Exercise

We're going to build an O/RM (Object Relational Mapper). ORMs are tools responsible for mapping data between a relational database and an object-oriented model. There are many popular O/RMs out there such as Hibernate and Entity Framework.

You can see a "super simplified" implementation of our imaginary ORM in the **proxy** package.

We use the **DbContext** class to read or write data to our database. The **Demo** class shows the typical workflow for using a **DbContext**

- Read an object from a database (`dbContext.getProduct()`)
- Change the properties of the in-memory object (`product.setName()`)
- Ask the dbContext to save the changes (`dbContext.saveChanges()`)

If you run the code in the **Demo** class, you'll only see a SELECT statement printed on the terminal. This simulates reading a product record from a database.

What is missing is the two UPDATE statements that should be generated when we save the changes. The reason this is happening is that our **DbContext** cannot keep track of the changed objects. So, when we call **saveChanges()**, nothing happens.

We can solve this problem using the proxy pattern. A proxy object looks like our target object (eg a Product object) but it adds some extra behavior to it. For example, when we call the **setName**() method, the proxy notifies the **DbContext** that it is changed. **DbContext** provides a method for this: **markAsChanged()**.

Use the proxy pattern to allow **DbContext** keep track of changed objects and persist them in a database.

NOTE: Some real-world O/RMs automatically generate these proxy objects based for you, so you don’t need to code them by hand.