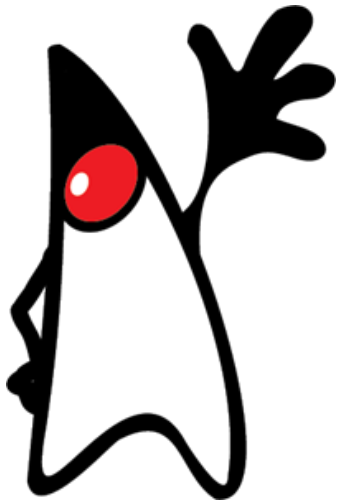
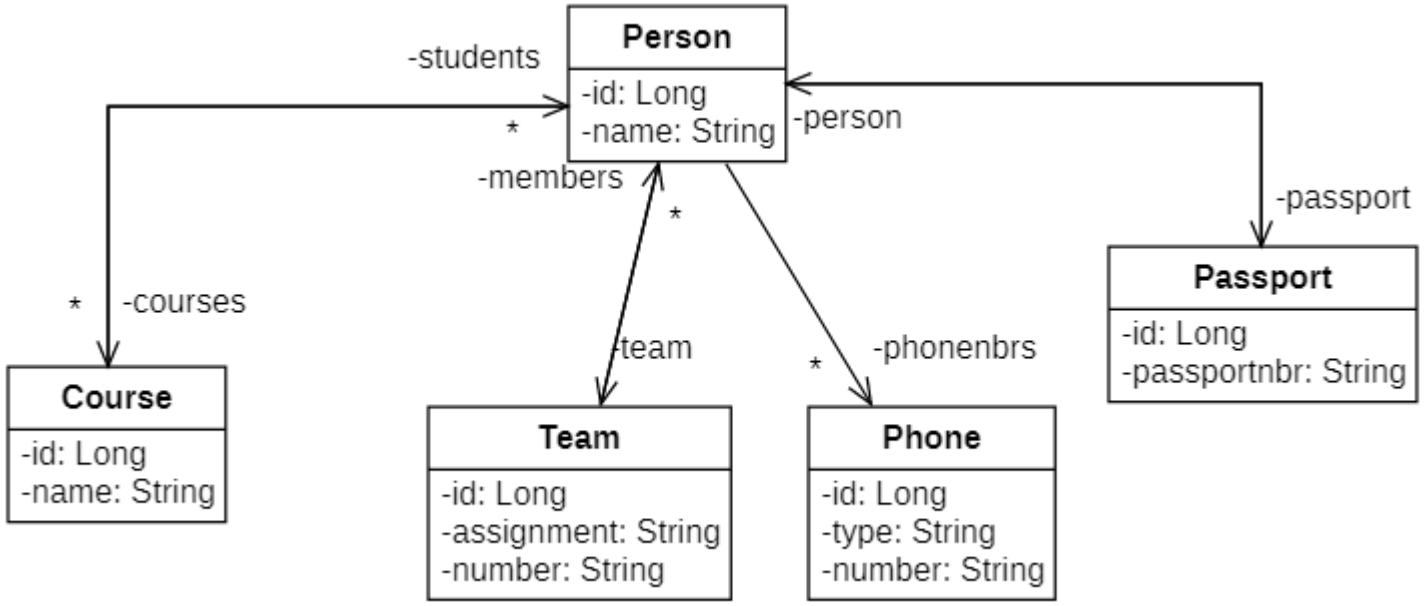
Associations
and
Additional annotations



Problem statement

- How does JPA deal with associations that may occur in a class diagram?
- How do you "map" the entities that have associations with corresponding tables in the database?



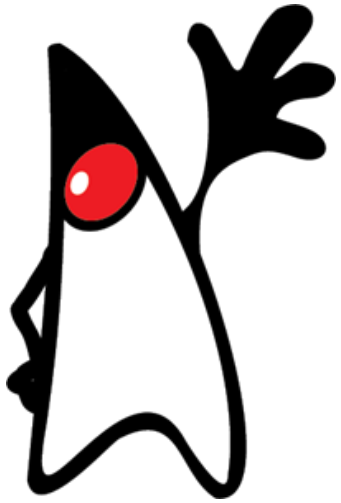
We distinguish 4 different cardinalities

	In the Entity	In the database
1 --- 1 (1 to 1)	@OneToOne Reference to one object	In the table corresponding to this entity there is a FK that establishes the association with the other entity
* --- 1 (many to 1)	@ManyToOne Reference to one object	In the table corresponding to this entity there is a FK that establishes the association with the other entity
1 --- * (1 in many)	@OneToMany Reference via collection attribute	In another table, the primary key of the table corresponding to this entity will be a FK. (that establishes the association with this entity).
--- (much on much)	@ManyToMany Reference via collection attribute	In another table, the primary key of the table corresponding to this entity will be a FK. (that establishes the association with this entity).

Structure

- Associations lesson example with explanation
    - 1 on 1: @OneToOne for Person - Passport
    - Many on 1: @ManyToOne for Person - Team
    - 1 to many: @OneToMany for Person - Phone
    - Much on much: @ManyToMany for Person - Course
- Direction of association
- Additional annotations

Making Associations Example



Contents

- This presentation will guide you through the implementation of associations in a JPA project.
- Follow these instructions carefully and you will learn what to do and how to do it.
- The project that came with this item is only for use in case you don't manage to make it yourself. It is best to use the (tele)coaching than...

- Good luck!

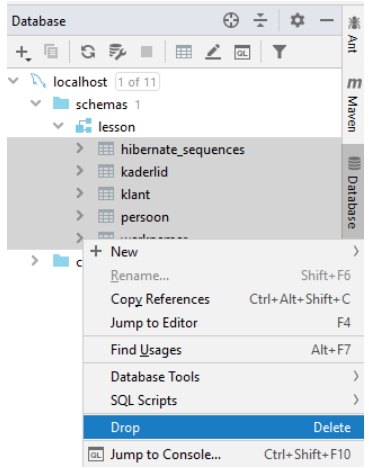
Structure

- This presentation will show you how to implement associations in JPA. The following cardinalities are dealt with in this example:

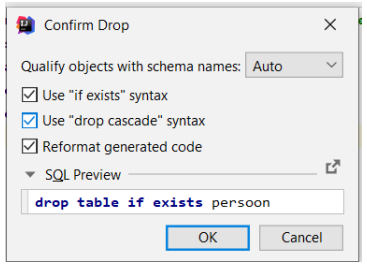
  - 1 to 1: @OneToOne for Person - Passport
  - Many to 1: @ManyToOne for Person - Team
  - 1 to many: @OneToMany for Person - Phone
  - Many to many: @ManyToMany for Person - Course

Making a lesson example: Associations

- Step 0: Delete/drop the person table in your database
  - In the inheritance project, you created a person table. In this project, you will also create a person table, but it must look different from the one in the previous project. Therefore, you should first remove all tables from the database:
    - Select all tables, right-click on a table and choose Drop



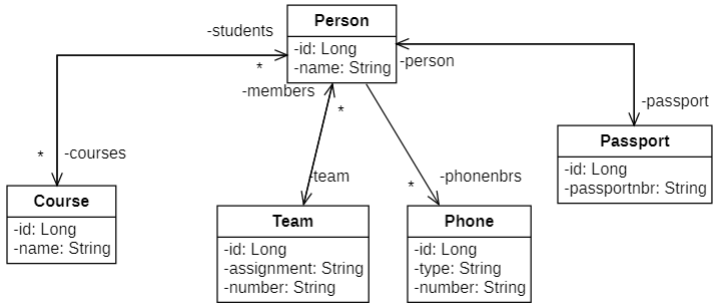
    - Then click on the following checkboxes and click OK:



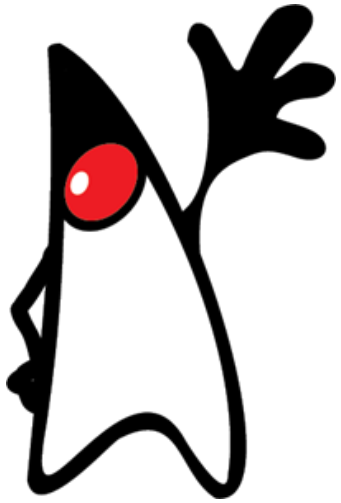
Making of the lesson example: Associations

- Step 1: Creating a web project
  - You can check this procedure again in the document at Start Java Advanced -> Create project in IntelliJ.

- Step 2: Create the entities "Person", "Passport", “Course", "Team" and "Phone".
  - See presentation *Lesson 3 JPA* to know how to create entities
  - Complete with the attributes shown in the class diagram
  - Generate the no-arg constructor and getters and setters in each entity
  - Implement **all** associations according to the indicated cardinality
    - @OneToOne for Person - Passport
    - @ManyToOne for Person - Team
    - @OneToMany for Person - Phone
    - @ManyToMany for Person - Course

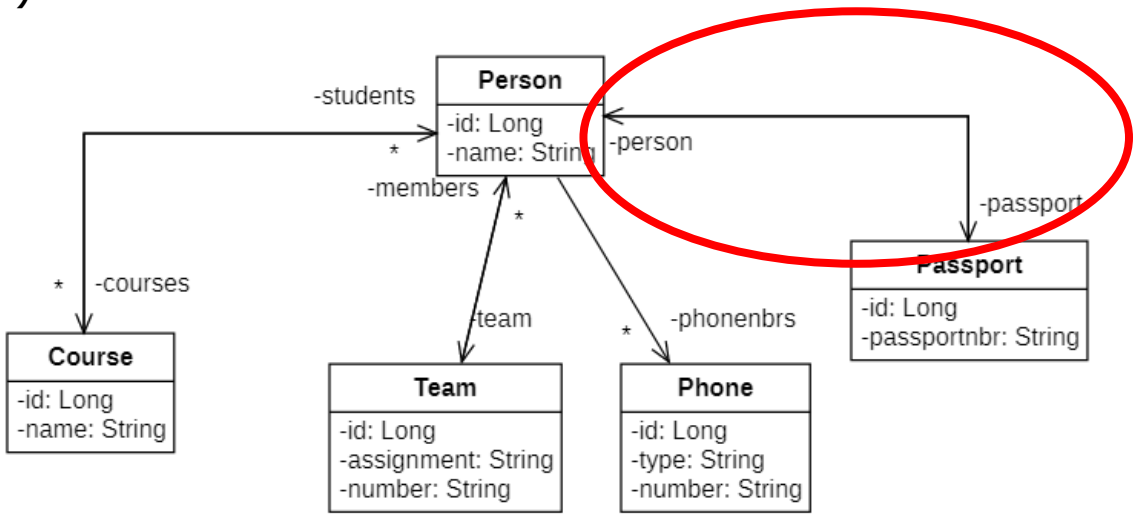


1 - 1:
@OneToOne



Edit Person.java for @OneToOne implementation

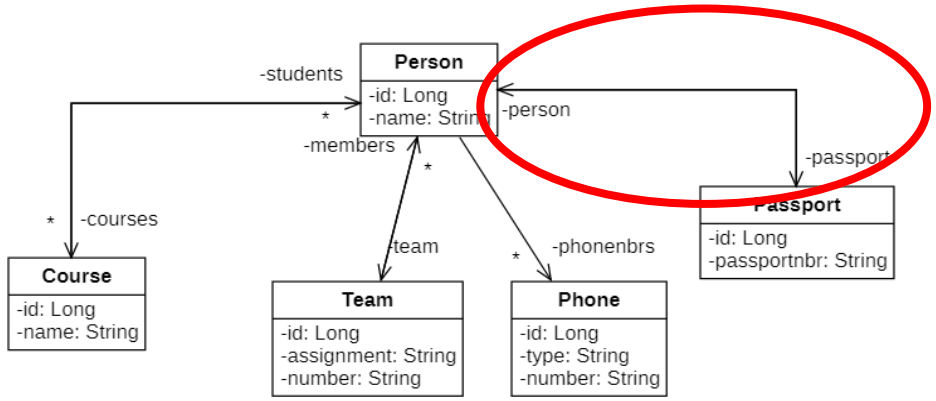
```
@Entity
public class Person{
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String name;

    @OneToOne (cascade={CascadeType.ALL})
    private Passport;

    ...
    @Override
    public String toString() {
        return name;
    }
}
```



Also generate the getter and setter for passport!

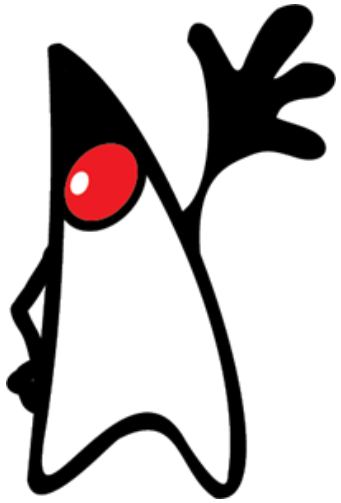
Modify Passport.java for @OneToOne implementation

```
@Entity
public class Passport{
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private Long id;

   @OneToOne(mappedBy= "passport")
   private Person person;
   ...
   private String passportnbr;

}
```



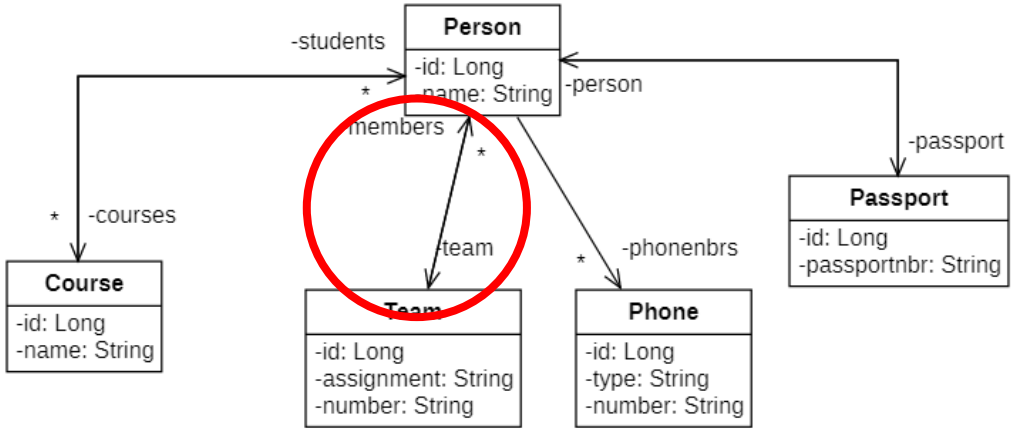
Also generate the getter and setter for person!

* - 1:
@ManyToOne



Person.java

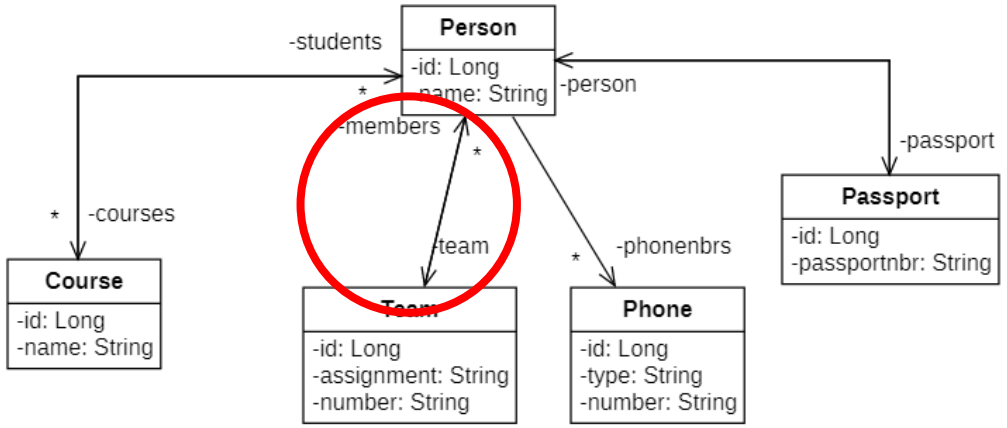
```
@Entity
public class Person {
  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private Long id;
  private String name;
  @OneToOne(cascade={CascadeType.ALL})
  private Passport;
  @ManyToOne
  private Team team;
  . . .
}
```



Also generate the getter and setter for team!

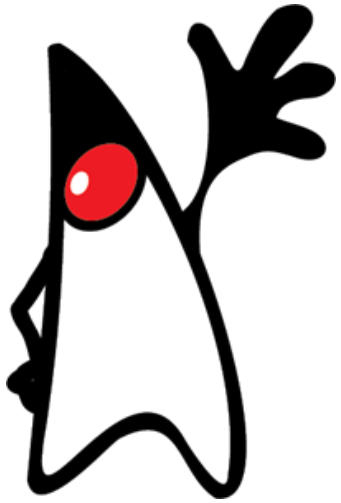
Team.java

```
@Entity
public class Team{
  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private Long id;
  private String assignment;
  private String number;
  @OneToMany(mappedBy="team")
  private List<Person> members = new ArrayList<>();
. . . //instantiate that collection immediately
  public String toString() {
    return assignment + " ("+number+")";
  }
}
```



Also generate the getter and setter for members!

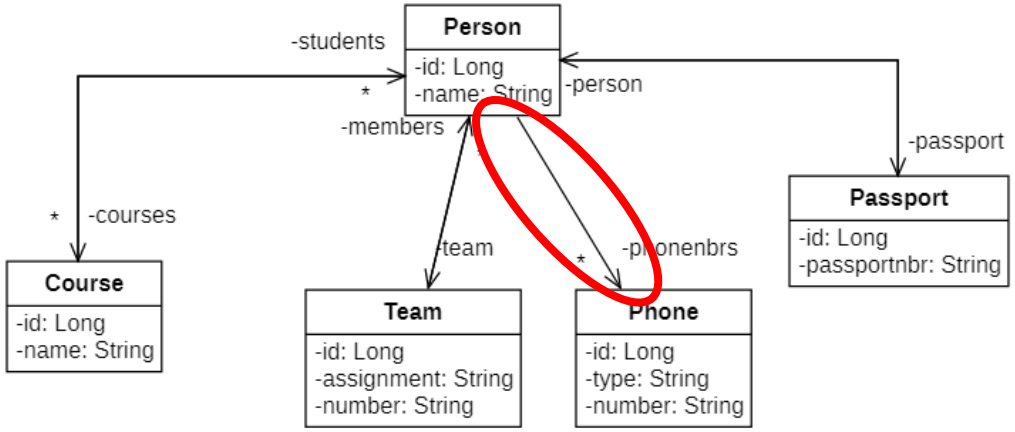
1 - *:
@OneToMany



Person.java

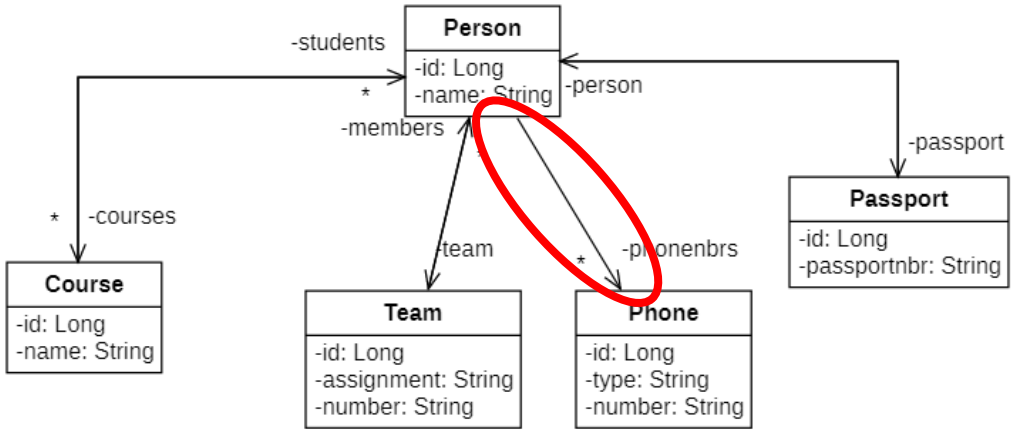
```
@Entity
public class Person{
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String name;
    @OneToOne (cascade={CascadeType.ALL})
    private Passport passport;
    @ManyToOne
    private Team team;
    @OneToMany
    private List<Phone> phonenbrs = new ArrayList<>();
                    // Always instantiate that collection immediately!
...
    public void addPhonenbr(String type, String number){
        Phone phone = new Phone();
        phone.setType(type);
        phone.setNumber(number);
        this. phonenbrs.add(phone);
     }
}
```

Also generate the getter and setter for phonenbrs!

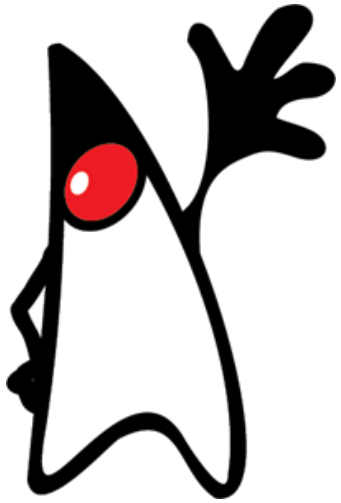


Phone.java

```
@Entity
public class Phone{
  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private Long id;
  private String type;
  private String number;
  //No need to add an extra attribute: the association is only drawn in one direction in the class diagram!

  public Phone() {
  }
. . .

}
```

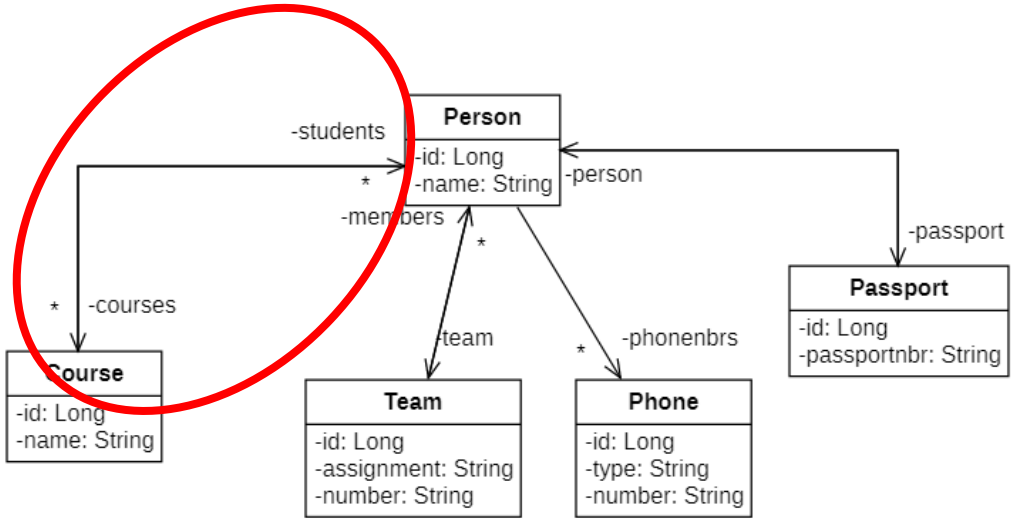


* - *:
@ManyToMany



Person.java

```
@Entity
public class Person{
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String name;
    @OneToOne (cascade={CascadeType.ALL})
    private Passport passport;
    @ManyToOne
    private Team team;
    @OneToMany
    private List<Telephone> phonenrs = new ArrayList<>();
    @ManyToMany
    private List<Course> courses = new ArrayList<>(); ...
}
```

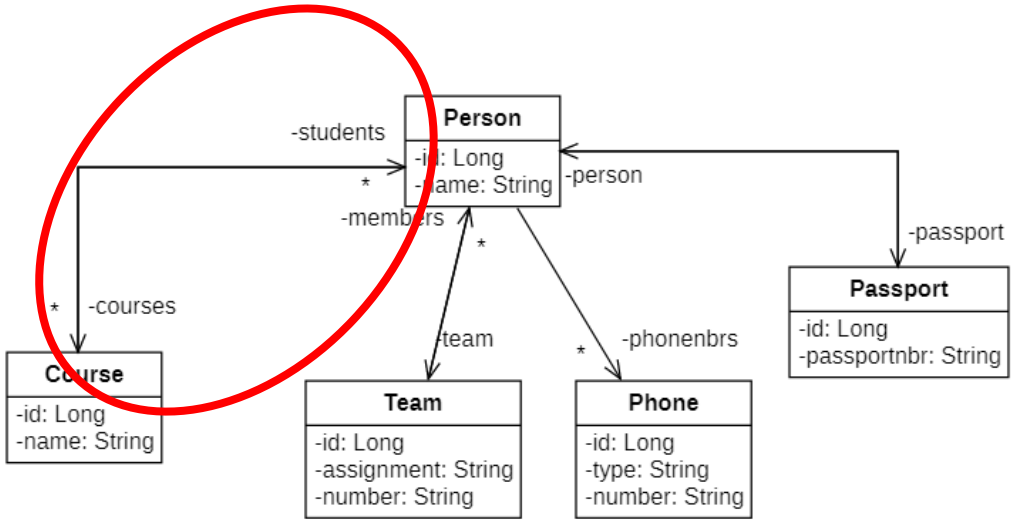


Also generate the getter and setter for boxes!

Course.java

```
@Entity
public class Course{
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String name;
    @ManyToMany(mappedBy="courses")
    private List<Person> students = new ArrayList<>();
        // Always instantiate that collection immediately!

...
}
```

Also generate the getter and setter for students!



Making a lesson example: Associations

- Step 3: Create repository for entities "Person" and "Team
  - See presentation *Lesson 3 JPA* to know how to do this
  - => PersonRepository and TeamRepository

Step 4: Create MainController

- Now create the MainController:
  - See presentation *Lesson 0 MVC* to know how to create a controller
  - Create association with PersonRepository and TeamRepository
  - Use the following methods:
    - Method to display the **index** page
    - Method to display the **addTeam** page where you can enter the details of a new Team
    - Method to create a new Team object with these data after passing the data from the addTeam page, store this team in the database and then display the index page
    - Method to display the **addPerson** page where you can enter the data of a new Person
    - Method, after passing the data from the addPerson page, to create a new Person object with this data, to store this person in the database and then to display the index page
    - Method to display the **addPhone** page where you can enter a new phone number for a person
    - Method, after forwarding the data from the addPhone page, to associate a new telephone number with the selected person, to store this modified person in the database and then to display the index page

Step 4: Create MainController

```
@Controller
public class MainController {
    private PersonRepository personRepository;
    private TeamRepository teamRepository;

    public MainController(PersonRepository personRepository,
TeamRepository teamRepository) {
        this.personRepository = personRepository;
        this.teamRepository = teamRepository;
    }

    @RequestMapping("/")
    public String index(Model model) {
        List<Team> list = teamRepository.findAll();
        model.addAttribute("teamList", list);
        return "index";
    }

    @RequestMapping("/addTeam")
    public String addTeam() {
        return "addteam";
    }

    @RequestMapping("/processAddTeam")
    public String processAddTeam(Model model, HttpServletRequest
request) {
        String assignment = request.getParameter("assignment");
        String number = request.getParameter("number");
        Team team = new Team();
        team.setAssignment(assignment);
        team.setNumber(number);
        teamRepository.save(team);
        List<Team> list = teamRepository.findAll();
        model.addAttribute("teamList", list);
        return "index";
    }
```

Step 4: Create MainController (continued)

```
@RequestMapping("/addPerson")
    public String addPerson(Model model) {
        List<Team> list = teamRepository.findAll();
        model.addAttribute("teamList", list);
        return "addPerson";
    }

    @RequestMapping("/processAddPerson")
    public String processAddPerson(Model model,
HttpServletRequest request) {
        String name = request.getParameter("name");
        long teamId =
Long.parseLong(request.getParameter("teamIndex"));
        Optional<Team> team = teamRepository.findById(teamId);
        Person person = new Person();
        person.setName(name);
        if (team.isPresent()) {
            person.setTeam(team.get());
        }
        personRepository.save(person);
        List<Team> list = teamRepository.findAll();
        model.addAttribute("teamList", list);
        return "index";
    }

    @RequestMapping("/addphone")
    public String addTelephone(Model model) {
        List<Person> list = personRepository.findAll();
        model.addAttribute("personList", list);
        return "addPhone";
    }

    @RequestMapping("/processAddPhone")
    public String processAddTelephone(Model model,
HttpServletRequest request) {
        String type = request.getParameter("type");
        String number = request.getParameter("number");
        long personId =
Long.parseLong(request.getParameter("personIndex"));
        Person = personRepository.findById(personId).get();
        person.addPhonenbr(type, number);
        personRepository.save(person);
        List<Team> list = teamRepository.findAll();
        model.addAttribute("teamList", list);
        return "index";
    }
}
```

Making a lesson example: Associations

- Step 5: Create user interface:
  - index.html
  - addTeam.html
  - addPerson.html
  - addPhone.html

Step 5: Create user interface index.html

Now it remains to create a user interface. Adjust the body of index.html so that new teams and persons can be registered and so that an overview is given of already registered teams and their members with their telephone numbers:

```
<body>.
< ol>
    <li th:each="team : ${teamlist}" ><span th:text="${team.get.getAssignment()}" />
        < ol>
            <li th:each="member : ${team.getMembers()}">
                <span th:text="${member.getName()}" />
                <span th:text="'(passportnbr: '+${member.getPassport().getPassportnbr()+')'}"></span>
                < ul>
                    <li th:each="tel : ${member.getPhonenrs()}">
                        <span th:text="${tel.getType()} + ': ' + ${tel.getNumber()}" />
                    </li>
                </ul>
            </li>
        </ol>
    </li>
</ol>
< br/>
<p><a href="/addTeam">Add team</a></p>
<p><a href="/addPerson">Add a person to a team</a></p>
<p><a href="/addPhone">Add phone number</a></p>
</body>.
```

Step 5: Create user interface addTeam.html

Now we have to add the pages addTeam.html and addPerson.html. We start with the body of addTeam.html:

```
<body>.
<h1>Make Team</h1>
<form action="/processAddTeam" >
    <p>
        <label for="assignment">Assignment:</label>
        <input type="text" name="assignment" id="assignment">
    </p>
    <p>
        <label for="number">Number:</label>
        <input type="text" name="number" id="number">
    </p>
    <p>
        <input type="submit" value="Save" name="save">.
    </p>
</form>.
</body>.
```

Step 5: Create user interface addPerson.html

We will now work on addPerson.html:

```
<body>.
<h1>Make Person</h1>
<form action="/processAddPerson" >
    <p>
        <label for="name">Name:</label>
        <input type="text" name="name" id="name">
    </p>
     <p>
        <label for="passport">Passport number:</label>
        <input type="text" name="passport" id="passport">
    </p>
    <p>
        <select name="teamIndex">
            <option value = "-1">Choose a team </option>
            <option th:each= "team: ${teamList}" th:text="${team.toString()}" th:value="${team.getId()}"  ></option>
        </select>.
    </p>
    <p>
        <input type="submit" value="create team member" name="create team member">.
    </p>
</form>.
</body>
```

Step 5: Create user interface addPhone.html

We are now working on addPhone.html:

```
<body>.
<h1>Create a Phone Number for a Person</h1>.
<form action="/processAddPhone" method="post">
    <p>
        <select name="personIndex">
            <option th:each= "person: ${personlist}" th:text="${person.toString()}" th:value="${person.getId()}"  ></option>
        </select>.
    </p>
    <p>
        <label for="type">Type:</label>.
        <input type="text" name="type" id="type">
    </p>
    <p>
        <label for="number">Number:</label>
        <input type="text" name="number" id="number">
    </p>
    <p>
        <input type="submit" value="create phone number" name="create phone number">.
    </p>
</form>.
</body>.
```

Application testing

We use the just created html pages to create 2 teams:

- Team 1: Assignment: Internationalisation, Number: 21
- Team 2: Assignment: Herbalist, Number: 22

We populate each team with some team members

- Team member 1: Name: Mark Roets, passport number: 12587-258648-21 Team: Internationalisation (21)
- Team member 2: Name: Jo Goossens, passport number: 587469-256413-58 Team: Internationalisation (21)
- Team member 3: Name: Bert Gevers, passport number: 2584-258947-36 Team: Herbalist (22)
- Team member 4: Name: Pieter De Belder, passport number: 25489-369587-59 Team: Herbalist (22)
- Team member 5: Name: Kenny De Boeck, passport number: 589748-958674-29 Team: Internationalisation (21)

index.html

Result:

1. Internationalisation
    1. Mark Roets (passportnbr: 12587-258648-21)
    2. Jo Goossens (passportnbr: 587469-256413-58)
    3. Kenny De Boeck (passportnbr: 589748-958674-29)
2. Herbalist
    1. Bert Gevers (passportnbr: 2584-258947-36)
    2. Pieter De Belder (passportnbr: 25489-369587-59)

Add team

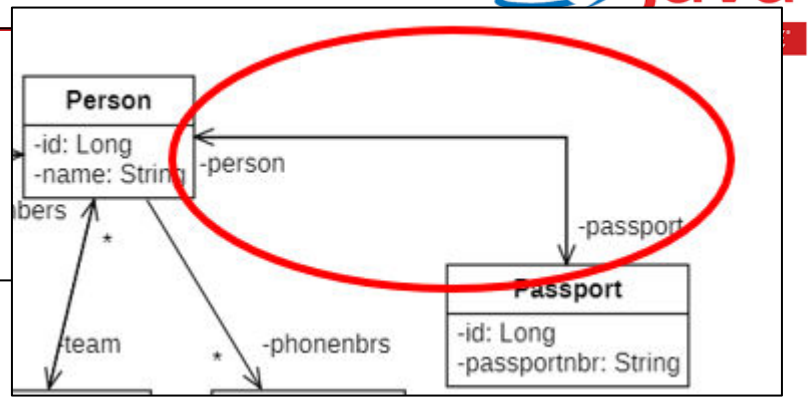
Add person to a team

Add phonenumber

In the database

In the class Person:
@OneToOne(cascade={CasadeType.ALL})
private Passport passport;

In the Passport class:
@OneToOne(mappedBy="passport")
private Person person;



	In the Entity	In the database
1 --- 1 (1 to 1)	@OneToOne Reference to one object	In the table corresponding to the entity there is an FK referring to the primary key of a record in another table

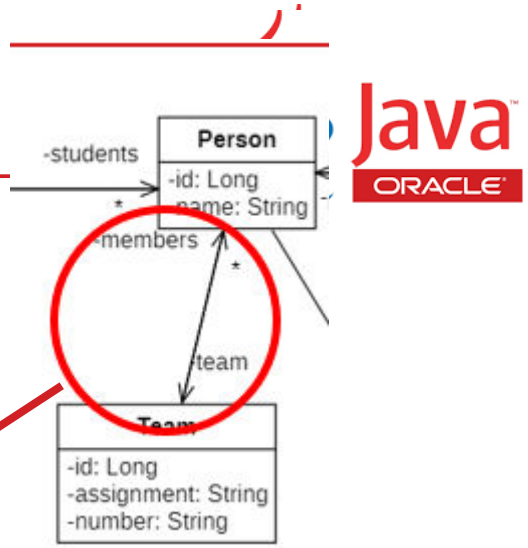
	id	name	passport_id	team_id
1	1	Mark Roets	1	1
2	2	Jo Goossens	2	1
3	3	Bert Gevers	3	2
4	4	Pieter De Belder	4	2
5	5	Kenny De Boeck	5	1

	id	passportnbr
1	1	12587-258648-21
2	2	587469-256413-58
3	3	2584-258947-36
4	4	25489-369587-59
5	5	589748-958674-29

In the database

In the class Person:
@ManyToOne
private Team team;

In the Team class:
@OneToMany(mappedBy="team")
private List<Person> members = new ArrayList<>();

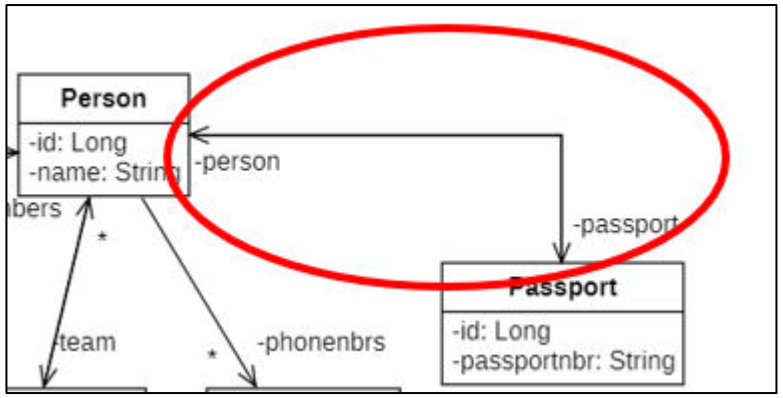


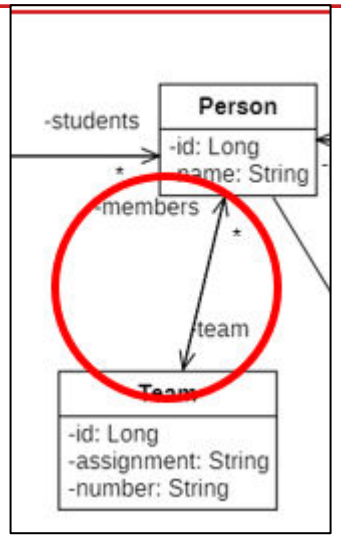
* --- 1 (many to 1)	@ManyToOne Reference to one object	In the table corresponding to the entity there is an FK referring to the primary key of a record in another table
1 --- * (1 in many)	@OneToMany Reference via collection attribute	In another table, the primary key of the table corresponding to the entity will be recorded as FK. (The table corresponding to the entity will not show this).

	id	name	passpor...	team_id
1	1	Mark Roets	1	1
2	2	Jo Goossens	2	1
3	3	Bert Gevers	3	2
4	4	Pieter De Belder	4	2
5	5	Kenny De Boeck	5	1

	id	assignment	number
1	1	Internationalisation	21
2	2	Herbalist	22

Bidirectional @OneToOne and @OneToMany





- Each person has one passport and one passport belongs to one person:
  - Bidirectional so we are able to access the passport of a person and access the person to which the passport belongs
  - In class Person and Passport we have @OneToOne annotation
- Each person belongs to exactly one team. Each team consists of several members.
  - Bidirectional so we are able to access the members of each team and also want to know to which team each person belongs
  - In class Team we have an @OneToMany annotation
  - In class Person we use the @ManyToOne annotation

Person.java

```
@Entity
public class Person{
  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private Long id;
  private String name;
  @OneToOne (cascade={CascadeType.ALL})
  private Passport;
  @ManyToOne
  private team;
  . . .
}
```

Table PERSON gets FK to table PASSPORT and when saving a new person, the associated passport will also be created and saved (cascade=…)

Table PERSON gets FK to table TEAM and when saving a new person, the team to which you want to link this new person already exists (and doesn't need to be created => no cascade)

Passport.java

```
@Entity
public class Team{
  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private Long id;
@OneToOne(mappedBy="passport")
  private Person person;
   . . .
}
```

Association is achieved by FK in table PERSON in the column defined by the attribute passport

Team.java

```
@Entity
public class Team{
  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private Long id;
  private String assignment;
  private String number;
  @OneToMany(mappedBy="team")
  private List<Person> members = new ArrayList<>();
   . . .
}
```

Association is achieved by FK in table PERSON in the column defined by the attribute team

Application testing

We now also add a telephone number:

Make a phonenumber for a person

Mark Roets

Type: homephone

Number: 014/562356

make phonenbr

We get this error message:

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Thu Apr 16 21:03:14 CEST 2020
There was an unexpected error (type=Internal Server Error, status=500).
org.hibernate.TransientObjectException: object references an unsaved transient instance - save the transient instance be
object references an unsaved transient instance - save the transient instance before flushing: fact.it.vbassociaties.model.

Application testing

Error message: "Object references an unsaved transient instance - save the transient instance before flushing: fact.it.vbassociations.model.Phone

Explanation: We write a person object to the database, but this person has a phone object attached to it which is not yet in the database. We have to make sure that the phone is written to the database together with the person.

To solve this problem, we adjust the annotation to the telephone numbers as follows.

```
@OneToMany(cascade={CascadeType.ALL})
   private List<Phone> phonbrs = new ArrayList<>();
```

This addition ensures that any database operation on person (such as saving and deleting) is also done on the phone numbers (in the same way as we did for «passport»). If we now save a Person object in the database, the phone numbers are automatically saved with it thanks to this annotation

CascadeType

@OneToMany(cascade={CascadeType.ALL})
private List<Phone> phonenbrs = new ArrayList<>(); @

- In this example we specify that when we call a database operation on the entity Person we will also call it for the associated entity Phone.

- cascade may have the following values for CascadeType:
  PERSIST, MERGE, REMOVE, REFRESH, ALL.
- If you choose CascadeType.PERSIST (in stead of CascadeType.ALL), creating a Person with a phonenbr will result in creating a Phone but deleting a person will not result in deleting the associated phone, for that we have CascadeType.REMOVE)

CascadeType

- It is possible to combine several values e.g. cascade={CascadeType.REFRESH, CascadeType.MERGE}
- Declaring CascadeType only makes sense in a composition/aggregation or Parent-Child association, where we write the annotation in Parent.
  => CascadeType in Person because if e.g. Person is deleted, then also his Phonenbrs have to be deleted. In the same way when a Person is deleted, his Passport must also be deleted but conversely if the Passport is deleted then the associated Person does not need to be deleted
- At OneToMany - ManyToOne association => CascadeType at the @OneToMany annotation (and not at @ManyToOne)
- If no CascadeType is specified, the order of writing is very important.
  See lesson example with OneToMany association
  - between Team and Person where Person is the "Owner" (this table also contains the FK). We first have to make Team persistent before we can store Person in the database.

Application testing

We restart the application and now add some phone numbers:

1. Internationalisation
    1. Mark Roets (passportnbr: 12587-258648-21)
        - homephone: 014/562356
        - GSM: 0478/985698
    2. Jo Goossens (passportnbr: 5874-25413-58)
    3. Kenny De Boeck (passportnbr: 589748-958674-29)
        - GSM: 0497/251436
        - office phone: 03/9582356
2. Herbalist
    1. Bert Gevers (passportnbr: 2584-258947-36)
        - homephone: 014/306598
    2. Pieter De Belder (passportnbr: 25489-369587-59)

Make a phonenumber for a person

Mark Roets

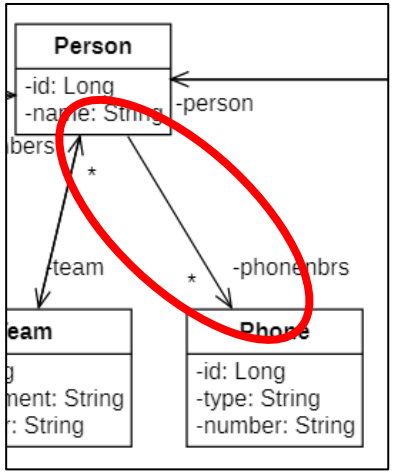
Type: homephone

Number: 014/562356

make phonenbr

In the database

1 --- * (1 in many)	@OneToMany Reference via collection attribute	In another table, the primary key of the table corresponding to the entity will be recorded as FK. (The table corresponding to the entity will not show this).



person

	id	name	passport_id	team_id
1	1	Mark Roets	1	1
2	2	Jo Goossens	2	1
3	3	Bert Gevers	3	2
4	4	Pieter De Be…	4	2
5	5	Kenny De Boe…	5	1

phone

	id	number	type
1	1	014/562356	homephone
2	2	0478/985698	GSM
3	3	014/306598	homephone
4	4	0497/251436	GSM
5	5	03/9582356	office phone

person_phonenbrs

	person_id	phonenbrs_id
1	1	1
2	1	2
3	3	3
4	5	4
5	5	5

Table person_phonenbrs is automatically created by JPA created:

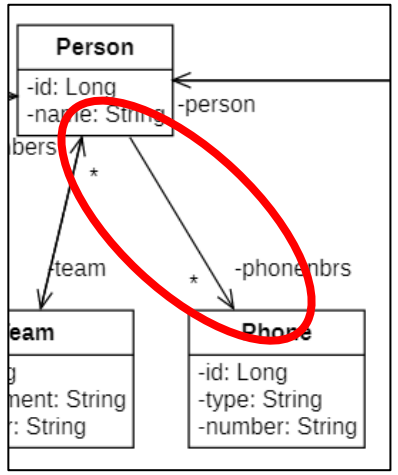
- person
- person_phonenbrs
- phone

OneToMany: Unidirectional

- One-way association between Person and Phone
- Unidirectional because a Phone object does not need to know who its owner is.
- Each person can have several phone numbers
- @OneToMany association: only annotation in class Person:
  - Person keeps a list of phones:

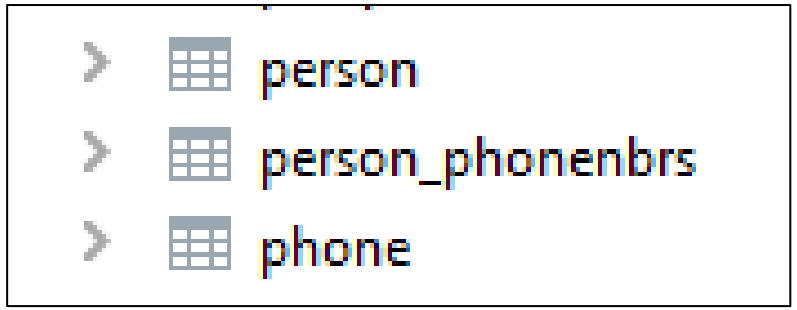
```
@OneToMany (cascade={CascadeType.ALL})
private List<Telephone> phonenbrs = new ArrayList<>();
```

  - There is no reference to Person in Phone



OneToMany: Unidirectional

- We end up with 3 tables:
  - person
  - phone
  - person_phonenbrs (will be created automatically)



person

	id	name	passport_id	team_id
1	1	Mark Roets	1	1
2	2	Jo Goossens	2	1
3	3	Bert Gevers	3	2
4	4	Pieter De Be…	4	2
5	5	Kenny De Boe…	5	1

person_phonenbrs

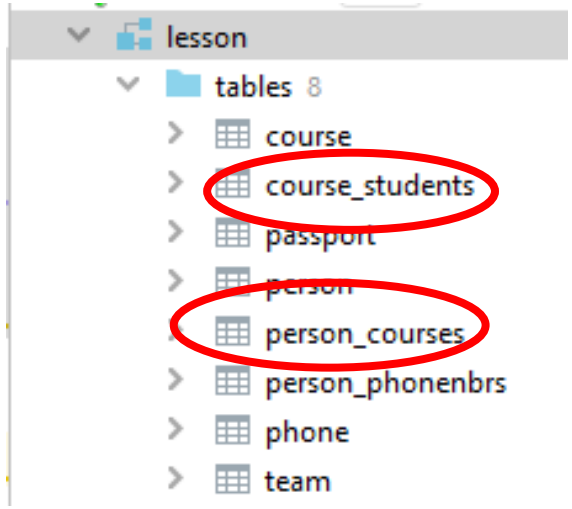
	person_id	phonenbrs_id
1	1	1
2	1	2
3	3	3
4	5	4
5	5	5

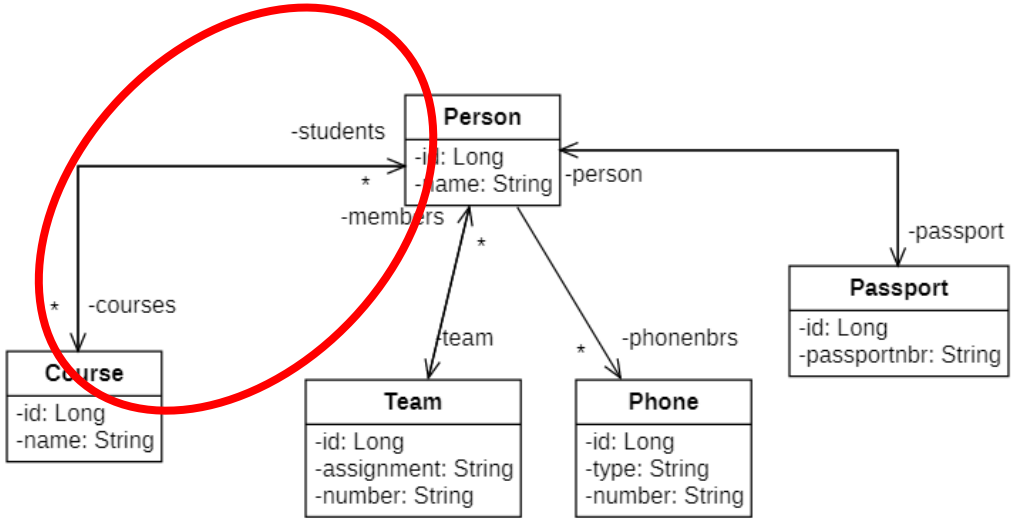
phone

	id	number	type
1	1	014/562356	homephone
2	2	0478/985698	GSM
3	3	014/306598	homephone
4	4	0497/251436	GSM
5	5	03/9582356	office phone

Application testing

- The table structure is currently as follows:



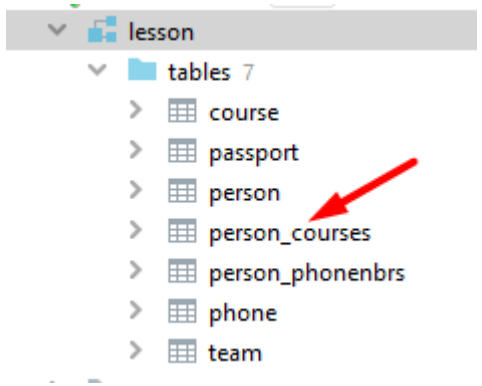


- We note that for the *-* relationship between person and course, 2 association tables were created.
- Adding courses to a person or adding persons a course will cause runtime errors because this method can cause inconsistencies (more explanation about this later in this presentation)
- We adapt the annotation to the students attribute in the entity Course as follows:

```
@ManyToMany(mappedBy="courses")
private List<Person> students = new ArrayList<>();
```

Application testing

- We remove all tables from the database and run the application again. Result in the database:



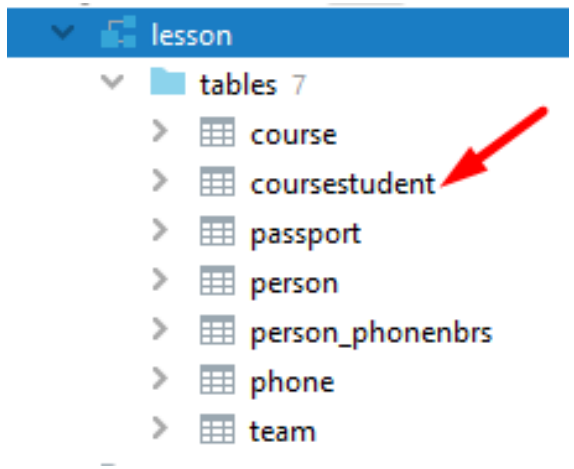
Only the association table person_courses remains!

- We now also modify the annotation for the courses attribute in the entity person as follows:

```
@ManyToMany
@JoinTable(name="coursestudent")
private List<Course> courses = new ArrayList<>();
```

Application testing

- We remove all tables from the database and run the application again. Result in the database:



The association table person_courses has been renamed to coursestudent

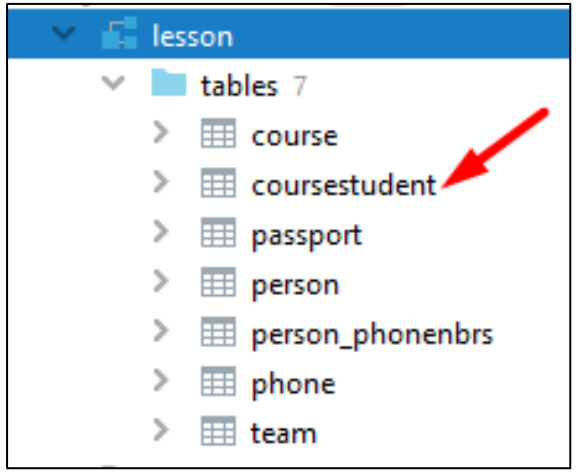
@ManyToMany: In summary

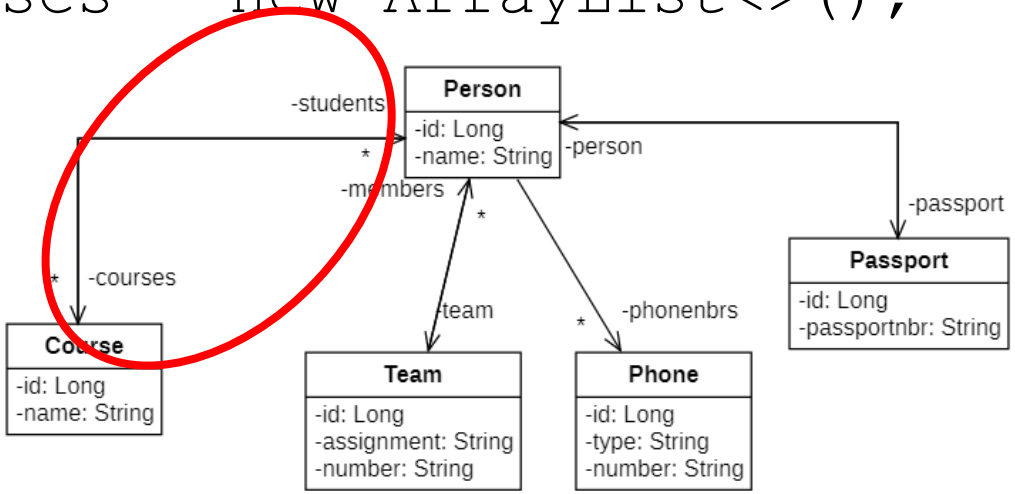
In the Entity **Course**:

```
@ManyToMany(mappedBy="courses")
private List<Person> students = new ArrayList<>();
```

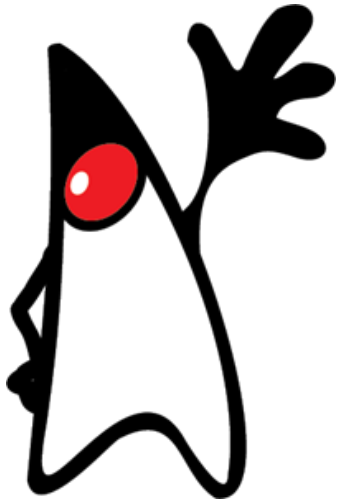
In the Entity **Person**:

```
@ManyToMany
@JoinTable(name="coursestudent")
private List<courses> courses = new ArrayList<>();
```



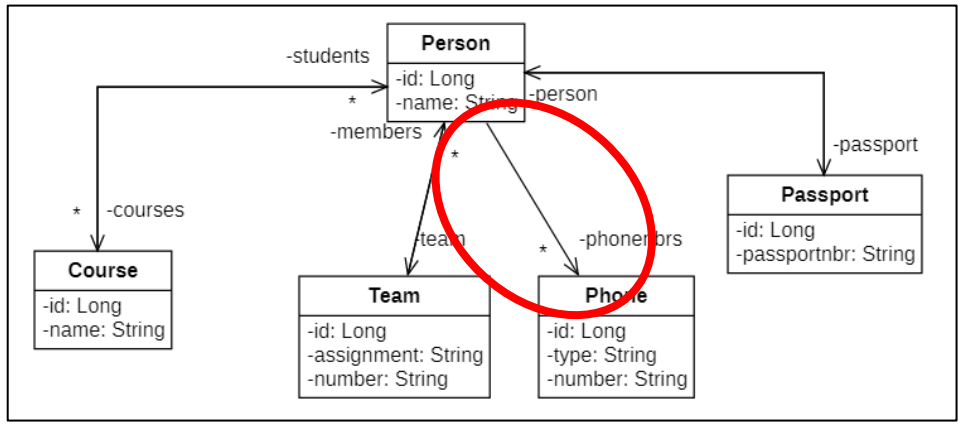


Direction of association



Association with direction: unidirectional

- If the association has a direction (= an arrowhead in the class diagram) then it is **Unidirectional**
  - => only one entity keeps a reference to another entity. The other entity does not keep a reference to this entity.
  - The entity that does keep a reference indicates which multiplicity this association has.
    - 1-1 => @OneToOne above the reference to one object
    - 1-* => @OneToMany above the reference to a collection attribute (List)
    - *-1 => @ManyToOne above the reference to one object
    - *-* => @ManyToMany above the reference to a collection attribute (List)
  - In the other entity we do not mention anything
- In the lesson example we had the association between Person and Phone in this way



Two-way association: bidirectional

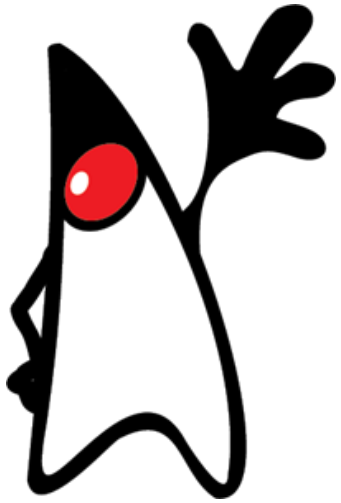
- If the association is to be worked out in the 2 directions (= there is no arrowhead in the class diagram) then it is **Bidirectional**
- Both entities in the association maintain a reference to the other
- One must then ensure that the association remains **consistent at all times**
  - For example, if a certain person has an association with a certain passport, that same passport must also have an association with that same person.
  - How?
    - When writing out an object that has an association with another object, only one place in the database will be used to enter the relationship, and this is the FK...
    - **One** of the 2 entities in the association **must** contain the "**MappedBy**" attribute.
      - Which entity with which reference depends on the type of association:
        - 1-1 => you may choose in which entity
        - *-1 => no MappedBy
        - 1-* => this entity must contain the MappedBy attribute if it is bidirectional
        - 1-* => this entity does not have a MappedBy attribute if it is unidirectional
        - *-* => you may choose in which entity
    - The MappedBy attribute will determine where the FK will be = Owner of the relation/association

Optional and FetchType

- **= Additional properties to the association annotations**
- **Optional**: default value = true
  - Indicates whether the FK can be null (NA/NNA)
  - If you do not specify, FK: NA (default)
  - For example. @ManyToOne(optional = false) => the FK is mandatory (NNA)

- **FetchType:** FetchType.EAGER or FetchType.LAZY
  - FetchType indicates whether you want to immediately fetch the associated objects from the database (EAGER) or not (LAZY)
  - Eg @OneToMany(mappedBy="team", fetch=FetchType.EAGER)
  - Default values for fetchtype in jpa 2.0
    - OneToMany: LAZY
    - ManyToOne: EAGER
    - ManyToMany: LAZY
    - OneToOne: EAGER
  - ManyToMany and OneToMany have default FetchType Lazy to avoid the initialisation of a lot of unused objects and the extra database queries that have to be run to initialise the associated objects. However, sometimes this goes wrong in jpa and the array list with associated objects is never filled up (empty ArrayList). This can be solved by specifying FetchType.EAGER with these associations.

Additional annotations



Contents

Annotations

In JPA, we use annotations to determine how the entities on the tables should be mapped.

- There are 89 different annotations for persistence
- We have already seen:
  - @Entity, @Id, @GeneratedValue, @Inheritance, @OneToOne, @ManyToOne, @ManyToMany, @JoinTable
- We see 3 additional annotations in this presentation:
  - @Transient, @Enumerated, @Temporal
- The full list of annotations can be found at:
  - http://docs.oracle.com/javaee/7/api/javax/persistence/package-summary.html under annotation types summary

@Transient

- @Transient
  - Can be added to an attribute you do not want to make persistent.
  - For example:

```
@Transient
private int calculatedValue;
```

  - => NO field “calculatedValue” will be created in the table corresponding to the Entity.

Enumeration

- You often want a variable that can only take a value from a fixed set of possibilities, for example a weekday
- You can use an enumeration for this.

```
enum Lesson day {
    MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY}
```

  This defines a new type of Lesson Day, for variables that can only have a value listed between the curly brackets.

- MONDAY, TUESDAY,... we call enumeration constants
- Look up all the necessary information to know how to work with an Enumeration class:
  - https://docs.oracle.com/javase/tutorial/java/javaOO/enum.html
  - https://www.w3schools.com/java/java_enums.asp
  - ...

Enumeration

- Points of attention for enumeration in connection with persistence:
  - Make enum a separate Java class
  - Then use the enumeration in your Entity but tell it how to make the info persistent in the database
  - You do this as follows:

```
@Entity
public class Course {
. . .
  @Enumerated(EnumType.STRING)
  private Lesson day
```

@Temporal

- When you use a java.util.Date or java.util.Calender datatype you must use the @Temporal annotation.

For example.

```
@Temporal(TemporalType.DATE)
private Date start date;

@Temporal(TemporalType.DATE) = equivalent to java.sql.Date
@Temporal(TemporalType.TIME) = equivalent to java.sql.Time
@Temporal(TemporalType.TIMESTAMP) = equivalent to java.sql.Timestamp
```

- If you forget to mention this, you will not get error messages immediately, but later, especially when you want to use the dates you entered... It is often difficult to know where the error came from. So do not forget this!

@Embeddable and @Embedded

- In an OO application we often have more classes than there are tables in the database.
- Suppose we have the class Period (quarter and year) and the class ProjectPhase (name and duedate). When persisting, we want the Period to be written as part of the ProjectPhase table.

```
@Embeddable
public class Period {
  private int quarter, year;
  public Period () {
  }
 . . .
}
```

@Embeddable and @Embedded

```
@Entity
public class ProjectPhase
. . .
  @Embedded
  private Period duedate;
. . .
}
```