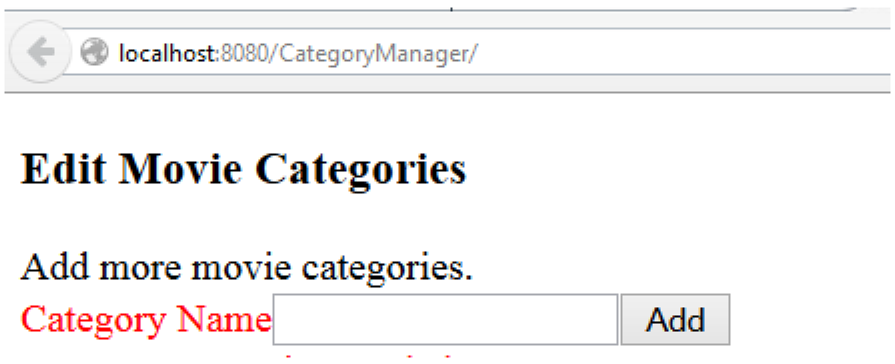
W. P. CAREY
SCHOOL *of* BUSINESS
ARIZONA STATE UNIVERSITY

CIS 494 – Business Systems Development with Java – Spring 2015

In-class Exercise 12: Exchanging data across JSF pages using Sessions

Due on Blackboard: Wednesday, April 29 by 11:59 PM

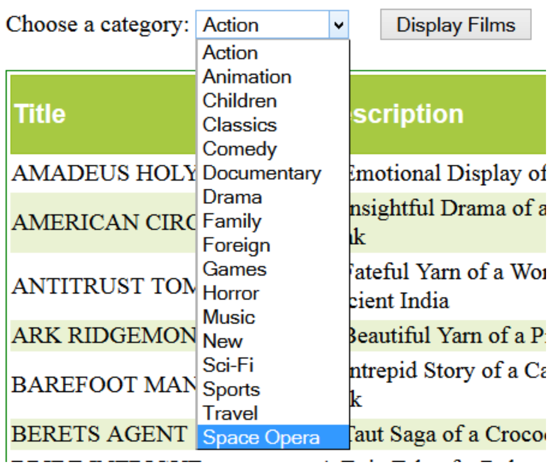


localhost:8080/CategoryManager/

Edit Movie Categories

Add more movie categories.

Category Name [ ] Add



Choose a category: Action Display Films

Title	scription
AMADEUS HOLY	Emotional Display of
AMERICAN CIRC	nsightful Drama of a k
ANTITRUST TOM	ateful Yarn of a Wo ient India
ARK RIDGEMON	Beautiful Yarn of a P
BAREFOOT MAN	ntrepid Story of a Ca k
BERETS AGENT	aut Saga of a Croco

For this project, you will create a very basic interface to update the film Categories in the Sakila database that is supplied with MySQL. The last inclass exercise showed a list of categories and then movies in each category. When this inclass starts working, the categories added from this project should reflect accurately in your previous project.

One of the primary purposes of this exercise is to see how JSF manages movement across different pages and how developers can exchange data across pages but still ensure that each page is referring only to a user's own data. This is established by using sessions.

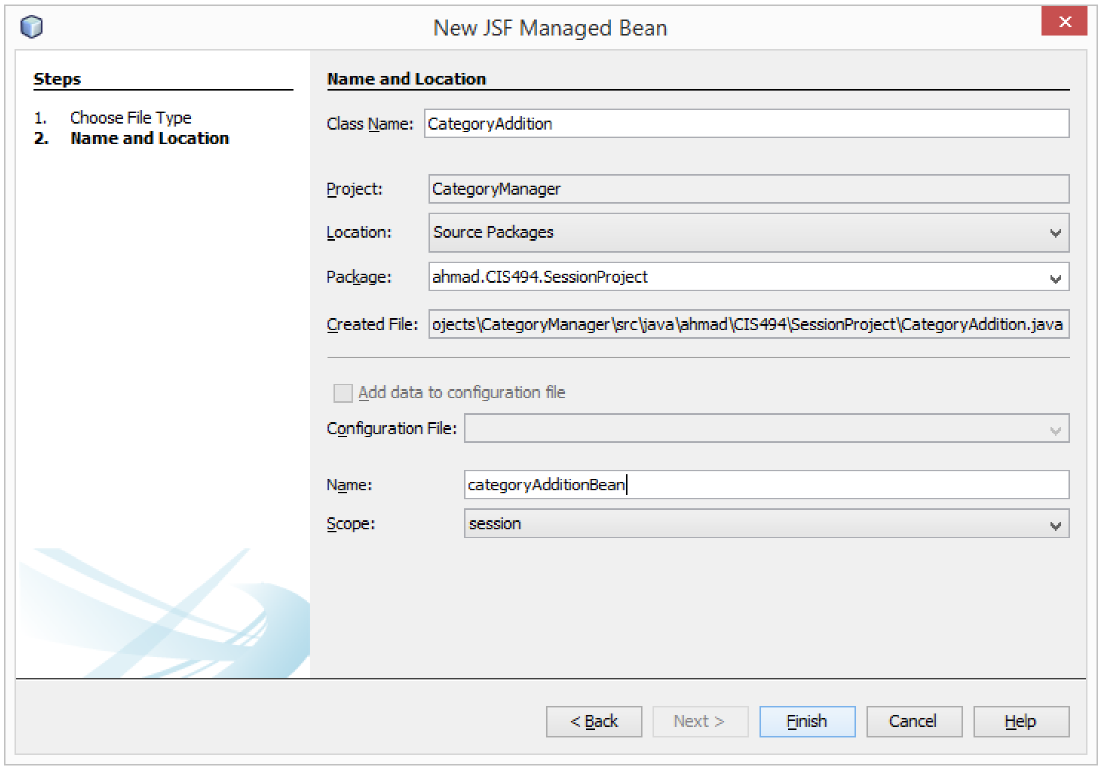
Before You Begin

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 "CategoryAddition".
2. 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. 3.1 Create a Managed Bean in NetBeans called categoryAdditionBean. Choose **session** scope. Change the Name at the bottom (this is the name of the instance, not Class name) to "categoryAdditionBean".



Class Name: CategoryAddition

Project: CategoryManager

Location: Source Packages

Package: ahmad.CIS494.SessionProject

Created File: ojects\CategoryManager\src\java\ahmad\CIS494\SessionProject\CategoryAddition.java

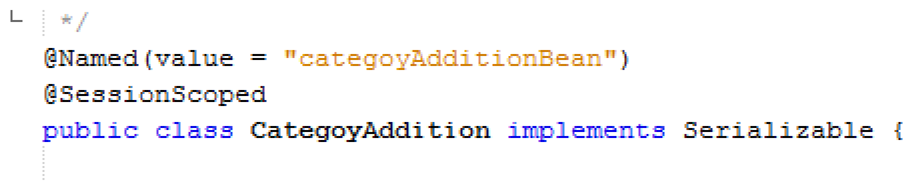
Configuration File:

Name: categoryAdditionBean

Scope: session

4. We are going to use JDBC – add java.sql.* to the list of import statements.
5. Verify that your auto-generated code contains @Name and @SessionScoped attributes and that your instance name is set to categoryAdditionBean.

   Note that if you create a session-scoped bean, the Class must implement the Serializable interface.



```
 */
@Named(value = "categoyAdditionBean")
@SessionScoped
public class CategoyAddition implements Serializable {
```

6. Create the following instance variables:

   - statement, of type PreparedStatement
   - updateStatus, of type String   // this will store a status update
   - categoryName, of type String // this will be list be the user input
   -

- Generate a getter/setter pair for updateStatus and categoryName. *Use the Refactor menu!*

7.1 Create a method called initializeJDBC. 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 – it should be nearly identical in setup.

- 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 INSERTS a record with the specified **name** into **category** table. Use a question mark instead of the actual value so that the field can be filled in using a PreparedStatement object.

INSERT INTO category(name) VALUES (?)

- Use your connection object to call the prepareStatement method. Give it your query variable as an argument and store the return value of the method into your *statement* variable.

  Make sure you use try/catch

7.2 Call initializeJDBC from the constructor of the class.

8. Create the following methods:

a) isRequiredFieldFilled() with return type Boolean.

    If categoryName is null return false, else return true.

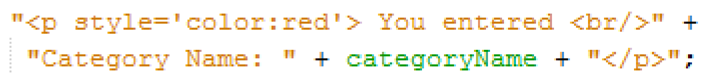
b) processSubmit() with return type String. This method will return the name of a JSF page based on whether the required field is filled in or not. If the required field is absent, nothing happens. Otherwise, the user will be directed to a new page.

    Call isRequiredFieldFilled – if it returns true,
        return "ConfirmCategory"
    else, return an empty string. *Do not change the content of the string. This string is the name of a webpage, so it needs to be accurate.*

c) getRequiredFields() with return type String. *This method will return an error message if the input field is not filled out.*
    Call isRequiredFieldFilled – if it returns true, return an empty string
    Else, return an error message such as "Category name is required."

d) getInput() with return type String

This method is for printing out the input to the html page. Therefore it requires only the categoryName. However, because we want this to appear on a web page, we should insert the categgoryName within HTML tags. Return the following string or something close to it.



```
"<p style='color:red'> You entered <br/>" +
 "Category Name: " + categoryName + "</p>";
```

e) storeCategory with return type String

This method is for storing the categoryName into the database by running an INSERT query. It will return the name of a JSF page based on whether it was able to execute an insert statement or not. If there is an error, execution goes into a catch. Otherwise, the user will be directed to a new page.

Within a try block,

Call the SetString method of your preparedStatement object and provide it with an index number, 1, and the categoryName. This will insert the category name properly into the preparedStatement object.

Run the query by calling the executeUpdate method. *This method is different because it is running an update command rather than a select query.*

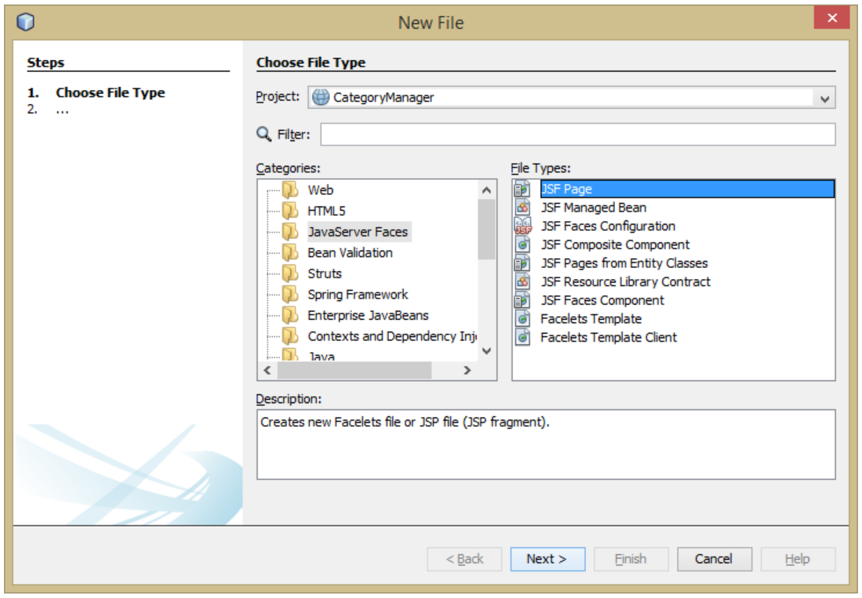
Set the value of the updateStatus field to say that categoryName has been updated. Use string concatenation to create your string.

Remember to have a catch block.

After the try-catch, return "CategoryStored". *Do not change the content of the string. This string is the name of a webpage, so it needs to be accurate.*

9. *Creating JSF Web Pages.*

You will have started out with an index.xhtml page. Add 2 more JSF pages. Go into "Other" after the new menu if you do not see it listed within the context menu when you right-click. Name your two pages "**ConfirmCategory**" and "**CategoryStored**". Make sure you select Facelets rather than JSP. (Facelet is the newer technology).

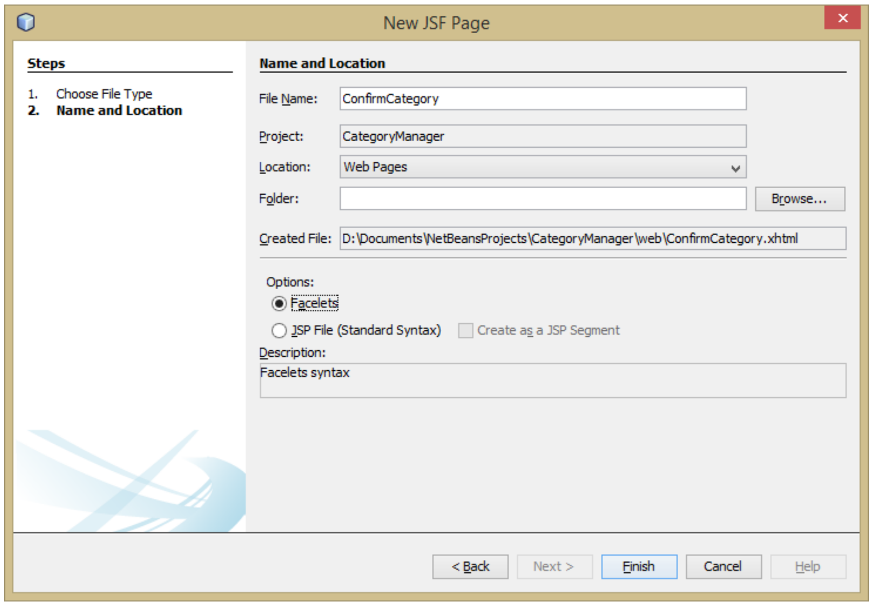


Project: CategoryManager

Filter:

Description:

Creates new Facelets file or JSP file (JSP fragment).



File Name: ConfirmCategory

Project: CategoryManager

Location: Web Pages

Folder:

Created File: D:\Documents\NetBeansProjects\CategoryManager\web\ConfirmCategory.xhtml

[✓ Facelets]

JSP File (Standard Syntax)

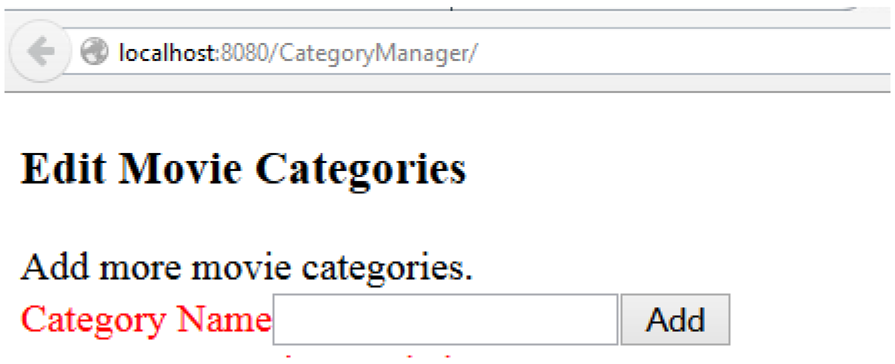
Create as a JSP Segment

Description:

Facelets syntax

a) **index.xhtml Page**

This page is the initial page that users see, which allows them to enter text to update the query. It should look like this:



Edit Movie Categories

Add more movie categories.

Category Name

Create a form tag.

Within the form tag,

- Create an h3 tag and write "Edit Movie Categories" within it.
- Add a statement, which tells users to add more movie categories. This is within the body tag, but not inside any other tag.

Create the following tags. Whenever you are assigning a value to the "value" attribute and need to use a JSF expression, use the curly braces and # sign. Use the name of the Managed Bean, categoryAddtionBean and then access property/method name.

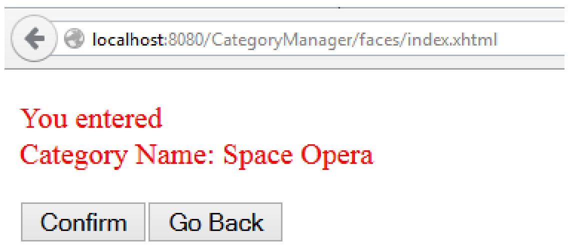
- outputLabel tag
  - Attribute name: value		value: Category Name
  - Attribute name: style		value: color:red
- inputText tag
  - Attribute name: id		value: categoryNameText
  - Attribute name: value		value: categoryName property
- commandButton tag
  - Attribute name: value		value: Add
  - Attribute name: action		value: processSubmit method call

  Note: the action property is the "event handler" for the button. Its value property determines what is going to happen when the user presses a button. In this case, it's a method call.

- outputText tag
  - Attribute name: escape		value: false
  - Attribute name: style		value: color:red
  - Attribute name: value		value: requiredFields property

b) ConfirmCategory.xhtml Page

This page is the second page that users see, which presents them with a choice of whether to cancel the update or whether to confirm and proceed with the insert category update.


localhost:8080/CategoryManager/faces/index.xhtml

You entered
Category Name: Space Opera

Page Title: Confirm Category Addition

Create a form tag. Within the form tag, create the following tags. Again, use the specified attributes and properties for each tag.

- outputText tag
  - Attribute name: escape value: false
  - Attribute name: style value: color:red
  - Attribute name: value value: input property

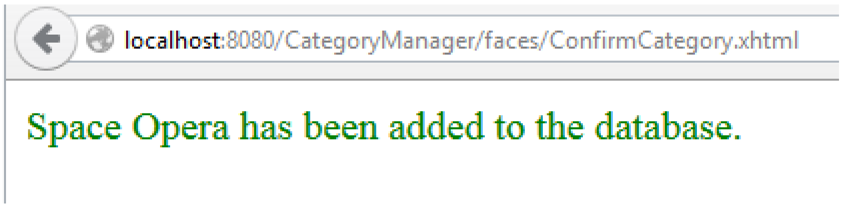
  This label retrieves the name of the category and displays it to the user.
- commandButton tag
  - Attribute name: value value: Confirm
  - Attribute name: action value: storeCategory method call

  This button calls storeCategory() if the user confirms that they want to proceed with the update.
- commandButton tag
  - Attribute name: value value: Go Back
  - Attribute name: action value: index

This button redirects the user back to index.xhtml if they hit "Go Back"

c) CategoryStored.xhtml Page

This page is the final page that users see, which presents them with a status message informing them that the data has been updated to the database.



localhost:8080/CategoryManager/faces/ConfirmCategory.xhtml

Space Opera has been added to the database.

Page Title: Category Stored?

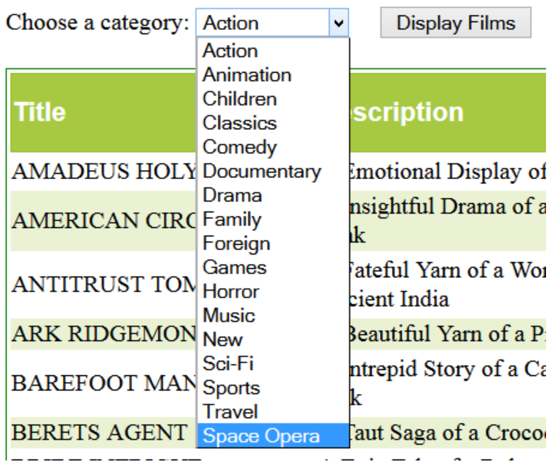
Create a form tag. Within the form tag, create the following tags. Again, use the specified attributes and properties for each tag.

- outputText tag
  - Attribute name: escape value: false
  - Attribute name: style value: color:green
  - Attribute name: value value: updateStatus property

    This label retrieves the updateStatus property and displays it to the user. This was the HTML text that was prepared with the "<P>" tags

10. Run the project!

When you are able to run the project and are able to go through the three pages and update to the database, the next time you run the FilmDB project from the previous in-class, you should see an updated list of categories in that application.



Choose a category: Action Display Films

Title	scription
AMADEUS HOLY	Emotional Display of
AMERICAN CIRC	nsightful Drama of a k
ANTITRUST TOM	ateful Yarn of a Wor cient India
ARK RIDGEMON	Beautiful Yarn of a P
BAREFOOT MAN	ntrepid Story of a Ca k
BERETS AGENT	aut Saga of a Croco