/training/etc
CARE TO LEARN

7150 Riverwood Drive, Suite J
Columbia, MD 21046
Tel: 410-290-8383
Fax: 410-290-9427
http://www.trainingetc.com

INTRODUCTION TO HIBERNATE

email: info@trainingetc.com

Course # TE2412
Rev. 6/1/2013

©2013 /training/etc Inc. REPRODUCTION OF THESE MATERIALS IS PROHIBITED.

This Page Intentionally Left Blank

Evaluation Copy

©2013 /training/etc Inc. REPRODUCTION OF THESE MATERIALS IS PROHIBITED.

Course Objectives

- At the conclusion of this course, students will be able to:
  - Describe the purpose and benefits of an object/relational mapping tool
  - Configure database connection properties in the Hibernate configuration file
  - Use JPA and Hibernate Annotations to map Java classes to database tables
  - Create, save, update, and delete entities
  - Distinguish between entity and value types
  - Configure primary key generators for persistent classes
  - Describe and use the Hibernate strategies for mapping inheritance hierarchies
  - Map collections and associations
  - Write queries using Hibernate Query Language (HQL)

Evaluation Copy

©2013 /training/etc Inc. REPRODUCTION OF THESE MATERIALS IS PROHIBITED.

This Page Intentionally Left Blank

Evaluation Copy

Table of Contents

CHAPTER 1: GETTING STARTED WITH HIBERNATE
What is Hibernate? 1-2
Preparing to Use Hibernate 1-3
Configuring Hibernate 1-5
JDBC and Datasource Properties 1-6
Other Configuration Properties 1-7
Hibernate Sessions 1-9
Writing Classes for Hibernate Applications 1-12
Sample Class and Database Table 1-13
Sample Class and Mapping Diagram 1-16
Creating and Saving a New Entity 1-17
Locating an Existing Entity 1-18
Updating an Existing Entity 1-20
Deleting an Entity 1-21
Executing an HQL Query 1-22
Schema Generation 1-23
Programmatic Configuration 1-24

CHAPTER 2: MAPPING PERSISTENT CLASSES
Class Annotations 2-2
Access Type 2-3
Property Annotations 2-4
Hibernate Types 2-6
Entities and Values 2-8
Mapping Embeddable Types 2-9
Compound Keys 2-11
Generated Keys 2-13

CHAPTER 3: INHERITANCE
Mapping Class Inheritance 3-2
Table Per Class Hierarchy 3-7
Table Per Subclass 3-9
Table Per Concrete Class 3-11

CHAPTER 4: COLLECTIONS AND ASSOCIATIONS
Mapping Collections 4-2
Initializing Collections 4-5
Sample Application - UML Diagram 4-6
Sample Application - Database Schema 4-7
Implementing Associations 4-9
Mapping Associations 4-13

CHAPTER 5: HIBERNATE QUERY LANGUAGE
HQL Basics 5-2
HQL Expressions 5-5
HQL Functions 5-6

Evaluation Copy

©2013 /training/etc Inc. REPRODUCTION OF THESE MATERIALS IS PROHIBITED.

Polymorphic Queries 5-9
Executing Queries 5-10
Scrollable Results 5-13
Named Queries 5-14
Associations and Joins 5-16
Inner Joins 5-18
Outer Joins 5-20
Sample Queries 5-23

APPENDIX A: WORKING WITH THE LAB FILES
Importing the Eclipse Project A-2
Working with MySQL A-3
Creating Tables for the Training Application A-4
Hibernate Console Configuration A-5

Evaluation Copy

©2013 /training/etc Inc. REPRODUCTION OF THESE MATERIALS IS PROHIBITED.

Chapter 1: Getting Started with Hibernate

1) What is Hibernate? 1-2
2) Preparing to Use Hibernate 1-3
3) Configuring Hibernate 1-5
4) JDBC and Datasource Properties 1-6
5) Other Configuration Properties 1-7
6) Hibernate Sessions 1-9
7) Writing Classes for Hibernate Applications 1-12
8) Sample Class and Database Table 1-13
9) Sample Class and Mapping Diagram 1-16
10) Creating and Saving a New Entity 1-17
11) Locating an Existing Entity 1-18
12) Updating an Existing Entity 1-20
13) Deleting an Entity 1-21
14) Executing an HQL Query 1-22
15) Schema Generation 1-23
16) Programmatic Configuration 1-24

Evaluation Copy

What is Hibernate?

- Hibernate is an object/relational mapping tool for the Java environment. **Object/relational mapping** (ORM) is the mapping of data represented by objects to relational database tables and SQL schemas.

- While Hibernate's chief functionality is saving and loading objects, it offers many other features including the following.
  - Automatic mapping from Java data types to SQL data types
  - Hibernate Query Language (HQL)
  - Schema generation
  - Reverse engineering from an existing database
  - Connection pooling
  - Two-level data caching

- Hibernate is based on the "Plain Old Java Objects" model (POJO).

- Hibernate can be used with or without an application server, such as WebLogic.

- Hibernate is an open source product and can be obtained from:

  `http://www.hibernate.org`

Evaluation Copy

Preparing to Use Hibernate

- Download Hibernate and unzip. Add `hibernate3.jar` to the `CLASSPATH`.

- Add all of the `.jar` files in the `lib/required` directory to the `CLASSPATH`:
  - `antlr-2.7.6.jar`
  - `commons-collections-3.1.jar`
  - `dom4j-1.6.1.jar`
  - `javassist-3.12.0.GA.jar`
  - `jta-1.1.jar`
  - `slf4j-api-1.6.1.jar`

- Add the `.jar` file in the `lib/jpa` directory to the `CLASSPATH`:
  - `hibernate-jpa-2.0-api-1.0.1.Final.jar`

- Add one of the SLF4J (Simple Logging Facade for Java) bindings to the `CLASSPATH`:
  - `slf4j-nop-1.6.x.jar` - discards all logging
  - `slf4j-simple-1.6.x.jar` - outputs messages of level `INFO` and higher to `System.err`
  - `slf4j-log4j12-1.6.x.jar` - for `log4j` version 1.2
  - `slf4j-jdk14-1.6.x.jar` - for JDK 1.4 logging
  - `slf4j-jcl-1.6.x.jar` - for Jakarta Commons Logging

Evaluation Copy

Preparing to Use Hibernate

- See `slf4j.org` for more information and to download SLF4J.

- The JDBC driver for your database must also be available on the `CLASSPATH`.
  - The JDBC driver for MySQL 5.1 is located in `mysql-connector-java-5.1.18-bin.jar`.
  - "Connector/J" for MySQL can be downloaded from `mysql.com/downloads/connector`.

- Additional `.jar` files may be required if using Hibernate with an application server.

Evaluation Copy

©2013 /training/etc Inc. REPRODUCTION OF THESE MATERIALS IS PROHIBITED.

Configuring Hibernate

- Hibernate can be configured using either a properties file called `hibernate.properties` or an XML configuration file called `hibernate.cfg.xml`, placed in a root directory of the classpath.
  - The XML file is generally the preferred approach because its DTD allows the contents to be verified by a validating XML parser.

Evaluation Copy

hibernate.properties

```
 1. hibernate.connection.driver_class
 2.    com.mysql.jdbc.Driver
 3.
 4. hibernate.connection.url
 5.    jdbc:mysql://localhost/test
```

hibernate.cfg.xml

```
 1. <!DOCTYPE hibernate-configuration PUBLIC
 2.  "-//Hibernate/Hibernate Configuration DTD
 3.  3.0//EN" "http:// www.hibernate.org/dtd/
 4.  hibernate-configuration-3.0.dtd">
 5.
 6. <hibernate-configuration>
 7.   <session-factory>
 8.     <property name="connection.driver_class">
 9.        com.mysql.jdbc.Driver
10.     </property>
11.     <property name="connection.url">
12.        jdbc:mysql://localhost/test
13.     </property>
14.
15.     ...
16.   </session-factory>
17. </hibernate-configuration>
```

©2013 /training/etc Inc. REPRODUCTION OF THESE MATERIALS IS PROHIBITED.

JDBC and Datasource Properties

- JDBC Properties

Property Name	Value
hibernate.connection.driver_class	JDBC driver class
hibernate.connection.url	JDBC URL
hibernate.connection.username	database user name
hibernate.connection.password	database user password
hibernate.connection.pool_size	maximum number of connections in pool

- DataSource Properties

Property Name	Value
hibernate.connection.datasource	DataSource JNDI name
hibernate.jndi.url	URL of JNDI provider
hibernate.jndi.class	InitialContext factory class name
hibernate.connection.username	database user name
hibernate.connection.password	database user password

Evaluation Copy

Other Configuration Properties

Property Name	Value
hibernate.show_sql	When set to true, all SQL statements are shown in the console window.
hibernate.generate_statistics	When set to true, Hibernate collects statistics useful for performance tuning.
hbm2ddl.auto	When set to create, Hibernate drops and creates tables on startup. When set to create-drop, tables are dropped when SessionFactory is closed.

Evaluation Copy

©2013 /training/etc Inc. REPRODUCTION OF THESE MATERIALS IS PROHIBITED.

Other Configuration Properties

Property Name	Value
`hibernate.dialect`	class name of a `Dialect` that allows Hibernate to generate SQL optimized for a particular database

- Dialect class names for selected databases are shown below. All dialect classes have a common superclass, `org.hibernate.dialect.Dialect`.

Database	Dialect
DB2	`org.hibernate.dialect.DB2Dialect`
MySQL	`org.hibernate.dialect.MySQLDialect`
Oracle	`org.hibernate.dialect.OracleDialect`
Pointbase	`org.hibernate.dialect.PointbaseDialect`
HSQLDB	`org.hibernate.dialect.HSQLDialect`
Microsoft SQL Server	`org.hibernate.dialect.SQLServerDialect`

Evaluation Copy

Hibernate Sessions

- A **session** is the central interface between an application and Hibernate.

- A Hibernate session is a single-threaded short-lived object that represents a conversation between an application and the database.

- A session generally corresponds to a unit of work (or use case) and is typically equivalent to a database transaction. The session is opened before the unit of work begins, and after the unit of work is completed, the session is closed.

- Sessions are obtained from a `SessionFactory`, which is a threadsafe object intended to be shared by all application threads. A `SessionFactory` is created once, usually on application startup.

- To get a `SessionFactory`, create an instance of the `Configuration` class, and then call its `buildSessionFactory()` method.
  - The no-argument constructor for `Configuration` assumes that the Hibernate configuration file is **`hibernate.cfg.xml`**. Other constructors allow you to specify alternate file names.

- A utility class for this purpose is shown on the next page.

Evaluation Copy

©2013 /training/etc Inc. REPRODUCTION OF THESE MATERIALS IS PROHIBITED.

Hibernate Sessions

HibernateUtils.java

```
 1.  package examples.util;
 2.
 3.  import org.hibernate.*;
 4.  import org.hibernate.cfg.*;
 5.
 6.  public class HibernateUtils {
 7.
 8.     private static Configuration myConfig = null;
 9.     private static SessionFactory sessionFactory =
10.          null;
11.
12.     public static Configuration configure() {
13.        if (myConfig == null) {
14.           myConfig = new Configuration();
15.           myConfig.configure();
16.        }
17.        return myConfig;
18.     }
19.
20.     public static SessionFactory getSessionFactory() {
21.        if (myConfig == null) {
22.           configure();
23.        }
24.        if (sessionFactory == null) {
25.           sessionFactory =
26.               myConfig.buildSessionFactory();
27.        }
28.        return sessionFactory;
29.     }
30.
31.     public static void shutdown() {
32.        sessionFactory.close();
33.     }
34.  }
```

Evaluation Copy

Hibernate Sessions

- Your application can create more than one `SessionFactory` if you are using more than one database.

- Once you have a `SessionFactory`, you can call one of the following methods to obtain a `Session`.
  - `getCurrentSession()`
  - `openSession()`

- Next, get a `Transaction` object by calling `beginTransaction()` on the `Session` object.

- Interact with the database by calling `Session` methods or by calling setter methods on mapped objects.

- Call `commit()` on the `Transaction` object.

- Call `close()` on the `Session` object.
  - Note that this step is not required for a `Session` obtained from `getCurrentSession()`. It will be automatically closed when calling `commit()` on the Transaction.

Evaluation Copy

©2013 /training/etc Inc. REPRODUCTION OF THESE MATERIALS IS PROHIBITED.

Writing Classes for Hibernate Applications

- Write a Java class (POJO) for each important business entity.
- The class must have a no-argument constructor.
- The class should have a primary key or other unique identifier.
- Declare each persistent field as a `private` data item of the class.
- Provide a `public` getter and setter for each persistent field.
- Annotate each class with JPA and/or Hibernate annotations.
- Add the annotated class to the Hibernate configuration file.
  - If class is in a root directory of the classpath

    ```
    <mapping class="Product"/>
    ```

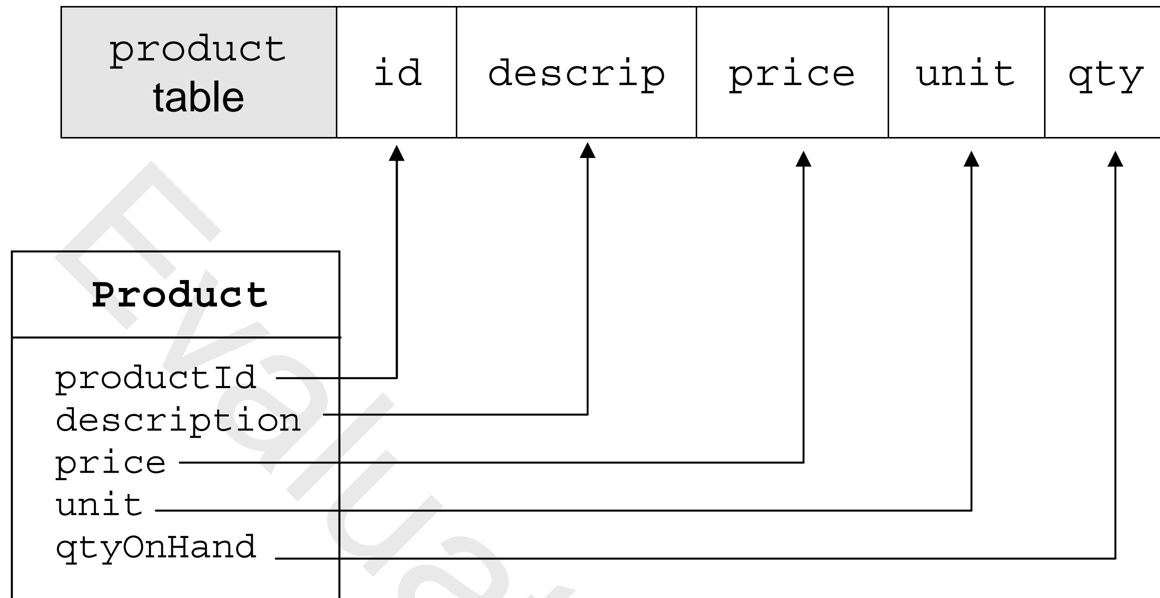
  - Or specify a path relative to the classpath.

    ```
    <mapping class="examples.starting.Product"/>
    ```

- A class mapping can also be specified programmatically (covered later in this chapter).

Evaluation Copy

Sample Class and Database Table



Evaluation Copy

Product.java

```
 1. package examples.starting;
 2.
 3. import javax.persistence.Column;
 4. import javax.persistence.Entity;
 5. import javax.persistence.Id;
 6.
 7. @Entity
 8. public class Product {
 9.
10.     private String productId;
11.     private String description;
12.     private String unit;
13.     private double price;
14.     private int    qtyOnHand;
15.
16.     public Product() { }
17.
```

©2013 /training/etc Inc. REPRODUCTION OF THESE MATERIALS IS PROHIBITED.

Sample Class and Database Table

Product.java (continued)

```
18.      @Id
19.      @Column(name="id")
20.      public String getProductId() {
21.          return productId;
22.      }
23.      public void setProductId (String id) {
24.          productId = id;
25.      }
26.
27.      @Column(name="descrip")
28.      public String getDescription() {
29.          return description;
30.      }
31.      public void setDescription (String desc) {
32.          description = desc;
33.      }
34.
35.      @Column
36.      public String getUnit() {
37.          return unit;
38.      }
39.      public void setUnit (String unit) {
40.          this.unit = unit;
41.      }
42.
43.      @Column
44.      public double getPrice() {
45.          return price;
46.      }
47.      public void setPrice (double price) {
48.          this.price = price;
49.      }
50.
```

Evaluation Copy

©2013 /training/etc Inc. REPRODUCTION OF THESE MATERIALS IS PROHIBITED.

Sample Class and Database Table

Product.java (continued)

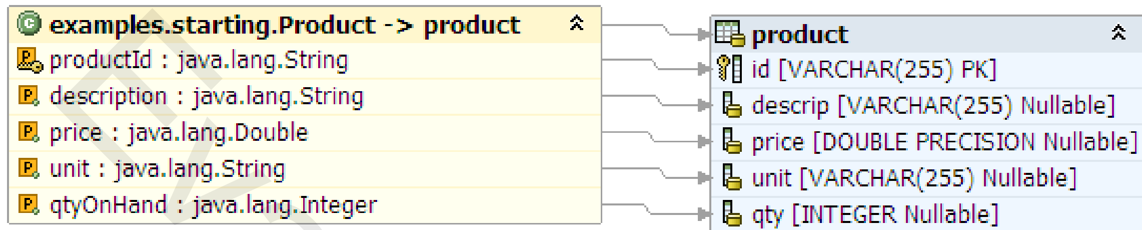
```
51.      @Column(name="qty")
52.      public int getQtyOnHand() {
53.          return qtyOnHand;
54.      }
55.      public void setQtyOnHand (int qty) {
56.          qtyOnHand = qty;
57.      }
58.
59.      public String toString() {
60.         return "Id: " + productId + " Desc: "
61.             + description + " Unit: " + unit
62.             + " Price: " + price + " Qty: "
63.             + qtyOnHand;
64.      }
65.  }
```

- The `@Entity` annotation specifies that persisted instances of this class will be managed by Hibernate. By default, the name of the database table is the same as the name of the class.
- The `@Id` annotation specifies the primary key field.
- The `@Column` annotations specify the other persistent fields.
- The `name` element is not required where the name of a field is the same as the name of the corresponding column in the database.

Evaluation Copy

Sample Class and Mapping Diagram

- A Mapping Diagram generated by the Hibernate Tools for Eclipse is shown below.



Evaluation Copy

Creating and Saving a New Entity

CreateProduct.java

```
 1. ...
 2.
 3. Configuration config =
 4.     HibernateUtils.configure();
 5.
 6. config.addAnnotatedClass(Product.class);
 7.
 8. config.setProperty
 9.    ("hibernate.hbm2ddl.auto", "create");
10.
11. Session mySession =
12.     HibernateUtils.getSessionFactory().
13.     getCurrentSession();
14.
15. mySession.beginTransaction();
16.
17. Product prod = new Product();
18.
19. prod.setProductId ("101");
20. prod.setDescription ("COPY PAPER");
21. prod.setUnit ("CS");
22. prod.setPrice (29.00);
23. prod.setQtyOnHand (100);
24.
25. mySession.save (prod);
26.
27. mySession.getTransaction().commit();
28.
29. HibernateUtils.shutdown();
30.
31. ...
```

Evaluation Copy

©2013 /training/etc Inc. REPRODUCTION OF THESE MATERIALS IS PROHIBITED.

Locating an Existing Entity

LocateProduct.java

```
 1. ...
 2.
 3. Configuration config =
 4.     HibernateUtils.configure();
 5.
 6. config.addAnnotatedClass(Product.class);
 7.
 8. Session mySession =
 9.     HibernateUtils.getSessionFactory().
10.     getCurrentSession();
11.
12. mySession.beginTransaction();
13.
14. Product prod = new Product();
15.
16. mySession.load (prod, "101");
17.
18. System.out.println ("Located Product # " +
19.     prod.getProductId());
20.
21. System.out.println ("Description = " +
22.     prod.getDescription());
23.
24. System.out.println ("Unit = " +
25.     prod.getUnit());
26.
27. System.out.println ("Price = " +
28.     prod.getPrice());
29.
30. System.out.println ("Qty = " +
31.     prod.getQtyOnHand());
32.
33. mySession.getTransaction().commit();
34. HibernateUtils.shutdown();
35.
36. ...
```

Evaluation Copy

©2013 /training/etc Inc. REPRODUCTION OF THESE MATERIALS IS PROHIBITED.

Locating an Existing Entity

- A variety of methods are available in the Session interface for locating an existing entity in the database.

- The `load()` methods should be used only to retrieve an instance that you assume exists and non-existence would be an error. If the instance does not exist, the `load()` methods throw `HibernateException`.
  - `void load (Object obj, Serializable id)`
    - This method was used in the example.
  - `Object load (Class theClass, Serializable id)`
  - `Object load (String entityName, Serializable id)`
    - Entity names will be discussed in the next chapter.

- The `get()` methods can be used when you are not sure if the instance exists. If the instance does not exist, the `get()` methods return `null`.
  - `Object get (Class theClass, Serializable id)`
  - `Object get (String entityName, Serializable id)`

Evaluation Copy

Updating an Existing Entity

UpdateProduct.java

```
 1. ...
 2.
 3. Session mySession =
 4.     HibernateUtils.getSessionFactory().
 5.     getCurrentSession();
 6.
 7. mySession.beginTransaction();
 8.
 9. Product prod = new Product();
10.
11. mySession.load (prod, "101");
12.
13. prod.setDescription("New Description");
14. prod.setUnit("XX");
15. prod.setPrice(9.99);
16. prod.setQtyOnHand(1);
17.
18. mySession.save (prod);
19.
20. mySession.getTransaction().commit();
21.
22. HibernateUtils.shutdown();
23.
24. ...
```

Evaluation Copy

©2013 /training/etc Inc. REPRODUCTION OF THESE MATERIALS IS PROHIBITED.

Deleting an Entity

DeleteProduct.java

```
 1. ...
 2.
 3. Session mySession =
 4.     HibernateUtils.getSessionFactory().
 5.     getCurrentSession();
 6.
 7. mySession.beginTransaction();
 8.
 9. Product prod = new Product();
10.
11. mySession.load (prod, "101");
12.
13. mySession.delete (prod);
14.
15. mySession.getTransaction().commit();
16.
17. HibernateUtils.shutdown();
18.
19. ...
```

Evaluation Copy

Executing an HQL Query

SelectProducts.java

```
 1. ...
 2.
 3. Session mySession =
 4.     HibernateUtils.getSessionFactory().
 5.     getCurrentSession();
 6.
 7. mySession.beginTransaction();
 8.
 9. String query =
10.     "from Product where price > 10.0";
11.
12. Query myQuery =
13.     mySession.createQuery(query);
14.
15. List<?> myList = myQuery.list();
16.
17. for (Object obj : myList) {
18.     System.out.println(obj);
19. }
20.
21. mySession.getTransaction().commit();
22.
23. HibernateUtils.shutdown();
24.
25. ...
26.
```

Evaluation Copy

©2013 /training/etc Inc. REPRODUCTION OF THESE MATERIALS IS PROHIBITED.

Schema Generation

- Data Definition Language (DDL) can be generated from the Hibernate mapping files by the `SchemaExport` utility.

- The general form of the command is:

```
java -cp hibernate-classpath
     org.hibernate.too.hbm2ddl.SchemaExport
     [options] [mapping-files]
```

- Options include the following.
  - `--drop` only drop the tables
  - `--create` only create the tables
  - `--text` show DDL, but do not export to the database
  - `--output=filename`
    output DDL to file
  - `--format` format the generated SQL nicely

Evaluation Copy

©2013 /training/etc Inc. REPRODUCTION OF THESE MATERIALS IS PROHIBITED.

Programmatic Configuration

- Hibernate mapping files and configuration properties can be provided dynamically by calling methods on the `Configuration` object.

```
config.addAnnotatedClass("Product.class");
config.setProperty("hibernate.show_sql", "true");
```

- The database schema can also be exported programmatically using the following `Configuration` instance.

```
SchemaExport se = new SchemaExport(config);
se.execute(false, true, true, true);
```

  - The boolean parameters for the `execute()` method specify whether or not to:
    - print DDL to the console;
    - export script to the database;
    - run the schema creation script; and
    - run the schema drop script.
  - See Hibernate API docs for `SchemaExport` in the `org.hibernate.tool.hbm2ddl` package.

Evaluation Copy

©2013 /training/etc Inc. REPRODUCTION OF THESE MATERIALS IS PROHIBITED.

Exercises

1. Write an `Employee` class with the (private) attributes shown below. Include getter and setter methods and a `toString()` method.

```
int    employeeId;
String firstName;
String lastName;
float  hourlyRate;
Date   hireDate;
```

- Solution: See next exercise.

2. Add annotations to map the `Employee` class to a database table called `workers`, with columns named `emplid`, `fname`, `lname`, `rate`, and `hiredate`.

- Solution: `solutions/starting/Employee.java`

3. Write programs (or a single program) to:

- add several records to the `workers` table;
- locate an employee by id number;
- update an existing employee;
- delete an employee; and
- list all employees.

- Solution: `solutions/starting/TestEmployee`

Evaluation Copy

Exercises

4. Add the `hibernate.generate_statistics` property to the Hibernate configuration file (with a value of `true`). Modify `SelectProducts.java` so that you call `getStatistics()` on the `Session` object before the transaction is comitted. (Check the Hibernate API docs for information on this method.) Then, display the entity count and the entity keys.
   - Solution: `solutions.starting.SelectProducts`

Evaluation Copy

©2013 /training/etc Inc. REPRODUCTION OF THESE MATERIALS IS PROHIBITED.

Chapter 3: Inheritance

1) Mapping Class Inheritance 3-2

2) Table Per Class Hierarchy 3-7

3) Table Per Subclass 3-9

4) Table Per Concrete Class 3-11

Evaluation Copy

©2013 /training/etc Inc. REPRODUCTION OF THESE MATERIALS IS PROHIBITED.

Mapping Class Inheritance

- There are three approaches to inheritance, represented by the `enum` type `javax.persistence.InheritanceType`.

Table per class hierarchy	**`InheritanceType.SINGLE_TABLE`**
Table per subclass	**`InheritanceType.JOINED`**
Table per concrete class	**`InheritanceType.TABLE_PER_CLASS`**

Evaluation Copy

- Consider the following classes.

Item.java

```
 1. package examples.inheritance1;
 2.
 3. import javax.persistence.*;
 4.
 5. @Entity
 6. @Table(name = "items1")
 7.
 8. public class Item {
 9.     protected String itemId;
10.     protected String description;
11.     protected String unit;
12.     protected double price;
13.
14.     public Item() {}
15.
16.     @Id
17.     public String getItemId() {
18.        return itemId;
19.     }
20.     public void setItemId(String id) {
21.        itemId = id;
22.     }
23.
24.     @Column(name = "descrip")
25.     public String getDescription() {
26.        return description;
27.     }
```

Mapping Class Inheritance

Item.java (continued)

```
28.     public void setDescription(String desc) {
29.        description = desc;
30.     }
31.
32.     @Column
33.     public String getUnit() {
34.        return unit;
35.     }
36.     public void setUnit(String unit) {
37.        this.unit = unit;
38.     }
39.
40.     @Column
41.     public double getPrice() {
42.        return price;
43.     }
44.     public void setPrice(double price) {
45.        this.price = price;
46.     }
47.
48.     public String toString() {
49.        StringBuffer sb = new StringBuffer(64);
50.        sb.append ("Id: ");
51.        sb.append (itemId);
52.        sb.append (" Desc: ");
53.        sb.append (description);
54.        sb.append (" Unit: ");
55.        sb.append (unit);
56.        sb.append (" Price: ");
57.        sb.append (price);
58.        return sb.toString();
59.     }
60. }
```

Evaluation Copy

Mapping Class Inheritance

WarehouseItem.java

```
 1.  package examples.inheritance1;
 2.
 3.  import javax.persistence.*;
 4.
 5.  @Entity
 6.  public class WarehouseItem extends Item {
 7.
 8.     private String location;
 9.
10.     public WarehouseItem() {}
11.
12.     @Column(name = "loc")
13.     public String getLocation() {
14.        return location;
15.     }
16.     public void setLocation(String location) {
17.        this.location = location;
18.     }
19.
20.     public String toString() {
21.        StringBuffer sb = new StringBuffer(128);
22.        sb.append (super.toString());
23.        sb.append ("  Location: ");
24.        sb.append (location);
25.        return sb.toString();
26.     }
27.  }
```

Evaluation Copy

©2013 /training/etc Inc. REPRODUCTION OF THESE MATERIALS IS PROHIBITED.

Mapping Class Inheritance

CustomItem.java

```
 1.  package examples.inheritance;
 2.
 3.  import javax.persistence.*;
 4.
 5.  @Entity
 6.  public class CustomItem extends Item {
 7.
 8.     private String vendorName;
 9.
10.     public CustomItem() { }
11.
12.     @Column(name = "vendor")
13.        public String getVendorName() {
14.        return vendorName;
15.     }
16.     public void setVendorName(String vendor) {
17.        vendorName = vendor;
18.     }
19.
20.     public String toString() {
21.        StringBuffer sb = new StringBuffer(128);
22.        sb.append (super.toString());
23.        sb.append (" Vendor: ");
24.        sb.append (vendorName);
25.        return sb.toString();
26.     }
27.  }
```

Evaluation Copy

©2013 /training/etc Inc. REPRODUCTION OF THESE MATERIALS IS PROHIBITED.

Mapping Class Inheritance

TestItems.java

```
 1.  ...
 2.
 3.  Item item = new Item();
 4.  item.setItemId ("100");
 5.  item.setDescription ("JUST AN ITEM");
 6.  item.setUnit ("EA");
 7.  item.setPrice (.99);
 8.  mySession.save (item);
 9.
10.  WarehouseItem wItem = new WarehouseItem();
11.  wItem.setItemId ("101");
12.  wItem.setDescription ("WAREHOUSE ITEM");
13.  wItem.setUnit ("EA");
14.  wItem.setPrice (9.99);
15.  wItem.setLocation ("Bin 37");
16.  mySession.save (wItem);
17.
18.  CustomItem cItem = new CustomItem();
19.  cItem.setItemId ("701");
20.  cItem.setDescription ("CUSTOM ITEM");
21.  cItem.setUnit ("EA");
22.  cItem.setPrice (99.99);
23.  cItem.setVendorName ("ABC Fabricators");
24.  mySession.save (cItem);
25.
26.  ...
27.
28.  List<?> myItems =
29.     mySession.createQuery("from Item")list();
30.
31.  for (Object obj : myItems) {
32.     System.out.println(obj);
33.  }
34.
35.  ...
```

Evaluation Copy

©2013 /training/etc Inc. REPRODUCTION OF THESE MATERIALS IS PROHIBITED.

Table Per Class Hierarchy

- One table is used for all types. This table contains columns for all properties of all classes in the hierarchy.

- Parent type can be abstract class or interface.

- An extra column (discriminator) is added to the table to identify sub-types. This value is not available to the application code.

- This is a good strategy in terms of simplicity and performance.

- One limitation of this strategy is that columns declared by the subclasses may not have NOT NULL constraints.

`items1`

ID	ITEM_TYPE	DESCRIP	PRICE	UNIT	LOC	VENDOR
100	I	JUST AN ITEM	0.99	EA	**[null]**	**[null]**
101	WI	WAREHOUSE ITEM	9.99	EA	Bin 37	**[null]**
701	CI	CUSTOM ITEM	99.99	EA	**[null]**	ABC Fabricators

Evaluation Copy

Table Per Class Hierarchy

Item.java

```
 1.  package examples.inheritance1;
 2.
 3.  import javax.persistence.*;
 4.
 5.  @Entity
 6.  @Table(name = "items1")
 7.  @Inheritance(strategy = InheritanceType.SINGLE_TABLE)
 8.  @DiscriminatorColumn(name = "item_type",
 9.      discriminatorType=DiscriminatorType.STRING)
10.  @DiscriminatorValue("I")
11.
12.  public class Item {
13.      ...
14.  }
```

CustomItem.java

```
 1.  package examples.inheritance1;
 2.
 3.  import javax.persistence.*;
 4.
 5.  @Entity
 6.  @DiscriminatorValue("CI")
 7.
 8.  public class CustomItem extends Item {
 9.      ...
10.  }
```

WarehouseItem.java

```
 1.  package examples.inheritance1;
 2.
 3.  import javax.persistence.*;
 4.
 5.  @Entity
 6.  @DiscriminatorValue("WI")
 7.
 8.  public class WarehouseItem extends Item {
 9.      ...
10.  }
```

Evaluation Copy

©2013 /training/etc Inc. REPRODUCTION OF THESE MATERIALS IS PROHIBITED.

Table Per Subclass

- One table is used for the superclass. Each subclass has a table with a foreign key to the primary key in the superclass table.

- The primary advantage of this strategy is that the database is normalized.

- The disadvantage of this strategy is that performance can be unacceptable for complex class hierarchies. Note that many queries may require a join across two (or more) tables.

items2

ID	DESCRIP	PRICE	UNIT
100	JUST AN ITEM	0.99	EA
101	WAREHOUSE ITEM	9.99	EA
701	CUSTOM ITEM	99.99	EA

witems2

WITEM_ID	LOC
101	Bin 37

citems2

CITEM_ID	VENDOR
701	ABC Fabricators

Evaluation Copy

©2013 /training/etc Inc. REPRODUCTION OF THESE MATERIALS IS PROHIBITED.

Table Per Subclass

Item.java

```
 1.  package examples.inheritance2;
 2.
 3.  import javax.persistence.*;
 4.
 5.  @Entity
 6.  @Table(name = "items2")
 7.  @Inheritance(strategy = InheritanceType.JOINED)
 8.
 9.  public class Item {
10.      ...
11.  }
```

CustomItem.java

```
 1.  package examples.inheritance2;
 2.
 3.  import javax.persistence.*;
 4.
 5.  @Entity
 6.  @Table(name = "citems2")
 7.  @PrimaryKeyJoinColumn(name = "citem_id")
 8.
 9.  public class CustomItem extends Item {
10.      ...
11.  }
```

WarehouseItem.java

```
 1.  package examples.inheritance2;
 2.
 3.  import javax.persistence.*;
 4.
 5.  @Entity
 6.  @Table(name = "witems2")
 7.  @PrimaryKeyJoinColumn(name = "witem_id")
 8.
 9.  public class WarehouseItem extends Item {
10.      ...
11.  }
```

Evaluation Copy

©2013 /training/etc Inc. REPRODUCTION OF THESE MATERIALS IS PROHIBITED.

Table Per Concrete Class

- One table is used for each non-abstract class. The table for a particular class has all of the properties of that class, including inherited properties.

- One limitation of this strategy is that if a property is mapped for the superclass, the column name must be the same in all of the subclass tables.

Evaluation Copy

`items3`

ID	DESCRIP	PRICE	UNIT
100	JUST AN ITEM	0.99	EA

`witems3`

ID	DESCRIP	PRICE	UNIT	LOC
101	WAREHOUSE ITEM	9.99	EA	Bin 37

`citems3`

ID	DESCRIP	PRICE	UNIT	VENDOR
701	CUSTOM ITEM	99.99	EA	ABC Fabricators

©2013 /training/etc Inc. REPRODUCTION OF THESE MATERIALS IS PROHIBITED.

Table Per Concrete Class

Item.java

```
 1.  package examples.inheritance3;
 2.
 3.  import javax.persistence.*;
 4.
 5.  @Entity
 6.  @Table(name = "items3")
 7.  @Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
 8.
 9.  public class Item {
10.      ...
11.  }
```

CustomItem.java

```
 1.  package examples.inheritance3;
 2.
 3.  import javax.persistence.*;
 4.
 5.  @Entity
 6.  @Table(name = "citems3")
 7.
 8.  public class CustomItem extends Item {
 9.      ...
10.  }
```

WarehouseItem.java

```
 1.  package examples.inheritance3;
 2.
 3.  import javax.persistence.*;
 4.
 5.  @Entity
 6.  @Table(name = "witems3")
 7.
 8.  public class WarehouseItem extends Item {
 9.      ...
10.  }
```

Evaluation Copy

Exercises

1. Working with the `Employee` class that you wrote in an exercise from a previous chapter, create two subclasses as follows.

```
public class TempEmployee extends Employee {

    private Date endDate;

    // add getter, setter, toString()
}

public class DeptManager extends Employee {

    private String title;

   // add getter, setter, toString()
}
```

   - Solution: `solutions/inheritance/TempEmployee`
     `solutions/inheritance/DeptManager`

2. Annotate the three classes using the table per class hierarchy approach. Test your mappings using the `TestEmployee` class in the `starters.inheritance` package.

   - Solution: `solutions/inheritance1`

3. Repeat the previous exercise using the table per subclass approach.

   - Solution: `solutions/inheritance2`

Evaluation Copy

Exercises

4. Design and test a simple three-level class hierarchy with an interface in the first level, an abstract class in the second level, and two concrete classes in the third level. Use the table per class hierarchy strategy.

   - Note: The "solution" is a suggested solution only. Feel free to create classes from a problem domain of your choice.
   - Solution: `solutions/inheritance/levels`

Evaluation Copy

©2013 /training/etc Inc. REPRODUCTION OF THESE MATERIALS IS PROHIBITED.