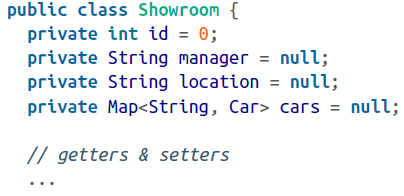
**Persisting Maps**

When you have a requirement to represent name/value pairs, your first choice should be Maps. The Map data structures are like dictionaries where you have a key (word) and related values (meanings). Maps are the de facto choice for key/value-paired data, such as bank accounts (value) of a single customer (key) or stock quotes for an issuer.

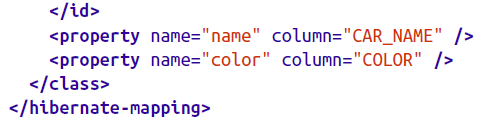
Continuing with our car showroom example, next we’ll add the capability for the showroom to hold potential customers’ reservations for test driving cars. We can best implement this functionality by employing a Map data structure, linking customers to car reservations:



Each car is reserved for a customer, and all the cars belong to the showroom. We can implement the customer-to-cars data type as a *Map<String, Car>* type.

The main meat is in the mapping, which is defined here:





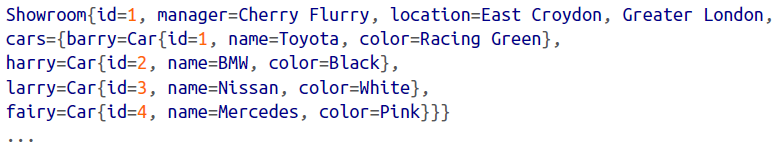
As expected, the showroom’s cars variable is represented by a *map* element referring to a table, CARS\_MAP, in the mapping definition. The map element defines a foreign key (SHOWROOM\_ID, in this case). The *map-key* attribute defines the key of the map – the customer, in our case. The car class mapping is a simple and straightforward one. Note that Hibernate would add a couple of more columns to the CARS\_MAP table – SHOWROOM\_ID and CUST\_NAME – in addition to the name and *color* columns*.*

Once the mapping is done, we need to demonstrate its working using a test client as follows:



Here we create a map with a customer name and the cars to test drive. We then attach them to the showroom. As you can see in our *Map* data structure, we have a brand new car corresponding to a customer.

As expected, the following output would be printed to the console if we run the *retriveMaps* method on the client:



The output shows all the cars in the showroom are reserved to customers for a test drive.