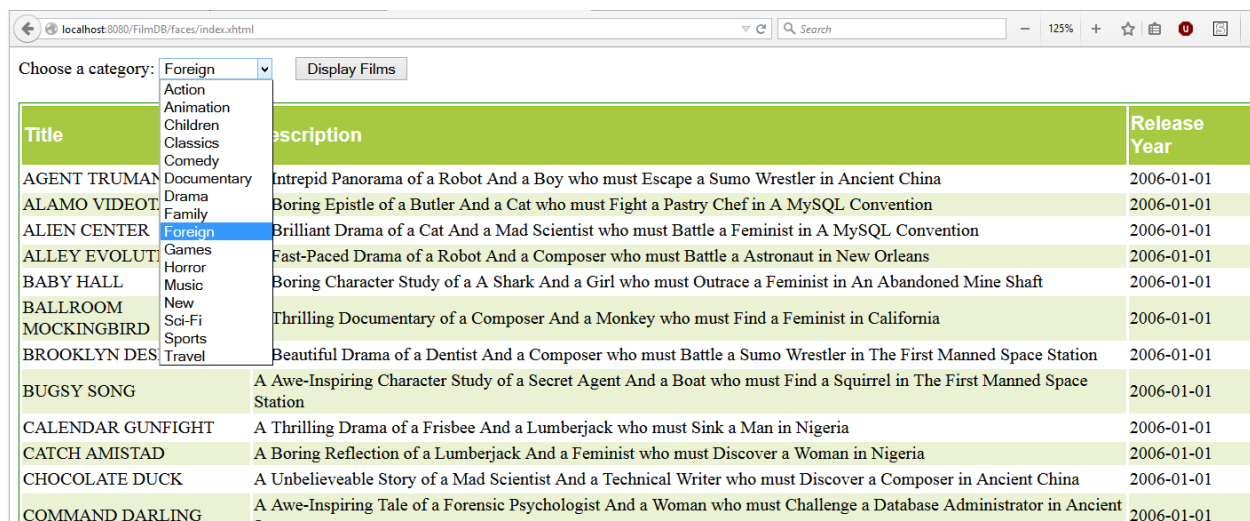


CIS 494 – Business Systems Development with Java – Spring 2015

In-class Exercise 11: JSF Web Application Using JDBC

Due on Blackboard: Friday, April 24 by 11:59 PM



Title	Description	Release Year
AGENT TRUMAN	Intrepid Panorama of a Robot And a Boy who must Escape a Sumo Wrestler in Ancient China	2006-01-01
ALAMO VIDEOT	Boring Epistle of a Butler And a Cat who must Fight a Pastry Chef in A MySQL Convention	2006-01-01
ALIEN CENTER	Brilliant Drama of a Cat And a Mad Scientist who must Battle a Feminist in A MySQL Convention	2006-01-01
ALLEY EVOLUT	Fast-Paced Drama of a Robot And a Composer who must Battle a Astronaut in New Orleans	2006-01-01
BABY HALL	Boring Character Study of a A Shark And a Girl who must Outrace a Feminist in An Abandoned Mine Shaft	2006-01-01
BALLROOM	Thrilling Documentary of a Composer And a Monkey who must Find a Feminist in California	2006-01-01
MOCKINGBIRD	Beautiful Drama of a Dentist And a Composer who must Battle a Sumo Wrestler in The First Manned Space Station	2006-01-01
BROOKLYN DES	A Awe-Inspiring Character Study of a Secret Agent And a Boat who must Find a Squirrel in The First Manned Space Station	2006-01-01
BUGSY SONG	A Thrilling Drama of a Frisbee And a Lumberjack who must Sink a Man in Nigeria	2006-01-01
CALENDAR GUNFIGHT	A Boring Reflection of a Lumberjack And a Feminist who must Discover a Woman in Nigeria	2006-01-01
CATCH AMISTAD	A Unbelievable Story of a Mad Scientist And a Technical Writer who must Discover a Composer in Ancient China	2006-01-01
CHOCOLATE DUCK	A Awe-Inspiring Tale of a Forensic Psychologist And a Woman who must Challenge a Database Administrator in Ancient	2006-01-01
COMMAND DARLING		

This project will involve creating a JSF web application that shows users a list of film categories in the database. Users will be able to select the name of a category from a combo box. The web application client (facelet running in the web browser) will submit the category to the server (JSF Managed Bean).

The Bean will form an SQL query and run it off the database using JDBC. The Bean will then “return” a list of all the movies in the selected category, which will be filled into the facelet. JSF Table controls will take care of populating a table with the data!

You will also attach a Cascading Style Sheet to the facelet to format the table nicely.

Before You Begin

You will use the following tools: JDBC, MySQL, NetBeans (and its associated subcomponents). Make sure you have these tools properly configured. The sample database sakila will be used for this exercise.

Use your previous NetBeans project to recollect how to create JSF Facelets, elements, JSF expressions, etc.

Instructions

1. Create a new NetBeans project called “FilmDB”.
- 2.1 Add a reference to cdi-api.jar within the libraries just like you did for the previous JDBC exercise. It is found in the Modules folder of Glassfish. This will ensure the correct classes are used.
- 2.2 Add a reference to the MySQL Java Connector jar within the libraries. You did this previously in Eclipse for the JDBC in-class. The Connector J is stored in the MySQL folder within Program Files (x86).
3. Create the Managed Bean class
- 3.1 Create a Managed Bean in NetBeans called CategoryJSFBean. Choose **application** scope. Change the Name at the bottom (this is the name of the instance, not Class name) to “categoryBean”.
- 3.2 We are going to use JDBC – add java.sql.* to the list of import statements. You will also need an ArrayList, so add an import statement for that too.
- 3.3 Verify that your auto-generated code contains @Name and @ApplicationScoped attributes and that your instance name is set to categoryBean.
- 3.4
 - Create the following instance variables:
 - categoryStatement, of type PreparedStatement
 - choice, of type String // this will store which category the user selected
 - categories, of type array of Strings // this will be list of categories being displayed
 - Generate a getter/setter pair for *choice* and *categories*. Use the Refactor menu!

3.5 Create a method called `loadCategories`. This method will query the database and prepare the SQL statements. It will use JDBC. You may look up your old JDBC code from the Eclipse project. Since we are still using Java, the JDBC code does not change! *This is one of the benefits of using a fully equipped programming language rather than a scripting language dedicated to one functional task – you can port code to various devices and/or platforms.*

- Load the JDBC Driver. *You must have added a reference to the JDBC jar file!*
- Create a connection to the database. Use your MySQL database running on localhost and the sakila database. You will need to hardcode your username and password.
- Create a String *query* and store in it an SQL statement that retrieves the column “name” from all rows of the Category table.
- Declare a PreparedStatement object called *statement* and run the `prepareStatement` method on your connection object. Supply *query* as an argument. Store the result of `prepareStatement` in *statement*.
- Execute the query on the statement object. Store the result in a ResultSet object.
- Declare and instantiate an ArrayList of Strings called “list”.
- Using a While loop, loop through the resultSet while there are more rows
 - Add the names column from each row to the arraylist using its add method.
Remember that you are accessing the resultSet using index numbers of columns, not names.
- Instantiate a new String array. Use the size of the ArrayList as the size of the array. The size method gives you the size. Store the reference in your *categories* array, which you declared as an instance variable.
- Call the `toArray` method of your ArrayList, list. Supply it with your *categories* array. This method will go through all the elements in the ArrayList and add them as elements in the array i.e. “*ArrayList to Array*”
- Using your existing query variable, create a new SQL statement that selects the 1) title, 2) description, and 3) release year of all films, from the film table in the sakila database for the category that has been selected. Use “ = ?” for category name. The question mark will later be filled in using a method. You will need to join 3 tables in you SQL statement to go from having the category name to being able to come up with the film name, description, and title.
- Call the `prepareStatement` method of your connection object again and pass it your query variable. Store the result in *categoryStatement*, which is the instance variable. *Remember: this is NOT the local variable statement.*
- All of these objects and methods throw exceptions – enclose everything you wrote in this method in a try block and catch a generic Exception.
- Call your `loadCategories` method from the constructor of the class.

3.6 Create a method called `getFilms`. It should return an object reference of type `ResultSet`. After the name of the method, type in “throws `SQLException`”, which tells the compiler that this method is going to throw that Exception. In reality, we will merely not catch exceptions. If an exception should occur, it will just keep on being thrown, in the hope that whoever called this method would catch it. Writing “throws” prevents you from needing to do the try-catch.

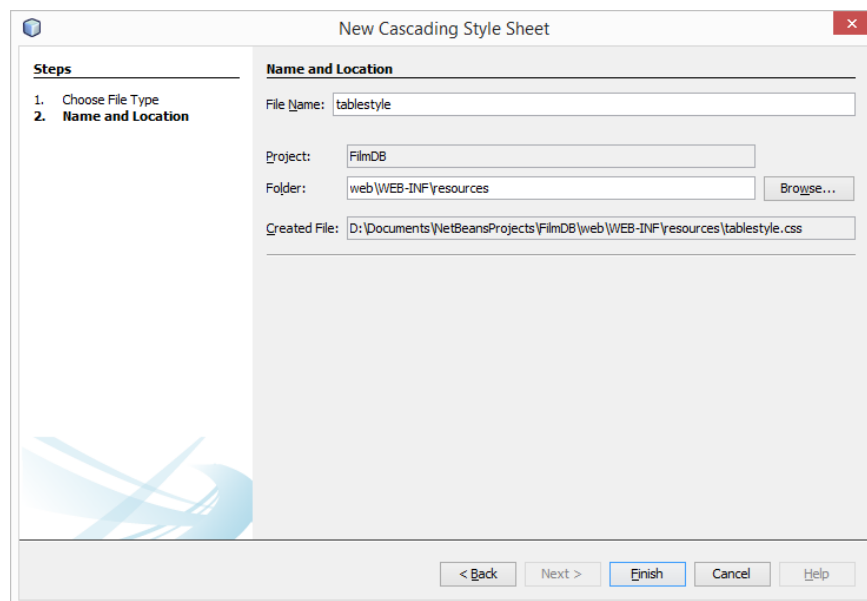
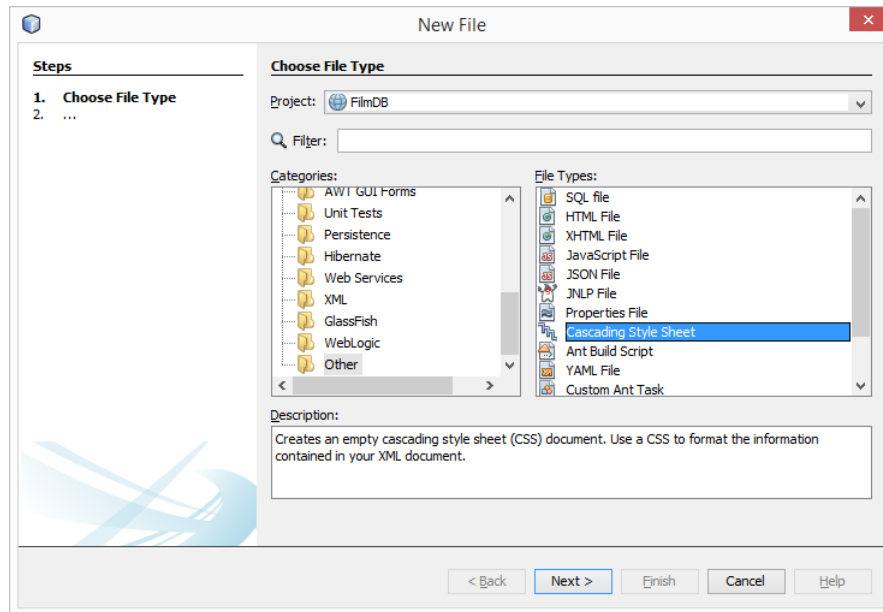
Method logic:

- if choice is null
 - if categories array's length is zero
 - return null. *If the user hasn't submitted a choice (i.e. a category name) and the categories array is empty, then there is nothing to process. Return null!*
 - Else if the categories array's length is not zero
 - *Let's just presume that the user selected the first category.*
 - Call the `setString` method of `categoryStatement`. Pass it 1 as the index number, and the first element of the categories array as the second argument. *I refers to the first question mark in the query. So, this will form the SQL query to retrieve all films for the first category.*
- Else if choice is not null
 - *The user sent a category name!*
 - Call the `setString` method of `categoryStatement`. Pass it 1 as the index number, and the variable choice as the second argument. *This will form the SQL query to retrieve films for the category specified by the user.*
- Call the `executeQuery` method of `categoryStatement` and return its return value as the return value of the method. *This line is not part of any if statement.*

4. Create a Cascading Style Sheet.

- Create a new resources folder under “Web Pages”.

Right-click and add a new file. Go into Other and in the dialog box that appear, again go into Other. Select Cascading Style Sheet. Call your CSS file tablestyle.css



- Paste the following CSS instructions into the CSS file. You can modify the CSS instructions as you see fit and to your taste. *You probably work with CSS in the CIS 425 web development class.*

```
.tableHeader {  
  
    font-family:Arial, Times;  
    border-collapse:collapse;  
    font-size:1.1em;  
    text-align:left;  
    padding-top:5px;  
    padding-bottom:5px;  
    background-color:#A7C942;  
    color: white;  
    border:1px solid #98bf21;  
}  
  
.oddTableRow {  
    border:1px solid #98bf21;  
}  
  
.evenTableRow{  
    background-color: #eeeeee;  
    font-size:1em;  
    padding: 3px 7x 2px 7px;  
    color: #000000;  
    background-color: #EAF2D3;  
}  
  
.table {  
    border:1px solid green;  
}
```

5. Edit the index.html facetlet. We are trying to create the following “view” or HTML page.

Title	Description	Release Year
AGENT TRUMAN	Intrepid Panorama of a Robot And a Boy who must Escape a Sumo Wrestler in Ancient China	2006-01-01
ALAMO VIDEOT	Boring Epistle of a Butler And a Cat who must Fight a Pastry Chef in A MySQL Convention	2006-01-01
ALIEN CENTER	Brilliant Drama of a Cat And a Mad Scientist who must Battle a Feminist in A MySQL Convention	2006-01-01
ALLEY EVOLUT	Fast-Paced Drama of a Robot And a Composer who must Battle a Astronaut in New Orleans	2006-01-01
BABY HALL	Boring Character Study of a A Shark And a Girl who must Outrace a Feminist in An Abandoned Mine Shaft	2006-01-01
BALLROOM	Thrilling Documentary of a Composer And a Monkey who must Find a Feminist in California	2006-01-01
MOCKINGBIRD	Beautiful Drama of a Dentist And a Composer who must Battle a Sumo Wrestler in The First Manned Space Station	2006-01-01
BROOKLYN DES	A Awe-Inspiring Character Study of a Secret Agent And a Boat who must Find a Squirrel in The First Manned Space Station	2006-01-01
BUGSY SONG	A Thrilling Drama of a Frisbee And a Lumberjack who must Sink a Man in Nigeria	2006-01-01
CALENDAR GUNFIGHT	A Boring Reflection of a Lumberjack And a Feminist who must Discover a Woman in Nigeria	2006-01-01
CATCH AMISTAD	A Unbelievable Story of a Mad Scientist And a Technical Writer who must Discover a Composer in Ancient China	2006-01-01
CHOCOLATE DUCK	A Awe-Inspiring Tale of a Forensic Psychologist And a Woman who must Challenge a Database Administrator in Ancient	2006-01-01
COMMAND DARLING		

Note: the following instructions are omitting the “h:” in the names of tags. All tags names are prefixed with “h:” as part of the JSF GUI elements specification. Accordingly, if instructions call for a “column” tag, you are putting in “h:column” as the tag name.

Refer to the previous work from class for exact syntax.

Remember that all tags must be cleanly closed!

```
<h:outputStylesheet name="tablestyle.css"/>
```

Reference:

- outputStylesheet is the name of the tag. Its name is prefixed with “h:”
- name is an attribute.
- “tablestyle.css” is the value of the name attribute.
- The tag is cleanly closed by the slash character.

5.1 Within the Head tag, Change the title to display films. Also, create a tag of type outputStyleSheet. Assign its name attribute the value “tablestyle.css” which is the name of the CSS file you created.

5.2 Within the Body tag, create a form tag. The rest of the code that follows lies within the form tag.

- Create an outputLabel tag and assign it the value “Choose a category: “.
- Create a tag for a ComboBox. *Refer to code from last class!* A ComboBox is represented by the selectOneMenu element. As its value, open a JSF expression, e.g. “#{java code}”. Within the JSF expression, use the instance variable name of the Bean, categoryBean, and then access a property. In this case, we want the value of the combo box to be stored in choice. So, we will use choice. Consequently, our full expression will look like “#{categoryBean.choice}”.
 - Within the ComboBox tag, create a selectItems tag. This tag comes from a different library, it will be prefixed with “f:” rather than “h:”. As its value, assign it a JSF expression which retrieves *categories* (which is your array) property.
- Create a commandButton tag. Give it a Style attribute with the value “margin-left: 20px” and a value of “Display Films”.
- Create two line break tags.
- Create a dataTable tag. Add the following attributes for the tag.
 - value.
 - Assign it the value of a JSF expression e.g. “#{java code}”. Within the JSF expression, the set the “films” property from the categoryJSFBean (this is the name of the Bean you chose when creating the bean). You also created a getFilms() method. This uses the “property” for that getter metter.
 - var
 - Assign to it the value, the string “film”. This associates each row of the table to a film.
 - rowClasses
 - Assign to it the value, the string “oddTableRow, evenTableRow” as a value. *This applies alternating Styles from the CSS file to alternating odd and even table rows.*
 - headerClass
 - Assign to it the value, the string “tableHeader”. Again, this takes the tableHeader style from the CSS file for the table header.
 - styleClass
 - Assign to it the value, the string “table”, which sets the value of all other elements within the table to the table style from the CSS file.

This is the end of the dataTable tag. Do not put in a “/” to close the tag. Only put in a “>”. Within the dataTable tag, we will create multiple column tags.

5.3 Within the dataTable tag, create 3 column tags – one each for film title, description and release year.

Within each column tag, create a facet tag. Facet also comes from the alternate library and thus is prefixed with “f:”. Assign the value “header” to its name attribute. Put in the name of the table column. Then after the facet tag closes, insert a JSF expression that accesses the property name from film. An example of the title column is shown below.

```
<h:column>
    <f:facet name="header">Title</f:facet>
    #{film.title}
</h:column>
```

Create the description and release year columns just like the Title column shown above.

Remember to close the dataTable tag!

Remember to close the form tag!

Remember that all of this should have gone inside the body tag which should also be closed!

6. You are ready to run the application!