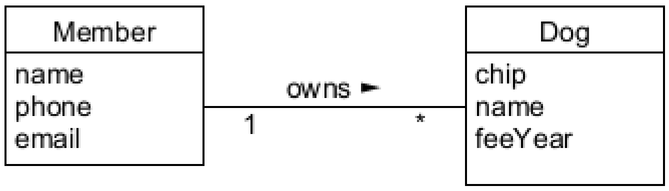
FFC - DAO Exercise

Recall the Fido Fitness system from the GUI Workshop. We are going to improve the system by adding the storage to the data in a database.

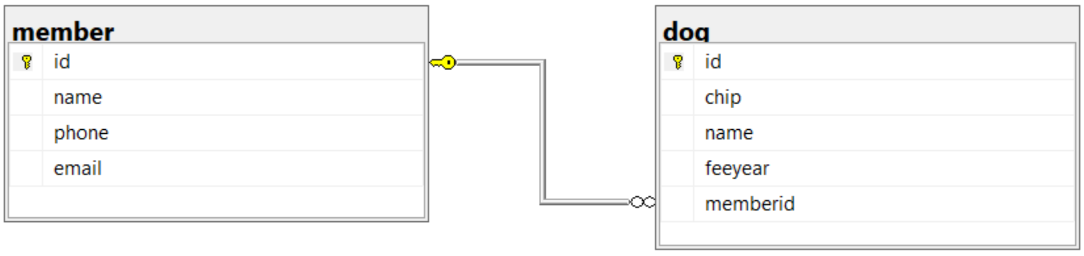
We consider the following **use cases** in prioritized order:

- CRUD Member - only create and read
- Register dog
- Update and unregister dog
- CRUD Member - update and delete

The system (part of it) is described by the following **domain model** (draft):



The database has the following schema. The id has been introduced in both the tables as surrogate keys.



The focus is on the model and database layers, but it is a good idea to implement the other layers too. Use unit testing to test your code.

1. PART

- Study the given SQL scripts that creates the database: **scripts/ffc.sql**.
- Execute them.
- Create a Java project. Remember to add the driver.
- Import/copy the code given to you (**src** folder) which contains the model layer and some classes of the data access layer.
- Update the DbConnection.java class with your credential.
- Create a package called test: add in the test package a test for checking the connection (see example); remember to add the JUnit libraries.

```
@Test
public void testGetConnection() {
   try {
      Connection c = DBConnection.getInstance().getConnection();
      assertNotNull(c);
   } catch (Exception e) {
      fail("issues with dbconnection");
   }
}
```

- Look at the given code and try to understand. Ask if something unclear!

- Create a new package called control.
- Add the class `MemberController` and implement the following methods by calling methods in the **dal** layer:
  - `public Member createMember(String name, String email, String phone)`
  - `public Member findByEmail(String email)`
  - `public List<Member> getAllMembers()`

Add some tests or a TryMe class to test everything is ok.

2. PART

You focus now on the register dog use case.
- Create the `DogDB` class that implements the interface `DogDBIF`.
- *You can create a TryMe class to directly call the methods and see everything is ok.*

- In `MemberDB`: update the `buildObject` method, so the method can retrieve the list of dogs if the `retrieveAssociation` parameter is true.

- In `MemberController` you can now implement the following methods
  - `public Member findByEmailWithDogs(String email)` that gets hold of the member together with the list of all his/her dogs
- Add the class `RegisterDogController`. The class needs to have an instance variable holding the current member. Implement the following methods:
  - `public Dog registerDog(String chip, String dogName, int feeYear)`
  - `public Member findByEmail(String email)`

- If you have time implement the GUI. Get inspiration from Workshop GUI

3. PART

- If you have time write the code for the rest of the use cases.