Introduction to Hibernate/JPA

- Object/Relational Mapping
- Hello, World in Hibernate

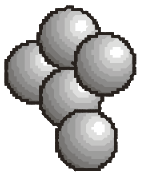
Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is Object/Relational Mapping?

- Object/relational mapping is the act of connecting data from a relational database with objects in an object-oriented language like Java

Object Space





Relational Space

STUDENT

STUDENTID	NAME	DATE
5	Jess Settles	1998-12-02
6	George Peeples	1966-05-19
7	Ben McGilmer	1965-03-13

Object orientation is a useful programming paradigm that facilitates reuse and easier maintenance. However, most of the world's data currently resides within relational databases managed by products like Oracle, DB2 and so forth.

O/R mapping lets developers model using object-oriented techniques while accessing/storing data in a relational database.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

O/R Techniques for Java

- Roll your own JDBC
- EJB2 entity beans with container-managed persistence
- MyBatis
- JPA
- Hibernate





Java programmers have several options to choose from when considering O/R mapping. One approach is to write the JDBC yourself -- while this gives the most flexibility and potentially best performance, it's code-intensive and tedious.

Another approach is to use entity Enterprise JavaBeans. Unfortunately, many architects and developers have decided that in EJB Version 2.0 and prior, entity beans are most often too heavyweight, have a steep learning curve and require too many associated files (e.g. primary-key classes, home interfaces, deployment descriptors and so forth). While this perception may not be entirely accurate (and is mostly addressed in EJB 3.0), the perception has hampered the acceptance of entity beans.

The final choice presented here is to use a light-weight O/R mapping framework. Such frameworks typically generate the tedious JDBC code, while being easier to code than entity beans. Two popular frameworks are iBATIS and Hibernate -- we cover Hibernate in this chapter.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The Impedance Mismatch

- Objects have state and behavior, while rows in a database are only state
- Most object-oriented languages support inheritance, but most relational databases do not
- Relational databases enforce **referential integrity** while objects do not
- Rows in a table usually have a distinct identity (primary key), while objects do not
- Relational databases have a query language (SQL), but there is no widely accepted equivalent in the object world

Why is O/R mapping a difficult process? The answer is that the two computing models are fundamentally different and considerable care must be taken converting between them. This page lists a few of the issues.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

O/R Issues

- Top down, bottom-up or "meet in the middle"?
- Caching of queries and objects
- Lazy instantiation of related objects
- Plain Old Java Objects (POJOs) versus inheritance-based frameworks

Here are some criteria to consider when evaluating an O/R approach.

The caching and lazy instantiation criteria have to do with performance, which is usually critical in O/R applications.

The reason that POJOs are advantageous is that they can more easily be reused and are easier to test. For example, since EJB2 entity beans are NOT POJOs, it's difficult to test them using a framework such as JUnit (the EJBs require the container runtime).

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to Hibernate

- Hibernate is a mature, open-source O/R persistence and query service for Java and .NET
- Hibernate's goal is to relieve developers from writing much of the tedious code required to access relational data using manual SQL and JDBC techniques
- **www.hibernate.org** is Hibernate's home page on the Web



The Hibernate project was started in 2001 and consists of over 76,000 lines of Java code and is licensed under the LGPL. Hibernate.org estimates that there are tens of thousands of Java projects worldwide using Hibernate.

The home page for Hibernate is www.hibernate.org. From there, you can download the distribution, browse or download the documentation or read discussion forums.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Hibernate Features

- Transparent persistence using POJOs
- Queries using the Hibernate Query Language, the JPA Object Query Language, or using standard SQL
- Rich support for object relationships
- Straightforward mapping between object space and relational space using either XML, annotations or both
- Complete support for transactions

You can read about some Hibernate tools at http://www.hibernate.org/255.html.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Database Independence using Dialects

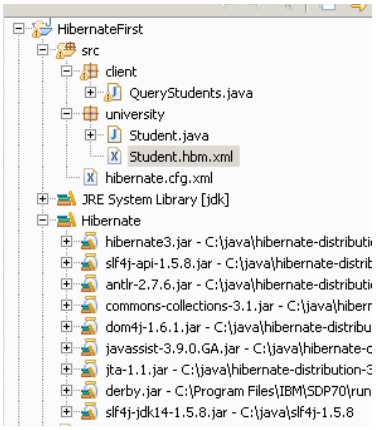
- Every database has its own idiosyncrasies -- Java programmers use the Data Access Object pattern to isolate the business layer from the data layer
- Hibernate extends this notion to define **dialect** classes, which encapsulate the differences in common databases including Oracle, DB2, PostgreSQL, HypersonicSQL, MySQL and others
- It's also possible to write custom dialects for other databases not directly supported by Hibernate

A Hibernate dialect is actually a class that extends the net.sf.hibernate.dialect.Dialect class. Hibernate3 ships with 16 dialects including DB2Dialect, HSQLDialect, MySQLDialect, Oracle9Dialect and PostgreSQLDialect.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Installing Hibernate

- Hibernate consists of a JAR file and a set of dependent JARs
- The JARs must be placed in the application's CLASSPATH along with a few configuration files
- For Web applications, the JARs must reside in the application's **WEB-INF/lib** directory



You can find most of the libraries in the Hibernate Core distribution.

Note that some application servers, including JBoss, ship with Hibernate -- in that case, applications do not need to configure any JARs into the CLASSPATH.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

A Simple POJO Entity

- Hibernate maps Java objects to rows in a database table
- The classes of the Java objects don't need any special inheritance or anything Hibernate-specific
- These classes often have something that makes each object unique from all others of its class

```
1    package university;
2
3    public class Student
4    {
5      private int studentID;  // unique across Students
6      private String name;
7      private double gpa;
8
9      // get/set methods, etc.
10   }
```

2 - 10

In this Student class, the "studentID' field will have a unique value in each Student object - this is a requirement so that Hibernate can map the studentID to a primary-key column in the database.

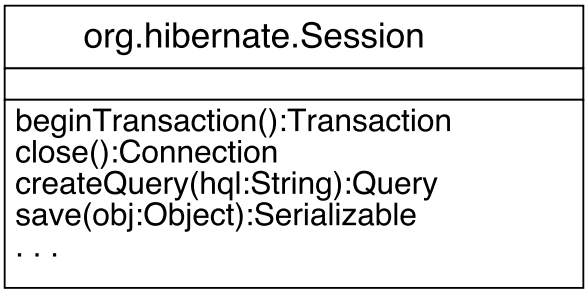
Most often, these classes are written in the form of a JavaBean, with standard get/set methods for the object's properties. Hibernate does require that the classes define a public, zero-argument constructor.

Since these classes don't have anything that ties them to Hibernate, you can easily unit-test them.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Hibernate Sessions

- The Hibernate **session** acts as a persistence manager and represents a set of objects for a particular unit of work
- The **org.hibernate.Session** interface is the main runtime interface between the application and Hibernate
- Applications must open a session to access persistent objects and should close the session when finished



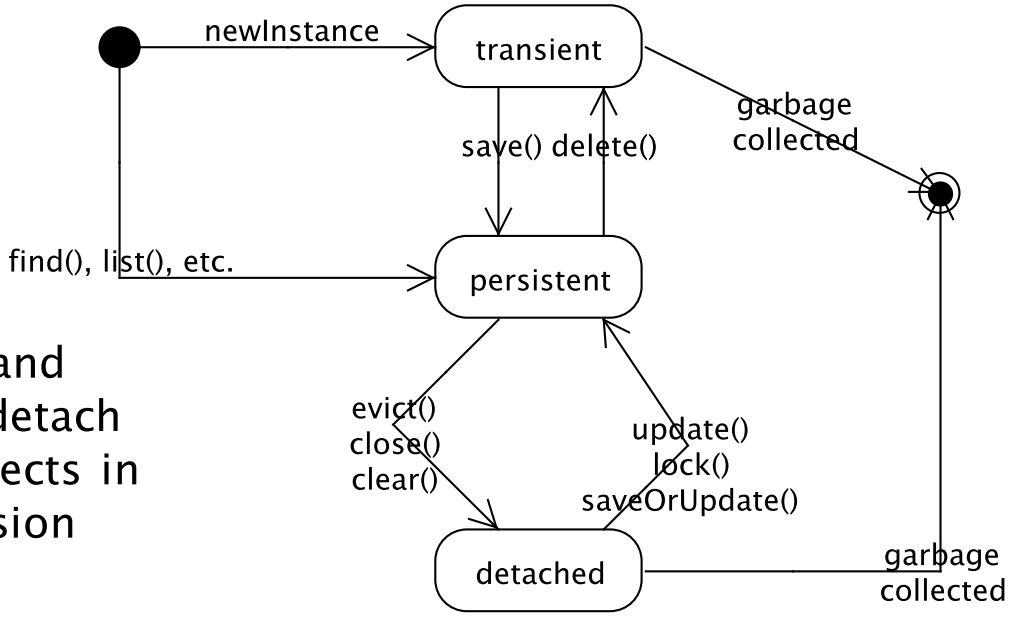
The Hibernate session is the primary API that applications use to save and load persistent objects. Do not confuse it with an HTTP session, which performs a different function.

In addition, Hibernate applications also typically use Hibernate transactions to organize units of work that can be committed or rolled-back as a group.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Entity States

- Hibernate entities transition through the **transient**, **persistent** and **detached** states



A "transient" object is not yet known by Hibernate, and its state will not be synchronized with the database. To make a new object persistent, you call the save() method on the Hibernate session.

A "persistent" object is managed by Hibernate and has an identity in the database.

A "detached" object was previously persistent, but is not currently managed by Hibernate. It still exists as a plain Java object, and could be re-attached to be a persistent object.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Opening and Closing a Session

- One way to open a session is to use **org.hibernate.SessionFactory**
- The SessionFactory is thread-safe and should be shared amongst all the parts of your application (the Session itself is NOT thread-safe)

```
1    SessionFactory sessionFactory =
2     new Configuration().configure().buildSessionFactory();
3    Session session = sessionFactory.openSession();
4
5    . . .
6
7    session.close();
8    sessionFactory.close();
```


Here we show simple code to open and close a Hibernate session. Many real-world Hibernate applications store the Session either in a thread-local variable or in JNDI so it can more easily be shared.

The org.hibernate.cfg.Configuration class looks for Hibernate configuration files in well known locations, then parses them and configures the session factory. Note that you can place the configuration files anywhere you want, but here we are assuming the default, well-known location.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Storing an Object

- To store an object, create and initialize it in memory, then use the session to save the object
- Saving also assigns an identity to the object

```
1    Transaction tx = session.beginTransaction();
2
3    Student s = new Student();
4    s.setGpa(2.25);
5    s.setName("Lou S.Y. Student");
6    session.save(s);
7
8    System.out.println("New student's ID: " +
9      s.getStudentID());
10
11   tx.commit();
```


Here we show the code to persist an object. We first create the object, which is a simple POJO, and then initialize its properties. To persist the object, we use the Hibernate Session's "save" method, which inserts a row into the associated table and assigns a unique identity to the object. This code demonstrates retrieving the identity, which in this case is an Integer property.

Note that we perform the insert within a Hibernate transaction.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Finding Objects

- You can easily find objects using the Hibernate Query Language and then use standard Java iteration techniques

```
1    List<Student> students =
2        session.createQuery("FROM Student").list();
3
4    for(Student s : students)
5    {
6       System.out.println(s);
7    }
```


Here we show code to retrieve a list of Student objects using an HQL query. The query itself is "from Student" in line 4 -- since we didn't specify any predicates, this query retrieves all Students. In reality, it actually queries all rows from the associated database table and then creates and initializes a Student object from each row.

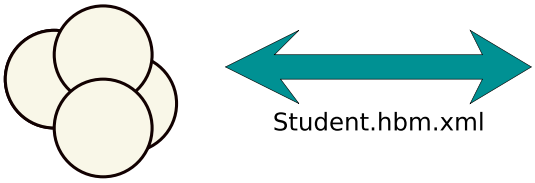
Note the obvious lack of any SQL or JDBC code!

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Mapping Objects to Tables

- To map object properties to a table, you must write a Hibernate configuration file either as a property file or as an XML file
- Most Hibernate applications use XML

Student POJO objects



STUDENT table (STUDENTID is primary key)

STUDENTID	NAME	GPA
1	Paul Westerberg	3.44
2	Joan Jett	3.22

Hibernate requires you to provide a file that maps properties from each POJO class into a database table. In many cases, you can use tools to help you write the configuration file, or they're not that hard to write from scratch. Most developers place the configuration files for each class in the same file system directory (i.e. package) as the class in question.

You can name these files anything you want, but many Hibernate developers use the naming convention Classname.hbm.xml.

At runtime, the configuration file must available to Hibernate, so if it's a Web application, you must include the configuration file in the application's WAR.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Mapping Objects to Tables, cont'd

- In the mapping file(s), Hibernate provides a rich syntax to map properties in the POJO to tables and rows in the database
- Hibernate provides support for generating values such as primary keys, including the simple auto-increment strategy shown here

```
1   <hibernate-mapping>
2     <class name="university.Student" table="STUDENT">
3       <id name="studentID" column="STUDENTID">
4         <generator class="increment" />
5       </id>
6       <property name="name" column="NAME" />
7       <property name="gpa" column="gpa" />
8     </class>
9   </hibernate-mapping>
```

STUDENT table (STUDENTID is primary key)

STUDENTID	NAME	GPA
1	Paul Westerberg	3.44
2	Joan Jett	3.22

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring Hibernate

- To specify connection properties and other configuration settings, you must write a Hibernate configuration file

```
1    hibernate.cfg.xml
2    -----------------
3
4    <hibernate-configuration>
5      <session-factory>
6        <property name="hibernate.dialect">
7          org.hibernate.dialect.DerbyDialect
8        </property>
9
10        . . .
11
12       <mapping resource="university/Student.hbm.xml" />
13     </session-factory>
14   </hibernate-configuration>
```

In addition to the per-class configuration file shown earlier, you must provide a configuration file that configures Hibernate for the datasource that contains the table(s) that are mapped to Java classes.

As with the other configuration file, this file must be available to Hibernate at runtime. The Configuration object shown earlier looks for a file named "hibernate.cfg.xml" in the application's CLASSPATH, so you use that name, you don't need to specify it anywhere in the code.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Hibernate Example

createstudents.sql

Student.java

CreateStudent.java

QueryStudents.java

hibernate.cfg.xml

Student.hbm.xml

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    DROP TABLE Student;
2
3    CREATE TABLE Student
4    (
5       StudentID   BIGINT NOT NULL,
6       Name        VARCHAR(200),
7       GPA         FLOAT,
8       CONSTRAINT  PK_Item  PRIMARY KEY(StudentID)
9    );
10
11
12   INSERT INTO Student VALUES (1, 'Paul Westerberg', 3.44);
13   INSERT INTO Student VALUES (2, 'Joan Jett', 3.22);
14
15   COMMIT;
```

```
1    package university;
2
3    public class Student
4    {
5        private int studentID;
6        private String name;
7        private double gpa;
8
9        public double getGpa()
10       {
11           return gpa;
12       }
13
14       public void setGpa(double gpa)
15       {
16           this.gpa = gpa;
17       }
18
19       public String getName()
20       {
21           return name;
22       }
23
24       public void setName(String name)
25       {
26           this.name = name;
27       }
28
29       public int getStudentID()
30       {
31           return studentID;
32       }
33
34       public void setStudentID(int studentID)
35       {
36           this.studentID = studentID;
37       }
38
39       @Override
40       public String toString()
41       {
42           return "Name: " + name +
43                  " ID: " + studentID +
44                  " GPA: " + gpa;
45       }
```

```
46
47    }
```

```
1    package client;
2
3    import java.util.Scanner;
4
5    import org.hibernate.Session;
6    import org.hibernate.SessionFactory;
7    import org.hibernate.Transaction;
8    import org.hibernate.cfg.Configuration;
9
10   import university.Student;
11
12   public class CreateStudent
13   {
14       public static void main(String[] args)
15       {
16           Scanner scanner = new Scanner(System.in);
17           System.out.print("Enter student name: ");
18           String name = scanner.nextLine();
19           System.out.print("Enter GPA: ");
20           String sGpa = scanner.nextLine();
21
22           SessionFactory factory =
23               new Configuration().configure().buildSessionFactory();
24           Session session = factory.openSession();
25           Transaction tx = session.beginTransaction();
26
27           Student s = new Student();
28           s.setGpa(Double.parseDouble(sGpa));
29           s.setName(name);
30           session.save(s);
31
32           System.out.println("New student's ID: " + s.getStudentID());
33
34           tx.commit();
35           session.close();
36           factory.close();
37       }
38   }
```

```
1    package client;
2
3    import java.util.List;
4
5    import org.hibernate.Session;
6    import org.hibernate.SessionFactory;
7    import org.hibernate.Transaction;
8    import org.hibernate.cfg.Configuration;
9
10   import university.Student;
11
12   public class QueryStudents
13   {
14
15       public static void main(String[] args)
16       {
17           SessionFactory factory =
18               new Configuration().configure().buildSessionFactory();
19           Session session = factory.openSession();
20
21           List<Student> students =
22               session.createQuery("FROM Student").list();
23
24           for(Student s : students)
25           {
26               System.out.println(s);
27           }
28
29           session.close();
30           factory.close();
31       }
32   }
```

```
1    <!DOCTYPE hibernate-configuration
2    SYSTEM "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
3    <hibernate-configuration>
4        <session-factory>
5            <property name="hibernate.connection.driver_class">
6                org.apache.derby.jdbc.EmbeddedDriver
7            </property>
8            <property name="hibernate.connection.url">
9                jdbc:derby:c:\springclass\dbhib\StudentDB
10           </property>
11           <property name="hibernate.dialect">
12               org.hibernate.dialect.DerbyDialect
13           </property>
14           <property name="show_sql">true</property>
15           <property name="format_sql">true</property>
16           <mapping resource="university/Student.hbm.xml" />
17       </session-factory>
18   </hibernate-configuration>
```

```
1    <?xml version="1.0"?>
2    <!DOCTYPE hibernate-mapping PUBLIC
3        "-//Hibernate/Hibernate Mapping DTD//EN"
4        "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
5    <hibernate-mapping>
6        <class name="university.Student" table="STUDENT">
7            <id name="studentID" column="STUDENTID">
8                <generator class="increment" />
9            </id>
10           <property name="name" column="NAME" />
11           <property name="gpa" column="gpa" />
12       </class>
13   </hibernate-mapping>
```

Introduction to JPA

- **Java Persistence** is the official Java API for object/relational mapping
- Originally only for use with EJB3, Java Persistence is now a separate API and can be used in Java SE applications
- JPA is a required part of Java Enterprise Edition, Version 5

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using Hibernate as a JPA Provider

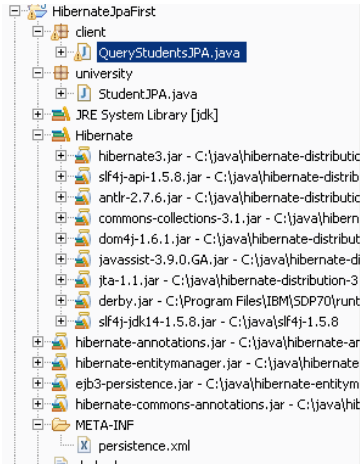
- Hibernate can act as a **JPA provider** allowing you to code to that JEE standard API
- JPA uses annotations for configuration instead (or in addition to) XML
- JPA uses an **EntityManager** type that's roughly equivalent to the Hibernate **Session**

```
1    EntityManagerFactory emf = Persistence
2      .createEntityManagerFactory("mypersistunit");
3
4    EntityManager entityManager =
5      (EntityManager)emf.createEntityManager();
6
7     . . .
8
9    emf.close();
10   entityManager.close();
```


Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JPA/Hibernate Configuration

- The CLASSPATH for a program that uses Hibernate as a JPA provider must include the Hibernate JARs
- JPA requires a configuration file, **persistence.xml** in the application's META-INF folder



Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

A Simple Example: Entity POJO

```
1    package hello;
2
3    import javax.persistence.Entity;
4    import javax.persistence.Id;
5
6    @Entity
7    public class Hello
8    {
9        @Id
10       private int helloID;
11       private String msg;
12
13       public void setHelloID(int helloID)
14       {
15           this.helloID = helloID;
16       }
```


Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

A Simple Example: Entity POJO, cont'd

```
17        public int getHelloID()
18        {
19            return helloID;
20        }
21
22        public void setMsg(String msg)
23        {
24            this.msg = msg;
25        }
26
27        public String getMsg()
28        {
29            return msg;
30        }
31    }
```


Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Hibernate versus JPA

- JPA is a standard API that's part of Java Enterprise Edition 5, while Hibernate is a third-party project
- Hibernate is under active development and provides features not available in standard JPA
- Hibernate applications traditionally use XML for configuration, while JPA apps use annotations
- Hibernate can act as a JPA provider

Hibernate also provides a few proprietary annotations that extend the standard annotations from JPA.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JPA/Hibernate Example

StudentJPA.java

QueryStudentsJPA.java

persistence.xml

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    package university;
2
3    import javax.persistence.Entity;
4    import javax.persistence.Id;
5    import javax.persistence.Table;
6
7    @Entity
8    @Table(name="STUDENT")
9    public class StudentJPA
10   {
11       @Id
12       private int studentID;
13       private String name;
14       private double gpa;
15
16       public double getGpa()
17       {
18           return gpa;
19       }
20
21       public void setGpa(double gpa)
22       {
23           this.gpa = gpa;
24       }
25
26       public String getName()
27       {
28           return name;
29       }
30
31       public void setName(String name)
32       {
33           this.name = name;
34       }
35
36       public int getStudentID()
37       {
38           return studentID;
39       }
40
41       public void setStudentID(int studentID)
42       {
43           this.studentID = studentID;
44       }
45
```

```
46      @Override
47      public String toString()
48      {
49          return "Name: " + name +
50                 " ID: " + studentID +
51                 " GPA: " + gpa;
52      }
53
54  }
```

```
1    package client;
2
3    import java.util.List;
4
5    import javax.persistence.EntityManager;
6    import javax.persistence.EntityManagerFactory;
7    import javax.persistence.Persistence;
8    import javax.persistence.Query;
9
10   import university.StudentJPA;
11
12   public class QueryStudentsJPA
13   {
14       public static void main(String[] args)
15       {
16           System.out.println("Starting up, JPA version");
17           EntityManagerFactory emf = Persistence
18                   .createEntityManagerFactory("default");
19
20           EntityManager entityManager =
21               (EntityManager) emf.createEntityManager();
22
23           Query q = entityManager.createQuery(
24                   "SELECT s FROM StudentJPA s");
25           List<StudentJPA>students = q.getResultList();
26           for(StudentJPA s : students)
27           {
28               System.out.println(s);
29           }
30
31           entityManager.close();
32           emf.close();
33       }
34   }
```

```
1    <?xml version="1.0" encoding="UTF-8"?>
2    <persistence version="1.0"
3        xmlns="http://java.sun.com/xml/ns/persistence"
4        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5        xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
6          http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
7        <persistence-unit name="default"
8            transaction-type="RESOURCE_LOCAL">
9            <provider>org.hibernate.ejb.HibernatePersistence</provider>
10
11           <class>university.StudentJPA</class>
12           <properties>
13               <property name="hibernate.connection.driver_class"
14                   value="org.apache.derby.jdbc.EmbeddedDriver" />
15               <property name="hibernate.connection.url"
16                   value="jdbc:derby:c:\springclass\dbhib\StudentDB"/>
17               <property name="hibernate.dialect"
18                   value="org.hibernate.dialect.DerbyDialect"/>
19               <property name="show_sql" value="true"/>
20               <property name="format_sql" value="true"/>
21
22           </properties>
23       </persistence-unit>
24   </persistence>
```

Chapter Summary

In this chapter, you learned:

- The fundamentals of object/relational mapping
- The fundamentals of Hibernate
- How to use Hibernate as a JPA provider

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum