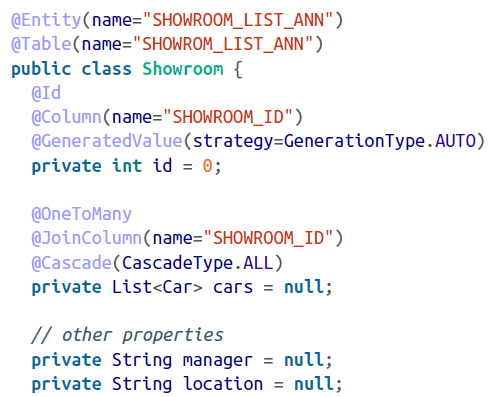
**Using a Foreign Key**

As we know, each showroom will have many cars, as represented by a one-to-many association. The *Showroom* entity consists of the collection of cars, showcasing them to customers. The *cars,* on the other hand, belong to a showcase; Hence, are modeled to have a foreign key relationship to the showroom.

Let’s first see the *Showroom* entity, which is defined as follows:

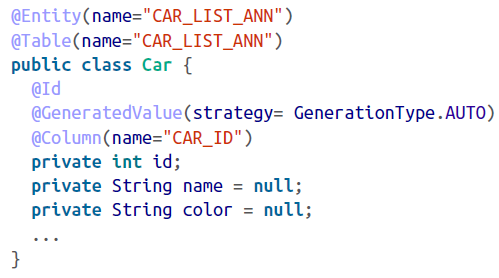


The class is declared as a persistable entity (via the @*Entity* annotation) mapping to a table identified with the @*Table* annotation. We define the identifier using an autogeneration strategy, meaning the identifier is set by one of the database’s functions, such as *auto\_increment* or *identity.*

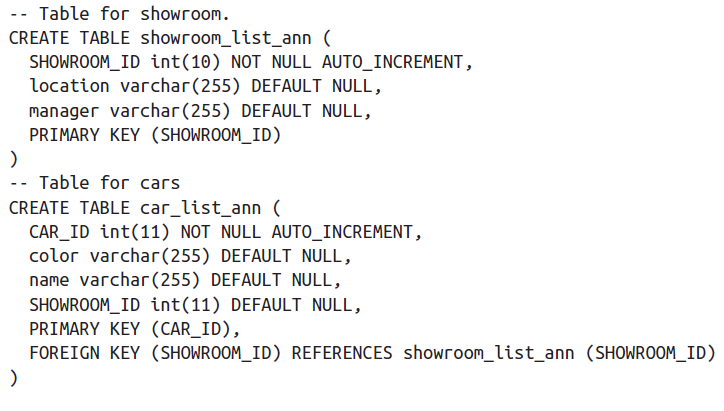
Let’s focus on one important property of the showroom: the collection of cars represented by a variable called *cars.* We use a java.util.List collection to hold the cars data. This variable is decorated with the @*OneToMany* annotation because each showroom will have many cars, and each car belong to a showroom.

To let Hibernate know about this dependency, we declare the *cars* variable along with an @*JoinColumn* annotation defining the foreign key. We must provide the column name SHOWROOM\_ID to pick up the list of cars from the cars table. The @*Cascade* annotation enables Hibernate to persist the collections associated with the main instance.

The Car entity is simple and straightforward:

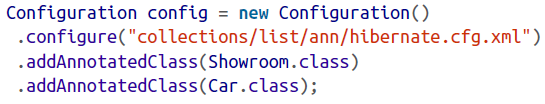


Here are the showroom and car database table scripts:



The cars table has its own primary key (CAR\_ID) plus a foreign key (SHOWROOM\_ID) referring to the main table.

Prepare your test case, adding the annotated classes to the configuration, as shown here.



As the client’s persist mechanism won’t be any different from what we saw when persisting lists, we will not repeat the test case here.

The output of the showroom is as follows:

