**Persisting Lists**

Lists are simple and easy data structure to hold item in an orderly manner. They can also keep track of an element’s position, using indexes. We can insert an element anywhere in the list, and retrieve the element using its index.

For example, let’s say we have persisted a list of manufacturers (Toyota, BMW, Mercedes, etc) via our Java Program. We should expect to retrieve the car data from the table in the same order in which it was inserted. So, if we run a query *list.get(n)*, We should get back the  *nth* element without fail.

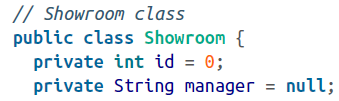
To satisfy this requirement, Hibernate maintains another table with the car’s indexes. So, when it fetches the cars from the main table, it also fetches the indexed order of these items from the additional table (called, say, CAR\_LIST). It will then associate and map these items together to find the order, and accordingly feed the client with the ordered responses.

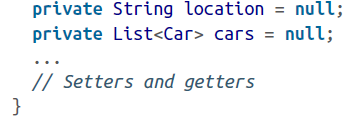
But enough theory. How can we persist our cars list using Hibernate? To get a better understanding, let’s see an example.

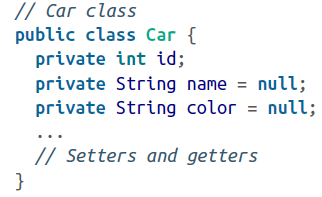
**List Example: Car Showroom**

Consider a simple case of a car showroom. A showroom consists of dozens of cars for customers to view and possibly purchase. We can represent with a simple model.

Every showroom will have a variety of cars to sell. Some cars may be brand new, while others are secondhand. We model these cars as *java.util.List*, as shown here in *Showroom’s* implementation (along with the Car definition):







The Showroom and Car classes are very simple POJOs. The only notable point is the declaration of a one-to-many association of Showroom with cars – that is, one showroom consists of many cars.

Once the POJOs are ready, we add the mapping definitions:



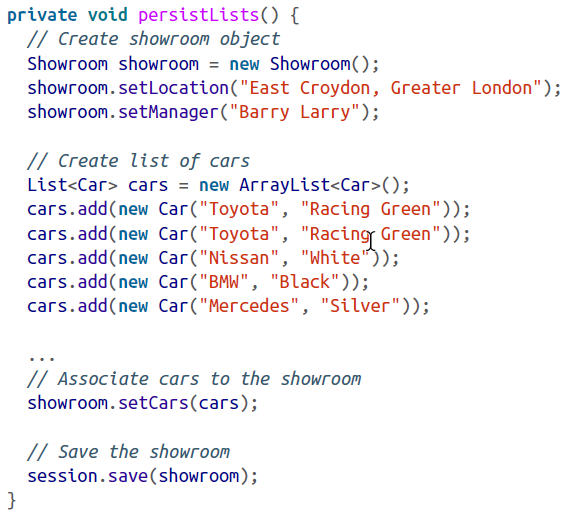
Notice the use of the *list* element in the preceding snippet. This element defines the mapping of the cars declared in the *Showroom* object to the table.

The main table, SHOWROOM\_LIST, will be created and populated as expected, so there are no surprises there. However, CARS\_LIST is an additional table gets created in the process. In addition to the CAR\_ID, NAME, and COLOR properties present in the CARS\_LIST table, which are directly declared on the object itself, Hibernate creates two other coloumns. One of them is a foreign key, SHOWROOM\_ID, while the other is the CAR\_INDEX column to hold the list indexes. CAR\_INDEX is populated with each list element’s position, and is used later to reconstruct the list elements in their original positions.

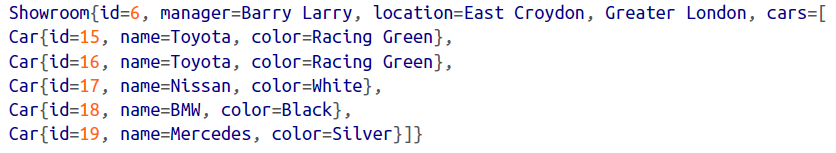
When retrieving the cars, at runtime, Hibernate recorders the records according to the index held in CAR\_INDEX. Let’s run the test client to check out how this works in practice.

**The Client for List Persistence**

Fire up a small test client to test the list persistence functionality, as shown here:



The test client is self-explanatory. Notice that I’m adding an extra Toyota to the list! When you run the test to retrieve the results, the following output is printed to the console (check out the duplicate Toyota cars too in the output!):



As you can see, Hibernate honors the insertion order of the cars in the list.