INTRODUCTION TO SPRING FRAMEWORK

email: info@trainingetc.com

Course # TE2410
Rev. 6/29/2013
Course Objectives

At the conclusion of this course, students will be able to:

- Describe the seven component modules of the Spring Framework.
- Understand the basic philosophies of Spring.
- Explain the purpose and benefits of dependency injection.
- Configure beans in a Spring configuration file.
- Use setter and constructor injection with Spring beans.
- Create property files for error messages and to support internationalization.
- Write validators and property editors for user-defined data.
- Understand basic concepts of aspect-oriented programming.
- Use the JDBC template to simplify database access.
- Use the Hibernate template to integrate Hibernate and Spring.
- Create Web applications using the Model-View-Controller architecture.
- Write forms and controllers for Spring Web applications.
# Table of Contents

**CHAPTER 1: INTRODUCTION TO SPRING**
- What is Spring? ................................................................. 1-2
- Overview of the Spring Framework .................................. 1-3
- Spring Philosophies ......................................................... 1-5
- Spring Documentation ...................................................... 1-6
- Java 5 Language Features ................................................ 1-7

**CHAPTER 2: A FIRST LOOK AT SPRING**
- A Simple Example .......................................................... 2-2
- Wiring Beans ....................................................................... 2-5
- Configuring a Properties File ........................................... 2-8
- Schema-Based Configuration ............................................. 2-10

**CHAPTER 3: BEANS AND CONTAINERS**
- Spring Containers ........................................................... 3-2
- Spring Configuration File ................................................ 3-3
- Spring Beans ...................................................................... 3-4
- Using the Container .......................................................... 3-5
- The BeanFactory Interface ............................................... 3-6
- Singleton vs. Prototype .................................................... 3-7
- Bean Naming ..................................................................... 3-10
- Dependency Injection ........................................................ 3-12
- Setter Injection .................................................................. 3-13
- Constructor Injection ....................................................... 3-17

**CHAPTER 4: THE APPLICATION CONTEXT**
- The ApplicationContext Interface .................................. 4-2
- Accessing Application Components ................................ 4-3
- Accessing Resources ....................................................... 4-5
- Internationalization with MessageSource .................... 4-9
- Application Events ........................................................... 4-13

**CHAPTER 5: DATA VALIDATION AND CONVERSION**
- The Validator Interface .................................................. 5-2
- The Errors Interface ....................................................... 5-3
- The ValidationUtils Class ................................................. 5-6
- Validator Example ........................................................... 5-7
- Testing the Validator ....................................................... 5-10
- Property Editors .............................................................. 5-12
- Custom Property Editors ................................................ 5-18

**CHAPTER 6: ASPECT-ORIENTED PROGRAMMING**
- Aspect-Oriented Programming ....................................... 6-2
- AOP Concepts ................................................................. 6-3
- AOP Proxies ..................................................................... 6-4
- The AOP Alliance ............................................................ 6-5
INTRODUCTION TO SPRING FRAMEWORK

Types of Advice ................................................................. 6-6
AOP Example ........................................................................ 6-7
Introductions ....................................................................... 6-14

CHAPTER 7: USING JDBC WITH SPRING
A Simpler Approach ........................................................... 7-2
The JdbcTemplate Class ...................................................... 7-3
Exception Translation ....................................................... 7-7
Updating with the JdbcTemplate ....................................... 7-9
Queries Using the JdbcTemplate ....................................... 7-13
Mapping Results to Java Objects .................................... 7-16

CHAPTER 8: USING HIBERNATE WITH SPRING
What is Hibernate? .......................................................... 8-2
Hibernate Sessions ......................................................... 8-3
Configuring Hibernate in Spring .................................... 8-4
The HibernateTemplate .................................................... 8-6
Sample Class with Annotations ....................................... 8-7
Sample Class and Mapping File ...................................... 8-8
Creating and Saving a New Entity ................................. 8-9
Locating an Existing Entity ............................................ 8-10
Updating an Existing Entity .......................................... 8-11
Hibernate Query Language ........................................... 8-12
Executing Queries ............................................................ 8-14

CHAPTER 9: SPRING WEB MVC - PART 1
Spring Web MVC .............................................................. 9-2
The DispatcherServlet .................................................... 9-3
Writing a Controller ........................................................ 9-4
A Simple View Page ........................................................ 9-6
Configuring the Controller ............................................ 9-7
Adding a View Resolver ................................................. 9-8
Adding a Message Bundle ............................................. 9-9
Adding Business Classes .............................................. 9-11
Adding Test Data ............................................................ 9-16
Accessing a Database ..................................................... 9-18
Adding a Form ............................................................... 9-21
Updating the Database .................................................. 9-27
Integrating Hibernate .................................................... 9-30

CHAPTER 10: SPRING WEB MVC - PART 2
Handler Mappings .......................................................... 10-2
View Resolution ............................................................. 10-4
Chaining View Resolvers .............................................. 10-7
Controllers ........................................................................ 10-8
AbstractWizardFormController .................................. 10-11

APPENDIX A: RESOURCES
Resources .......................................................................... A-2
Chapter 1:
Introduction to Spring

1) What is Spring? ........................................................................................................ 1-2
2) Overview of the Spring Framework....................................................................... 1-3
3) Spring Philosophies .............................................................................................. 1-5
4) Spring Documentation .......................................................................................... 1-6
5) Java 5 Language Features ..................................................................................... 1-7
What is Spring?

- The Spring Framework is a leading full-stack Java / Java EE application framework.

- Application frameworks help developers write and "glue" together the various parts of a software project. A full-stack framework includes all the necessary components to assemble a fully working, ready-to-use application.

- Spring provides significant benefits for many projects, increasing development productivity and runtime performance while improving application quality.

- In particular, Spring addresses the issue of dependencies between components.
Overview of the Spring Framework

- The Spring Framework is organized into seven modules.

<table>
<thead>
<tr>
<th>DAO</th>
<th>ORM</th>
<th>MVC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AOP</td>
<td>WEB</td>
</tr>
<tr>
<td></td>
<td>CONTEXT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CORE</td>
<td></td>
</tr>
</tbody>
</table>

- **Core** - The core container provides the essential functionality of the Spring framework. A primary component of the core container is the *BeanFactory*, an implementation of the Factory pattern. The *BeanFactory* applies the Inversion of Control (IOC) pattern to separate an application's configuration and dependency specification from the actual application code.

- **Context** - The Spring context is a configuration file that provides context information to the Spring framework.

- **AOP** - The Spring AOP module integrates aspect-oriented programming functionality directly into the Spring framework through its configuration management feature. The Spring AOP module provides declarative transaction management services for objects in any Spring-based application.
Overview of the Spring Framework

- **DAO** - The Spring JDBC Data Access Object abstraction layer offers an unchecked exception hierarchy for managing the exception handling and error messages thrown by JDBC operations. The DAO module simplifies error handling and greatly reduces the amount of "boilerplate" code you need to write, such as opening and closing connections.

- **ORM** - The Spring framework integrates nicely with several object-relational mapping tools (including Hibernate).

- **Web** - The Web context module builds on top of the application context module, providing contexts for Web-based applications.

- **MVC** - The Spring Model-View-Controller framework is a full-featured MVC implementation for building Web applications.
Spring Philosophies

- Spring is a technology dedicated to allowing you to build applications using POJOs ("Plain Old Java Objects").

- Programming to an interface rather than an implementation is always a good practice and is definitely encouraged by Spring.

- A central concept of the Spring framework is non-invasiveness, which means that you should not be forced to introduce framework-specific classes into your domain model. Spring (almost) always offers you the choice.

- In general, Spring is not prescriptive. While Spring makes it easy to use good practices, it avoids forcing a particular approach.

- Spring is consistent.

- Spring does not reinvent the wheel.

- "Make the right thing easy to do."
Spring Documentation

- Spring documentation can be found in the docs directory of the Spring distribution. There are two subdirectories, which are described below.

- spring-framework-reference - contains the reference guide in HTML and PDF format

- javadoc-api - contains the JavaDocs for the Spring classes, interfaces, and exceptions
Java 5 Language Features

- In some of the sample code in this course, you will see the use of new language features introduced in Java 5. A few examples are presented below as a review.

- Traversing an array

  ```java
  String array[] = {"A", "B", "C", "D"};
  ```

  - Traditional for loop
    ```java
    for (int i = 0; i < array.length; ++i) {
        System.out.println(array[i]);
    }
    ```

  - Enhanced for loop
    ```java
    for (String s : array) {
        System.out.println(s);
    }
    ```

- Creating a collection

  - Using a traditional collection
    ```java
    ArrayList list = new ArrayList();
    list.add("A");
    list.add("B");
    list.add("C");
    ```

  - Using a Java Generic
    ```java
    ArrayList<String> list = new ArrayList<String>();
    list.add("A");
    list.add("B");
    list.add("C");
    ```
Java 5 Language Features

- Traversing a collection
  - Using an iterator
    ```java
    Iterator iter = myList.iterator();
    while (iter.hasNext()) {
        String s = (String) iter.next();
        System.out.println(s);
    }
    ```
  - Enhanced for loop
    ```java
    for (String s : list) {
        System.out.println(s);
    }
    ```
Chapter 2: A First Look at Spring

1) A Simple Example .............................................................. 2-2
2) Wiring Beans ................................................................. 2-5
3) Configuring a Properties File ............................................. 2-8
4) Schema-Based Configuration ............................................. 2-10
A Simple Example

- We begin by defining an interface and an implementation class.

**HelloService.java**

```java
1. package examples.hello;
2. 
3. public interface HelloService {
4.   public void sayHello();
5. }
```

**HelloServiceImpl.java**

```java
1. package examples.hello;
2. 
3. public class HelloServiceImpl implements HelloService {
4.   private String greeting;
5. 
6.   public void sayHello() {
7.     System.out.println (greeting);
8.   }
9. 
10.   public void setGreeting(String greeting) {
11.     this.greeting = greeting;
12.   }
13. }
14. }
```
A Simple Example

- The implementation class is configured in a Spring beans configuration file.

- The value of the "greeting" property is specified in the beans configuration file. This setting triggers a call to the setGreeting() method in the implementation class.

- This mechanism is referred to as **dependency injection**.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING/DTD BEAN 2.0//EN" "http://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>
  <bean id="hello" class="examples.hello.HelloServiceImpl">
    <property name="greeting">
      <value>Hello Spring World!</value>
    </property>
  </bean>
</beans>
```

- Note the XML prolog (line 1), the reference to the Document Type Definition (lines 2-4), and the top-level <beans> element (lines 6 and 15). Although these lines will not always be shown in the course notes, they are always included in the Spring configuration file.
A Simple Example

The Spring container manages the configuration and lifecycle of beans. There are two basic kinds of containers, BeanFactory and ApplicationContext, and several different implementations of each.

We will begin by using the XmlBeanFactory, which reads configuration information from an XML file. We construct it using a ClassPathResource, which will look for the XML file on the classpath.

HelloTest.java

```java
1. package examples.hello;
2. import org.springframework.beans.factory.BeanFactory;
3. import org.springframework.beans.factory.xml.XmlBeanFactory;
4. import org.springframework.core.io.ClassPathResource;
5. public class HelloTest {
6.     public static void main(String[] args) {
7.         BeanFactory factory = new XmlBeanFactory
8.             (new ClassPathResource("hello.xml"));
9.         HelloService service = (HelloService)
10.            factory.getBean("hello");
11.         service.sayHello();
12.     }
13. }
```

Wiring Beans

• Now we will show an example where one bean depends on another. The process of configuring one bean that depends on another is called "wiring" beans.

• Here is an interface and implementation class to work with the HelloService bean.

HelloHelper.java

```
package examples.hello;

public interface HelloHelper {
    public String convertString (String original);
}
```

HelloHelperImpl.java

```
package examples.hello;

public class HelloHelperImpl implements HelloHelper {
    public String convertString (String original) {
        return original.toUpperCase();
    }
}
```
Wiring Beans

• Now here is a revised version of the HelloService bean.

HelloServiceImpl2.java

```
1. package examples.hello;
2. 
3. public class HelloServiceImpl2 implements HelloService {
4.    
5.    private String greeting;
6.    private HelloHelper helper;
7.    
8.    public void sayHello() {
9.        System.out.println
10.           (helper.convertString(greeting));
11.    }
12. 
13. 
14.    public void setGreeting(String greeting) {
15.        this.greeting = greeting;
16.    }
17. 
18.    public void setHelper(HelloHelper helper) {
19.        this.helper = helper;
20.    }
21. }
```
Wiring Beans

Here is the revised configuration.

```xml
hello.xml

```1. 
2. <bean id="hello2"
3. class="examples.hello.HelloServiceImpl2">
4. 
5. <property name="greeting">
6. <value>Hello Spring World!</value>
7. </property>
8. 
9. <property name="helper">
10. <ref bean="helloHelper"/>
11. </property>
12. </bean>
13. 
14. <bean id="helloHelper"
15. class="examples.hello.HelloHelperImpl"/>
16.
```

Note the use of the `<ref>` element to reference another bean by its configured id.
Configuring a Properties File

• Instead of hard-coding the greeting message in the beans configuration file, we will read it from a properties file.

• The `PropertyPlaceholderConfigurer` pulls values from a properties value and places them into bean configuration entries. The default placeholder syntax follows the JSP expression language style.

    `${property-name}`

**hello.xml**

```xml
1. 2. <bean id="propertyPlaceholderConfigurer"
3.     class="org.springframework.beans.factory.config."
4.     PropertyPlaceholderConfigurer">
5. 
6.     <property name="location">
7.         <value>hello.properties</value>
8.     </property>
9. </bean>
10. 
11. <bean id="hello3"
12.     class="examples.hello.HelloServiceImpl2">
13.     <property name="greeting">
14.         <value>${hello.message}</value>
15.     </property>
16.     <property name="helper">
17.         <ref bean="helloHelper"/>
18.     </property>
19. </bean>
20. 
```
Configuring a Properties File

Here is the properties file.

```properties
hello.properties
1. hello.message=Hello World from properties file
```

Here is the test program.

```java
HelloTest3.java
1. package examples.hello;
2. 
3. import org.springframework.context.ApplicationContext;
4. import org.springframework.context.support.ClassPathXmlApplicationContext;
5. 
6. 
7. public class HelloTest3 {
8. 
9.     public static void main(String[] args) {
10.         ApplicationContext ctx = new
11.             ClassPathXmlApplicationContext("hello.xml");
12.         
13.             HelloService service =
14.                 (HelloService) ctx.getBean("hello3");
15.         
16.             service.sayHello();
17.         }
18. }
```

In the test program above, note that BeanFactory has been replaced by ApplicationContext, which provides support for reading messages from a properties file. In an upcoming chapter, you will learn about other features of Spring's ApplicationContext.
Schema-Based Configuration

• Newer versions of Spring configuration files may use a schema rather than a DTD.

• While using the older DTD style is still fully supported, the schema-based configuration files are somewhat easier to read and allow for customization through the use of user-defined configuration tags.

• In the schema-based configuration file, the DTD reference is replaced by special attributes on the top-level <beans> tag (shown below).

**hello-schema.xml (excerpt)**

```xml
  1. <?xml version="1.0" encoding="UTF-8"?>
  2. 
  3. <beans xmlns="http://www.springframework.org/schema/beans"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xmlns:aop="http://www.springframework.org/schema/aop"
              xmlns:tx="http://www.springframework.org/schema/tx"
              xsi:schemaLocation="http://www.springframework.org/schema/beans
                 http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
                 http://www.springframework.org/schema/aop
                 http://www.springframework.org/schema/aop/spring-aop-2.0.xsd
                 http://www.springframework.org/schema/tx
                 http://www.springframework.org/schema/tx/spring-tx-2.0.xsd">
  4.       <!-- Your bean definitions go here -->
  5. </beans>
```
Exercises

1. Add a second method to the HelloService interface.

   public void sayGoodBye();

Implement the method in the HelloServiceImpl class. Revise the Spring configuration file so that the message displayed by the sayGoodBye() method is obtained from the properties file.

   Solution: solutions.hello.HelloService.java
   solutions.hello.HelloServiceImpl.java
   solutions.hello.HelloTest.java

2. Write a class that implements the HelloHelper interface such that the convertString() method removes all spaces from the string. Modify the Spring configuration file so that your class is used as the HelloHelper.

   Solution: solutions.hello.HelloHelperNoSpaces.java
   solutions.hello.HelloTest.java

3. Revise your solution to the previous exercise so that the character to be removed can be specified in the Spring configuration file.

   Solution: solutions.hello.HelloHelperRemoveChar.java
   solutions.hello.HelloTest.java
Exercises

4. Write a class that implements the following interface.

   public interface WordTranslator {

       // This method translates a word if necessary and returns the replacement word; if translation is not required, the word is returned unchanged.

       public String translate (String word);
   }

   Your class should translate "cat" to "dog," leaving all other words unchanged. Write a test program and test your class as a Spring bean.

   Solution: solutions.hello.WordTranslator.java
              solutions.hello.WordTranslatorImpl.java
              solutions.hello.WordTranslatorTest.java

5. Revise your solution to the previous exercise so that the target and replacement words are injected through <property> settings in the Spring configuration file.

   Solution: solutions.hello.WordTranslatorImpl2.java
              solutions.hello.WordTranslatorTest2.java
Exercises

6. Revise the HelloHelperImpl class so that the convertString() method splits the string into words and passes each word to the translate() method in your implementation of the WordTranslator interface. Wire the two classes together in the Spring configuration file.

Solution: solutions.hello.HelloHelperTranslate.java
           solutions.hello.HelloTest.java
Chapter 8:
Using Hibernate with Spring

1) What is Hibernate? ................................................................. 8-2
2) Hibernate Sessions ................................................................. 8-3
3) Configuring Hibernate in Spring .............................................. 8-4
4) The HibernateTemplate ......................................................... 8-6
5) Sample Class with Annotations .............................................. 8-7
6) Sample Class and Mapping File ............................................. 8-8
7) Creating and Saving a New Entity .......................................... 8-9
8) Locating an Existing Entity .................................................... 8-10
9) Updating an Existing Entity ................................................... 8-11
10) Hibernate Query Language .................................................. 8-12
11) Executing Queries ............................................................... 8-14
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.

- Like Spring, Hibernate is based on the "Plain Old Java Objects" model (POJO) and can be used with or without an application server.

- For more information about Hibernate, see:
  
  http://www.hibernate.org

- As you might expect, Spring makes it easier to use Hibernate, primarily in the form of the following support classes designed for Hibernate 2 and Hibernate 3, respectively.
  
  - org.springframework.orm.hibernate.HibernateTemplate
  - org.springframework.orm.hibernate3.HibernateTemplate

- Like the JdbcTemplate, the HibernateTemplate provides convenience methods, and also wraps Hibernate exceptions in the unchecked Spring exception org.springframework.dao.DataAccessAccessException (or one of its subclasses).
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 persistent data store.

- A Session generally corresponds to a unit of work 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 flushed and closed.

- A Session is obtained from a SessionFactory. Spring provides the LocalSessionFactoryBean for creating a Hibernate SessionFactory from a DataSource.
  - Spring also provides the AnnotatedSessionFactoryBean if you are using Hibernate annotations.

- As shown below, the SessionFactory is passed to the HibernateTemplate via dependency injection.
Configuring Hibernate in Spring

hibern8.xml

```xml
<bean id="hibernateTemplate" class="org.springframework.orm.hibernate3.HibernateTemplate">
  <property name="sessionFactory">
    <ref bean="mySessionFactory"/>
  </property>
</bean>

<bean id="mySessionFactory" class="org.springframework.orm.hibernate3.annotations.AnnotatedSessionFactoryBean">
  <property name="dataSource" ref="myDataSource"/>
  <property name="annotatedClasses">
    <value>examples.hibern8.Product</value>
  </property>
  <property name="hibernateProperties">
    <props>
      <prop key="hibernate.dialect" value="org.hibernate.dialect.MySQLDialect"/>
      <prop key="hibernate.show_sql" value="true"/>
    </props>
  </property>
</bean>

<bean id="myDataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
  <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
  <property name="url" value="jdbc:mysql://localhost/test"/>
  <property name="username" value="root"/>
</bean>
```
Configuring Hibernate in Spring

- The `annotatedClasses` property specifies the fully-qualified names of the annotated classes.

- The `hibernateProperties` property allows you to configure Hibernate settings, which in a non-Spring environment would typically be done in the Hibernate configuration file `hibernate.cfg.xml`.
  - When set to true, `hibernate.show_sql` indicates that all SQL statements generated by Hibernate are displayed in the console window.
  - `hibernate.dialect` specifies the name of a `Dialect` class that allows Hibernate to generate SQL optimized for a particular database. `Dialect` class names for selected databases are shown in the table below.

<table>
<thead>
<tr>
<th>Database</th>
<th>Dialect</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2</td>
<td><code>org.hibernate.dialect.DB2Dialect</code></td>
</tr>
<tr>
<td>MySQL</td>
<td><code>org.hibernate.dialect.MySQLDialect</code></td>
</tr>
<tr>
<td>Oracle</td>
<td><code>org.hibernate.dialect.OracleDialect</code></td>
</tr>
<tr>
<td>Pointbase</td>
<td><code>org.hibernate.dialect.PointbaseDialect</code></td>
</tr>
<tr>
<td>HSQLDB</td>
<td><code>org.hibernate.dialect.HSQLDialect</code></td>
</tr>
<tr>
<td>Microsoft SQL Server</td>
<td><code>org.hibernate.dialect.SQLServerDialect</code></td>
</tr>
</tbody>
</table>
The HibernateTemplate

• The examples in this chapter will illustrate a number of the methods in the HibernateTemplate class.

• Creating, updating, and deleting database entities
  ▸ Serializable save (Object entity)
  ▸ void load (Object entity, Serializable id)
  ▸ void update (Object entity)
  ▸ void delete (Object entity)

• Executing queries
  ▸ List find (String query)
  ▸ List findByNamedParam (String query, String paramName, Object value)
  ▸ void setMaxResults (int maxResults)
Sample Class with Annotations

- Database Schema

<table>
<thead>
<tr>
<th>product table</th>
<th>id</th>
<th>Descrip</th>
<th>price</th>
<th>unit</th>
<th>qty</th>
</tr>
</thead>
</table>

Product.java

```
1. package examples.hibern8;
2. import javax.persistence.*;
3. @Entity
4. public class Product {
5.     @Id
6.     @Column(name="id")
7.     private String productId;
8.     @Column(name="descrip")
9.     private String description;
10.    @Column
11.    private String unit;
12.    @Column
13.    private double price;
14.    @Column(name="qty")
15.    private int qtyOnHand;
16.    // getters and setters not shown
17. }
```
Sample Class and Mapping File

• The `@Entity` annotation specifies that the persistent data fields of the class should be managed by Hibernate.

• The `@Id` annotation specifies the primary key field.

• The `@Column` annotations specify the other persistent fields.

• The `name` option is not required where the name of a field is the same as the name of the corresponding column in the database.
Creating and Saving a New Entity

CreateProduct.java

```java
1. package examples.hibern8;
2. import org.springframework.beans.factory.BeanFactory;
3. import org.springframework.beans.factory.xml.XmlBeanFactory;
4. import org.springframework.core.io.ClassPathResource;
5. import org.springframework.orm.hibernate3.HibernateTemplate;
6. public class CreateProduct {
7.     private HibernateTemplate hibernateTemplate;
8.     public static void main (String[] args) {
9.         BeanFactory factory = new XmlBeanFactory
10.            (new ClassPathResource ("hibern8.xml"));
11.         CreateProduct demo = (CreateProduct)
12.            factory.getBean("createProduct");
13.         demo.run();
14.     }
15.     public void run() {
16.         Product prod = new Product();
17.         prod.setProductId ("101");
18.         prod.setDescription ("COPY PAPER");
19.         prod.setUnit ("CS");
20.         prod.setPrice (29.00);
21.         prod.setQtyOnHand (100);
22.         hibernateTemplate.save (prod);
23.     }
24.     public void setHibernateTemplate
25.            (HibernateTemplate hibernateTemplate) {
26.         this.hibernateTemplate = hibernateTemplate;
27.     }
28. }
```
Locating an Existing Entity

LocateProduct.java

```java
1. // details omitted
2.
3. Product prod = new Product();
4.
5. hibernateTemplate.load (prod, "101");
6.
7. System.out.println ("Located Product # " +
8.   prod.getProductId());
9.
10. System.out.println ("Description = " +
11.   prod.getDescription());
12.
13. System.out.println ("Unit = " +
14.   prod.getUnit());
15.
16. System.out.println ("Price = " +
17.   prod.getPrice());
18.
19. System.out.println ("Qty = " +
20.   prod.getQtyOnHand());
21.
```
Updating an Existing Entity

UpdateProduct.java

```java
1. // details omitted
2.
3. Product prod = new Product();
4.
5. hibernateTemplate.load (prod, "101");
6.
7. prod.setDescription("New Description");
8. prod.setUnit("XX");
9. prod.setPrice(9.99);
10. prod.setQtyOnHand(1);
11.
12. hibernateTemplate.update (prod);
13.
```

DeleteProduct.java

```java
1. // details omitted
2.
3. Product prod = new Product();
4.
5. hibernateTemplate.load (prod, "101");
6.
7. hibernateTemplate.delete (prod);
8.
```
Hibernate Query Language

- Hibernate uses an SQL-like query language called Hibernate Query Language (HQL). HQL is fully object-oriented and understands concepts such as inheritance, polymorphism, and association.

- HQL queries are case-insensitive, except for the names of Java classes and properties.

- The simplest possible Hibernate query is shown below. This query returns all `Product` instances.

  ```sql
  from Product
  ```

- Usually, you will want to assign an alias to represent the instances, so that you can use the alias later in the query. To be consistent with Java naming conventions, a query alias should be named beginning with a lowercase letter. As shown below, use of the keyword `as` is optional.

  ```sql
  from Product as prod
  from Product prod
  ```

- Queries may return any of the properties of an instance.

  ```sql
  select prod.description from Product as prod
  ```

  ```sql
  select prod.productId from Product as prod
  where prod.description like 'XL%'
  ```
Hibernate Query Language

- Queries may return multiple properties as an array of type `Object[]`.
  
  ```sql
  select prod.productId, prod.unit, prod.price
  from Product as prod
  ```

- Supported functions include the following.
  - `avg`
  - `sum`
  - `min`
  - `max`
  - `count`

- The `distinct`, `order by`, and `group by` keywords may also be used, as in standard SQL.

- Examples
  
  ```sql
  select avg(prod.price), sum(prod.qtyOnHand)
  from Product as prod
  ```

  ```sql
  select count(*) from Product
  ```

  ```sql
  select distinct prod.unit from Product as prod
  ```

  ```sql
  select count(distinct prod.description)
  from Product as prod
  ```
Executing Queries

QueryProduct.java

```java
1. List myList = null;
2. Product prod = null;
3. String query1 =
4. "from Product as prod where prod.price > ?";
5. String query2 =
6. "select prod.productId, prod.description, " +
7. "prod.price from Product as prod where " +
8. "prod.price > :lowPrice";
9. String query3 =
10. "select prod.productId, prod.description from " +
11. "Product as prod order by prod.description";
12. // No select clause, retrieve entire entity
13. myList = hibernateTemplate.find(query1,
14. new Double(10.0));
15. for (Object obj : myList) {
16. prod = (Product) obj;
17. System.out.print ("Product #" +
18. prod.getProductId());
19. System.out.print (" Desc: " +
20. prod.getDescription());
21. System.out.println (" Price: " +
22. prod.getPrice());
23. }
```
Executing Queries

QueryProduct (continued)

```java
34. // Retrieve selected fields only
35. // Where clause uses named parameter
36.
37. myList = hibernateTemplate.findByNamedParam(query2,
38.     "lowPrice", new Double(25.0));
39.
40. for (Object obj : myList) {
41.     Object[] fields = (Object[]) obj;
42.     System.out.print ("Product #" + fields[0]);
43.     System.out.print (" Desc: " + fields[1]);
44.     System.out.println (" Price: " + fields[2]);
45. }
46.
47. // Set maximum no. of rows to retrieve
48.
49. hibernateTemplate.setMaxResults(10);
50. myList = hibernateTemplate.find(query3);
51.
52. for (Object obj : myList) {
53.     Object[] fields = (Object[]) obj;
54.     System.out.print ("Product #" + fields[0]);
55.     System.out.println (" Desc: " + fields[1]);
56. }
```
Exercises

1. Add annotations for Hibernate to the Customer class (from the exercises in the previous chapter).
   - Solution: solutions.hibern8.Customer

2. Write a program (or set of programs) with the following functionality.
   - Create a new customer.
   - Locate and display an existing customer.
   - Update an existing customer.
   - Delete an existing customer.
   - Solution: solutions.hibern8.CustomerAccess

3. Write a program to perform the following queries on the customer table and display the results.
   - Find the id's and names of all customers in California (CA).
   - Find the name of the customer with id = '0114.'
   - Find all customers with zipcode beginning with '4' (show all fields).
     - Hint: Use the SQL like operator.
   - Find the name, city, and limit of the customer with highest credit limit.
     - Hint: Use two queries. The first query should use the SQL max function to get the highest value in the limit column.
   - Solution: solutions.hibern8.CustomerQueries