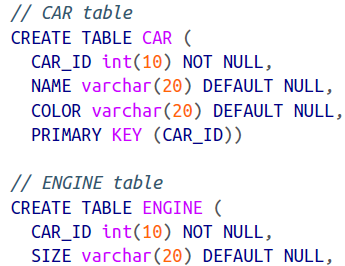
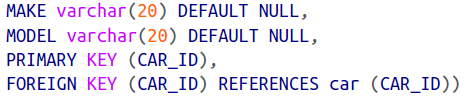
**Using a Primary Key**

Let’s consider the table schemas.

As mentioned earlier, we have couple of choices to express the one-to-one relationship: using either a primary key or a foreign key. For both options, while the Java classes remain the same, the table definitions differ.

The basic idea is both tables exhibit the one-to-one relationship by sharing the same primary key. The tables are designed to share the primary key, as shown in the following snippet:

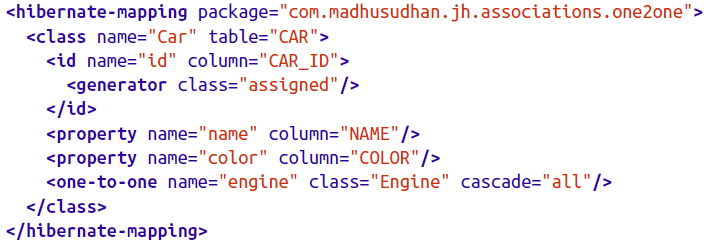




The *CAR* table is straightforward, with a CAR\_ID as the primary key. However, the interesting bits are in the ENGINE table. Two things you should have noticed there:

* The Primary key of the ENGINE is a CAR\_ID
* It has a foreign key constraint pointing to the primary key of the CAR table. So, an engine will always be created with the same *id* as that of a car. Thus, we say both tables share the same primary key.

Now let’s see how we do mapping for a one-to-one association. The mapping for the *Car* object against the *CAR* table is given here, in a *Car.hbm.xml file:*

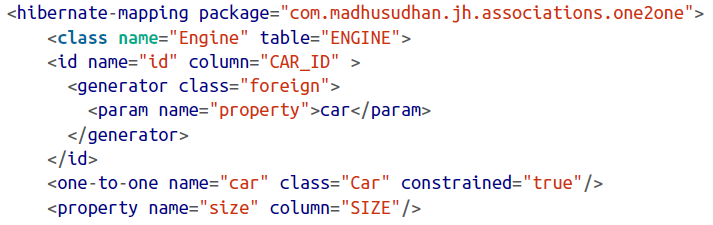


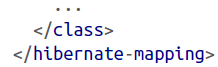
By now, this mapping should be familiar to you.

The *id* is set by the application, and the relevant properties are mapped to the table columns. Did you notice how the engine property was set? A one-to-one mapping tag is used to associate the engine to the *Car.* What this notation is truly saying is:

* The instance of the Car (it is defended for the *Car* Class) has a property called *engine.*
* *Car* and *Engine* exhibit a one-to-one association.
* *Engine* is set by values pulled from the ENGINE table (which is mapped to the *Engine* object).

The Engine mapping is a bit more involved, however:





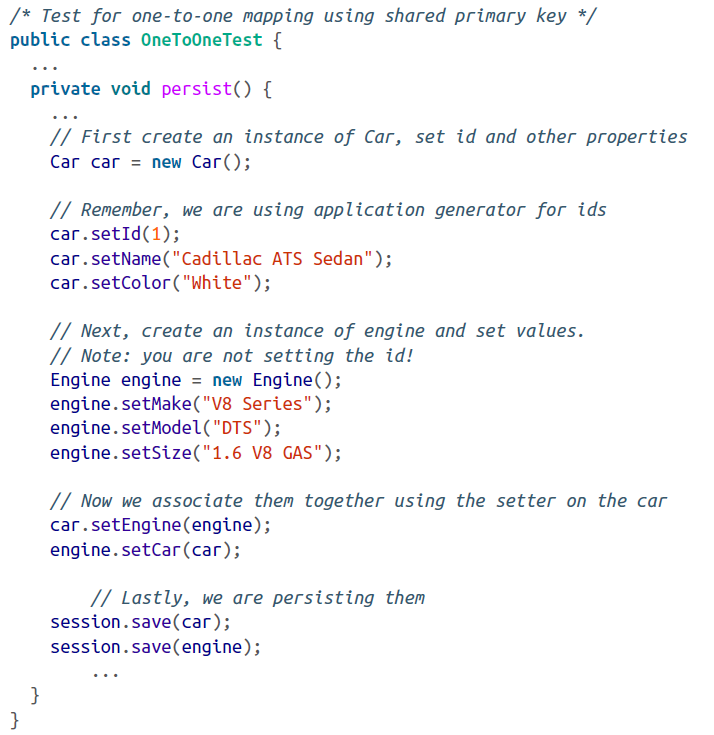
We know that Engine’s primary key (*id*) is the same as *Car’s id.* We should somehow mention this fact in our mapping to let Hibernate know how to deal with this situation. There is a special generator class called *foreign* for such purposes. This generator checks for a property named *Car* (which is defined via the one-to-one tag further down in the mapping) and picks up the *id* from that reference.

There’s one more attribute we’ve used in the one-to-one element: *constrained=”true”.* Simply put, this means that the primary key of the ENGINE table has a foreign key constraint, which is deduced from the primary key of the CAR table.

The hard work is done, so let’s test the application.

**Testing the Association**

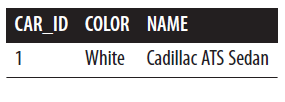
To demonstrate how this relationship works, we will create a test class as shown here:

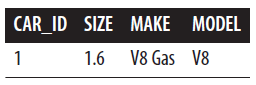


Although the code is doing lots of things, it is not difficult to grasp (read the comments to understand the operations). Note that the primary key of the *Engine* is not set on the object while we’re creating it. This is because we are borrowing and sharing the id from the *Car.* This way, during the process of creating the *Engine,* Hibernate grabs the *id* from the car and sets it onto the *Engine* object.

We are persisting both the car and engine in the preceding code. They are pretty much individual tables in this scenario except with primary-foreign key constraints. In our next case, we will see how the engine gets persisted by just persisting the car.

Below table show the output from the tables.





We can see the primary key *CAR\_ID*  is shared across both tables.