TP2 : Java persitent API (JPA)

JPA est une interface de programmation Java permettant aux développeurs d'organiser des données relationnelles dans des applications utilisant la plateforme Java.

Primary key

```
@Entity
public class Person{

private String identifier;

@id
public String getIdentifier(){
        return identifier;
}

public void set Identifier( String identifier){
        this. identifier = identifier;
}
}
```

A primary key automatically generated

```
@Entity
public class Article{

private Long id;

@Id
@GeneratedValue(strategy = GenerationType.AUTO)
public Long getId() {
        return id;
}

public void setId(Long id ){
    this.id = id;
}

}
```

Association Mapping

one to one ;
one to many ;
many to one ;
many to many ;
Les associations sont unidirectionnelles ou bidirectionnelles ;

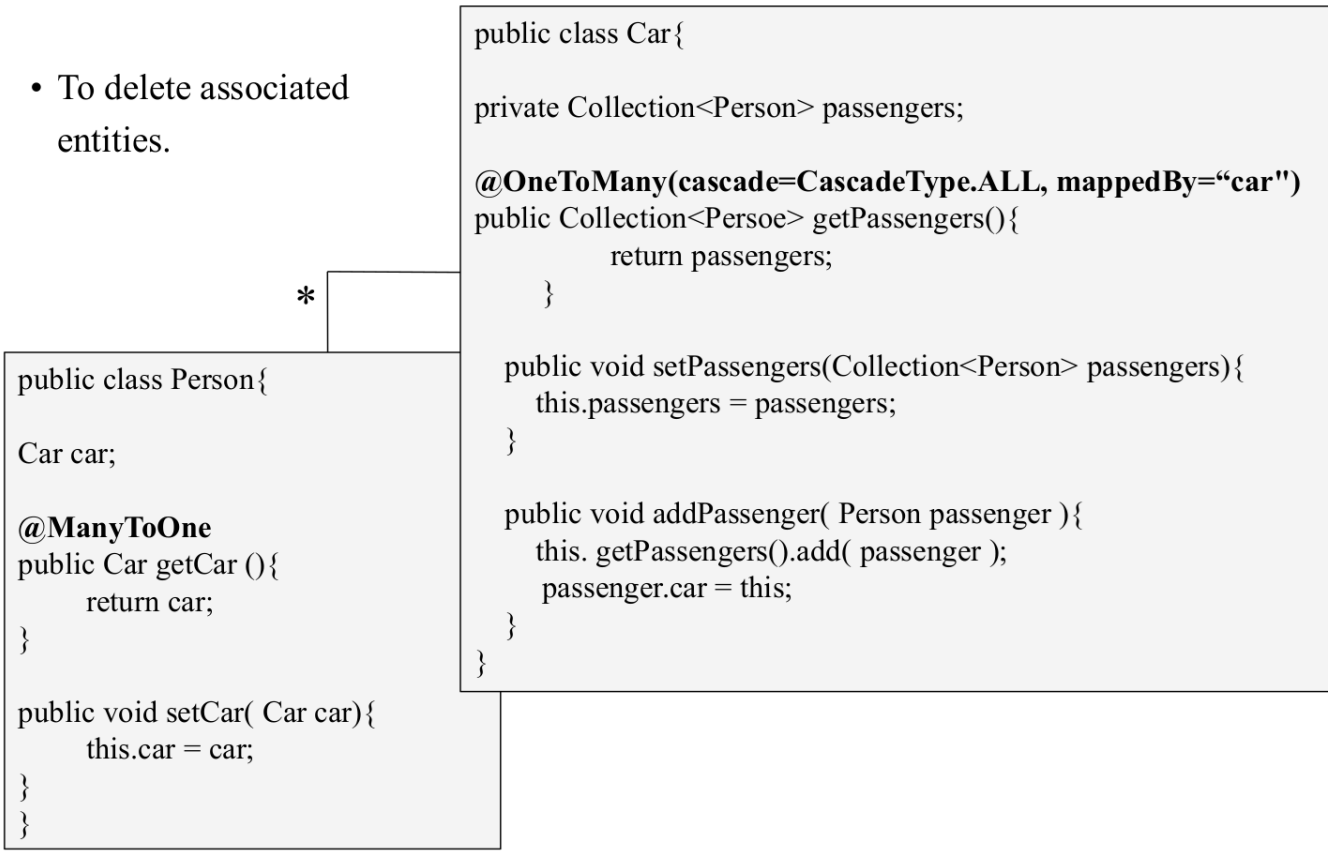
One to One

```
public class Person{

private Car car;

@OneToOne(mappedBy = “driver")
public Car getCar(){
        return car;
}

public void setCar( Car car ){
        this.car = car;
}
}
```

```
public class Car{

private Person driver;

@OneToOne
public Person getDriver(){
        return driver;
}

public void setDriver ( Person driver){
        this.driver = driver;
}
}
```

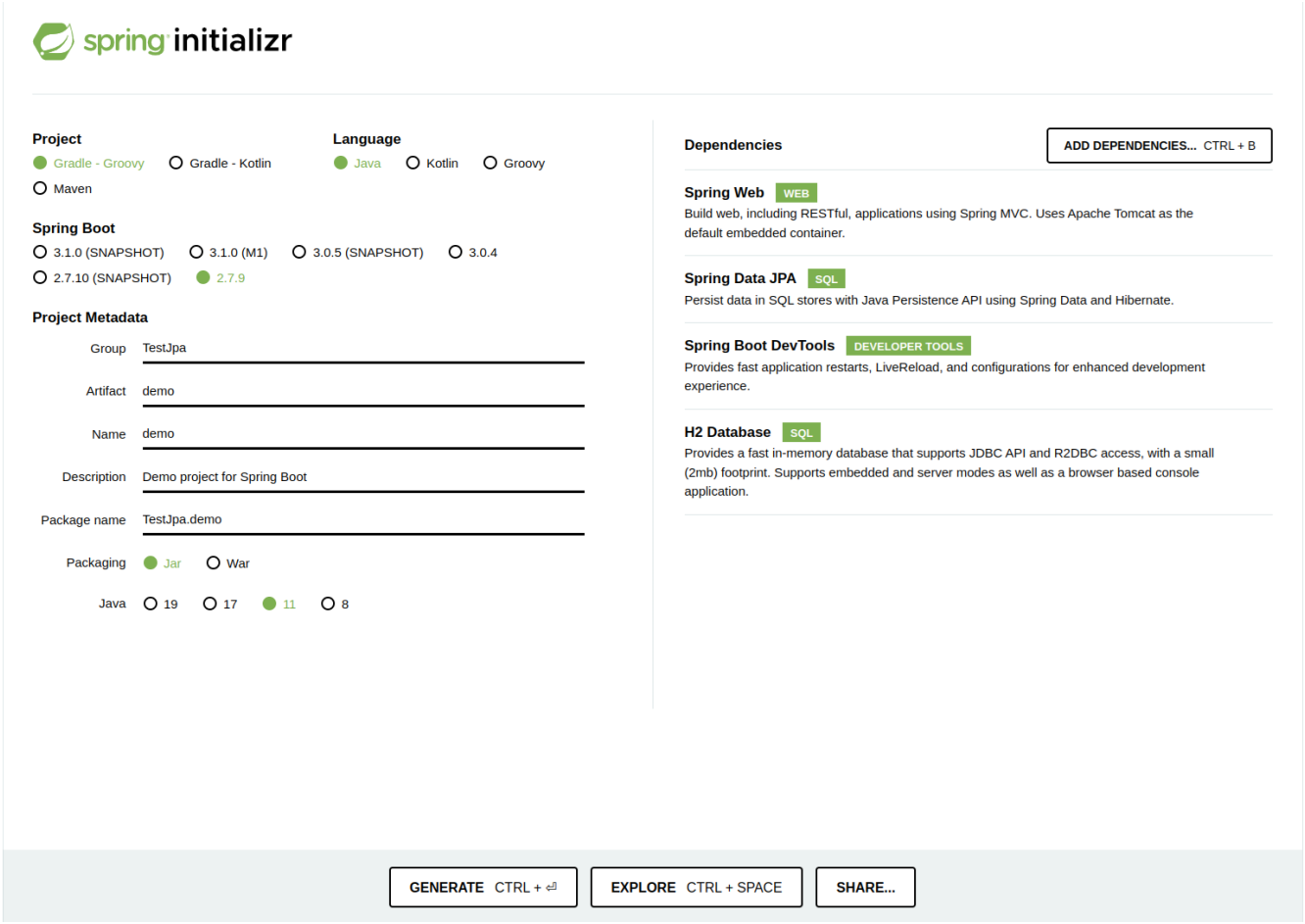
One to many

- To delete associated entities.



Travail à faire : Gestion des habitants d'une cité

1. Créer un projet avec Spring Initializer: https://start.spring.io/



2. Construire un diagramme de classe avec les entités City et Person.

Conditions: Une personne habite dans une cité.
Une cité peut habriter plusieurs personnes.

3. Créer une Entité Person avec les attributs suivants:

id: long
name: String
age: int
city: City (Entité à créer dans la question 4)

→ Définir un constructeur dans Person
→ Définir un constructeur pour initiliser uniquement les attributs name et age
→ Définir les Getters et Setters de Person
→ Surcharger la méthode toString
→ Faites que l'identifiant soit générer de manière automatique

→ Définisser la relation de Person avec l'entity City dans getCity à l'aide de la question 2.

4. Créer une entité City avec les attributs:

persons: List<Person>
id: long
name: String

→ Définir un constructeur par défaut de City
→ Définir un constructeur pour initialiser l'attribtu name
→ Définir les Getters et Setters de City
→ Surcharger la méthode toString
→ Générer l'identifiant de manière automatique
→ Définissez la relation de City par rapport à Person dans getPerson()

5. Créer une interface CityRepository qui hérite la classe CrudRepository<City, Long>

→ Définir dans l'interface CityRepository une méthode findByName de paramètre String name et de type de retour List<City>.

6. Créer un service RestWebService avec comme Comme attribut cityRepository de type CityRepository

→ Défnir un constructeur qui initialise l'attribut cityRepository.
→ cityRepository.findAll(): retourne la liste des cités de type Iterable<City>
cityRepository.findByName(name) retourne la liste des cités avec le nom name (List<City>)

A l'aide du TP1 (GetMapping) et des deux méthodes précédentes, créer une méthode getCities() qui retourne la liste des cités et getCitiesByName(@PathVariable("name") String name) qui retourne la liste des cités avec le nom name.

→ Compléter ce code pour rajouter une cité dans la base de données

```java
@PostMapping("/cities")
@Transactional(propagation = Propagation.REQUIRES_NEW, rollbackFor = Exception.class)
public Iterable<City> addCity(@RequestBody City city) throws Exception {
    ...
}
```

7. En vous inspirant du code en bas, créer des cités et des personnes.
   NB: Le code doit être mis dans le fichier principal après la méthode main.

```java
@Bean
public CommandLineRunner demo(CityRepository repository) {
    return (args) -> {

        // Définir la date
        SimpleDateFormat dateFormat = new SimpleDateFormat( pattern: "yyyy-MM-dd");
        Date date = dateFormat.parse( source: "2018-10-09");
        System.out.println(date.toString());

        // Créer une Cité
        City paris = new City( name: "Paris");
        Person tintin = new Person( name: "Tintin",  age: 20);
        paris.getPersons().add(tintin);
        tintin.setCity(paris);

        repository.save(paris);

        System.out.println("-------------------------------");
        System.out.println("Cities found with findAll():");
        for (City city : repository.findAll()) {
            System.out.println(city.toString());
        }

        System.out.println("-------------------------------");
        System.out.println("Persons associted with a city");
        Iterable<City> cities = repository.findAll();
        City c = cities.iterator().next();

        List<Person> persons = c.getPersons();
        System.out.println(persons.toString());

        System.out.println("-------------------------------------------");
        System.out.println("City found with findName('Paris'):");
        repository.findByName("Paris").forEach(city -> {
            System.out.println(city.toString());
        });

    };
}
```