CHAPTER 9

Writing to the ODMG 3.0

When on board H.M.S. Beagle, as naturalist, I was much struck with certain facts in the distribution of the organic beings inhabiting South America, and in the geological relations of the present to the past inhabitants of that continent. These facts, as will be seen in the latter chapters of this volume, seemed to throw some light on the origin of species—particularly I want to address an object-relational specification that has been a contributor to many other frameworks and systems, the Object Data Management Group. This in turn will throw some light on other frameworks that I will cover in the next chapters.

The Object Data Management Group (ODMG) is a committee that was formed in 1991 to create a specification that would allow developers to write objects that could be persisted transparently: independent of language and independent of mechanism. The ODMG’s belief is that a specification can be defined that both an object database and an object-to-relational layer can implement.

This would allow software to be developed that could change from a relational database storage mechanism to object database storage. The reverse would also be true. Essentially the goal is write once, store anywhere. By writing applications to the ODMG, a system should not be bound to one storage mechanism.

In order to achieve this goal, the ODMG has come up with several specifications. The specifications are all grouped together under the ODMG 3.0 standard, and they are implemented by many object databases and object-relational frameworks. The ODMG is also a reference for technologies like Java Data Objects (JDO) and similar technologies.

For this chapter, since we are discussing the ODMG, and the underlying storage mechanism should be irrelevant, I am going to primarily refer to Object Data Management Systems (ODMSs). By this, I mean that your objects will be stored in either a relational database or object database. The ODMG frees you from worrying about which it is. For the code in this chapter, I used both the commercial Poet FastObjects object database and the open source Jakarta ObjectRelationalBridge (OJB) product, which I cover in more detail in Chapter 11.

I will not go too far in depth into the ODMG because it is a large topic and implementations vary a great deal. For more information on the ODMG, I suggest going to their Web site at http://www.odmg.org. The ODMG specification is also available in book form from their site. The reason I want to address the ODMG is not only because it can be useful for Java persistence, but also because many persistence products are either based on it or claim compatibility with it.
Object Model

The ODMG's object model is based on the model defined by the Object Management Group (OMG), which is the creator of the Common Object Request Broker Architecture (CORBA) and is intended to be compatible with that specification. What this means is that they took a well-established object model (CORBA) and modified it as little as possible to use it for persistence. This in turn gives developers an easy path to incorporate this technology into their already existing distributed applications.

The object model really specifies how objects can relate to each other and how they interact. In addition, things like relationships, locking, and transactions are also defined in the ODMG's object model. This is particularly important to the ODMG since the specification is used across more languages than Java. An object stored using ODMG in Java should theoretically be retrievable from C++ or Smalltalk.

How objects relate and interact is already taken care of for you by the Java language itself. And this relationship is compatible with the ODMG's requirements. One object can contain references to another. You can use references to objects' call methods on other related objects to perform work.

As an example, Figure 9-1 shows an OMDG object model that contains an Employee class with two subclasses, Manager and Developer. The Java object model is exactly the same. That is because Java naturally supports the basic object model.

Inheritance works the same way in the OMDG's object model as in Java's. Figure 9-2 illustrates that an interface can inherit from any number of interfaces but cannot inherit from a class. A class, however, can inherit not only from any number of interfaces, but also from one other class. This is the same as Java's normal inheritance model.