CHAPTER 2

Integrating and Configuring Hibernate

Compared to other Java persistence solutions, integrating Hibernate into a Java application is easy. The designers of Hibernate avoided some of the more common pitfalls and problems with the existing Java persistence solutions, and created a clean but powerful architecture. In practice, this means that you do not have to run Hibernate inside any particular J2EE container or framework—Hibernate 3.5 only requires Java 2 Standard Edition (J2SE), version 5.0 (or later).

At first, adding Hibernate to your Java project looks intimidating—the distribution includes a large set of libraries. To get your first Hibernate application to work, you have to set up the database, the mapping files or annotations, the configuration, and your plain old Java objects (POJOs). After you have done all that, you need to write the logic in your application that uses the Hibernate session to actually do something! But once you learn how to integrate Hibernate with your application, the basics apply for any project that uses Hibernate.

If you already have an application that uses Hibernate 2, the migration path from Hibernate 2 to Hibernate 3.5 is easy. While Hibernate 3.5 is not completely backward-compatible, most of the changes are additional features that you can integrate into your existing application as you see fit. The Hibernate developers provided implementations of the core Hibernate 2 objects in Hibernate 3.5 with the Hibernate 2 methods for backward compatibility. We discuss the differences between Hibernate 2 and Hibernate 3.5 in more depth in Appendix D.

One of the key features of Hibernate’s design is the principle of least intrusiveness—the Hibernate developers did not want Hibernate to intrude into your application more than was necessary. This led to several of the architectural decisions made for Hibernate. In Chapter 1 you saw how Hibernate can be applied to solve persistence problems using conventional Java objects. In this chapter, we explain some of the configuration details needed to support this behavior.

The Steps Needed to Integrate and Configure Hibernate

This chapter explains configuration and integration in detail, but for a quick overview, refer to the following bulleted list to determine what you need to do to get your first Hibernate application up and running. Chapter 3 leads you through the building of a pair of small example applications that use Hibernate. The first of these is as simple as we could make it, so it is an excellent introduction to the following necessary steps:
1. Identify the POJOs that have a database representation.
2. Identify which properties of those POJOs need to be persisted.
3. Annotate each of the POJOs to map your Java object’s properties to columns in a database table (covered in more detail in Chapter 7).
4. Create the database schema using the schema export tool, use an existing database, or create your own database schema.
5. Add the Hibernate Java libraries to your application’s classpath (covered in this chapter).
6. Create a Hibernate XML configuration file that points to your database and your mapped classes (covered in this chapter).
7. In your Java application, create a Hibernate Configuration object that references your XML configuration file (covered in this chapter).
8. Also in your Java application, build a Hibernate SessionFactory object from the Configuration object (covered in this chapter).
9. Finally, retrieve Hibernate Session objects from the SessionFactory, and write your data access logic for your application (create, retrieve, update, and delete).

Don’t worry if you don’t understand every term or concept in the preceding list. After reading this chapter, and then going through the example in the next chapter, you will know what these terms mean and how they fit together.

**Understanding Where Hibernate Fits in Your Java Application**

You can call Hibernate from your Java application directly, or you can access Hibernate through another framework. You can call Hibernate from a Swing application, a servlet, a portlet, a JSP page, or any other Java application that has access to a database. Typically, you would use Hibernate to either create a data access layer for an application or replace an existing data access layer.

Hibernate supports Java Management Extensions (JMX), J2EE Connector Architecture (JCA), and Java Naming and Directory Interface (JNDI) Java language standards. Using JMX, you can configure Hibernate while it is running. Hibernate may be deployed as a JCA connector, and you can use JNDI to obtain a Hibernate session factory in your application. In addition, Hibernate uses standard Java Database Connectivity (JDBC) database drivers to access the relational database. Hibernate does not replace JDBC as a database connectivity layer—Hibernate sits on a level above JDBC.

In addition to the standard Java APIs, many Java web and application frameworks now integrate with Hibernate. Hibernate’s simple, clean API makes it easy for these frameworks to support Hibernate in one way or another. The Spring framework provides excellent Hibernate integration, including generic support for persistence objects, a generic set of persistence exceptions, and transaction management. Appendix C explains how Hibernate can be configured within a Spring application.

Regardless of the environment that you are integrating Hibernate into, certain requirements remain constant. You will need to define the configuration details that apply—these are then represented by a Configuration object. From the Configuration object, a single SessionFactory object is created; and from this, Session objects are instantiated, through which your application accesses Hibernate’s representation of the database.