

CS544

Enterprise Application Architecture

Lesson 4 – JPA Mapping – Part 1

Frameworks and Best Practices Used in Designing Large-Scale Software Systems

Payman Salek, M.S.

Original Material: Prof. Rene de Jong – July 2022

© 2022 Maharishi International University



MAPPING DATA TYPES

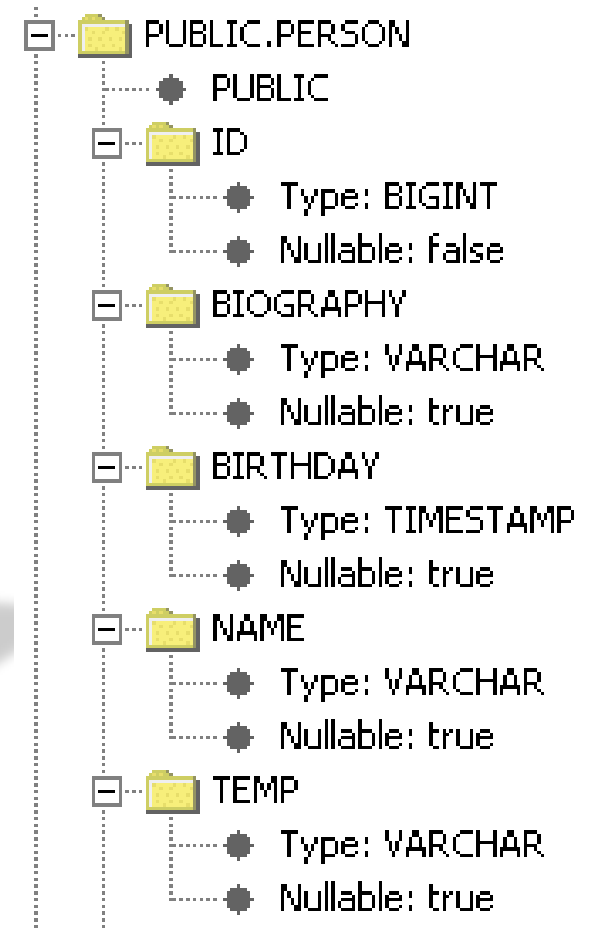
Annotation Types

- Use @Column to specify more details
- Use @Lob to indicate Large values
- Use @Transient to indicate that a property should ***not*** be persisted

Default mapping

```
@Entity
public class Person {
    @Id
    @GeneratedValue
    private long id;
    private String name;
    private Date birthday;
    private String biography;
    private String temp;

    ...
}
```



Specify different mapping

```
@Entity
public class Person {
    @Id
    @GeneratedValue
    private long id;
    @Column(name="FULLNAME", length=255, nullable=false)
    private String name;
    @Temporal(TemporalType.DATE)
    private Date birthday;
    @Lob
    private String biography;
    @Transient
    private String temp;

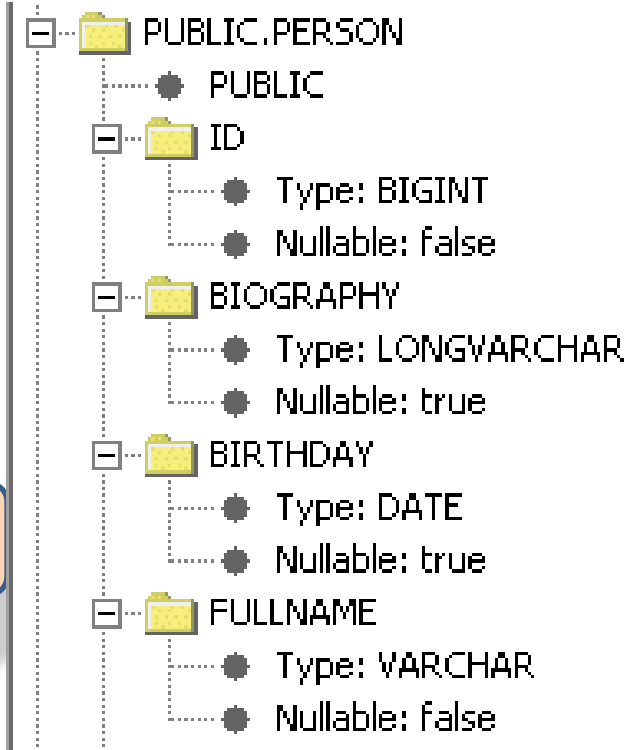
    ...
}
```

Name will be stored as:
FULLNAME VARCHAR(255) NOT NULL

Birthday will be
stored as a Date

Biography will be stored as CLOB
instead of VARCHAR

Temp will not be stored in the database



Property or Field Access

- JPA can access objects in two ways
 - property access gets and sets object values through getter /setter methods
 - field access gets and sets object values directly from / to the fields

```
@Entity
public class Person {
    @Id
    @GeneratedValue
    private long id;
    private String name;
    ...
}
```

JPA field access

```
@Entity
public class Person {
    private long id;
    private String name;

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

    @Id
    @GeneratedValue
    public long getId() { return id; }
    private void setId(long id) { this.id = id; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
}
```

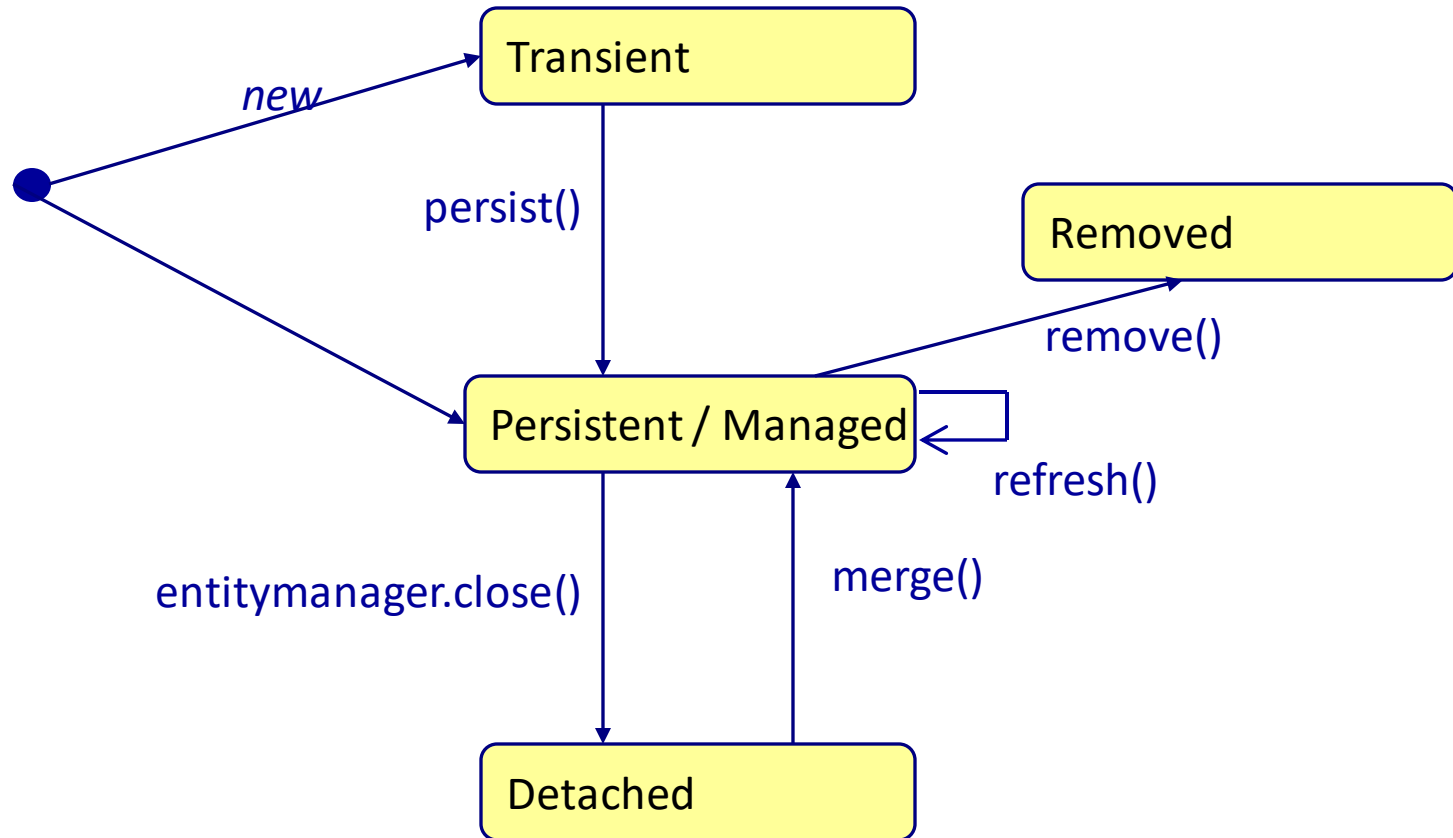
JPA property access

Specifying Access with Annotations

- The JPA specification lets you set the Access Type with the location of `@Id`
 - Placing `@Id` on a field specifies field access for the entire object
 - All other annotations should be on the fields
 - Placing `@Id` on a getter specifies property access for the entire object
 - All other annotations should be on the getters

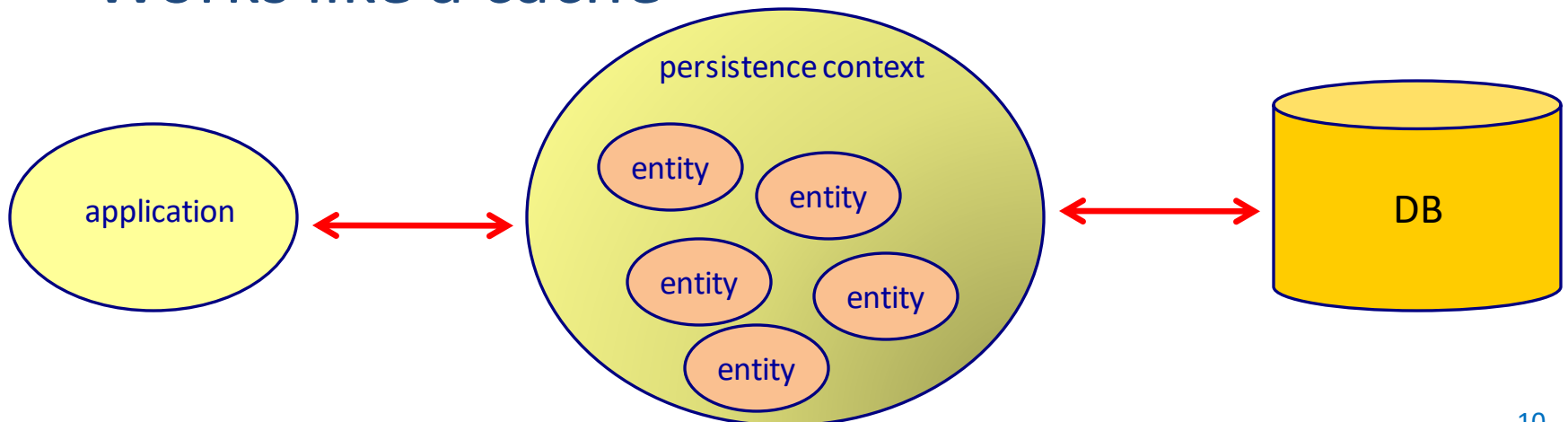
ENTITY OBJECT LIFECYCLE

JPA lifecycle of an entity



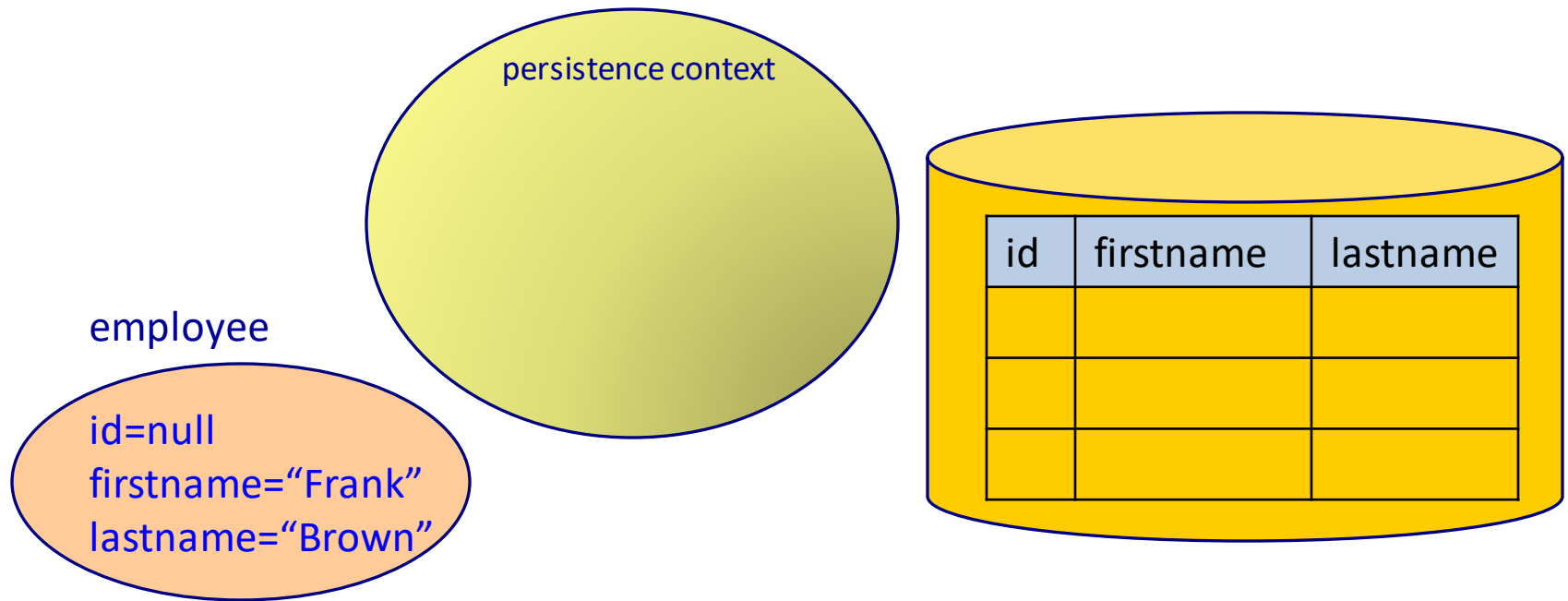
Persistence context

- Manages the entities
- Guarantees that managed entities will be saved in the database
- Tracks changes until they are pushed to the database
- Works like a cache



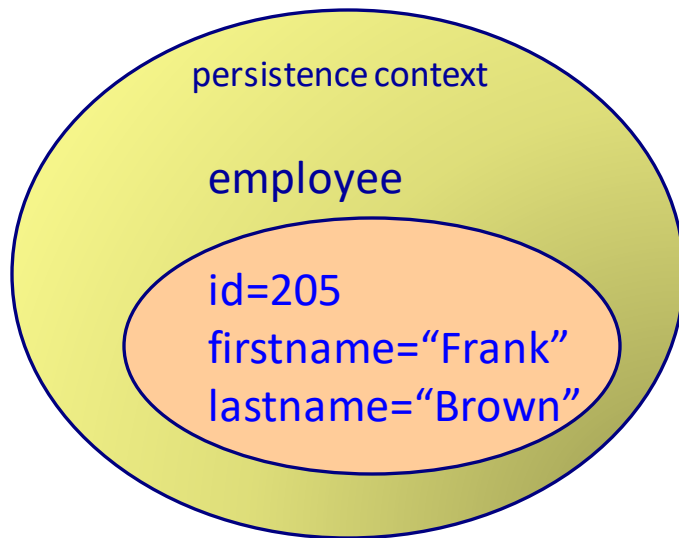
Transient entity

- A transient entity has no database identity



Managed entity

- A managed entity is managed by the persistence context and has a database identity

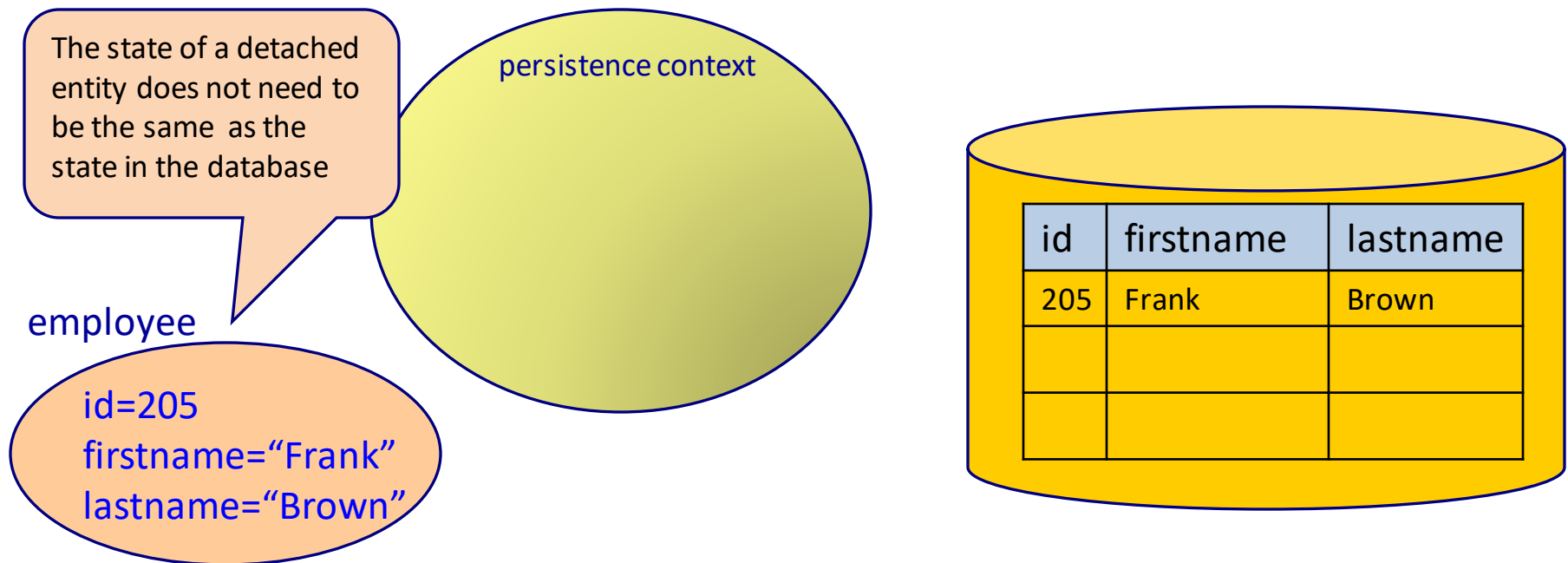


The diagram shows a yellow cylinder representing a database. Inside the cylinder is a table with three columns: "id", "firstname", and "lastname". The first row contains the values "205", "Frank", and "Brown". The second and third rows are empty.

id	firstname	lastname
205	Frank	Brown

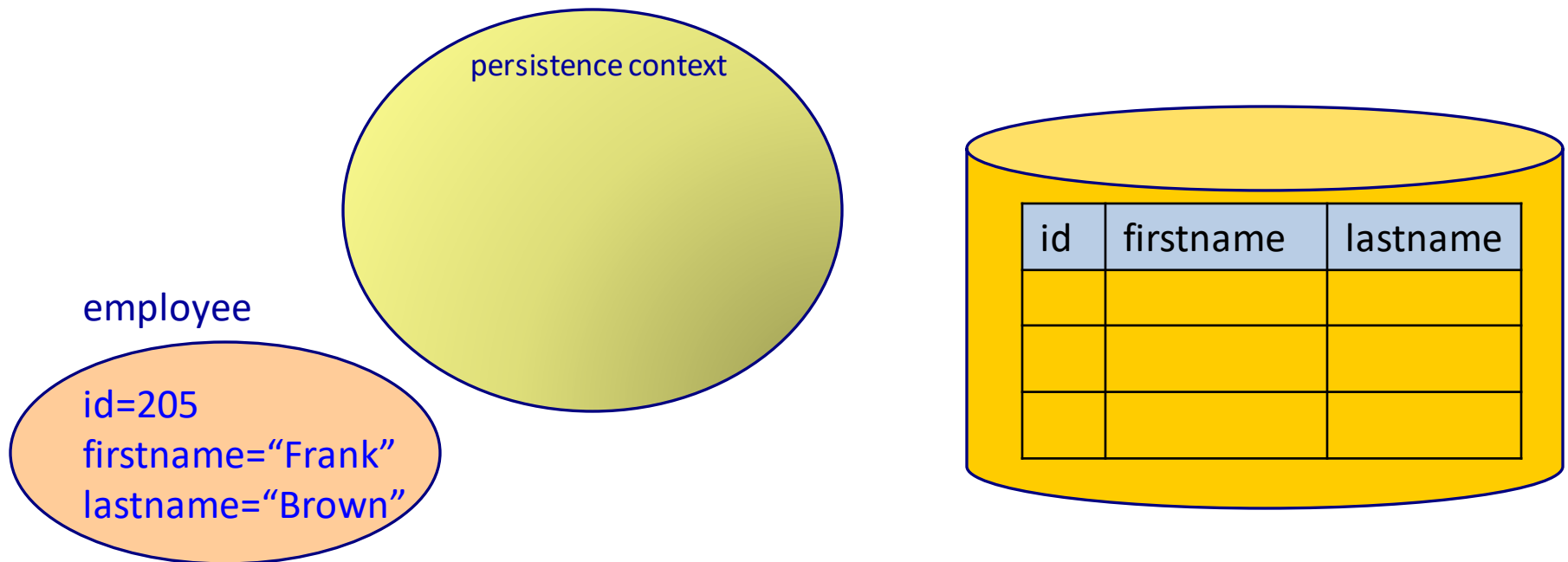
Detached entity

- A detached entity has a database identity, but is not managed by the current persistence context



Removed entity

- With a removed entity is the corresponding data removed from the database.



Association Mapping

ASSOCIATION MAPPING

Association Mapping

- In Java associations are made with object references

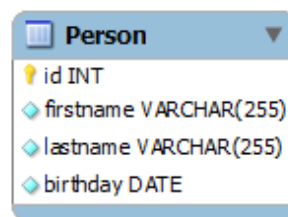
Person has a cars collection of references

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

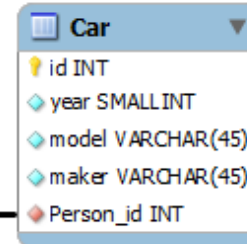
```
public class Car {  
    private int id;  
    private short year;  
    private String model;  
    private String maker;  
    private Person owner;  
    ...  
}
```

Car also has an owner reference back to its owner

- In a relational schema associations are made with Foreign keys



1



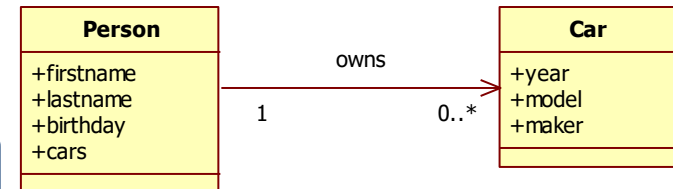
∞

Car has a foreign key to Person

- 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

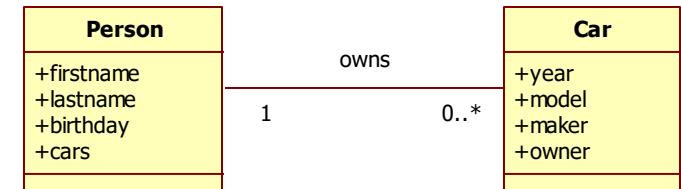
Person has a
collection of
references to
Car objects

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

```
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<Car>();
    ...
}
```

```
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

MANY TO ONE ASSOCIATIONS

Uni-Directional Many to One default mapping

@Entity
public class Car {
 @Id
 @GeneratedValue
 private int id;
 private short year;
 private String model;
 private String maker;
 @ManyToOne
 private Customer customer;
 ...
}

@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	580	1999	1

CUSTOMER table

ID	FIRSTNAME	LASTNAME
1	Frank	Brown

Use a foreign key column

Uni-Directional Many to One with JoinColumn

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

JoinColumn

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

CAR table

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

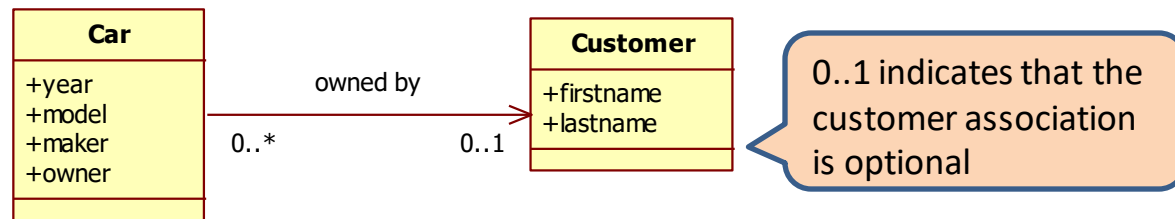
CUSTOMER table

ID	FIRSTNAME	LASTNAME
1	Frank	Brown

Use a foreign key column

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

Uni-Directional Many to One with JoinTable

JoinTable

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

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

CAR table

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

CAR_CUST table

CUSTOMER_ID	ID
1	1
1	2

CUSTOMER table

ID	FIRSTNAME	LASTNAME
1	Frank	Brown

Uni-Directional Many to One with JoinTable

```
@Entity
public class Car {
    @Id
    @GeneratedValue
    private int id;
    private short year;
    private String model;
    private String maker;
    @ManyToOne
    @JoinTable(name = "car_cust",
        joinColumns = { @JoinColumn(name = "car_id") },
        inverseJoinColumns = { @JoinColumn(name = "cust_id") })
    private Customer customer;
    ...
}
```

JoinTable

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

CAR table

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

CAR_CUST table

CUST_ID	CAR_ID
1	1
1	2

CUSTOMER table

ID	FIRSTNAME	LASTNAME
1	Frank	Brown

Mapping Summary



`@ManyToOne`

Default mapping
uses joincolumn

`@ManyToOne`

`@JoinColumn (name="c_id")`

`@ManyToOne`

`@JoinTable (name="car_cust")`

ONE TO MANY ASSOCIATIONS

Uni-directional One to Many default mapping

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

@OneToMany

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

PERSON table

ID	FIRSTNAME	LASTNAME
1	Frank	Brown

PERSON_CAR table

PERSON_ID	CARS_ID
1	1
1	2

CAR table

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

Use a link table

Uni-directional One to Many with JoinColumn

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

@JoinColumn

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

PERSON table

ID	FIRSTNAME	LASTNAME
1	Frank	Brown

CAR table

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

Use a foreign key column

Uni-directional One to Many with JoinTable

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

@JoinTable

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

PERSON table

ID	FIRSTNAME	LASTNAME
1	Frank	Brown

PERS_CAR table

P_ID	C_ID
1	1
1	2

CAR table

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

Use a link table

Many to One / One to Many (Bi)

```
@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<Car>();
    ...
}
```

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

```
@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

CAR table

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

Hibernate sees this bi-directional association as 2 independent associations

Both FK column contain the same information

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<Car>();

    ...
}
```

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

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

The bi-directional
association is stored in
one FK column

Mapping Summary



`@ManyToOne`

Default mapping
uses joincolumn

`@ManyToOne`

`@JoinColumn (name="c_id")`

`@ManyToOne`

`@JoinTable (name="car_cust")`

`@OneToMany`

Default mapping
uses jointable

`@OneToMany`

`@JoinColumn (name="p_id")`

`@OneToMany`

`@JoinTable (name="pers_car")`

BI-directional: Use `mappedBy` on the many side

ONE TO ONE ASSOCIATIONS

OneToOne with annotations

- JPA does not support a real 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

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

Workaround: @PrimaryKeyJoinColumn

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

Primary key
value not
generated

@PrimaryKeyJoinColumn
Join on PK value

```
@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

Mapping Summary



`@ManyToOne`

Default mapping
uses joincolumn

`@ManyToOne`

`@JoinColumn (name="c_id")`

`@ManyToOne`

`@JoinTable (name="car_cust")`

`@OneToMany`

Default mapping
uses jointable

`@OneToMany`

`@JoinColumn (name="p_id")`

`@OneToMany`

`@JoinTable (name="pers_car")`

`@OneToOne`

Same as `@ManyToOne`
Do not use `@PrimaryKeyJoinColumn`

BI-directional: Use `@MappedBy` on the many side

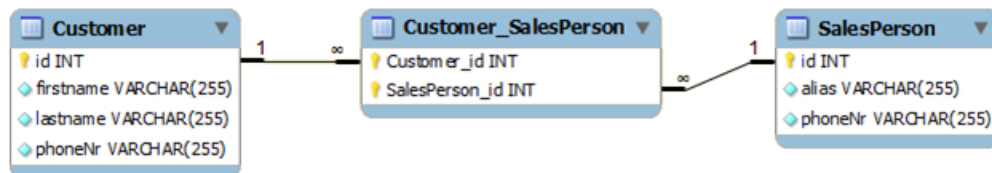
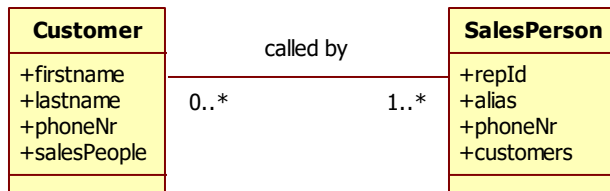
MANY TO MANY ASSOCIATIONS

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<SalesPerson>();
    ...
}
```

@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<Customer>();
    ...
}
```

mappedBy specifies that the other side is the owning side

Mapping Summary

`@ManyToOne`

Default mapping
uses joincolumn

`@ManyToOne`

`@JoinColumn (name="c_id")`

`@ManyToOne`

`@JoinTable (name="car_cust")`

`@OneToMany`

Default mapping
uses jointable

`@OneToMany`

`@JoinColumn (name="p_id")`

`@OneToMany`

`@JoinTable (name="pers_car")`

`@OneToOne`

Same as `@ManyToOne`

Do not use `@PrimaryKeyJoinColumn`

`@ManyToMany`

Default mapping
uses jointable

`@ManyToMany`

`@JoinTable (name = "Customer_SalesPerson")`

BI-directional: Use `@MappedBy` on the many side

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<Car>();
    ...
}
```

```
@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 JPA 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(cascade=CascadeType.PERSIST)
    private List<Car> cars = new ArrayList<Car>();
    ...
}
```

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(cascade={CascadeType.PERSIST, CascadeType.MERGE})
    private List<Car> cars = new ArrayList<Car>();
    ...
}
```

Cascade on persist and Merge

Cascade Types



JPA	Description
ALL	Cascade on all operations
PERSIST	Cascade on persist operations
MERGE	Cascade on merge operations
REMOVE	Cascade on remove operations
REFRESH	Cascade on refresh operations

Mapping Summary

<code>@ManyToOne</code> Default mapping uses joincolumn	<code>@ManyToOne</code> <code>@JoinColumn (name="c_id")</code>	<code>@ManyToOne</code> <code>@JoinTable (name="car_cust")</code>
<code>@OneToMany</code> Default mapping uses jointable	<code>@OneToMany</code> <code>@JoinColumn (name="p_id")</code>	<code>@OneToMany</code> <code>@JoinTable (name="pers_car")</code>
<code>@OneToOne</code>	Same as <code>@ManyToOne</code> Do not use <code>@PrimaryKeyJoinColumn</code>	
<code>@ManyToMany</code> Default mapping uses jointable	<code>@ManyToMany</code> <code>@JoinTable (name = "Customer_SalesPerson")</code>	
BI-directional:	Use <code>@MappedBy</code> on the many side	
Cascading:	By default no cascading <code>@OneToMany (cascade=CascadeType.PERSIST)</code>	

Main point

- One of the important aspects of using JPA is creating the correct mapping between the classes and the tables in the database.

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.

JPA default fetching

- @OneToOne defaults to **eager** loading
- @ManyToOne defaults to **eager** loading
- @OneToMany defaults to lazy loading
- @ManyToMany defaults to lazy loading

Changing the default fetching

```
@Entity
public class Course {
    @Id
    private String courseNumber;
    private String name;
    @OneToMany(fetch=FetchType.EAGER)
    @JoinColumn(name="courseid")
    private Collection<Student> students = new ArrayList<Student>();
}
```

Eager fetching

COLLECTION MAPPING

Collections



- Java collections:
 - Not ordered List (= Bag)
 - Set
 - List
 - Map

Mapping a not ordered List (1)

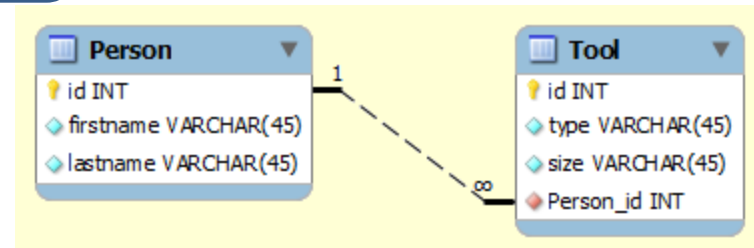
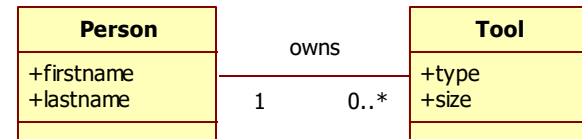
```
@Entity
public class Person {
    @Id
    @GeneratedValue
    private int id;
    private String firstname;
    private String lastname;
    @OneToMany(mappedBy="owner", cascade=CascadeType.PERSIST)
    private Collection<Tool> tools = new ArrayList<Tool>();
}
```

Hibernate will map a Collection as a Bag

We use an ArrayList since there is no official java Bag implementation

```
@Entity
public class Tool {
    @Id
    @GeneratedValue
    private int id;
    private String type;
    private String size;
    @ManyToOne
    private Person owner;
    ...
}
```

We've mapped this collection as a bi-directional one to many



Mapping a not ordered List (2)

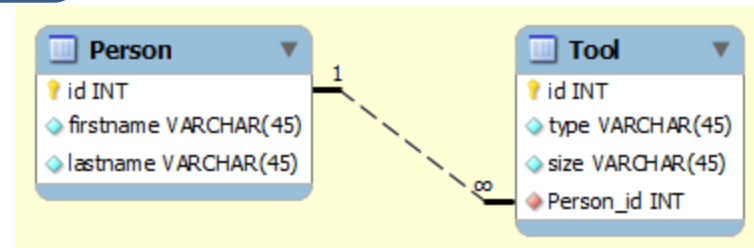
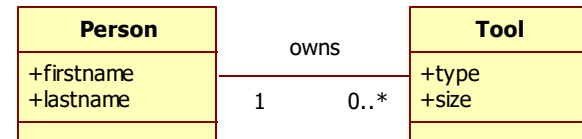
```
@Entity
public class Person {
    @Id
    @GeneratedValue
    private int id;
    private String firstname;
    private String lastname;
    @OneToMany(mappedBy="owner", cascade=CascadeType.PERSIST)
    private List<Tool> tools = new ArrayList<Tool>();
}
```

By default List also maps to a Bag

ArrayList is the most common List implementation

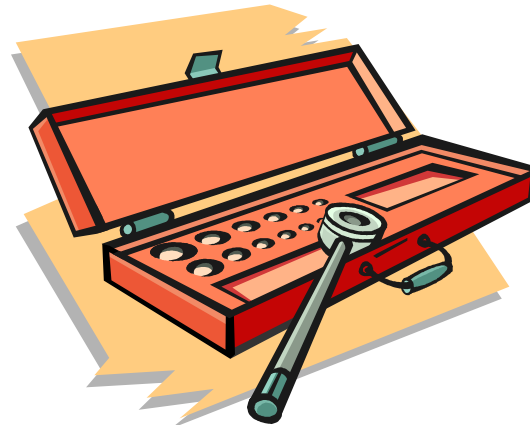
```
@Entity
public class Tool {
    @Id
    @GeneratedValue
    private int id;
    private String type;
    private String size;
    @ManyToOne
    private Person owner;
    ...
}
```

Same bi-directional one to many as last slide



Sets

- Sets are bags that can not contain duplicates:
 - A set still has no inherent order
 - A set can not contain duplicates
- Store bought toolboxes are generally a set
 - No duplicates
 - No inherent order*



Mapping a Set

■ java.util.Set maps as a Set

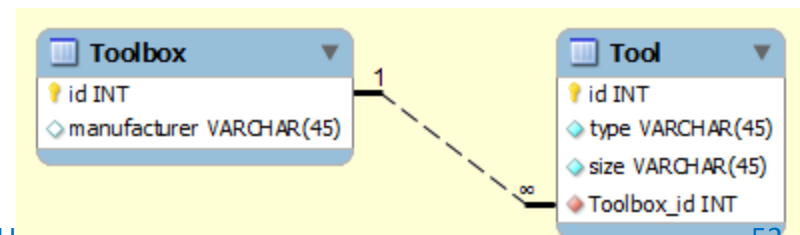
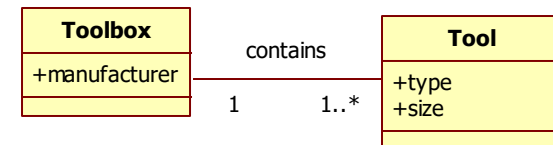
```
@Entity
public class Toolbox {
    @Id
    @GeneratedValue
    private int id;
    private String manufacturer;
    private String model;
    @OneToMany(mappedBy="toolbox", cascade=CascadeType.PERSIST)
    private Set<Tool> tools = new HashSet<Tool>();
}
```

Set maps as a set

HashSet is the most common Set implementation

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

Tool class completes the bi-directional many to one



Lists

- Lists have an inherent order:
 - A List has an inherent, arbitrary order
 - A List can still contain duplicates
- A shopping list is a typical list example
 - An inherent, although often arbitrary order
 - May contain duplicates



One to Many bi-directional List

```
@Entity
public class Person {
    @Id
    @GeneratedValue
    private int id;
    private String firstname;
    private String lastname;
    @OneToMany(cascade=CascadeType.PERSIST)
    @JoinColumn(name="buyer_id")
    @OrderColumn(name="sequence")
    private List<Item> shopList = new ArrayList<Item>();
    ...
}
```

```
@Entity
public class Item {
    @Id
    @GeneratedValue
    private int id;
    private String name;
    private String description;
    ...
}
```

@OrderBy

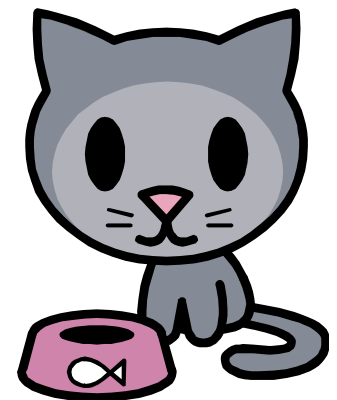
```
@Entity
public class Person {
    @Id
    @GeneratedValue
    private int id;
    private String firstname;
    private String lastname;
    @OneToMany(mappedBy="owner", cascade=CascadeType.PERSIST)
    @OrderBy(clause="type ASC")
    private List<Tool> tools = new ArrayList<Tool>();
    ...
}
```

Order the list of Tools by
the attribute 'type'

```
@Entity
public class Tool {
    @Id
    @GeneratedValue
    private int id;
    private String type;
    private String size;
    @ManyToOne
    private Person owner;
    ...
}
```

Maps

- A Map 'maps' a set of keys to a bag of values:
 - Each value in the bag has a unique key
 - Given a key, the map can quickly retrieve the value
 - No inherent order in either keys or values
- Pet owner ship can be modeled as a map.
 - Each pet has a unique name*
 - To find a pet, you use its name
 - No inherent order in names or pets



Map

@MapKey specifies the key column on the remote class

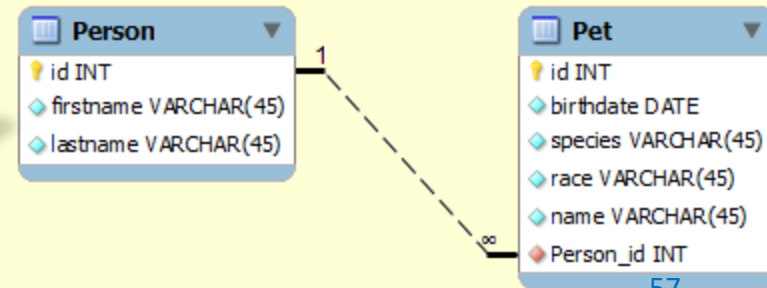
```
@Entity
public class Person {
    @Id
    @GeneratedValue
    private int id;
    private String firstname;
    private String lastname;
    @OneToMany(mappedBy="owner", cascade=CascadeType.PERSIST)
    @MapKey(name="name")
    private Map<String, Pet> pets = new HashMap<String, Pet>();
    ...
}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

Normal @OneToMany

Normal @ManyToOne

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

Specified by @MapKey, will be indexed



Main point

- When an ordered list is stored in the database and you want to persist the order of the elements, then you need to save the order in the database.

Science of Consciousness: There is order in creation. Everything in creation works according the laws of nature.

Connecting the parts of knowledge with the wholeness of knowledge

1. Using JPA requires that the OO domain model looks very similar as the Relational database model.
 2. Collections can be mapped as a Set, a Map, an unordered List and an ordered List
-
3. **Transcendental consciousness** is the most abstract field at the basis of all creation, with the greatest flexibility and power.
 4. **Wholeness moving within itself:** In Unity Consciousness, we see that all layers of creation, from completely abstract to completely relative are nothing but the Self.

