***OneToOne Relationship mapping with JPA***

@Repository

Public interface AddressRepository extends JpaRepository<Address, Long>{

}

Note: you have 2 tables in models similar (the address and the Student)

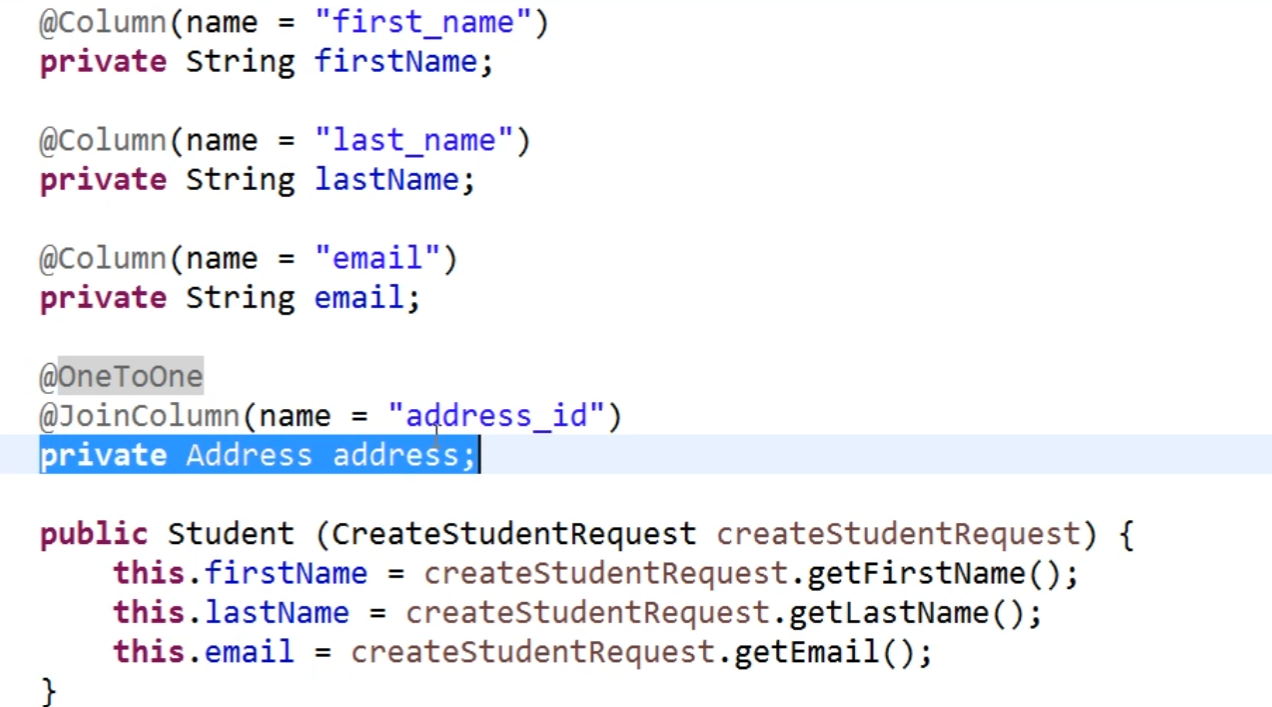
1. Address

A screenshot of a computer code

AI-generated content may be incorrect.

1. Student Table

Is just a class like the one above with the constructors and fields, each of the fields goes with the @Column annotation followed by the name. And create a field with the name of the class you want to share it, but you will use @OneToOne and @JoinColumn



***Insert record with OneToOne Relationship and JPA***

Inject the field in one of the classes with his own repository.

@Getter

@Setter

Public class StudentService{

@Autowired

StudentRepository studentRepository;

@Autowired

AddressRepository addressRepository;

**Note: in the create method of student create a new object of the other repository**

Public Student createStudent (CreateStudentRequest createStudentRequest){

Student student = new Student (createStudentRequest);

Address address = new Address();

address.setStreet(createStudentRequest.getStreet());

address.setCity(createStudentRequest.getCity());

address = addressRepository.save(address);

student.setAddress(address);

student = studentRepository.save(student);

return student;

}

}

***Join query with Spring Data JPA***

@Repository

List<Student> findByAddressCity(String city);

@Query(“From Student where address.city = : city ”)

List<student> getByAddressCity (String city);

@Service

Public List<Student> getByCity (String city){

Return studentRepository.findByAddressCity(city);

}

***Lazy loading in JPA***

On the relation @OneToOne use fetch, it means that the information from the related tables do not get automatically trigger when it check the principal entity. It will only charge when you access explicitly to the relation

@OneToOne(fetch = FetchType.LAZY)

***Bi-Directional OneToOne Relationship in JPA***

On the first relation declaration mark the @JoinColumn(name = “column name”) too, then move to the other table and create a field related to the previous table and add

@OneToOne(mappedBy = “address”) again

***One to Many relationships***

In a table create a field related to other table and use @ManyToOne relationship with @JoinColumn(name = “key”)

Example:

@ManyToOne

@JoinColumn(name = “student\_id”)

Private Student student;

Now you move to the related table that do not have the mark and use a related field

@OneToMany(mappedBy = “student”)

Private List<Subject> learningSubjects;

***Insert record with OneToMany relationship and JPA***

Note: remember for each entity class/ table we have a repository. In the service of the main related table on the creation request add a list of all “subjects”

A computer screen shot of a program

AI-generated content may be incorrect.

A computer screen shot of a program code

AI-generated content may be incorrect.