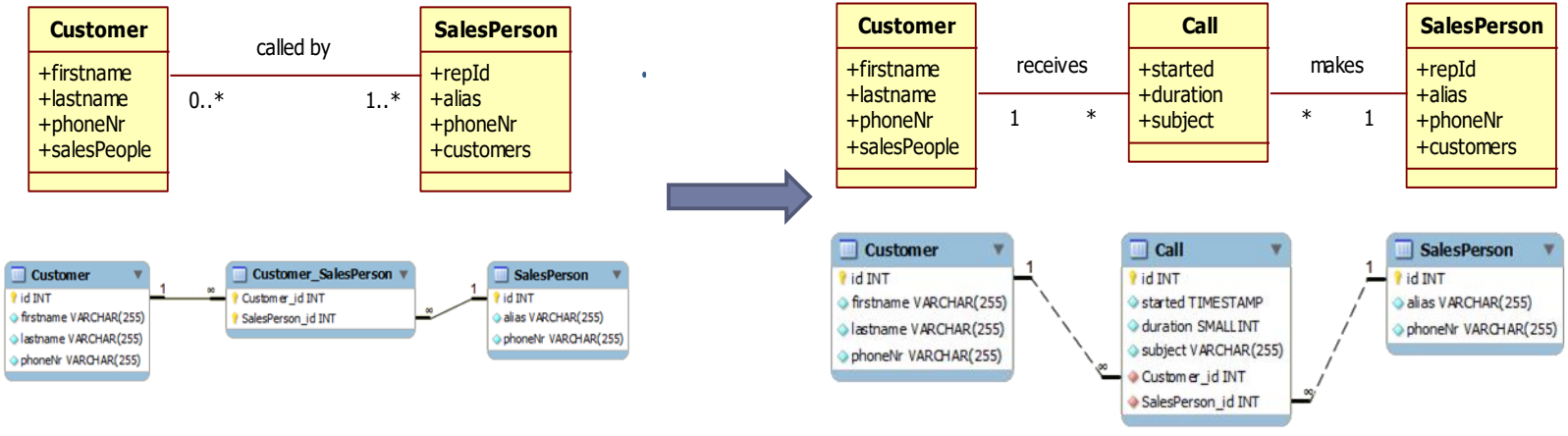


Both specify @ManyToMany
The owning side can also
specify @JoinTable to give names



Reconsider

- During Domain Analysis **consider changing** ManyToMany



- ManyToMany are often interesting connections
  - Maybe you want to keep data on how / what connected
  - Turn JoinTable into an entity

CS544 EA

Hibernate

Cascades

Cascades

- An operation cascades if it **follows references**
- By default non of the operations cascade:
  - em.persist(person) will not persist its car objects
  - em.merge(person) will not re-connect its car objects
  - em.remove(person) will not remove its car objects

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

        ...
```

```
@Entity
public class Car {
        @Id
        @GeneratedValue
        private Long id;
        private short year;
        private String model;
        private String maker;
        @ManyToOne
        private Customer owner;

        ...
```

CascadeType

- You can specify **which operations** cascade
  - Every association has the cascade option

```
@Entity
public class Customer {
        @Id
        @GeneratedValue
        private Long id;
        private String firstName;
        private String lastName;
        @ManyToMany(mappedBy="customers", cascade= {CascadeType.MERGE, CascadeType.PERSIST})
        private List<SalesPerson> salesPeople = new ArrayList<>();
        @OneToOne(cascade=CascadeType.ALL)
        private Address address;
```

Can be specified as a list

Or as a single value

Persisting a Customer now automatically also persists all linke SalesPerson and Address Objects

CascadeTypes
ALL
DETACH
MERGE
PERSIST
REFRESH
REMOVE

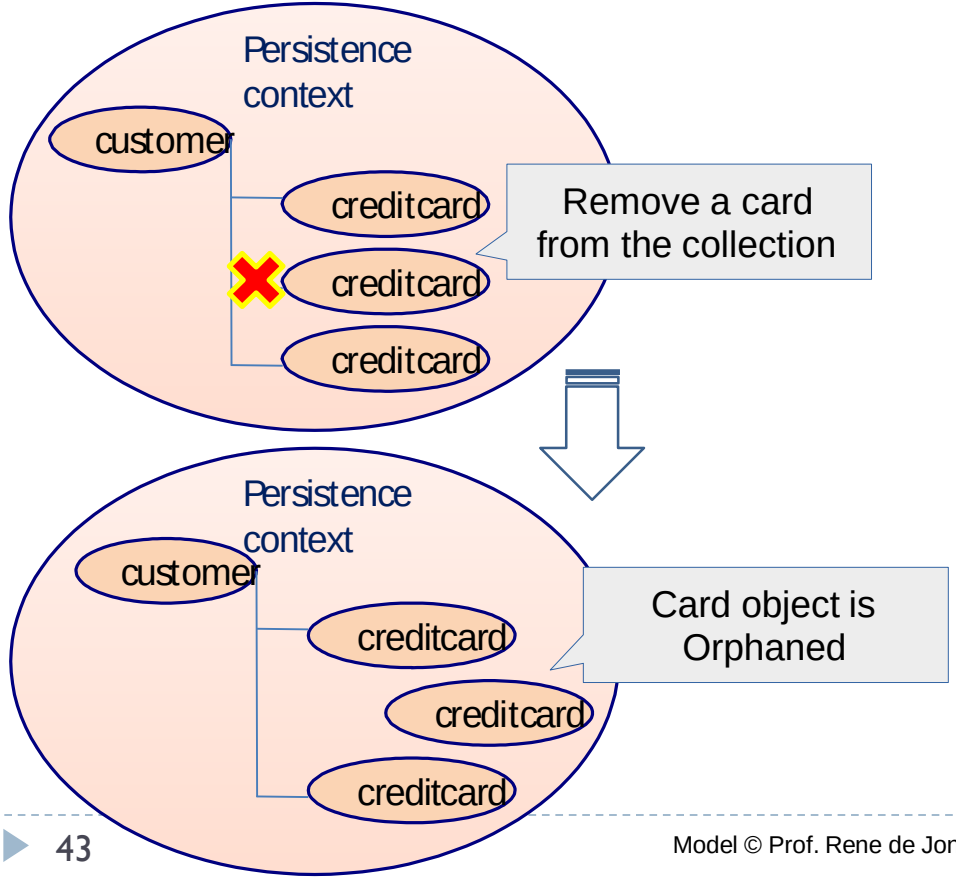
Orphan Removal

- Orphan removal is a topic related to cascades
  - Option on @OneToMany and @OneToOne
    - Both for Uni-directional and Bi-directional
  - When the connection / **reference is broken**, the entity that was referred to is **automatically removed**

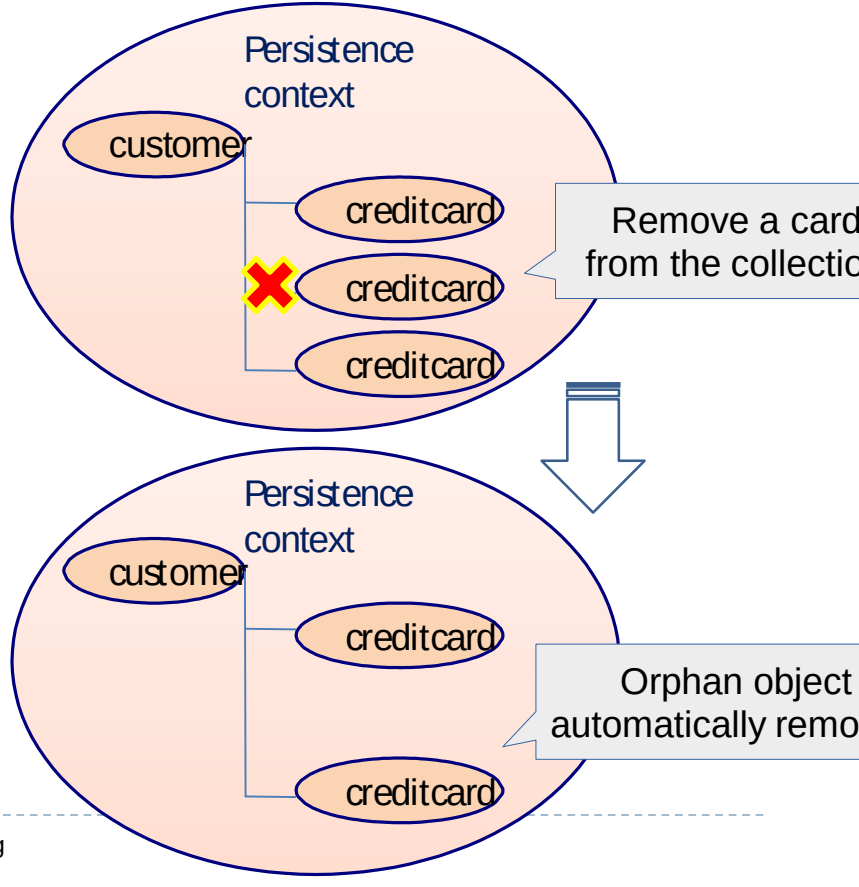
```
@Entity
public class Customer {
        @Id
        @GeneratedValue
        private Long id;
        private String firstname;
        private String lastname;
        @OneToMany(mappedBy="owner", orphanRemoval=true)
        private List<CreditCard> cards =
                new ArrayList<>();

        ...
```

Orphan Removal

One to Many without Orphan Removal



One to Many using Orphan Removal



Summary

- There are 7 types of associations
  - Bi-Directional associations need an owning side
  - Use mappedBy to give up control (not be owner)
- Mapping choices:
  - JoinTable of JoinColumn (OneToMany/ManyToOne)
  - Shared PK or FK (OneToOne)
- Cascades:
  - Allowing operations to follow references
- How connections are made is as important as the parts themselves The whole is greater than the sum of the parts

CS544 EA

Hibernate

Collections

Collections

- So far we've only used java.util.List for collections
  - Other options are available
- Very important to P2I
  - Hibernate provides its **own implementations**
  - Do not create get / set methods for a collection!

```
public List<Car> getCars() {
        return cars;
}

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

.setCars() can be used to **overwrite** Hibernate's collection **Impelentation**!

```
public boolean addCar(Car car) {
        if (cars.add(car)) {
                car.setOwner(this);
                return true;
        }
        return false;
}

public boolean removeCar(Car car) {
        if (cars.remove(car)) {
                car.setOwner(null);
                return true;
        }
        return false;
}
```

4 Types of Collections

- Collections Types:
  - **Bag** (collection without restrictions)
  - **Set** (Bag without duplicates)
  - **List** (Bag with an given arbitrary order)
  - **Map** (Set of keys to a bag of values)

- JPA defaults java.util.List to be a Bag!
  - Every collection so far was **mapped as a Bag**

2 Types

- **Non-Indexed** Collections:
  - Don't need anything extra
  - Collections: **Bag** and **Set**

- **Indexed** Collection:
  - Implemented with a DB index lookup
  - Collections: **Map** and **List**

@OrderBy

- Hibernate's **Bag**, **Set**, and **Map** can be ordered
  - Based on a property of the elements

```
@Entity
public class ToolBox {
        @Id
        @GeneratedValue
        private Long id;
        private String model;
        private String manufacturer;
        @OneToMany(mappedBy="owner")
        @OrderBy("size ASC")
        private Set<Tool> tools
                = new HashSet<>();
```

@OrderBy annotation

```
@Entity
public class Tool {
 @Id
 @GeneratedValue
 private int id;
 private String type;
 private String size;
 @ManyToOne
 private ToolBox toolbox;
```

Tools in the Toolbox are sorted based on this size in ascending order

- List cannot (be re-ordered)
  - It is always ordered based on its order column

CS544 EA

Hibernate

Collection: Bag

Bag

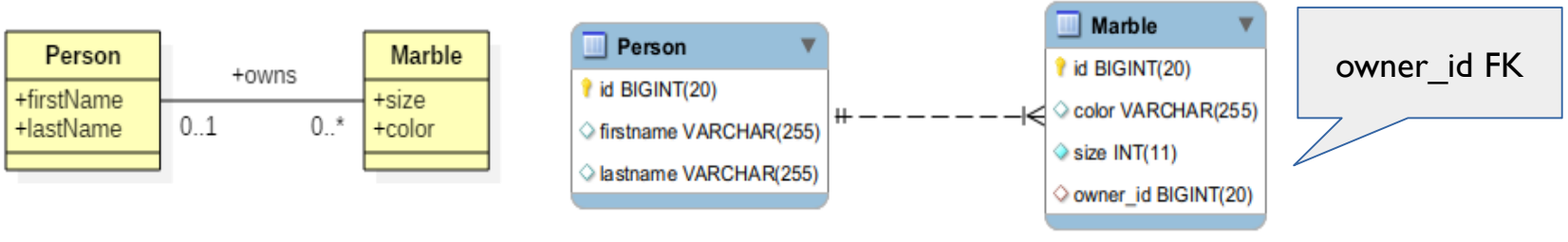
- The **most basic** collection is a bag
  - It is allowed to contain duplicates
  - It has no built-in order (can use @OrderBy)

- Like a bag of marbles
  - May **contain duplicates**
  - **No order**, although can be ordered

Bag Implementation

- **java.util.Collection** is a bag interface
- JPA also treats **java.util.List** as a bag
  - Java has no official bag implementation

- Bags are **non-indexed** collections
  - DB can implement it using a FK, no additional index



Code for Bag

- java.util.**Collection** maps as **Bag**

```
@Entity
public class Person {
        @Id
        @GeneratedValue
        private Long id;
        private String firstname;
        private String lastname;
        @OneToMany(mappedBy="owner")
        private Collection<Marble> marbles
                = new ArrayList<>();

        ...
```

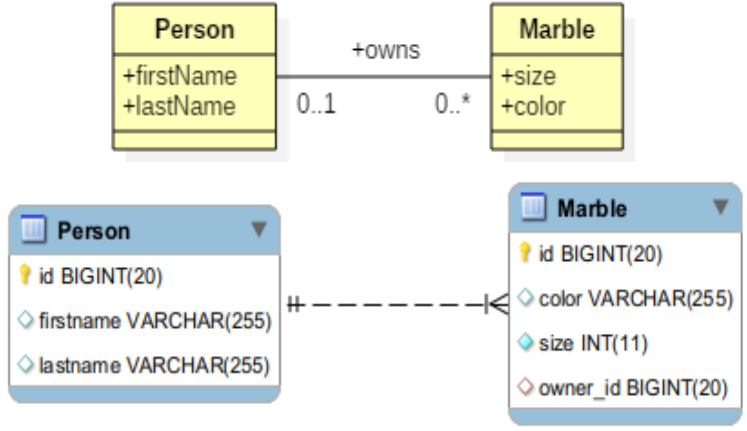
Collection maps as a Bag

Used ArrayList since no official Bag implementation.

Remember: Hibernate will replace with its own implementation

```
@Entity
public class Marble {
        @Id
        @GeneratedValue
        private Long id;
        private int size;
        private String color;
        @ManyToOne
        private Person owner;
```

Made this association bi-directional
Uni-directional also works



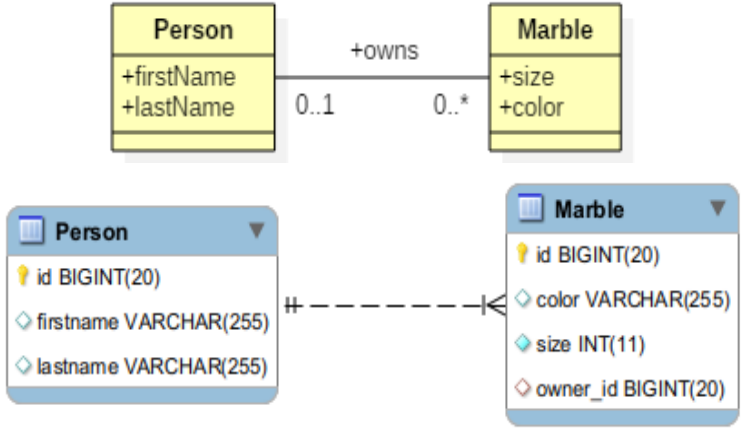
Code for Bag

- java.util.**List** maps as **Bag**

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

        ...
```

List also maps as a Bag

```
@Entity
public class Marble {
        @Id
        @GeneratedValue
        private Long id;
        private int size;
        private String color;
        @ManyToOne
        private Person owner;
```



CS544 EA

Hibernate

Collection: Set

Set

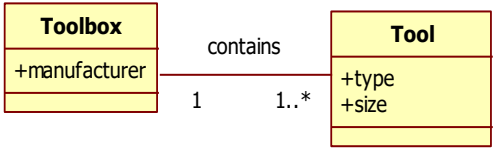
- Sets are bags that cannot contain duplicates
  - A set has **no built-in order** (can @OrderBy)
  - A set **cannot contain duplicates**

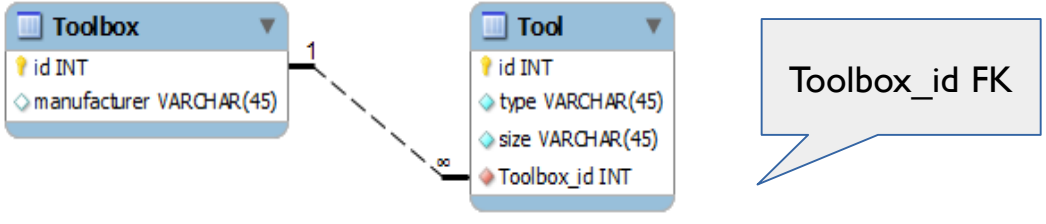
- Store bought Set of Tools:
  - No duplicates
  - No characteristic for order

Set Implementation

- Java has the **java.util.Set** interface
  - **java.util.HashSet** is a general implementation

- Like bags, sets are **non-indexed** collections
  - DB can implement it using FK, no extra index





Equals & HashCode

- Elements **in a Set** need to implement **.equals()** and **.hashcode()**
  - Otherwise Java cannot check uniqueness

```
@Override
public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + id;
        result = prime * result + ((size == null) ? 0 : size.hashCode());
        result = prime * result + ((type == null) ? 0 : type.hashCode());
        return result;
}

@Override
public boolean equals(Object obj) {
        if (this == obj)
                return true;
        if (obj == null)
                return false;
        if (getClass() != obj.getClass())
                return false;
        Tool other = (Tool) obj;
        if (id != other.id)
                return false;
        if (size == null) {
                if (other.size != null)
                        return false;
        } else if (!size.equals(other.size))
                return false;
        if (type == null) {
                if (other.type != null)
                        return false;
        } else if (!type.equals(other.type))
                return false;
        return true;
}
```

IDE can generate these

Code for Set

- java.util.**Set** maps as a Set

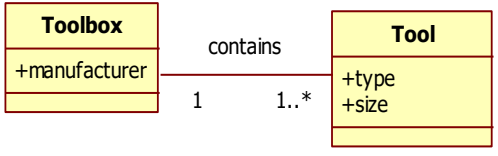
```
@Entity
public class ToolBox {
        @Id
        @GeneratedValue
        private Long id;
        private String manufacturer;
        @OneToMany(mappedBy="toolbox")
        private Set<Tool> tools
                = new HashSet<>();
```

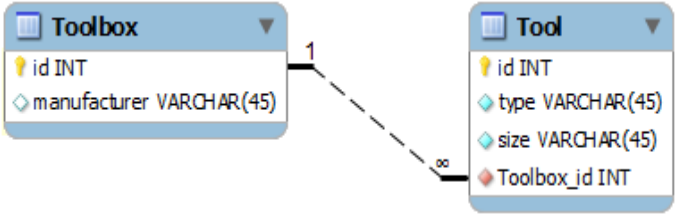
Set maps as a Set

HashSet is good implementation to start with…

Will be replaced!

```
@Entity
public class Tool {
 @Id
 @GeneratedValue
 private int id;
 private String type;
 private String size;
 @ManyToOne
 private ToolBox toolbox;
```





CS544 EA

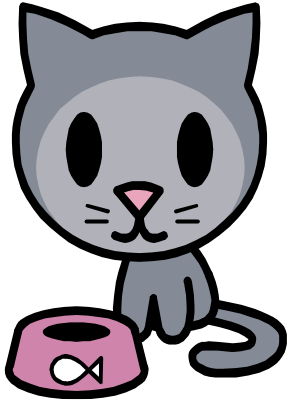
Hibernate

Collection: Map

Map

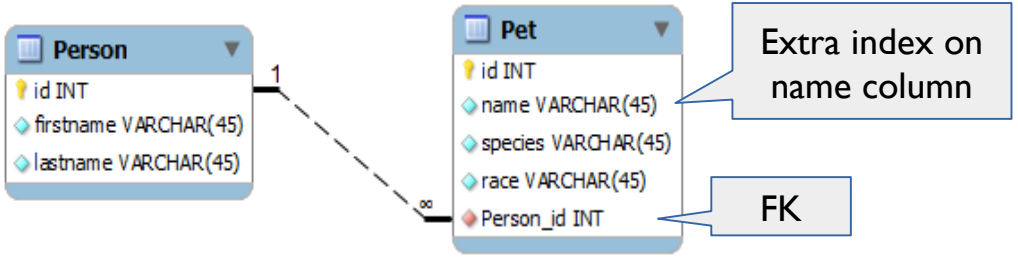
- A map 'maps' a set of Keys to a bag of values:
  - **Unique keys that lead to values** (not unique)
  - Given the key, map can quickly get value
  - No built-in order in keys or values (can use @OrderBy)

- Pet ownership can be modeled as a map
  - Each pet has a unique name*
  - To find a pet you use its name
  - No specific order in names or pets



Map Implementation

- Java has the **java.util.Map** interface
  - **java.util.HashMap** is a common implementation
- Maps are indexed collections
  - Need a **FK** and a **index** on another column
  - Can be column of entity, or additional column



Code for Map (no additional column)

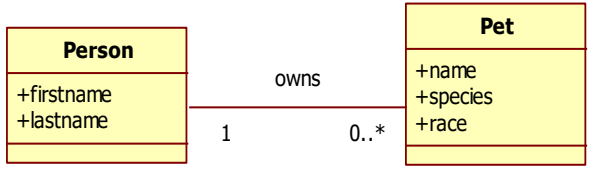
- **@MapKey** if the key is already part of the entity

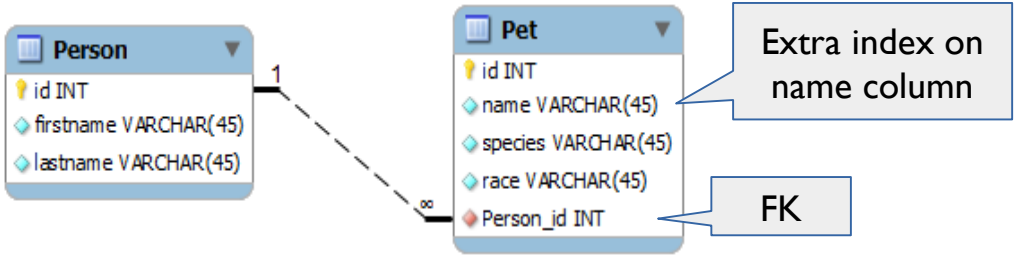
```
@Entity
public class Person {
        @Id
        @GeneratedValue
        private Long id;
        private String firstname;
        private String lastname;
        @OneToMany(mappedBy="owner")
        @MapKey(name="name")
        private Map<String, Pet> pets
                = new HashMap<>();
```

Specify property on other side

```
@Entity
public class Pet {
        @Id
        @GeneratedValue
        private Long id;
        private String species;
        private String race;
        private String name;
        @ManyToOne
        private Person owner;
```

Property used as key





Additional Column & Bi-Directional

- When a collection needs an **additional column**
  - It **needs to be the owning side** of the association
  - Only it knows the value that needs to be inserted

- Bi-directional: collection side **defaults to not being owner**!
  - Side with collection normally gives up control with mappedBy
  - Other side does not even have mappedBy option

Uni-Direct & Additional Column

- **@MapKeyColumn** no problem for uni-directional

```
@Entity
public class Person {
        @Id
        @GeneratedValue
        private Long id;
        private String firstname;
        private String lastname;
        @OneToMany
        @JoinColumn(name="Person_id")
        @MapKeyColumn(name="name")
        private Map<String, Pet> pets
                = new HashMap<>();
```

No mappedBy

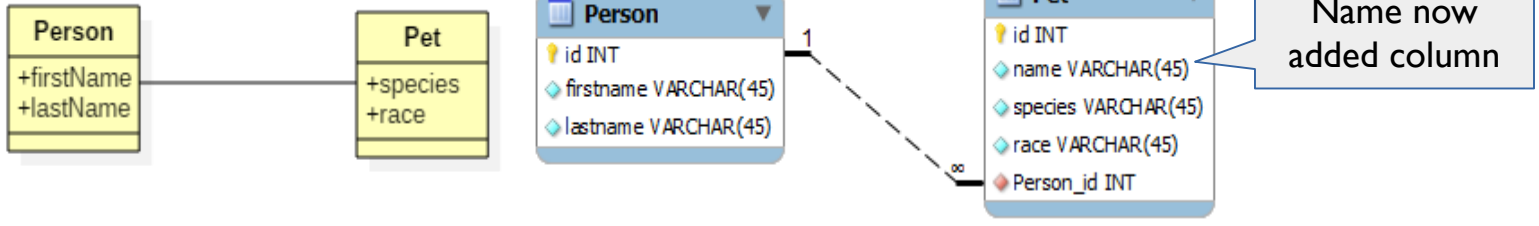
@JoinColumn to avoid join table!

@MaKeyColumn Specifies name of (indexed) column to add

```
@Entity
public class Pet {
        @Id
        @GeneratedValue
        private Long id;
        private String species;
        private String race;
```

No name property

No reference back (uni)



Bi-Direct @MapKeyColumn

- MappedBy emulation on @JoinColumn for @ManyToOne
  - **insertable=false, updatable=false**

```
@Entity
public class Person {
        @Id
        @GeneratedValue
        private Long id;
        private String firstname;
        private String lastname;
        @OneToMany
        @JoinColumn(name="Person_id")
        @MapKeyColumn(name="name")
        private Map<String, Pet> pets
                = new HashMap<>();
```

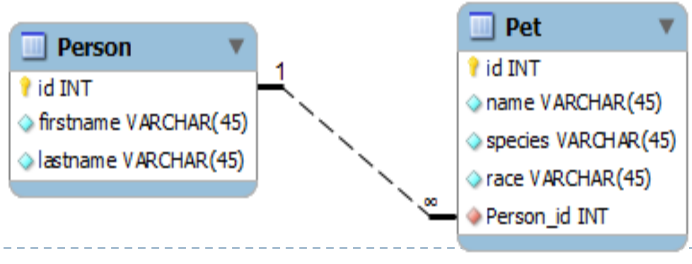
```
@Entity
public class Pet {
        @Id
        @GeneratedValue
        private Long id;
        private String species;
        private String race;
        private String name;
        @ManyToOne
        @JoinColumn(name="Person_id", insertable=false, updatable=false)
        private Person owner;
```

Gives up control without mappedBy

Same as unidirectional

Both map to same join column.

Avoids race condition by using insertable=false, updatable=false



CS544 EA

Hibernate

Collection: List

List

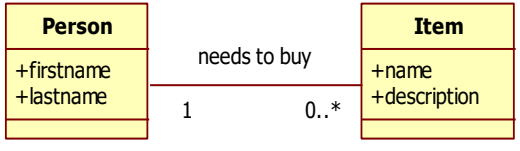
- Bag with the ability to keep an arbitrary order
  - **Built-in order** (not based on properties): no @OrderBy
  - List can **contain duplicates**

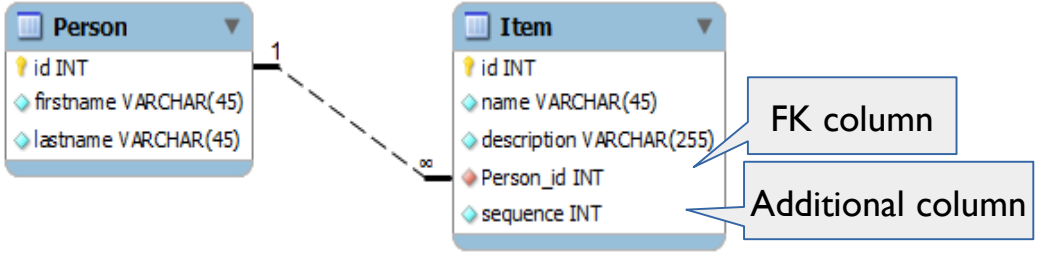
- A shopping list is a typical example
  - Built-in although often arbitrary order
  - May contain duplicates

List Implementation

- Java has the java.util.List interface
  - Java.util.ArrayList most common implementation

- Lists are indexed collections
  - Needs FK and an **additional indexed sequence column**
  - Same problem with Bi-Direct as additional column for map!





Code List Uni-Direct

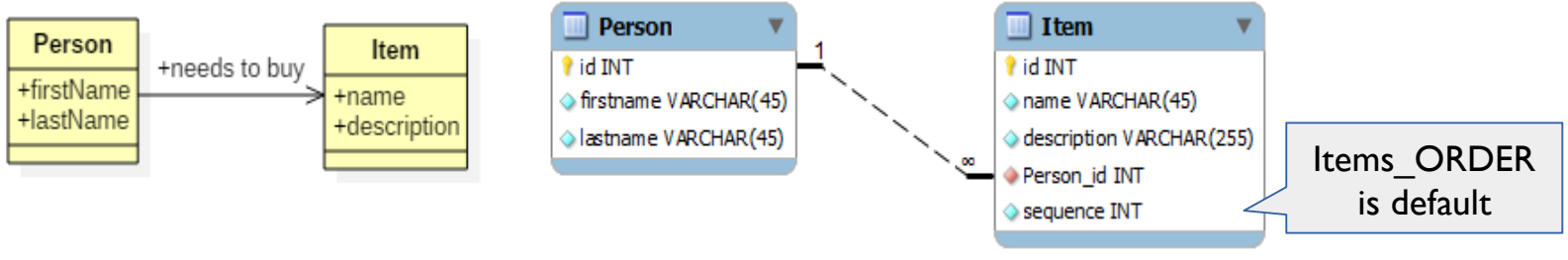
- **@OrderColumn** for additional indexed column
  - Is what makes it a 'real' list instead of bag

```
@Entity
public class Person {
        @Id
        @GeneratedValue
        private Long id;
        private String firstname;
        private String lastname;
        @OneToMany
        @JoinColumn(name="Person_id")
        @OrderColumn(name="sequence")
        private List<Item> items
                = new ArrayList<>();
```

@JoinColumn to avoid join table!

Optional name for order column

```
@Entity
public class Item {
        @Id
        @GeneratedValue
        private Long id;
        private String name;
        private String description;
```



Code List Bi-Direct

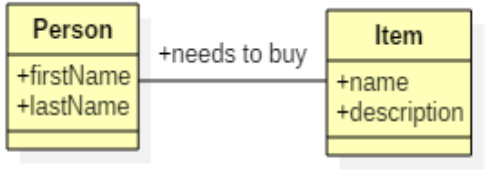
- **Emulation of mappedBy**: insertable, updateable

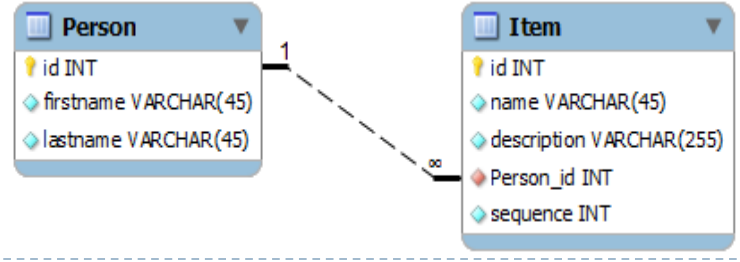
```java
@Entity
public class Person {
        @Id
        @GeneratedValue
        private Long id;
        private String firstname;
        private String lastname;
        @OneToMany
        @JoinColumn(name="Person_id")
        @OrderColumn(name="sequence")
        private List<Item> items
                = new ArrayList<>();
```

Same as uni-direct

```java
@Entity
public class Item {
        @Id
        @GeneratedValue
        private Long id;
        private String name;
        private String description;
        @ManyToOne
        @JoinColumn(name="Person_id", insertable=false, updatable=false)
        private Person person;
```

Using @JoinColumn insertable, updatable to give up control of association





CS544 EA

Hibernate

Element Collections

Element Collections

- To map **collections of primitive values**
  - Or of embeddable classes (discussed later)

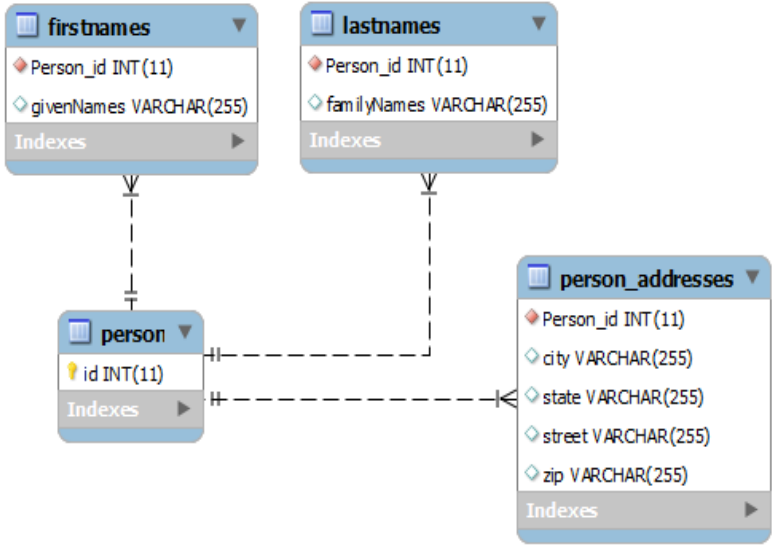
- Does not make sense from OO perspective
  - Still good to know about

@ElementCollection

```
@Entity
public class Person {
  @Id @GeneratedValue
  private int id;
  @ElementCollection
  @CollectionTable(name = "firstNames")
  private List<String> givenNames = new ArrayList<>();
  @ElementCollection
  @CollectionTable(name = "lastNames")
  private List<String> familyNames = new ArrayList<>();
  @ElementCollection
  private List<Address> addresses = new ArrayList<>();

  ...
```

Optional @CollectionTable: to specify table name

Default table name is: Classname_propertyname



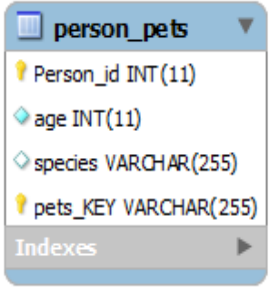
Map

- Maps need a key column
  - Here Pet is a @Embeddable class (more on this later)

```
@Entity
public class Person {
 @Id @GeneratedValue
 private int id;
 private String name;
 @ElementCollection
 @MapKeyColumn(name = "name")
 private Map<String, Pet> Pets = new HashMap<>();
 ...
```

Optional: specify the column name. Default name is: propertyname_KEY

```
@Embeddable
public class Pet {
        private int age;
        private String species;
        ...
```



Summary

- 4 Types of collections
  - Bag: duplicates, no order, can @OrderBy
  - Set: no duplicates, no order, can @OrderBy
  - Map: Set of Keys to a Bag of values
    - @MapKey or additional @MapKeyColumn (Bi-Direct diff)
  - List: duplicates, built-in order, no @OrderBy
    - Always additional column, makes Bi-Direct different

Main Point

1. A good ORM provides features that allow the developer to easily traverse object relationships.
2. ***Science of Consciousness:*** *When we practice the TM Technique, we tap an inner reserve of energy and intelligence that allows us to easily and flexibly manage the diverse activities of every day life*