An Introduction to Hibernate 3.5

Most significant development projects involve a relational database. The mainstay of most commercial applications is the large-scale storage of ordered information, such as catalogs, customer lists, contract details, published text, and architectural designs.

With the advent of the World Wide Web, the demand for databases has increased. Though they may not know it, the customers of online bookshops and newspapers are using databases. Somewhere in the guts of the application a database is being queried and a response is offered.

While the demand for such applications has grown, their creation has not become noticeably simpler. Some standardization has occurred around the Java Persistence API with the release of Enterprise Java Beans 3.0. Hibernate 3.5 is an implementation of the Java Persistence API standard, which replaced older Java persistence solutions such as the entity beans from Enterprise Java Beans 2.

There are solutions for which some sort of object-relational mapping (ORM) like Hibernate is appropriate, and some for which the traditional approach of direct access via the Java Database Connectivity (JDBC) API is appropriate. We think that Hibernate represents a good first choice, as it does not preclude the simultaneous use of these alternative approaches.

To illustrate some of Hibernate’s strengths, in this chapter we will show you a brief example using Hibernate and contrast this with the traditional JDBC approach.

Plain Old Java Objects (POJOs)

In our ideal world, it would be trivial to take any Java object and persist it to the database. No special coding would be required to achieve this, no performance penalty would ensue, and the result would be totally portable.

In this ideal world, we would perhaps perform such an operation in a manner like that shown in Listing 1-1.

Listing 1-1. A Rose-Tinted View of Object Persistence

```java
POJO pojo = new POJO();
ORMSolution magic = ORMSolution.getInstance();
magic.save(pojo);
```

There would be no nasty surprises, no additional work to correlate the class with tables in the database, and no performance problems.
Hibernate comes remarkably close to this, at least when compared with the alternatives—but alas, there are configuration files to create and subtle performance issues to consider. Hibernate does, however, achieve its fundamental aim—it allows you to store POJOs in the database. Figure 1-1 shows how Hibernate fits into your application between the client code and the database.

**Figure 1-1. The role of Hibernate in a Java application**

The common term for the direct persistence of traditional Java objects is object-relational mapping—that is, mapping the objects in Java to the relational entities in a database.

POJOs can be any Java object at all. Hibernate allows you to persist POJOs with very few constraints. Listing 1-2 is an example of a simple POJO to represent a message.

**Listing 1-2. The POJO Used in this Chapter’s Examples**

```java
public class Message {
    private Message() {
    }

    public Message(String messageText) {
        this.messageText = messageText;
    }

    public String getMessageText() {
        return messageText;
    }
}
```