



# INSTITUT TEKNOLOGI DEL

## MATERI PRAKTIKUM PEMROGRAMAN BERORIENTASI OBJEK Program Diploma

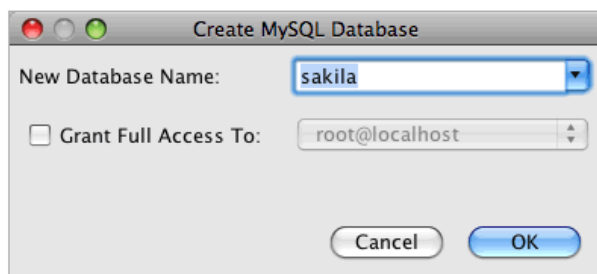
Topic	:	Hibernate											
Activity	:	coding											
Duration	:	Week 10											
Deliverables		DVDStoreAdmin.rar (project DVDStoreAdmin)											
Due Date	:	13 April 2018, 20.00											
Deliver to	:	Ecourse											
Goals	:	Students understand: ORM in Java and how to use Hibernate											
Requirements	:	<table><tr><th>Software or Resource</th><th>Version Required</th></tr><tr><td>NetBeans IDE</td><td>7.2, 7.3, 7.4, 8.0, Java</td></tr><tr><td>Java Development Kit (JDK)</td><td>version 7 or 8</td></tr><tr><td>MySQL database server</td><td>version 5.x</td></tr><tr><td>Sakila Database</td><td>plugin available from update center</td></tr></table>		Software or Resource	Version Required	NetBeans IDE	7.2, 7.3, 7.4, 8.0, Java	Java Development Kit (JDK)	version 7 or 8	MySQL database server	version 5.x	Sakila Database	plugin available from update center
Software or Resource	Version Required												
NetBeans IDE	7.2, 7.3, 7.4, 8.0, Java												
Java Development Kit (JDK)	version 7 or 8												
MySQL database server	version 5.x												
Sakila Database	plugin available from update center												

### Instructions

#### 1. Creating the Database

The sakila database is added to the list of databases in the Create MySQL database dialog box.

- Open the Plugins manager and install the Sakila Sample Database plugin.
- After installing the plugin, start the MySQL database server by expanding the Databases node in the Services window, right-clicking the MySQL Server node and choosing Start.
- Right-click the MySQL Server node and choose Create Database.
- Select the Sakila database from the New Database Name drop down list in the Create MySQL Database dialog box. Click OK.



When you click OK a Sakila node appears under the MySQL Server node.

- Right-click the Sakila node and choose Connect

When you click Connect a database connection node for the Sakila database

(jdbc:mysql://localhost:3306/sakila [username on Default]) is listed under the Databases node. When a connection is open you can view the data in the database by expanding the connection node.



# INSTITUT TEKNOLOGI DEL

## MATERI PRAKTIKUM PEMROGRAMAN BERORIENTASI OBJEK Program Diploma

### 2. Creating the Java Swing Application Project

In this exercise you create a simple Java Swing application project called DVDStoreAdmin.

- Choose File > New Project (Ctrl-Shift-N). Select Java Application from the Java category and click Next.
- Type DVDStoreAdmin for the project name and set the project location.
- Deselect the Use Dedicated Folder option, if selected.
- For this tutorial there is little reason to copy project libraries to a dedicated folder because you will not need to share libraries with other users.
- Deselect Create Main Class. Click Finish.
- When you click Finish, the IDE creates the Java application project. The project does not have a main class. You will create a form and then set the form as the main class.

### 3. Adding Hibernate Support to the Project

To add support for Hibernate to your project you need to add the Hibernate library to the project. The Hibernate library is included with the IDE and can be added to any project by right-clicking the 'Libraries' node in the Projects window, selecting 'Add Library' and then selecting the Hibernate library in the Add Library dialog box.

The IDE includes wizards to help you create the Hibernate files you may need in your project. You can use the wizards in the IDE to create a Hibernate configuration file and a utility helper class. If you create the Hibernate configuration file using a wizard the IDE automatically adds the Hibernate libraries to the project.

#### Creating the Hibernate Configuration File

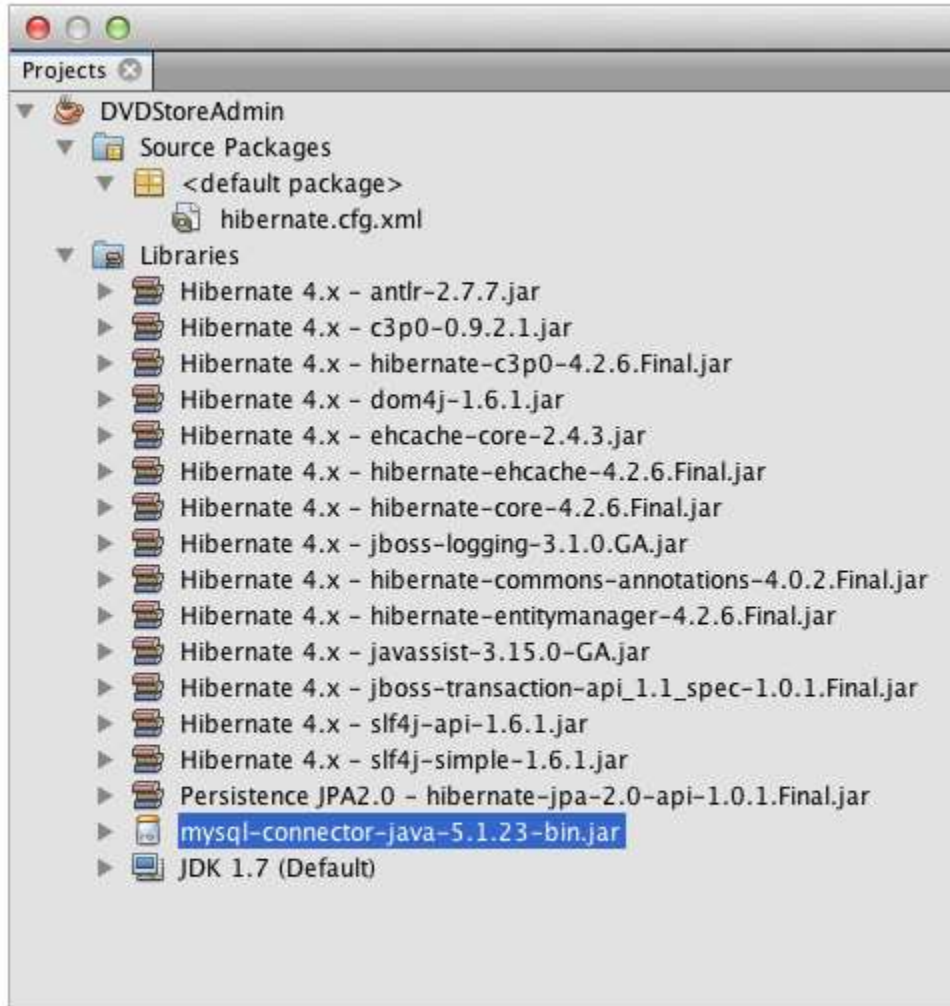
The Hibernate configuration file (hibernate.cfg.xml) contains information about the database connection, resource mappings, and other connection properties. When you create a Hibernate configuration file using a wizard you specify the database connection by choosing from a list of database connection registered with the IDE. When generating the configuration file the IDE automatically adds the connection details and dialect information based on the selected database connection. The IDE also automatically adds the Hibernate library to the project classpath. After you create the configuration file you can edit the file using the multi-view editor, or edit the XML directly in the XML editor.

- Right-click the Source Packages node in the Projects window and choose New > Other to open the New File wizard.
- Select Hibernate Configuration Wizard from the Hibernate category. Click Next.
- Keep the default settings in the Name and Location pane (you want to create the file in the src directory). Click Next.
- Select the sakila connection in the Database Connection drop down list. Click Finish.



When you click Finish the IDE opens hibernate.cfg.xml in the source editor. The IDE creates the configuration file at the root of the context classpath of the application (in the Files window, WEB-INF/classes). In the Projects window the file is located in the <default package> source package. The configuration file contains information about a single database. If you plan to connect to multiple databases, you can create multiple configuration files in the project, one for each database servers, but by default the helper utility class will use the hibernate.cfg.xml file located in the root location.

If you expand the Libraries node in the Projects window you can see that the IDE added the required Hibernate JAR files and the MySQL connector JAR.



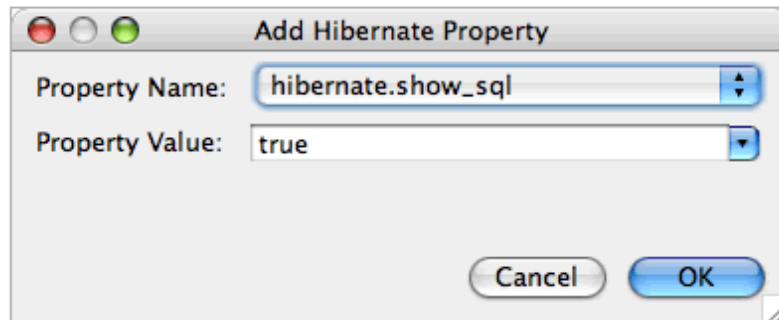
 **Note.** NetBeans IDE 8.0 bundles the Hibernate 4 libraries. Older versions of the IDE bundled Hibernate 3.

## Modifying the Hibernate Configuration File

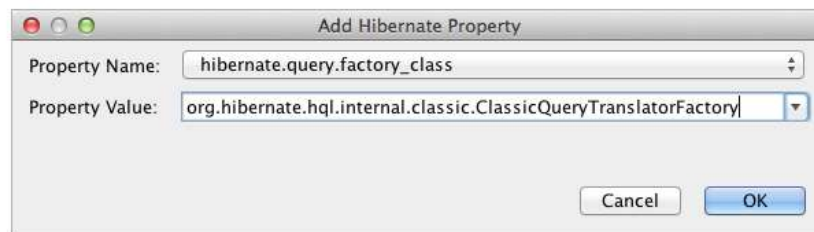
In this exercise you will edit the default properties specified in `hibernate.cfg.xml` to enable debug logging for SQL statements.

- Open `hibernate.cfg.xml` in the Design tab. You can open the file by expanding the Configuration Files node in the Projects window and double-clicking `hibernate.cfg.xml`.
- Expand the Configuration Properties node under Optional Properties.
- Click Add to open the Add Hibernate Property dialog box.

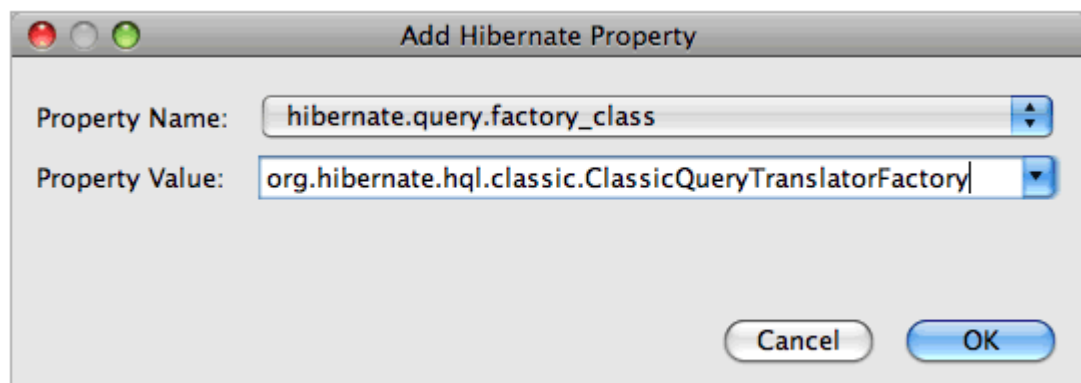
- d. In the dialog box, select the `hibernate.show_sql` property and set the value to `true`. Click OK. This enables the debug logging of the SQL statements.



- e. Click Add under the Miscellaneous Properties node and select `hibernate.query.factory_class` in the Property Name dropdown list.
- f. Type `org.hibernate.hql.internal.classic.ClassicQueryTranslatorFactory` as the Property Value. This is the translator factory class that is used in Hibernate 4 that is bundled with the IDE. Click OK.



⚠ If you are using NetBeans IDE 7.4 or earlier you should select `org.hibernate.hql.classic.ClassicQueryTranslatorFactory` as the Property Value in the dialog box. NetBeans IDE 7.4 and earlier bundled Hibernate 3.



If you click the XML tab in the editor you can see the file in XML view. Your file should look like the following:

```
<hibernate-configuration>
  <session-factory name="session1">
    <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
    <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</prope
    <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/sakila</p
    <property name="hibernate.connection.username">root</property>
    <property name="hibernate.connection.password">#####</property>
    <property name="hibernate.show_sql">true</property>
    <property name="hibernate.query.factory_class">org.hibernate.hql.internal.class
  </session-factory>
</hibernate-configuration>
```

- g. Save your changes to the file.

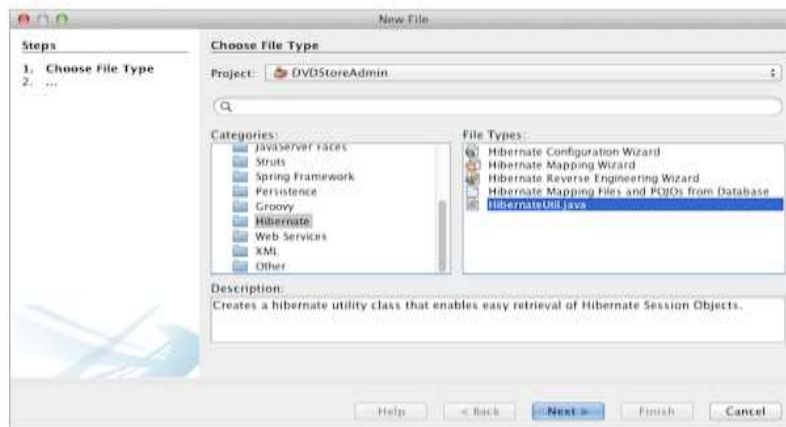
After you create the form and set it as the main class you will be able to see the SQL query printed in the IDE's Output window when you run the project.

## Creating the HibernateUtil.java Helper File

To use Hibernate you need to create a helper class that handles startup and that accesses Hibernate's SessionFactory to obtain a Session object. The class calls Hibernate's configure() method, loads the hibernate.cfg.xml configuration file and then builds the SessionFactory to obtain the Session object.

In this section you use the New File wizard to create the helper class HibernateUtil.java.

- a. Right-click the Source Packages node and select New > Other to open the New File wizard.
- b. Select Hibernate from the Categories list and HibernateUtil.java from the File Types list. Click Next.



- c. Type HibernateUtil for the class name and sakila.util as the package name. Click Finish.



# INSTITUT TEKNOLOGI DEL

## MATERI PRAKTIKUM PEMROGRAMAN BERORIENTASI OBJEK Program Diploma

When you click Finish, HibernateUtil.java opens in the editor. You can close the file because you do not need to edit the file.

#### 4. Generating Hibernate Mapping Files and Java Classes

In this tutorial you use a plain old Java object (POJO), Actor.java, to represent the data in the table ACTOR in the database. The class specifies the fields for the columns in the tables and uses simple setters and getters to retrieve and write the data. To map Actor.java to the ACTOR table you can use a Hibernate mapping file or use annotations in the class.

You can use the Reverse Engineering wizard and the Hibernate Mapping Files and POJOs from a Database wizard to create multiple POJOs and mapping files based on database tables that you select. Alternatively, you can use wizards in the IDE to help you create individual POJOs and mapping files from scratch.

Notes.

When you want to create files for multiple tables you will most likely want to use the wizards. In this tutorial you only need to create one POJO and one mapping file so it is fairly easy to create the files individually. You can see the steps for creating the POJOs and mapping files individually at the end of this tutorial.

##### Creating the Reverse Engineering File

The reverse engineering file (hibernate.reveng.xml) is an XML file that can be used to modify the default settings used when generating Hibernate files from the metadata of the database specified in hibernate.cfg.xml. The wizard generates the file with basic default settings. You can modify the file to explicitly specify the database schema that is used, to filter out tables that should not be used and to specify how JDBC types are mapped to Hibernate types.

- Right-click the Source Packages node and select New > Other to open the New File wizard.
- Select Hibernate from the Categories list and Hibernate Reverse Engineering Wizard from the File Types list. Click Next.
- Type hibernate.reveng for the file name.
- Keep the default src as the Location. Click Next.
- Select actor in the Available Tables pane and click Add. Click Finish.

The wizard generates a hibernate.reveng.xml reverse engineering file. You can close the reverse engineering file because you will not need to edit the file.

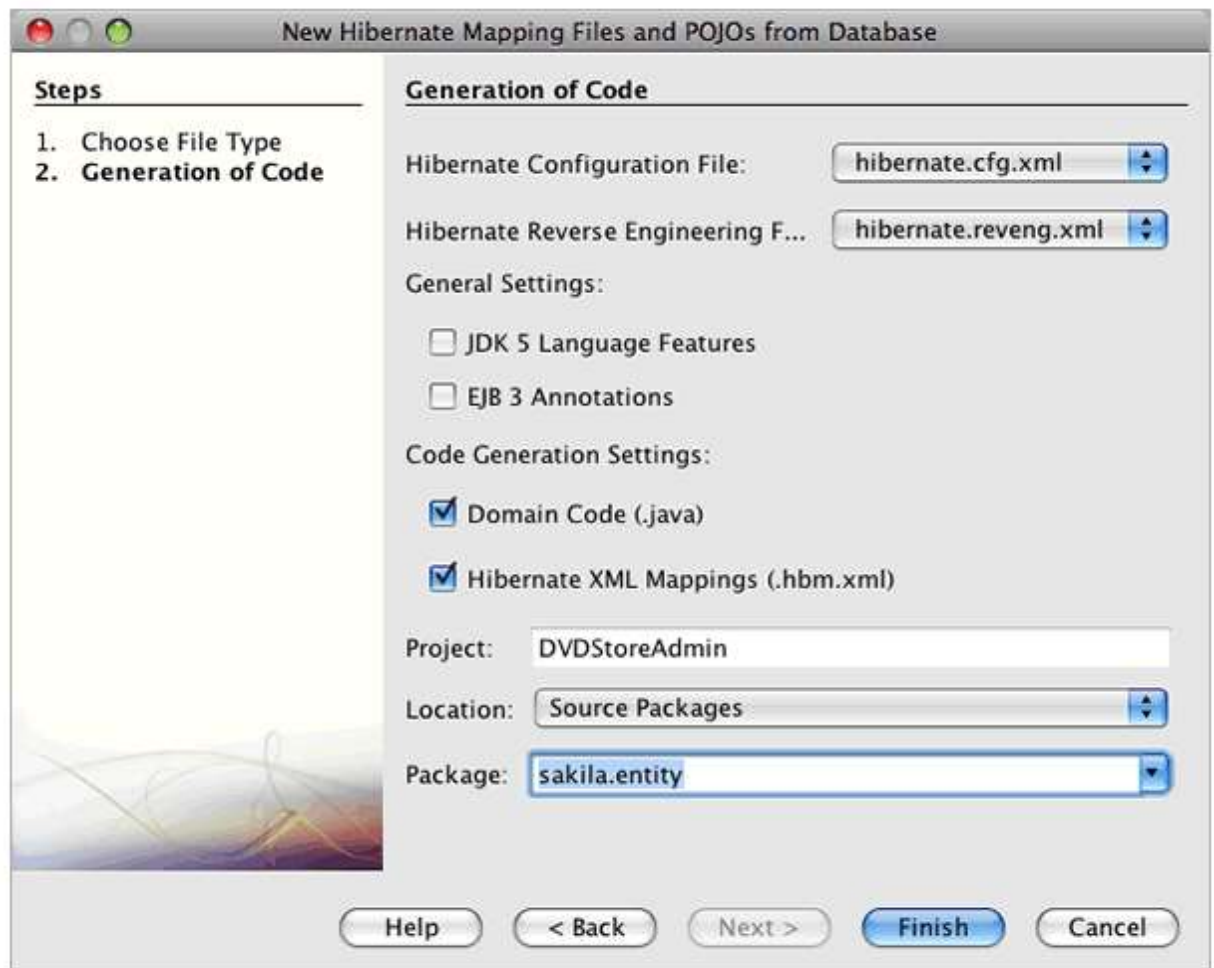
##### Creating Hibernate Mapping Files and POJOs From a Database

The Hibernate Mapping Files and POJOs from a Database wizard generates files based on tables in a database. When you use the wizard, the IDE generates POJOs and mapping files for you based on the database tables specified in hibernate.reveng.xml and then adds the mapping entries to



hibernate.cfg.xml. When you use the wizard you can choose the files that you want the IDE to generate (only the POJOs, for example) and select code generation options (generate code that uses EJB 3 annotations, for example).

- Right-click the Source Packages node in the Projects window and choose New > Other to open the New File wizard.
- Select Hibernate Mapping Files and POJOs from a Database in the Hibernate category. Click Next.
- Select hibernate.cfg.xml from the Hibernate Configuration File dropdown list, if not selected.
- Select hibernate.reveng.xml from the Hibernate Reverse Engineering File dropdown list, if not selected.
- Ensure that the Domain Code and Hibernate XML Mappings options are selected.
- Type sakila.entity for the Package name. Click Finish.







# INSTITUT TEKNOLOGI DEL

## MATERI PRAKTIKUM PEMROGRAMAN BERORIENTASI OBJEK Program Diploma

When you click Finish, the IDE generates the POJO Actor.java with all the required fields and generates a Hibernate mapping file and adds the mapping entry to hibernate.cfg.xml.

Now that you have the POJO and necessary Hibernate-related files you can create a simple Java GUI front end for the application. You will also create and then add an HQL query that queries the database to retrieve the data. In this process we also use the HQL editor to build and test the query.

### 5. Creating the Application GUI

In this exercise you will create a simple JFrame Form with some fields for entering and displaying data. You will also add a button that will trigger a database query to retrieve the data.

#### Creating the JFrame Form

- Right-click the project node in the Projects window and choose New > Other to open the New File wizard.
- Select JFrame Form from the Swing GUI Forms category. Click Next.
- Type DVDStoreAdmin for the Class Name and type sakila.ui for the Package. Click Finish.
- When you click Finish the IDE creates the class and opens the JFrame Form in the Design view of the editor.

#### Adding Elements to the Form

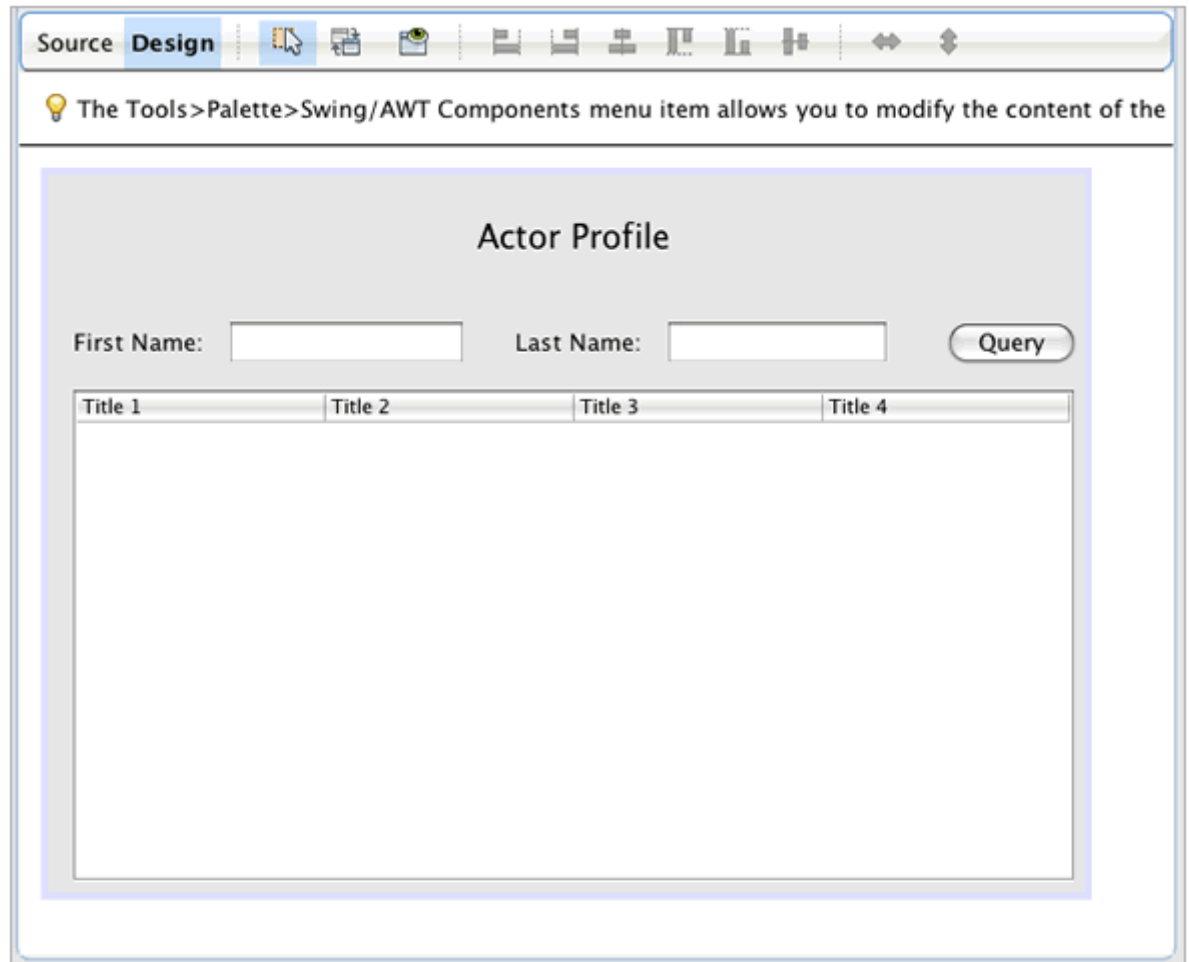
You now need to add the UI elements to the form. When the form is open in Design view in the editor, the Palette appears in the left side of the IDE. To add an element to the form, drag the element from the Palette into the form area. After you add an element to the form you need to modify the default value of the Variable Name property for that element.

- Drag a Label element from the Palette and change the text to Actor Profile.
- Drag a Label element from the Palette and change the text to First Name.
- Drag a Text Field element next to the First Name label and delete the default text.
- Drag a Label element from the Palette and change the text to Last Name.
- Drag a Text Field element next to the Last Name label and delete the default text.
- Drag a Button element from the Palette and change the text to Query.
- Drag a Table element from the Palette into the form.
- Modify the Variable Name values of the following UI elements according to the values in the following table. You can modify the Variable Name value of an element by right-clicking the element in the Design view and then choosing Change Variable Name. Alternatively, you can change the Variable Name directly in the Inspector window.

Element	Variable Name
First Name text field	<code>firstNameTextField</code>
Last Name text field	<code>lastNameTextField</code>
Query button	<code>queryButton</code>
Table	<code>resultTable</code>

- i. Save your changes.

In Design view your form should look similar to the following image.



The screenshot shows the NetBeans IDE in Design view. The form is titled 'Actor Profile'. It features a toolbar at the top with various icons for design and development. Below the toolbar, there is a message: 'The Tools>Palette>Swing/AWT Components menu item allows you to modify the content of the'. The form itself has a light blue background and contains the following elements:

- Two text input fields: 'First Name:' and 'Last Name:'.
- A 'Query' button.
- A table with four columns: 'Title 1', 'Title 2', 'Title 3', and 'Title 4'.

Now that you have a form you need to create the code to assign events to the form elements. In the next exercise you will construct queries based on Hibernate Query Language to retrieve data. After you construct the queries you will add methods to the form to invoke the appropriate query when the Query button is pressed.

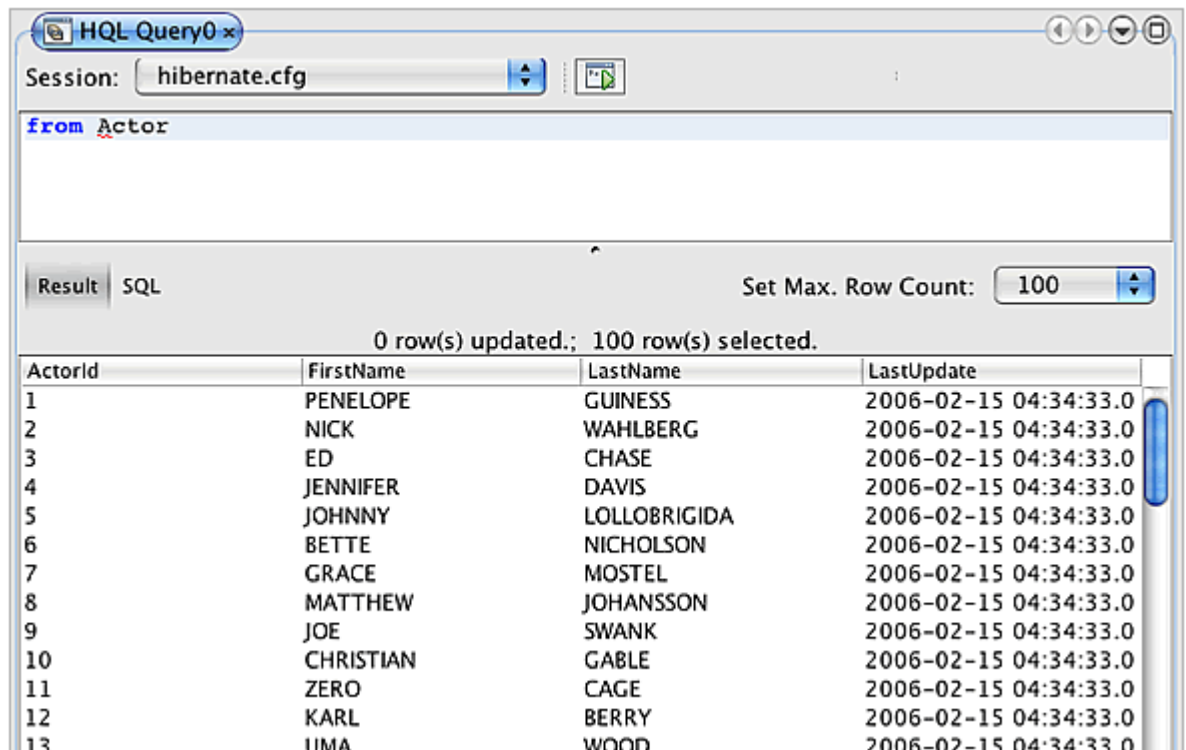
You do not need to assign Variable Name values to the Label elements.

## 6. Creating the Query in the HQL Query Editor

In the IDE you can construct and test queries based on the Hibernate Query Language (HQL) using the HQL Query Editor. As you type the query the editor shows the equivalent (translated) SQL query. When you click the 'Run HQL Query' button in the toolbar, the IDE executes the query and shows the results at the bottom of editor.

In this exercise you use the HQL Editor to construct simple HQL queries that retrieve a list of actors' details based on matching the first name or last name. Before you add the query to the class you will use the HQL Query Editor to test that the connection is working correctly and that the query produces the desired results. Before you can run the query you first need to compile the application.

- Right-click the project node and choose Build.
- Expand the <default package> source package node in the Projects window.
- Right-click hibernate.cfg.xml and choose Run HQL Query to open the HQL Editor.
- Test the connection by typing from Actor in the HQL Query Editor. Click the Run HQL Query button ( Run HQL Query button ) in the toolbar.



The screenshot shows the HQL Query Editor window with the session set to 'hibernate.cfg'. The query entered is 'from Actor'. The results are displayed in a table with 13 rows. The table has columns: ActorId, FirstName, LastName, and LastUpdate. The results show a list of actors with their IDs, first names, last names, and last update timestamps.

ActorId	FirstName	LastName	LastUpdate
1	PENELOPE	GUINNESS	2006-02-15 04:34:33.0
2	NICK	WAHLBERG	2006-02-15 04:34:33.0
3	ED	CHASE	2006-02-15 04:34:33.0
4	JENNIFER	DAVIS	2006-02-15 04:34:33.0
5	JOHNNY	LOLOBRIGIDA	2006-02-15 04:34:33.0
6	BETTE	NICHOLSON	2006-02-15 04:34:33.0
7	GRACE	MOSTEL	2006-02-15 04:34:33.0
8	MATTHEW	JOHANSSON	2006-02-15 04:34:33.0
9	JOE	SWANK	2006-02-15 04:34:33.0
10	CHRISTIAN	GABLE	2006-02-15 04:34:33.0
11	ZERO	CAGE	2006-02-15 04:34:33.0
12	KARL	BERRY	2006-02-15 04:34:33.0
13	LIMA	WOOD	2006-02-15 04:34:33.0

When you click Run HQL Query you should see the query results in the bottom pane of the HQL Query Editor.



# INSTITUT TEKNOLOGI DEL

## MATERI PRAKTIKUM PEMROGRAMAN BERORIENTASI OBJEK Program Diploma

- e. Type the following query in the HQL Query Editor and click Run HQL Query to check the query results when the search string is 'PE'.

```
from Actor a where a.firstName like 'PE%'
```

The query returns a list of actors' details for those actors whose first names begin with 'PE'. If you click the SQL button above the results you should see the following equivalent SQL query.

```
select actor0_.actor_id as col_0_0_ from sakila.actor actor0_ where (actor0_.first_name
```

- f. Open a new HQL Query Editor tab and type the following query in the editor pane. Click Run HQL Query.

```
from Actor a where a.lastName like 'MO%'
```

The query returns a list of actors' details for those actors whose last names begin with 'MO'.

Testing the queries shows that the queries return the desired results. The next step is to implement the queries in the application so that the appropriate query is invoked by clicking the Query button in the form.

### 7. Adding the Query to the Form

You now need to modify DVDStoreAdmin.java to add the query strings and create the methods to construct and invoke a query that incorporates the input variables. You also need to modify the button event handler to invoke the correct query and add a method to display the query results in the table.

- a. Open DVDStoreAdmin.java and click the Source tab.  
b. Add the following query strings (in bold) to the class.

```
public DVDStoreAdmin() {  
    initComponents();  
}  
  
private static String QUERY_BASED_ON_FIRST_NAME="from Actor a where a.firstName like '"  
private static String QUERY_BASED_ON_LAST_NAME="from Actor a where a.lastName like '"
```

**TIP** It is possible to copy the queries from the HQL Query Editor tabs into the file and then modify the code.

- c. Add the following methods to create the query based on the user input string.



# INSTITUT TEKNOLOGI DEL

## MATERI PRAKTIKUM PEMROGRAMAN BERORIENTASI OBJEK Program Diploma

```
private void runQueryBasedOnFirstName() {  
    executeHQLQuery(QUERY_BASED_ON_FIRST_NAME + firstNameTextField.getText() + "%");  
}  
  
private void runQueryBasedOnLastName() {  
    executeHQLQuery(QUERY_BASED_ON_LAST_NAME + lastNameTextField.getText() + "%");  
}
```

The methods call a method called `executeHQLQuery()` and create the query by combining the query string with the user entered search string.

- d. Add the `executeHQLQuery()` method.

```
private void executeHQLQuery(String hql) {  
    try {  
        Session session = HibernateUtil.getSessionFactory().openSession();  
        session.beginTransaction();  
        Query q = session.createQuery(hql);  
        List resultList = q.list();  
        displayResult(resultList);  
        session.getTransaction().commit();  
    } catch (HibernateException he) {  
        he.printStackTrace();  
    }  
}
```

The `executeHQLQuery()` method calls Hibernate to execute the selected query. This method makes use of the `HibernateUtil.java` utility class to obtain the Hibernate Session.

- e. Right-click in the editor and choose Fix Imports (Ctrl-Shift-I; ⌘-Shift-I on Mac) to generate import statements for the Hibernate libraries (`org.hibernate.Query`, `org.hibernate.Session`) and `java.util.List`. Save your changes.
- f. Create a Query button event handler by switching to the Design view and double-clicking the Query button.
- g. The IDE creates the `queryButtonActionPerformed` method and displays the method in the Source view.
- h. Modify the `queryButtonActionPerformed` method in the Source view by adding the following code so that a query is run when the user clicks the button.



# INSTITUT TEKNOLOGI DEL

## MATERI PRAKTIKUM PEMROGRAMAN BERORIENTASI OBJEK Program Diploma

```
private void queryButtonActionPerformed(java.awt.event.ActionEvent evt) {  
    if(!firstNameTextField.getText().trim().equals("")) {  
        runQueryBasedOnFirstName();  
    } else if(!lastNameTextField.getText().trim().equals("")) {  
        runQueryBasedOnLastName();  
    }  
}
```

- i. Add the following method to display the results in the JTable.

```
private void displayResult(List resultList) {  
    Vector<String> tableHeaders = new Vector<String>();  
    Vector tableData = new Vector();  
    tableHeaders.add("ActorId");  
    tableHeaders.add("FirstName");  
    tableHeaders.add("LastName");  
    tableHeaders.add("LastUpdated");  
  
    for(Object o : resultList) {  
        Actor actor = (Actor)o;  
        Vector<Object> oneRow = new Vector<Object>();  
        oneRow.add(actor.getActorId());  
        oneRow.add(actor.getFirstName());  
        oneRow.add(actor.getLastName());  
        oneRow.add(actor.getLastUpdate());  
        tableData.add(oneRow);  
    }  
    resultTable.setModel(new DefaultTableModel(tableData, tableHeaders));  
}
```

- j. Right-click in the editor and choose Fix Imports (Ctrl-Shift-I; ⌘-Shift-I on Mac) to generate an import statement for java.util.Vector and java.util.List. Save your changes.

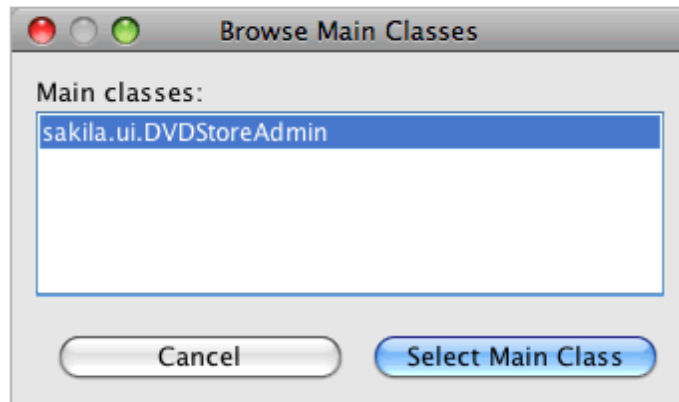
After you save the form you can run the project

### 8. Running the Project

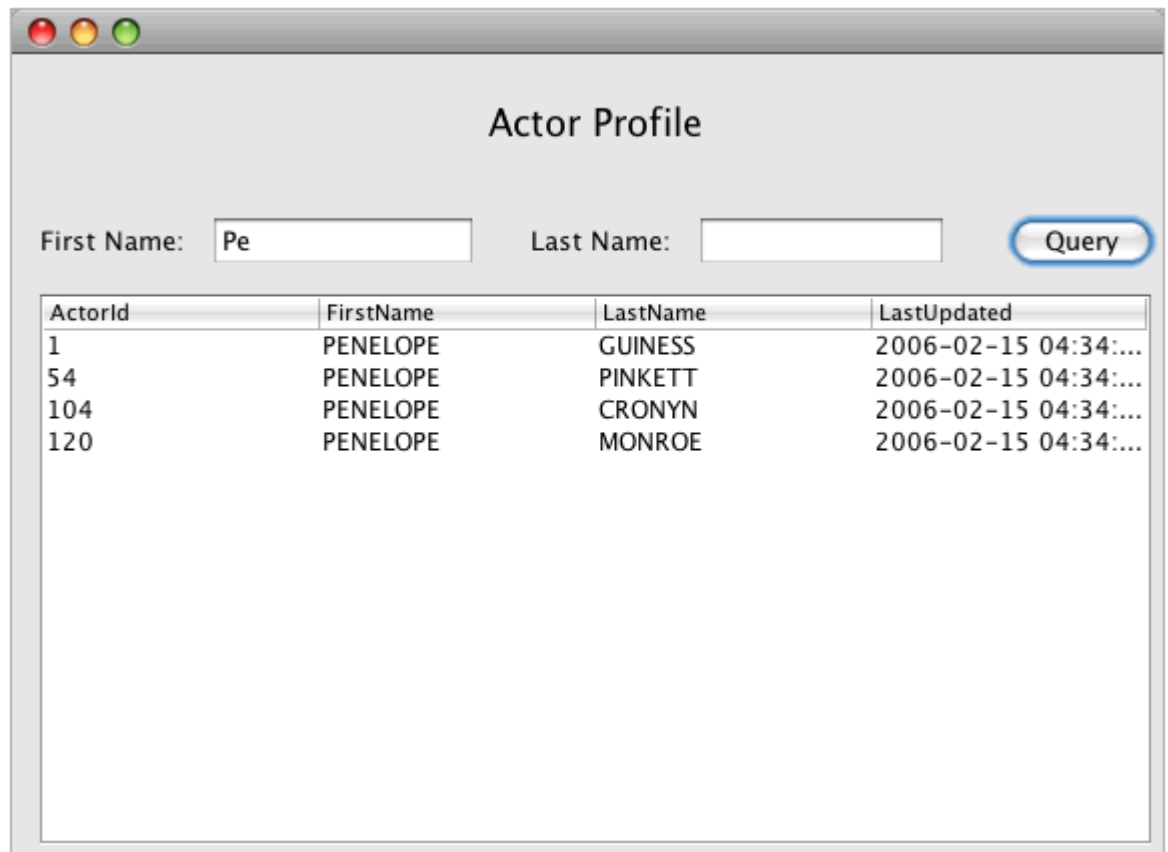
Now that the coding is finished, you can launch the application. Before you run the project, you need to specify the application's Main Class in the project's properties dialog box. If no Main Class is specified, you are prompted to set it the first time that you run the application.

- a. Right-click the project node in the Projects window and choose Properties.
- b. Select the Run category in the Project Properties dialog box.
- c. Type sakila.ui.DVDStoreAdmin for the Main Class. Click OK.
- d. Alternatively, you can click the Browse button and choose the main class in the dialog box.





- e. Click Run Project in the main toolbar to launch the application.  
Type in a search string in the First Name or Last Name text field and click Query to search for an actor and see the details.



- f. If you look in the Output window of the IDE you can see the SQL query that retrieved the displayed results.



# INSTITUT TEKNOLOGI DEL

## MATERI PRAKTIKUM PEMROGRAMAN BERORIENTASI OBJEK Program Diploma

### 9. Creating POJOs and Mapping Files Individually

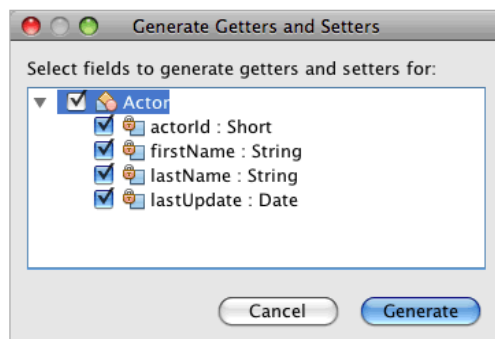
Because a POJO is a simple Java class you can use the New Java Class wizard to create the class and then edit the class in the source editor to add the necessary fields and getters and setters. After you create the POJO you then use a wizard to create a Hibernate mapping file to map the class to the table and add mapping information to hibernate.cfg.xml. When you create a mapping file from scratch you need to map the fields to the columns in the XML editor.

Note. This exercise is optional and describes how to create the POJO and mapping file that you created with the Hibernate Mapping Files and POJOs from Database wizard.

- Right-click the Source Packages node in the Projects window and choose New > Java Class to open the New Java Class wizard.
- In the wizard, type Actor for the class name and type sakila.entity for the package. Click Finish.
- Make the following changes (displayed in bold) to the class to implement the Serializable interface and add fields for the table columns.

```
public class Actor implements Serializable {  
    private Short actorId;  
    private String firstName;  
    private String lastName;  
    private Date lastUpdate;  
}
```

- Right-click in the editor and choose Insert Code (Alt-Insert; Ctrl-I on Mac) and select Getter and Setter in the popup menu to generate getters and setters for the fields.
- In the Generate Getters and Setters dialog box, select all the fields and click Generate

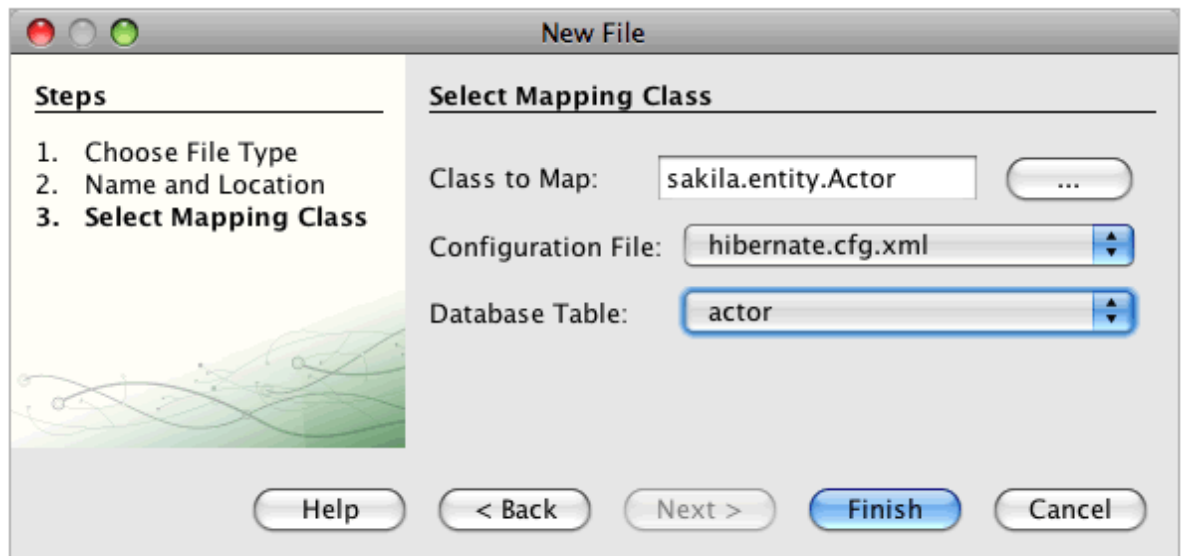


**TIP:** In the Generate Getters and Setters dialog box, you can use the Up arrow on the keyboard to move the selected item to the Actor node and then press the Space bar to select all fields in Actor.

- Fix your imports and save your changes.

After you create the POJO for the table you will want to create an Hibernate Mapping File for Actor.java.

- Right-click the sakila.entity source packages node in the Projects window and choose New > Other to open the New File wizard.
- Select Hibernate Mapping Wizard in the Hibernate category. Click Next.
- Type Actor.hbm for the File Name and check that the Folder is src/sakila/entity. Click Next.
- Type sakila.entity.Actor for the Class to Map and select actor from the Database Table drop down list. Click Finish.



When you click Finish the Actor.hbm.xml Hibernate mapping file opens in the source editor. The IDE also automatically adds an entry for the mapping resource to hibernate.cfg.xml. You can view the entry details by expanding the Mapping node in the Design view of hibernate.cfg.xml or in the XML view. The mapping entry in the XML view will look like the following:

```
<mapping resource="sakila/entity/Actor.hbm.xml"/>
</session-factory>
</hibernate-configuration>
```

- Map the fields in Actor.java to the columns in the ACTOR table by making the following changes (in bold) to Actor.hbm.xml



# INSTITUT TEKNOLOGI DEL

## MATERI PRAKTIKUM PEMROGRAMAN BERORIENTASI OBJEK Program Diploma

```
<hibernate-mapping>
  <class name="sakila.entity.Actor" table="actor">
    <id name="actorId" type="java.lang.Short">
      <column name="actor_id"/>
      <generator class="identity"/>
    </id>
    <property name="firstName" type="string">
      <column length="45" name="first_name" not-null="true"/>
    </property>
    <property name="lastName" type="string">
      <column length="45" name="last_name" not-null="true"/>
    </property>
    <property name="lastUpdate" type="timestamp">
      <column length="19" name="last_update" not-null="true"/>
    </property>
  </class>
</hibernate-mapping>
```

**TIP** You can use code completion in the editor to complete the values when modifying the mapping file.

Note: By default, the generated class element has a closing tag. Because you need to add property elements between the opening and closing class element tags, you need to make the following changes (displayed in bold). After making the changes you can then use code completion between the class tags.

```
<hibernate-mapping>
  <class name="sakila.entity.Actor" table="actor">
    </class>
</hibernate-mapping>
```

- f. Click the Validate XML button in the toolbar and save your changes.
- Creating individual POJOs and Hibernate mapping files might be a convenient way to further customizing your application.



# **INSTITUT TEKNOLOGI DEL**

**MATERI PRAKTIKUM  
PEMROGRAMAN BERORIENTASI OBJEK  
Program Diploma**