Java EE, Beyond the Basics Hands-On-Lab

Exercise Manual

David Delabassee
Oracle
@delabassee

David R. Heffelfinger
Ensode Technology, LLC
dheffelfinger@ensode.com
http://www.ensode.com
@ensode

Bob Larsen
Metaformers, Inc.
@direHerring

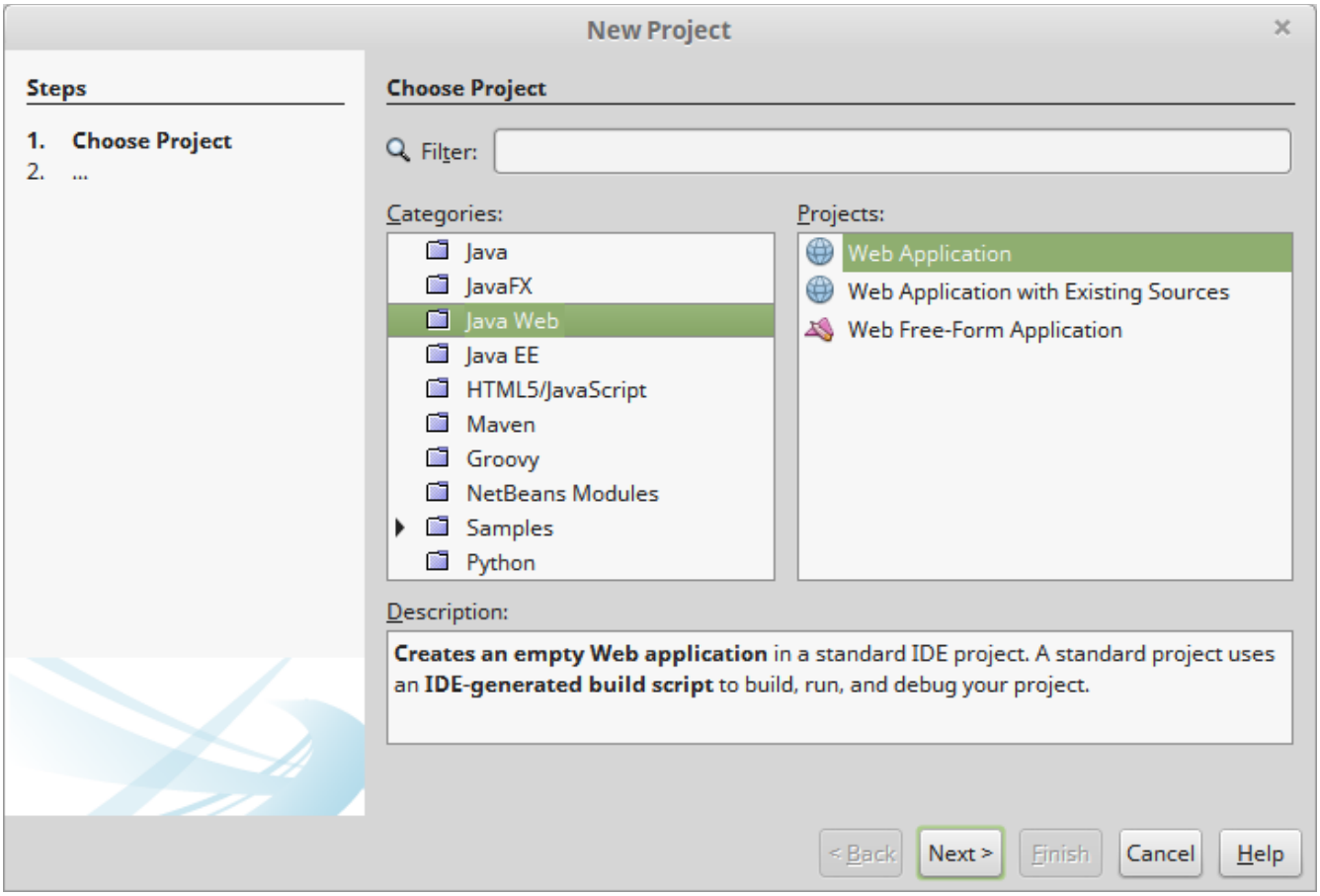
Sven Reimers
Airbus Defence and Space
@SvenNB

Exercise 1

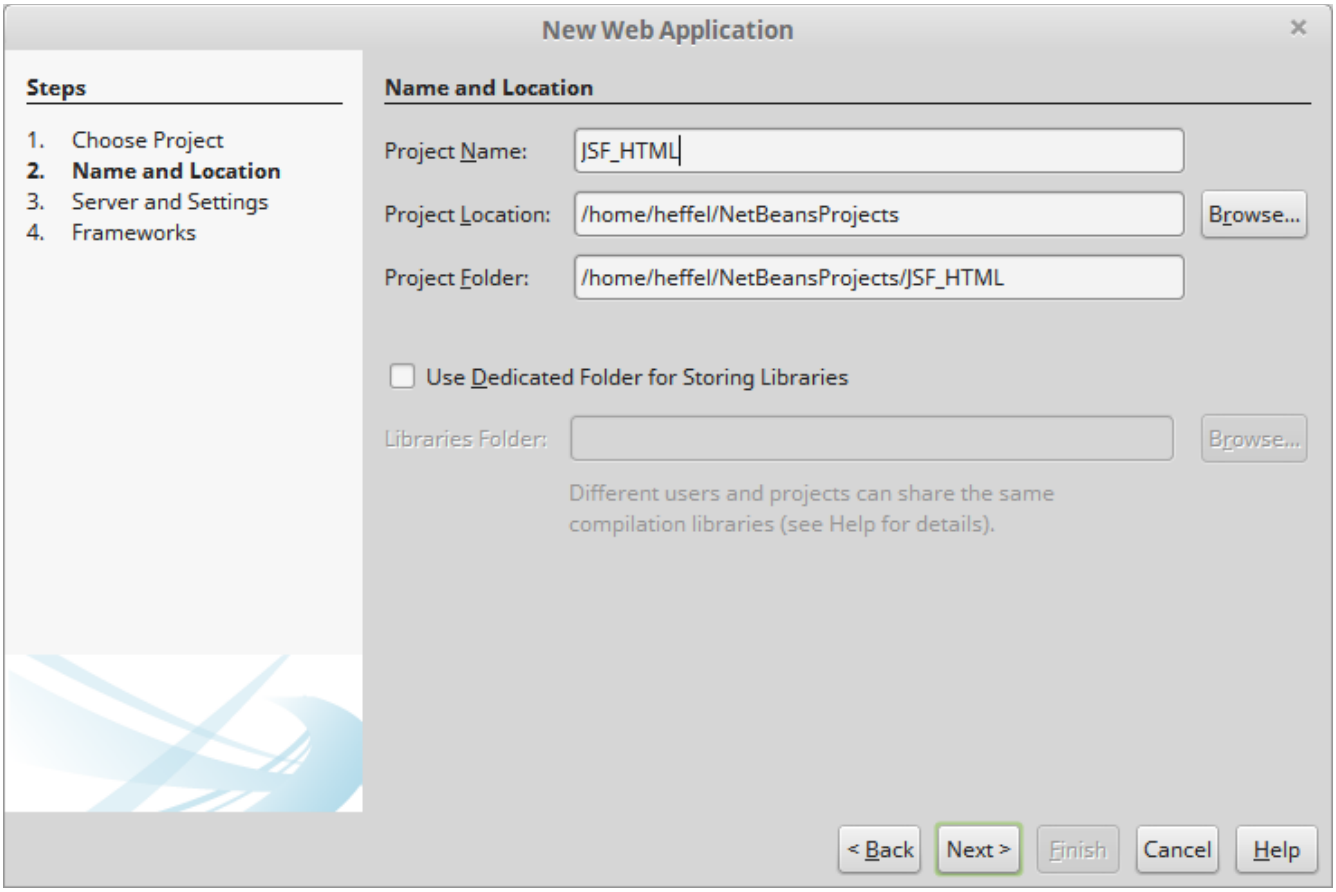
In this exercise, we will develop a JSF application using HTML markup.

Create the project

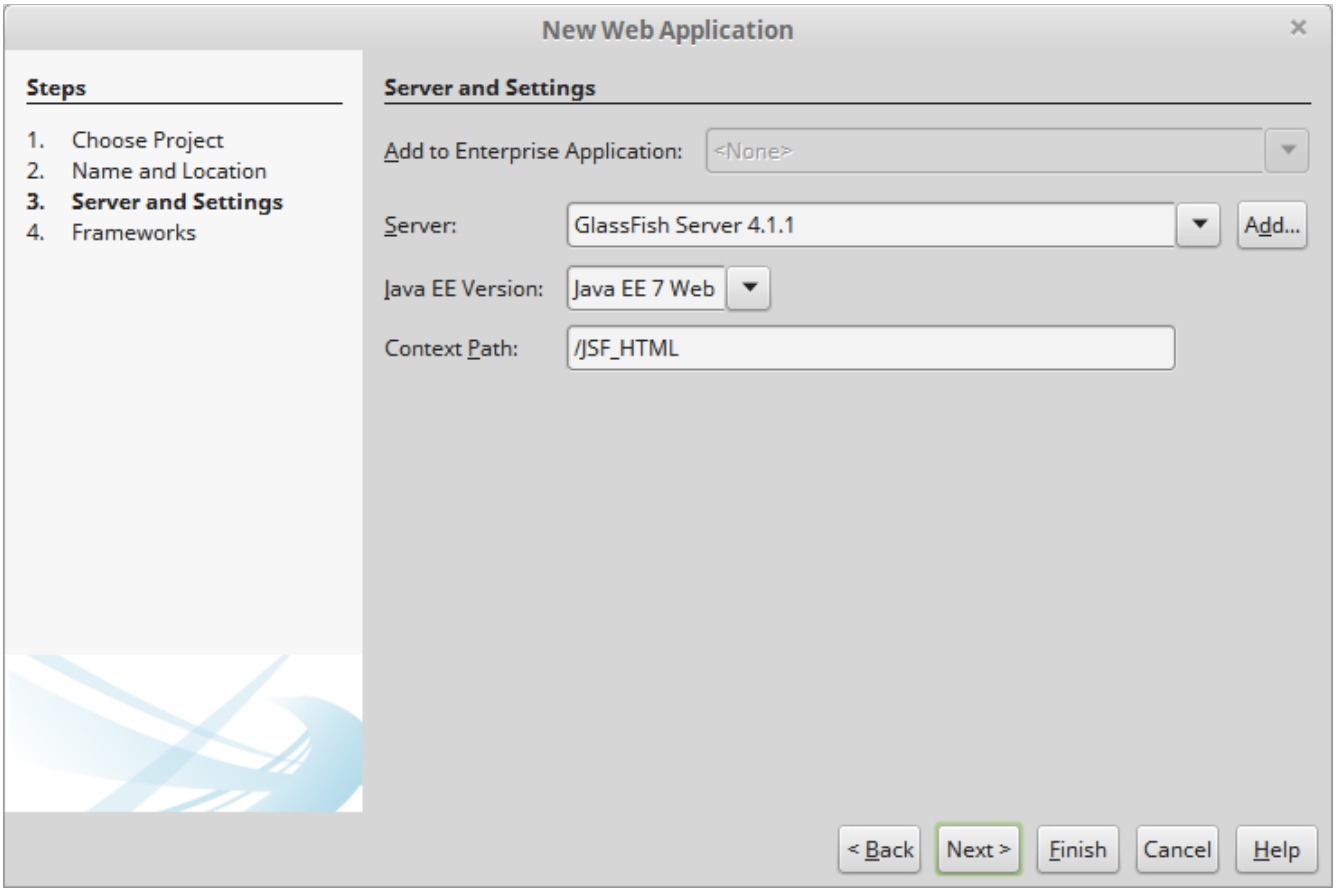
1. Create a new Web Application Project (File | New Project)
2. Under “Categories”, select “Java Web”
3. Under “Projects” select “Web Application”



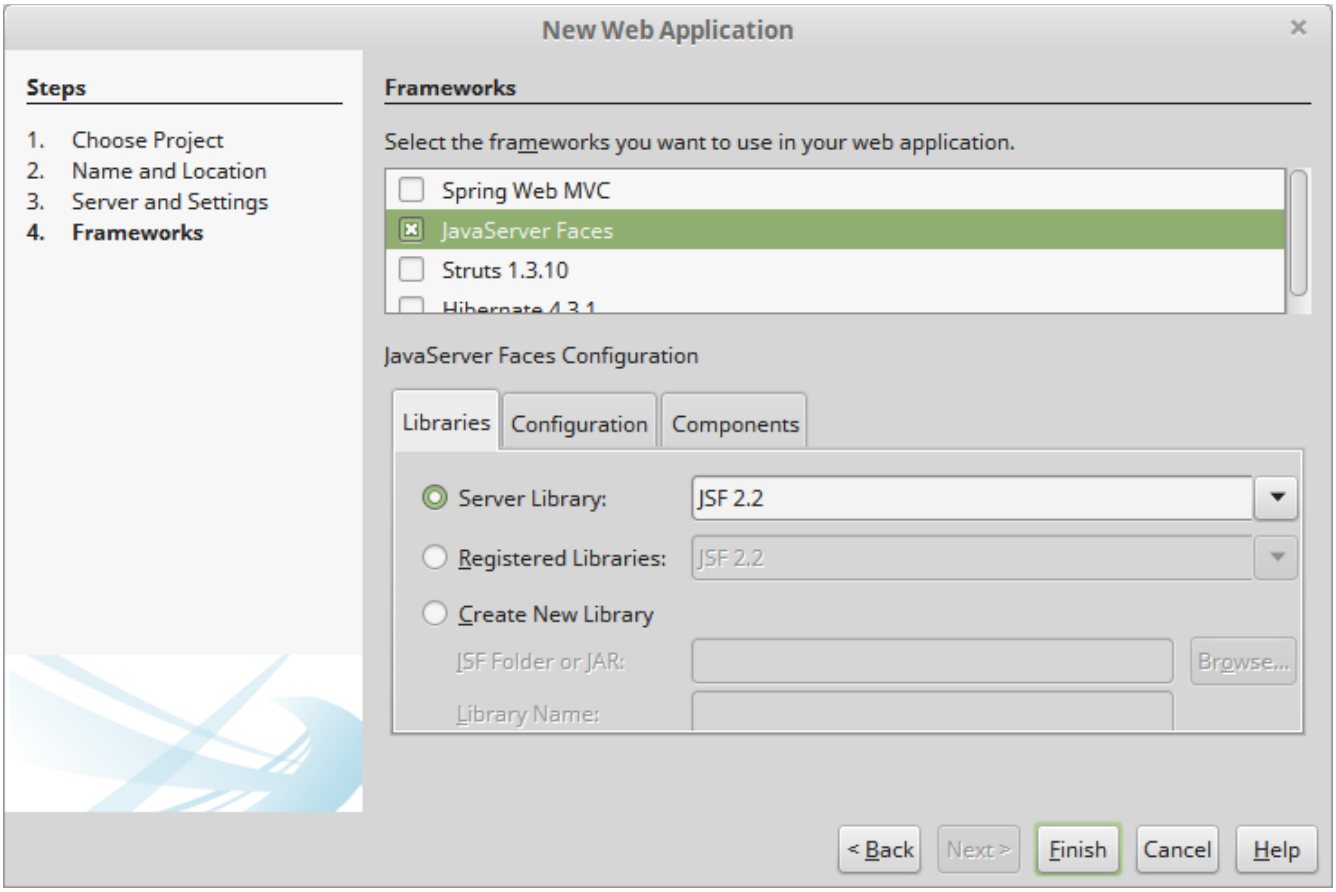
4. Click “Next”



5. Name your project “JSF_HTML” and click “Next”.



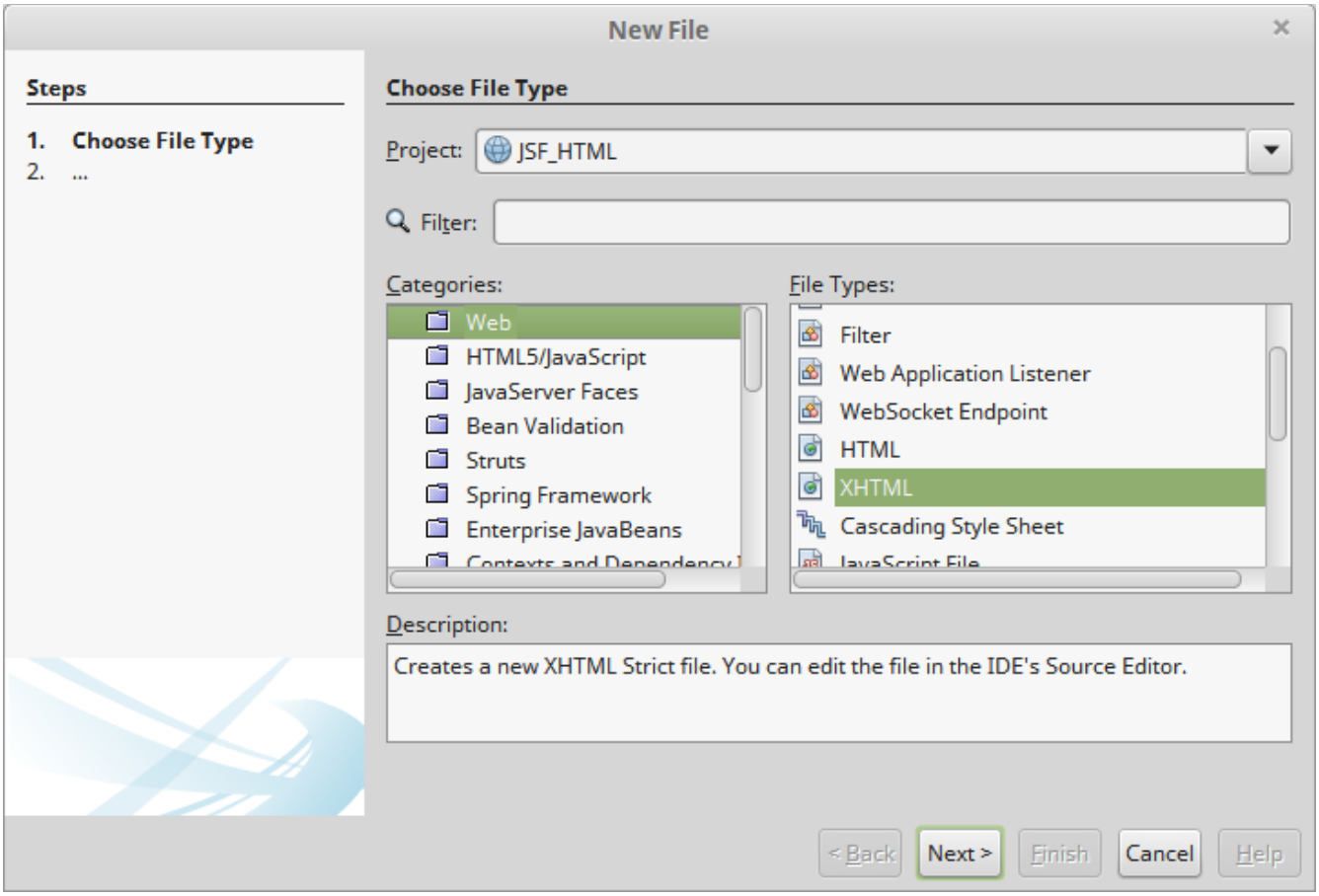
6. Accept the defaults and click “Next”.



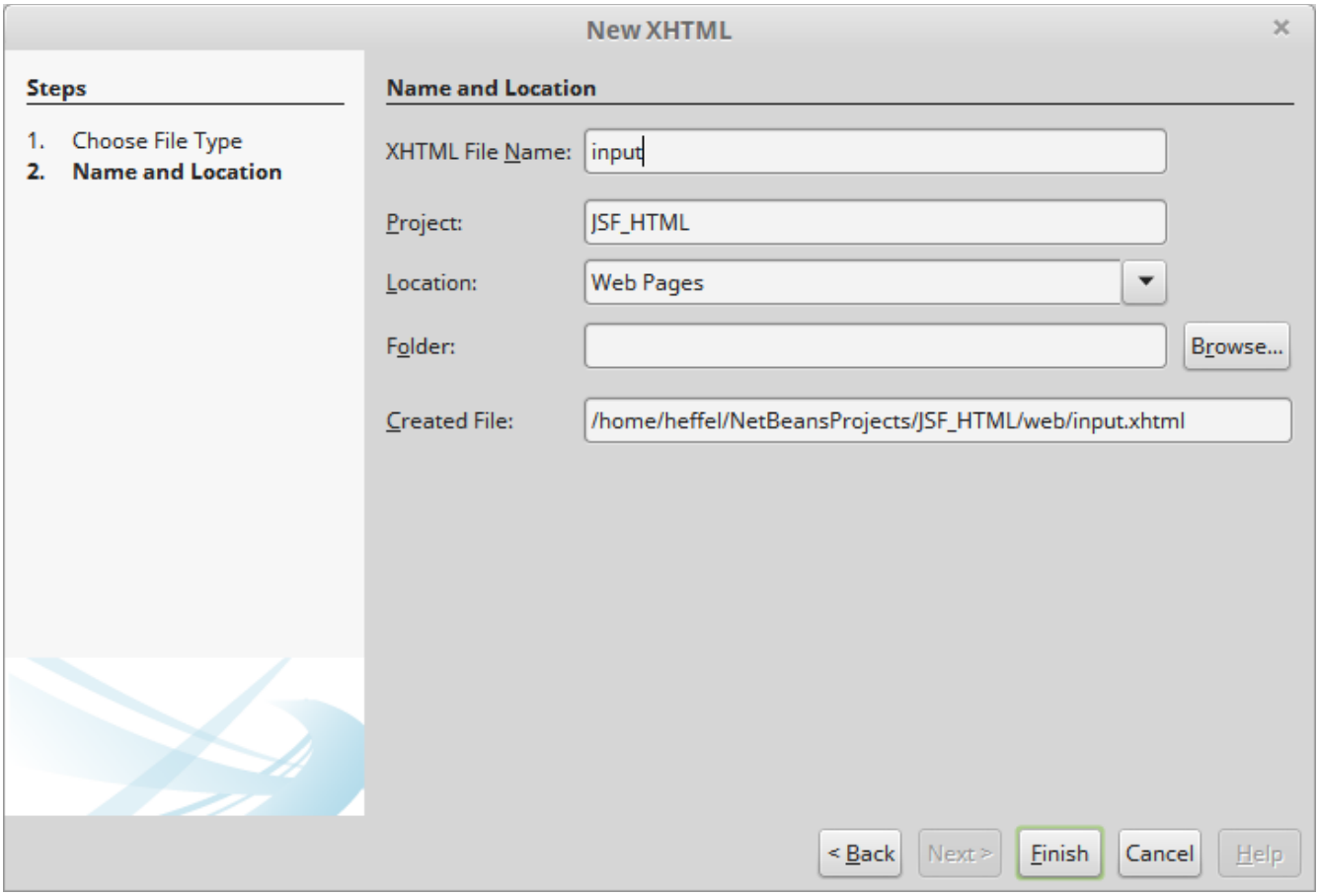
7. Select the “JavaServer Faces” framework and click “Finish”.

Develop the User Interface

1. Create a new XHTML file (File | New, “Web” category and “XHTML” file type).



2. Name your new file “input” (NetBeans automatically adds the .xhtml file extension).



3. Add the "jsf" namespace to the generated input.xhtml file. Your `<html>` tag should look like this:

```
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:jsf="http://xmlns.jcp.org/jsf">
```

4. Replace the generated `<div>` tag with a `<form>` tag, add a `jsf:id` attribute to the tag so that the JSF engine identifies it as a JSF form tag.

```
<form jsf:id="mainForm">

</form>
```

5. Insert the following HTML markup inside your `<form>` tag. Note that we are using standard HTML tags with JSF specific attributes.

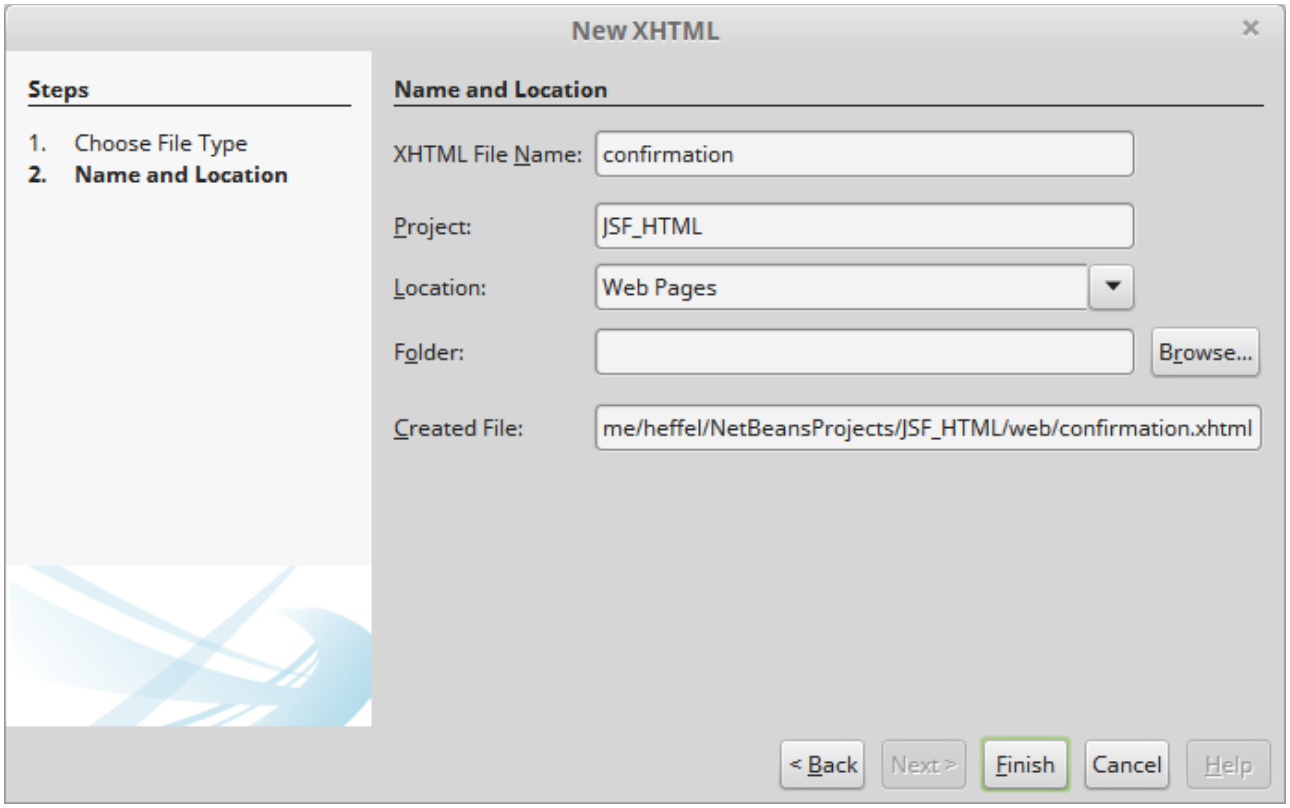
```
<label for="firstName">First Name:</label><input type="text"
id="firstName" jsf:value="#{customer.firstName}"/><br/>

<label for="lastName">Last Name:</label><input type="text"
id="lastName" jsf:value="#{customer.lastName}"/>

<input type="submit" jsf:action="#{controller.navigate}"
```

```
value="Submit"/>
```

6. Create a new XHTML file, name it “confirmation”.



7. Replace the body of the `<div>` tag in the generated file with the following markup:

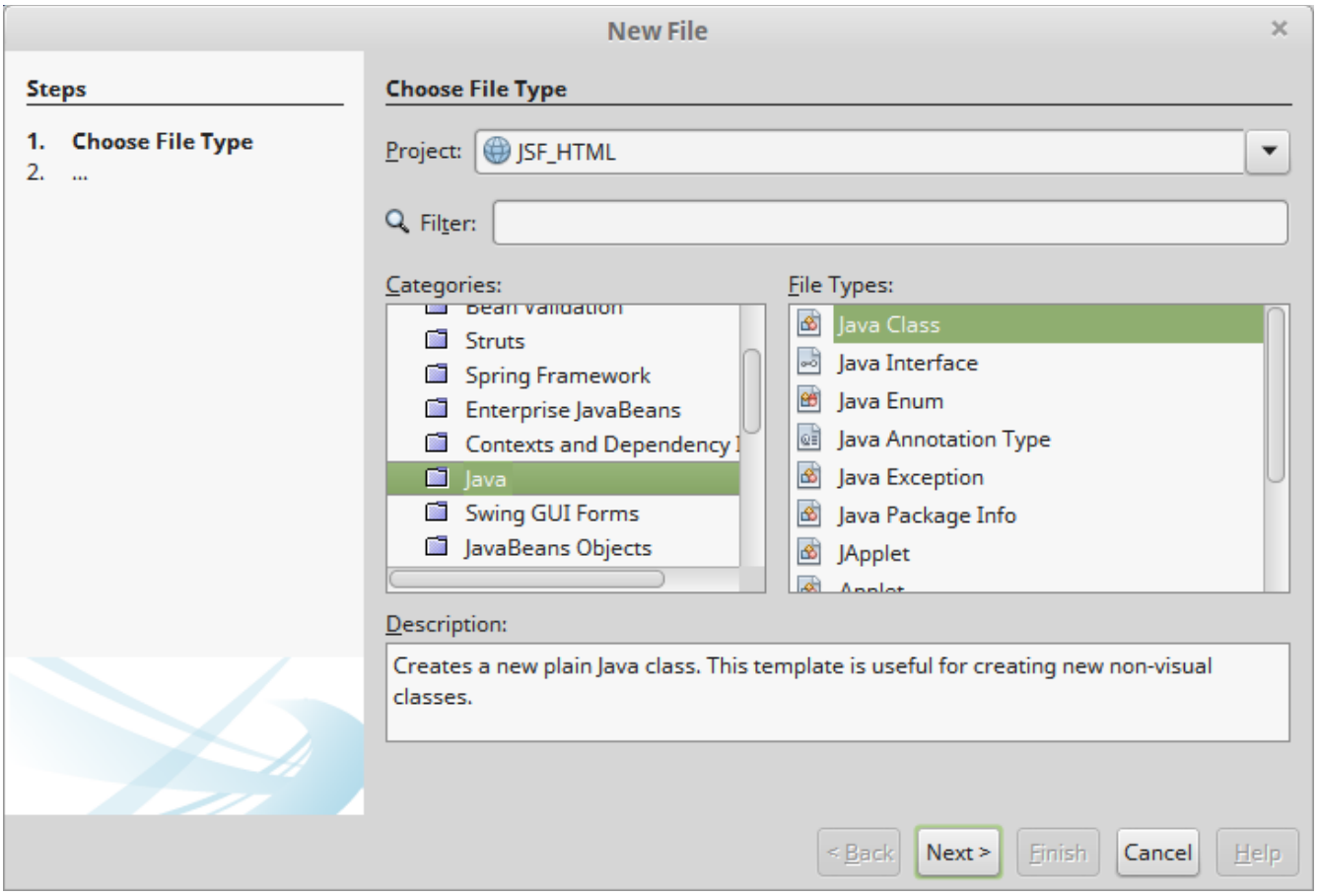
```
First Name: ${customer.firstName}<br/>
Last Name: ${customer.lastName}
```

Develop CDI Beans

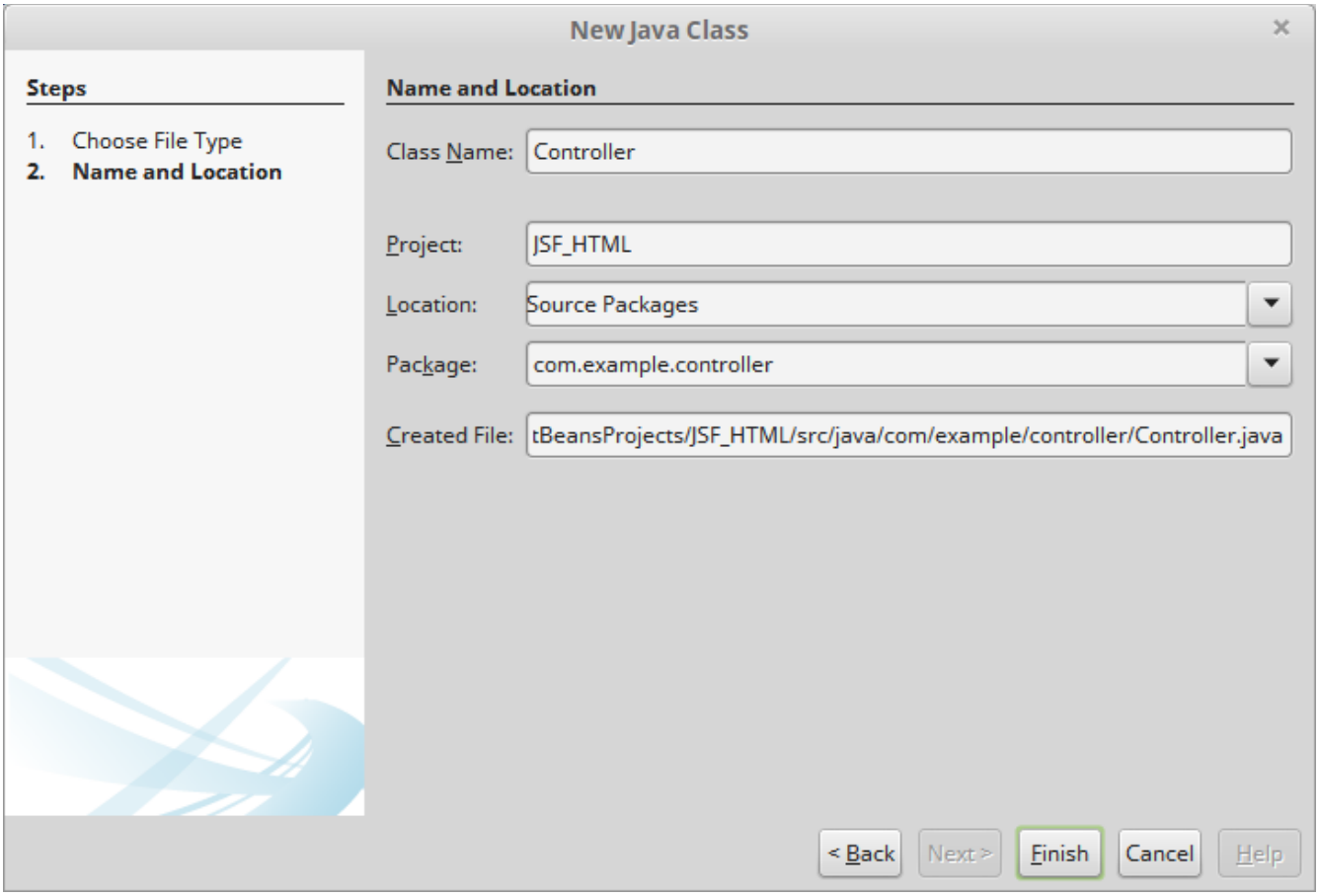
In this section, we will develop CDI beans to interact with the HTML pages we just created.

Controller class

1. Create a new Java class (File | New File.., Under “Categories”, select “Java”, under “File Types”, select “Java Class”.



Name your new class “`Controller`” and put it under the `com.example.controller` package, make your class implement the `java.io.Serializable` interface (Session scoped CDI managed beans should always implement Serializable).



2. Annotate the Controller class with the `@Named` and `@SessionScoped` annotations.
3. Add a new public method to the `Controller` class, name the method `navigate()`, have it return the string `"confirmation"`. Notice how this method is invoked in the `jsf:action` attribute of the submit button on the input page we created on the previous method.

Model Class

1. Create a new class (File | New File …, "Java" Category, "Java Class" File Type).
2. Name your new class "`Customer`" and put it in the `com.example.model` package.
3. Annotate your customer class with the `@Model` annotation (the `@Model` annotation is a built-in CDI stereotype, it is equivalent to annotating the class with `@Named` and `@RequestScoped`).
4. Add two private String variables to the `Customer` class, name them "`firstName`" and "`lastName`".
5. Add getter and setter methods for the "firstName" and "lastName" variables (hint: if using NetBeans standard keybindings, getters and setters can be generated by hitting Alt+Ins, then selecting "Getter and Setter...")

Run the project

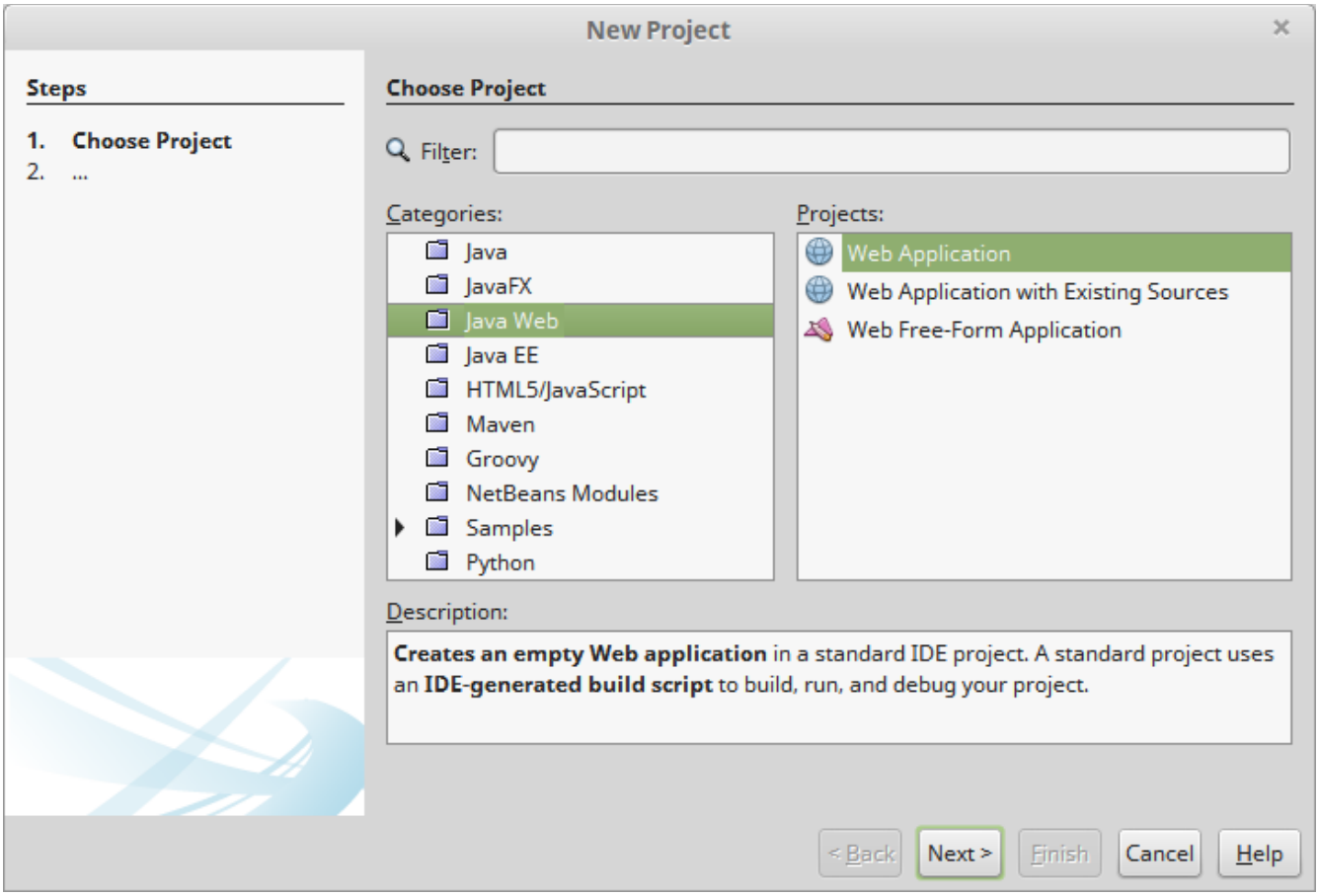
1. Right click on the project and select “Run”
2. GlassFish will automatically start, the browser will pop up and render your project's index.xhtml file automatically.
3. Replace the URL in your browser with `http://localhost:8080/JSF_HTML/faces/input.xhtml`.
4. Enter any arbitrary values into the “First Name” and “Last Name” input fields.
5. Click on the button labeled “Submit”
6. Verify that your confirmation page displays the values you entered in the input page.

Exercise 2

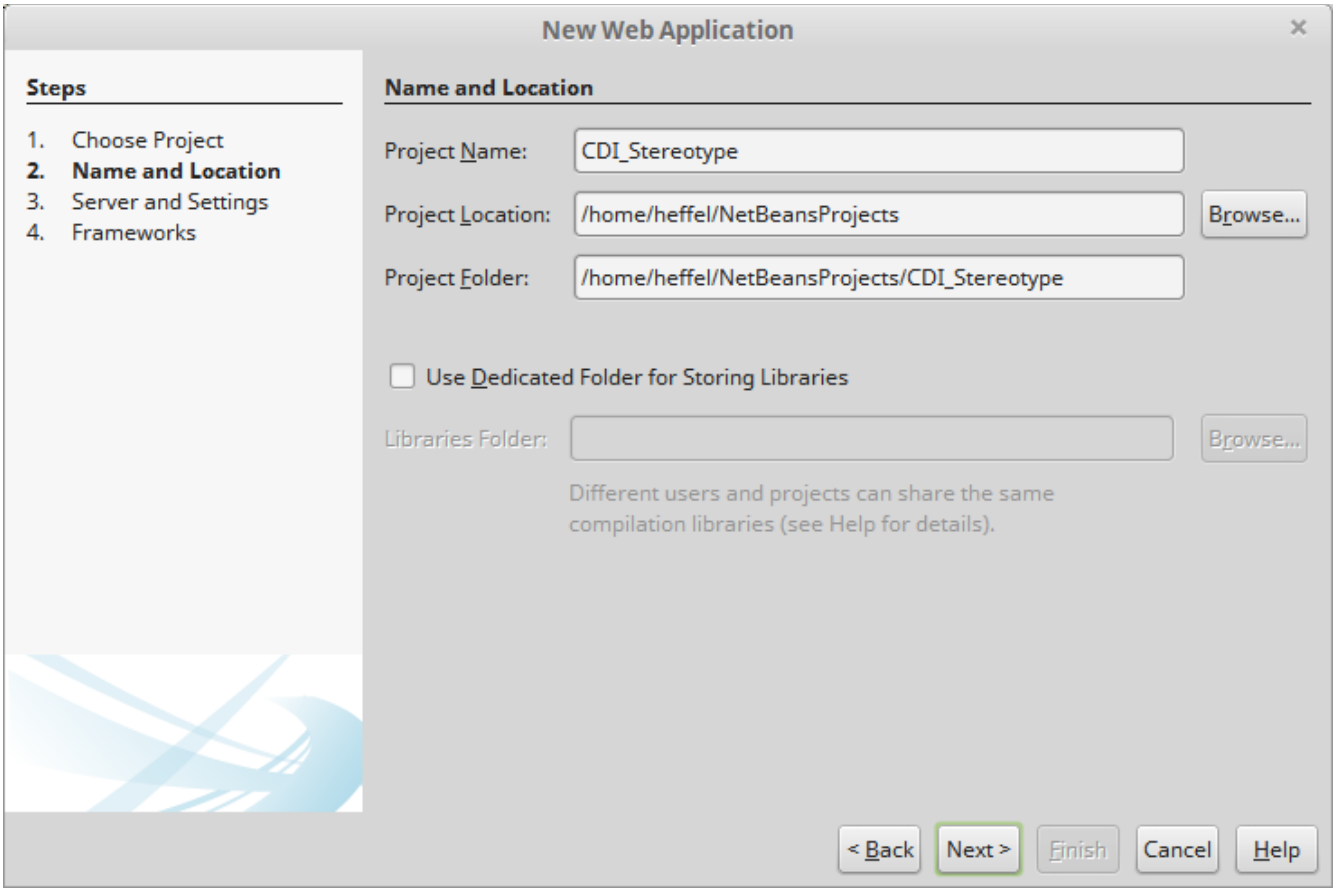
In this exercise, we will develop a CDI stereotype.

Create the project

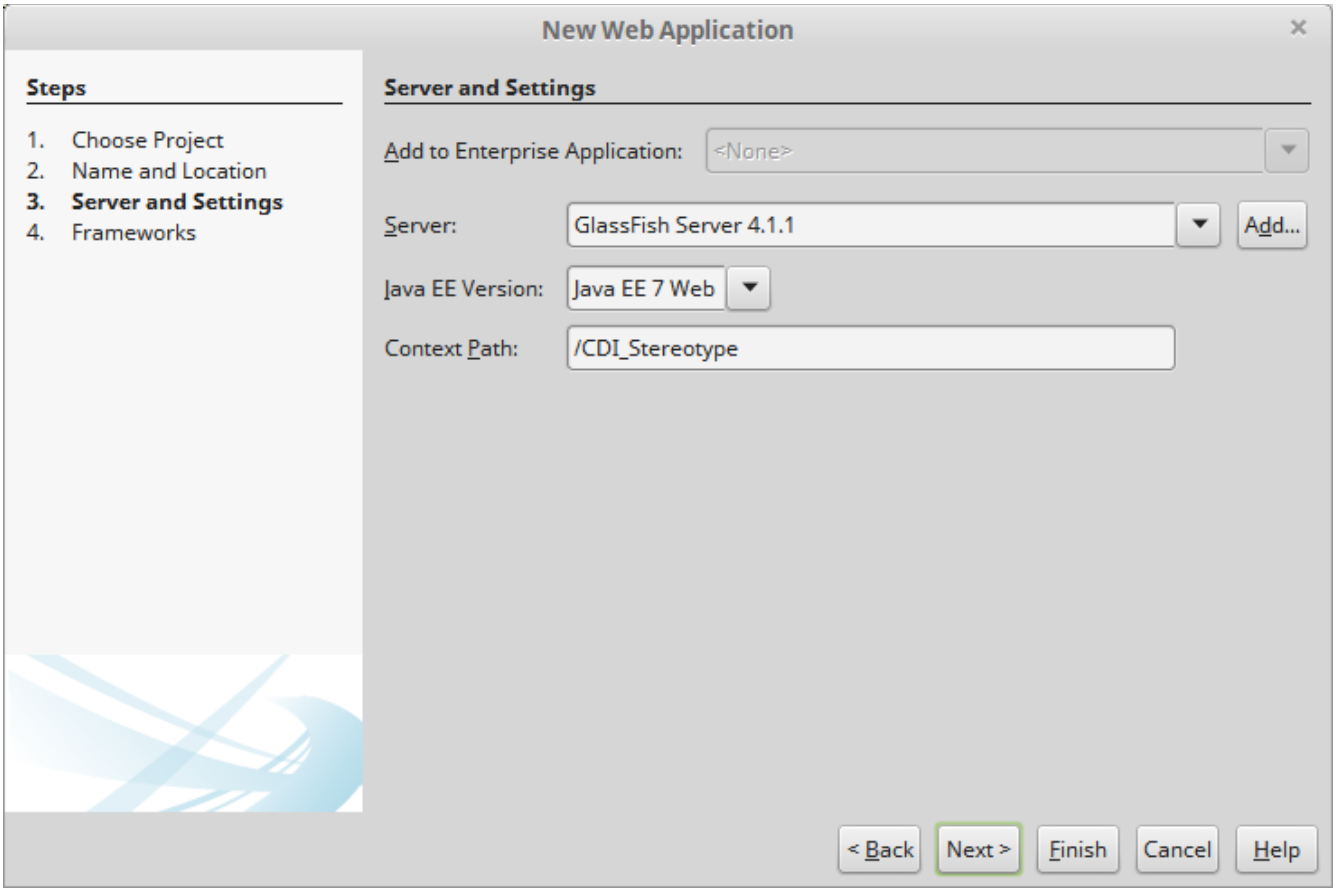
1. Create a new Web Application Project (File | New Project)
2. Under “Categories”, select “Java Web”
3. Under “Projects” select “Web Application”


New Project
Steps
1. Choose Project
2. ...
Choose Project
Filter:
Categories:
Java
JavaFX
Java Web
Java EE
HTML5/JavaScript
Maven
Groovy
NetBeans Modules
Samples
Python
Projects:
Web Application
Web Application with Existing Sources
Web Free-Form Application
Description:
Creates an empty Web application in a standard IDE project. A standard project uses an IDE-generated build script to build, run, and debug your project.
< Back
Next >
Finish
Cancel
Help

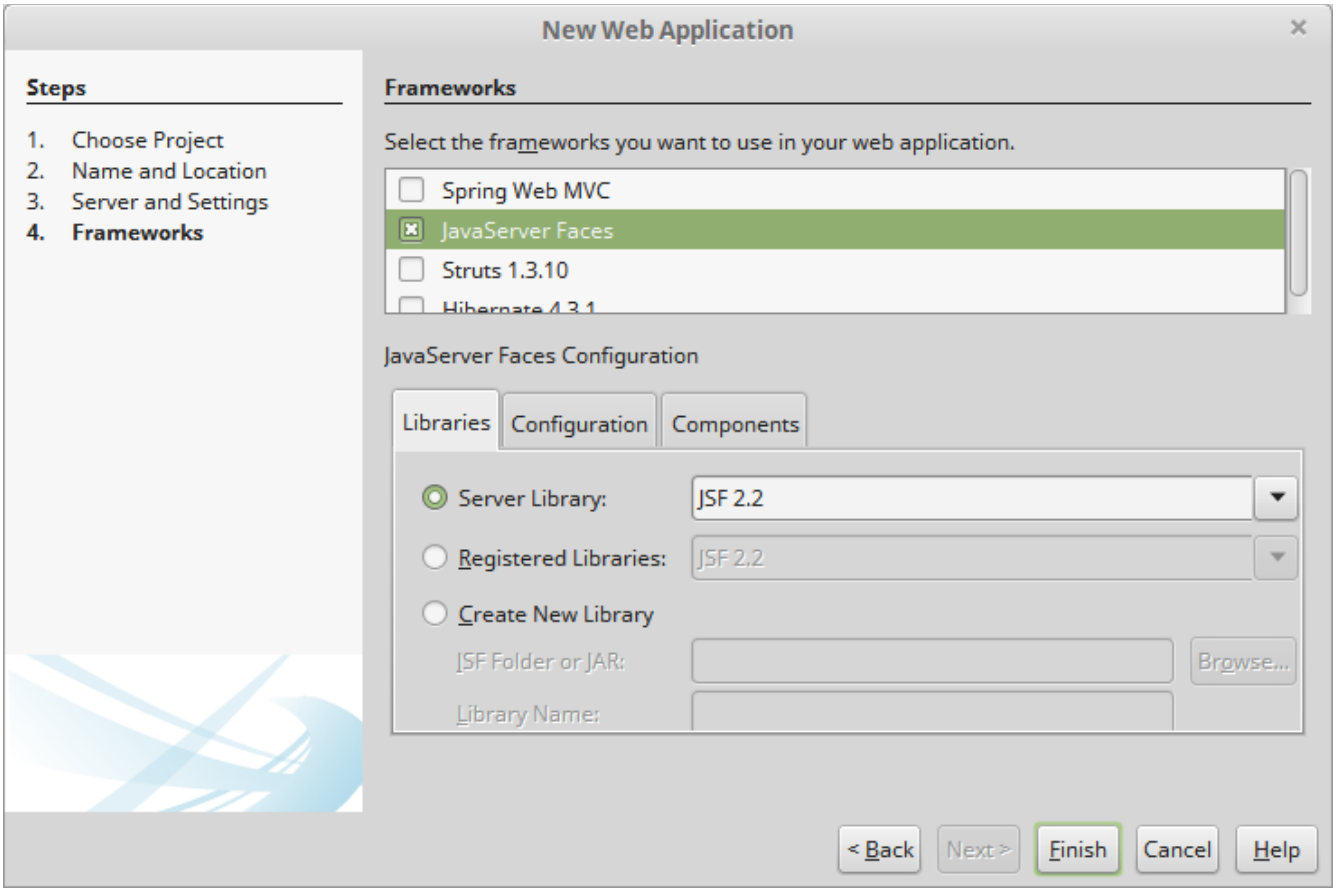
4. Click “Next”



5. Name your project “CDI_Stereotype” and click “Next>”

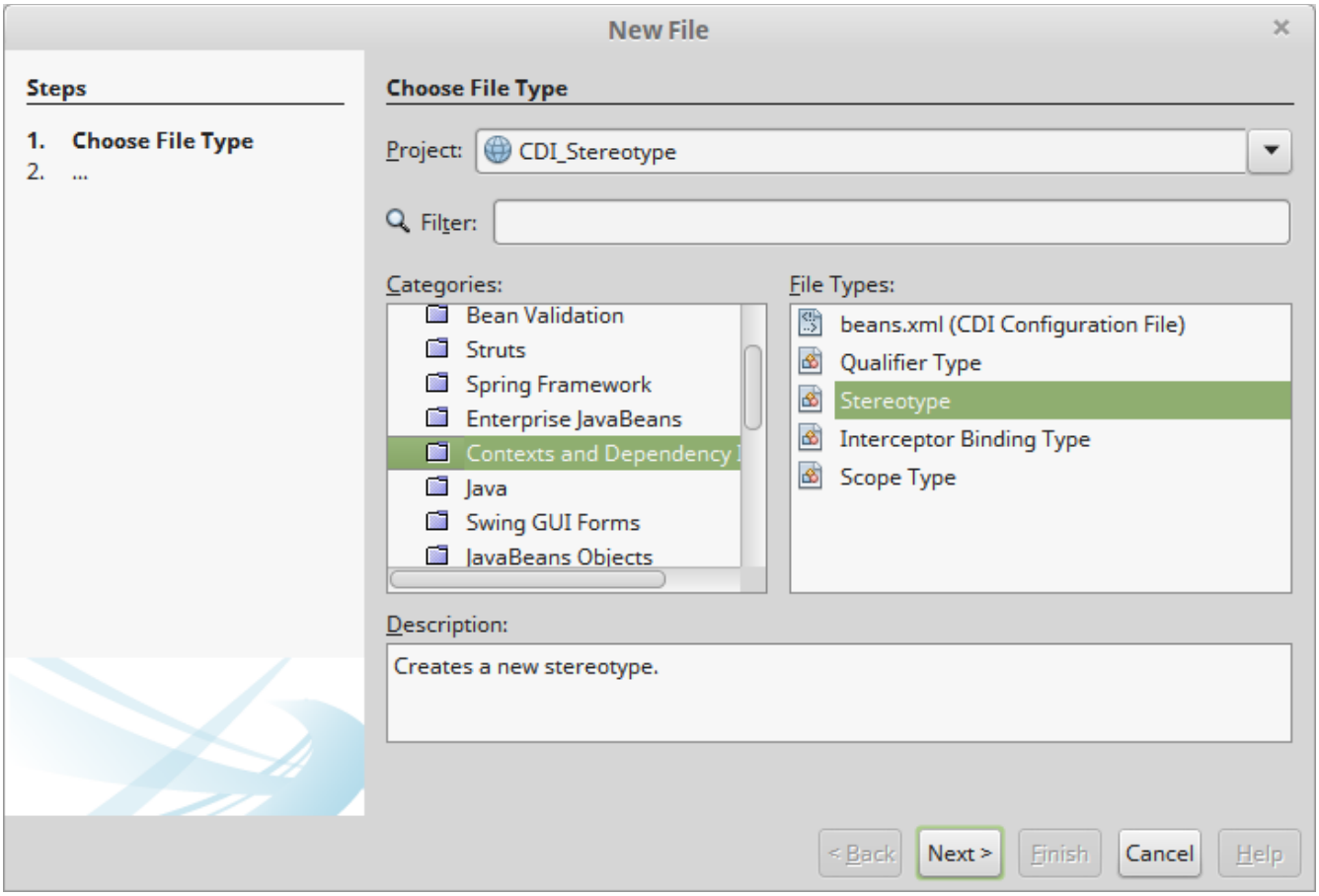


6. Accept the defaults and click “Next>”

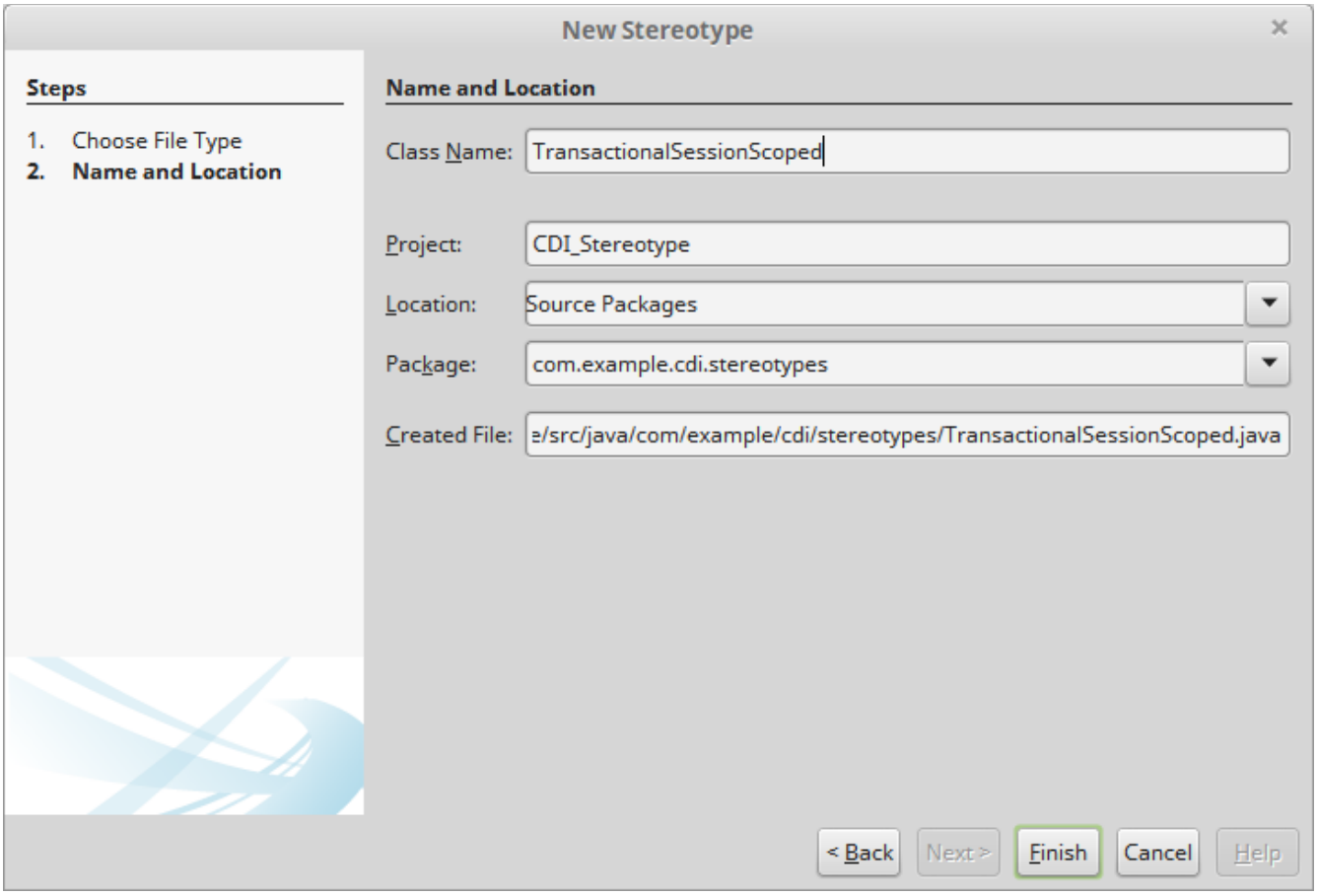


7. Select the “JavaServer Faces” framework and click “Finish”.

Develop A CDI Stereotype



1. Click on “File | New | Other”, select the “Contexts and Dependency Injection” category, and “Stereotype” file type, then click “Next>”



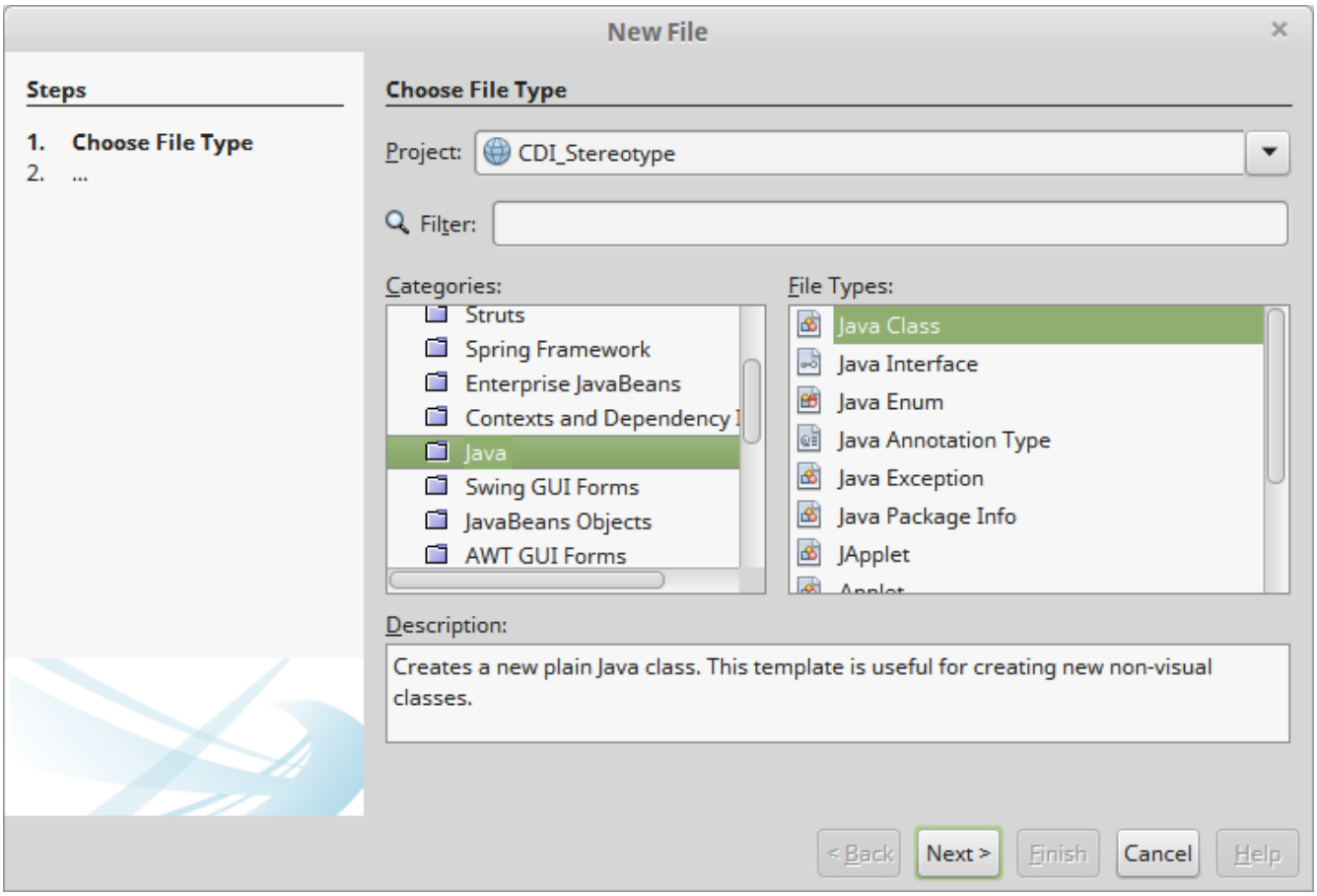
2. Enter “TransactionalSessionScoped” as the class name, and “com.example.cdi.stereotypes” as the package, click “Finish”

3. Add the @Transactional annotation (in package javax.transaction.transactional) to the generated CDI stereotype.

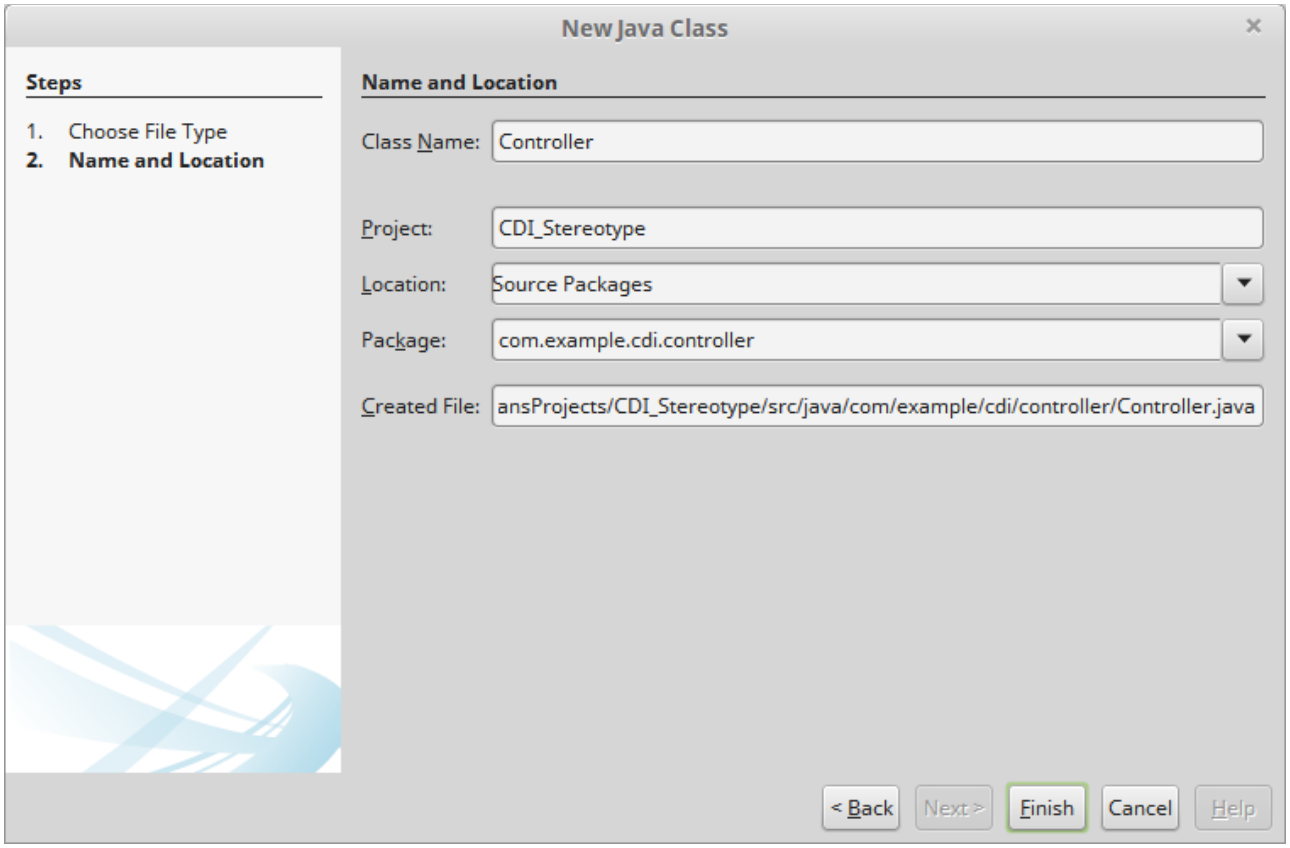
   Tip: CDI Beans can take advantage of container managed transactions by annotating them with the @Transactional annotation.

4. Add the @SessionScoped annotation(in package javax.enterprise.context) to the generated CDI stereotype.

Develop a CDI Bean



1. Create a new Java class (File | New File.., “Java” category and “Java Class” file type.



2. Name your class “Controller” and put it in package “com.example.cdi.controller”
3. Annotate your Controller class with the @Named and @TransactionalSessionScoped annotations.
4. Make your class implement the java.io.Serializable interface.
5. Add a boolean class level variable, name it “rollbackTransaction”
    1. initialize your “rollbackTransaction” variable to true.
6. Add a new public method to your controller class, name your method “navigate()”, and have it return a String (i.e. your method signature would be “public String navigate()”).
7. Add the following code to the body of your “navigate() method:

```
if (rollbackTransaction) {
      throw new RuntimeException();
}

return "confirmation";
```

Tip: Throwing a RuntimeException will force the transaction to roll back.

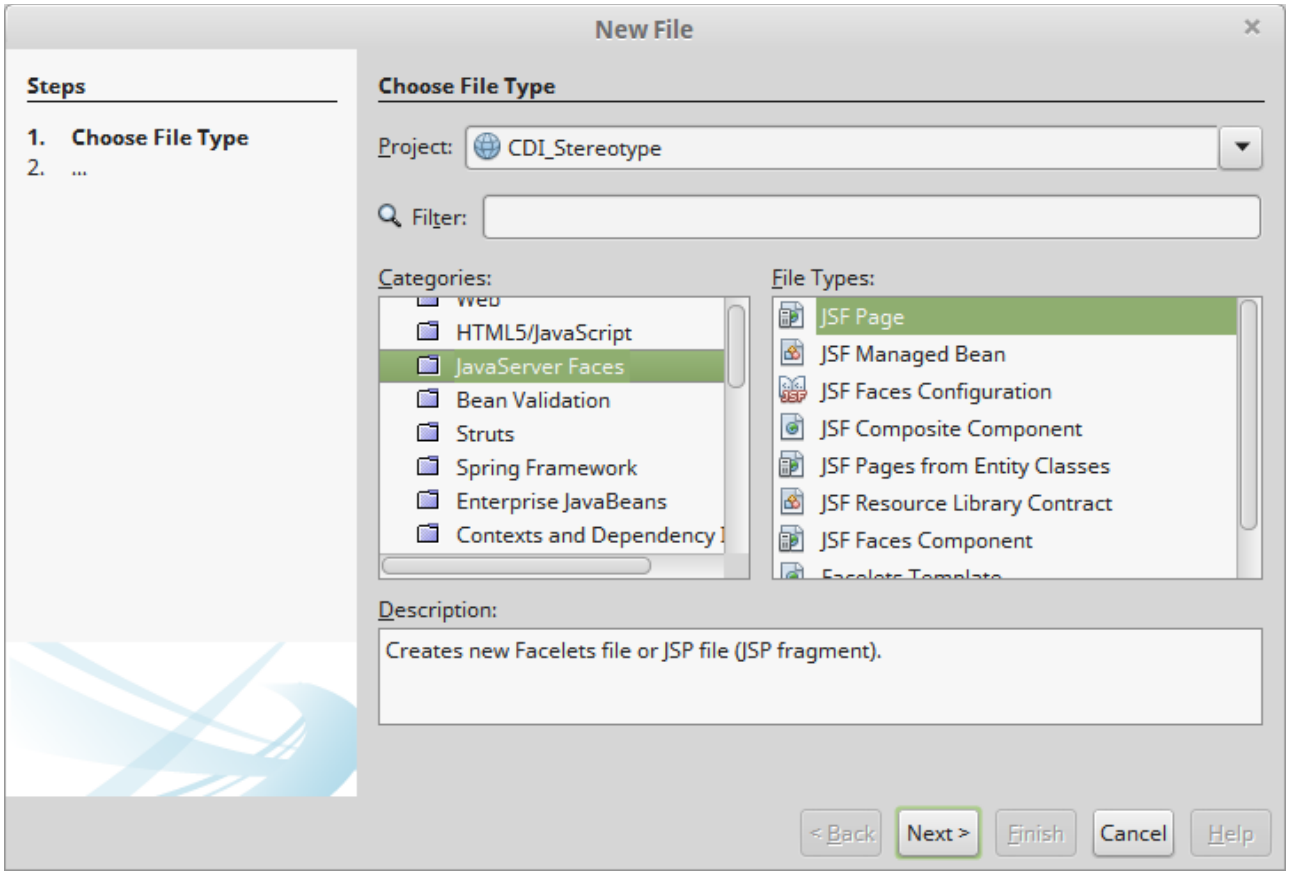
Update index.xhtml

1. Modify the body of the generated index.xhtml (automatically generated when we created the project, found under the “Web Pages” filder), to have the following markup inside its <h:body>

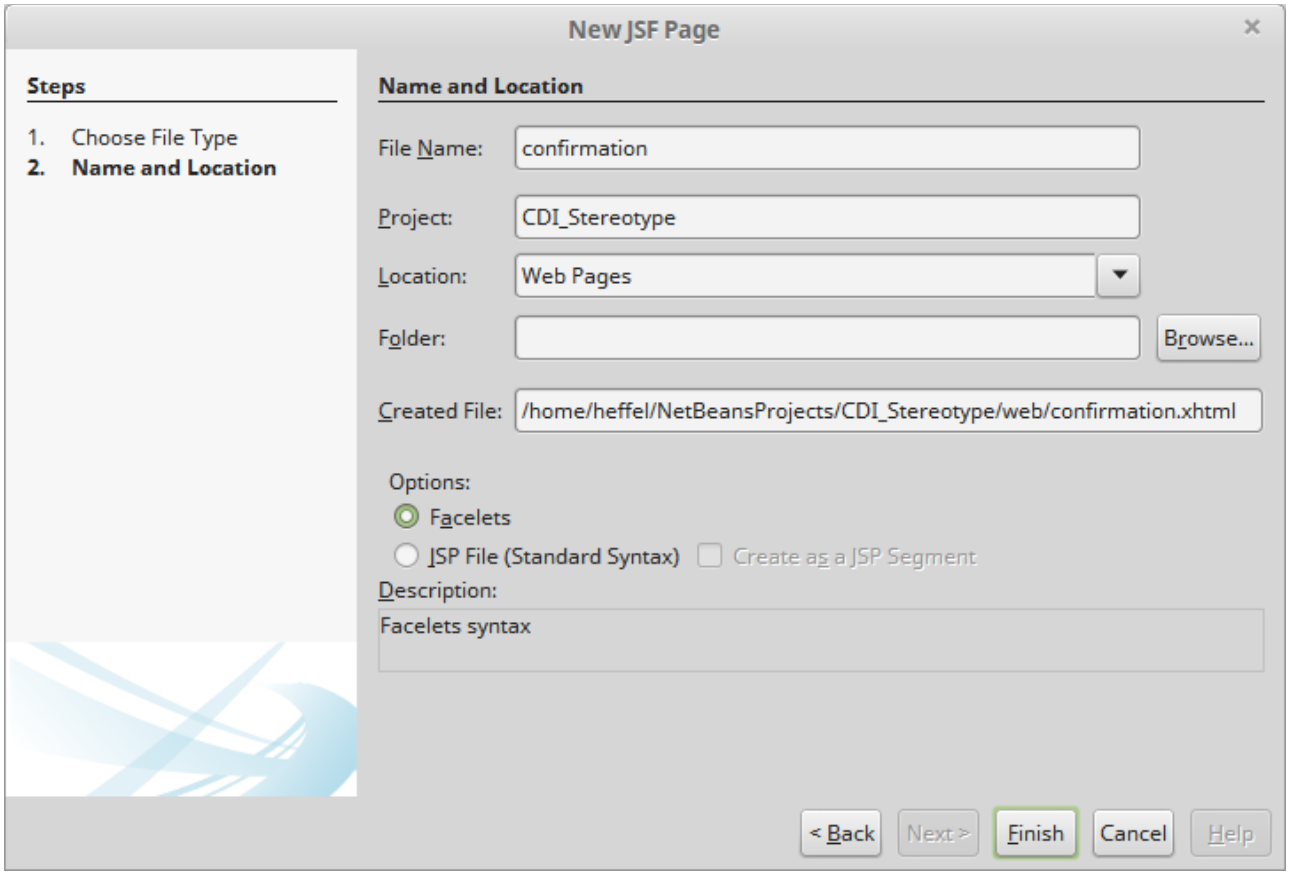
tag:

```
<h:form>
  <h:commandButton action="#{controller.navigate()}" value="Submit"/>
</h:form>
```

Develop a Confirmation Page



1. Click on “File | New File...”, select the “JavaServer Faces” category and “JSF Page” file type, click “Next>”



2. Name your file “confirmation” (NetBeans automatically adds the .xthml file extension), accept all other defaults and click “Finish”.

3. Change the body of the <h:body> tag in your confirmation page as follows:

```
<h2>If you see this, the transaction didn't roll back!</h2>
```

4. Right click on the project and select “Run”
5. Click on the “Submit” button on your browser, your transaction should have rolled back due to the RunTimeException we are throwing in the Controller class.
6. Modify Controller.java so that the “rollBackTransaction” variable is initialized to “false”.
7. Rerun your project.
8. Click on the “Submit” button.
9. Verify that the confirmation page is displayed.

Exercise 3

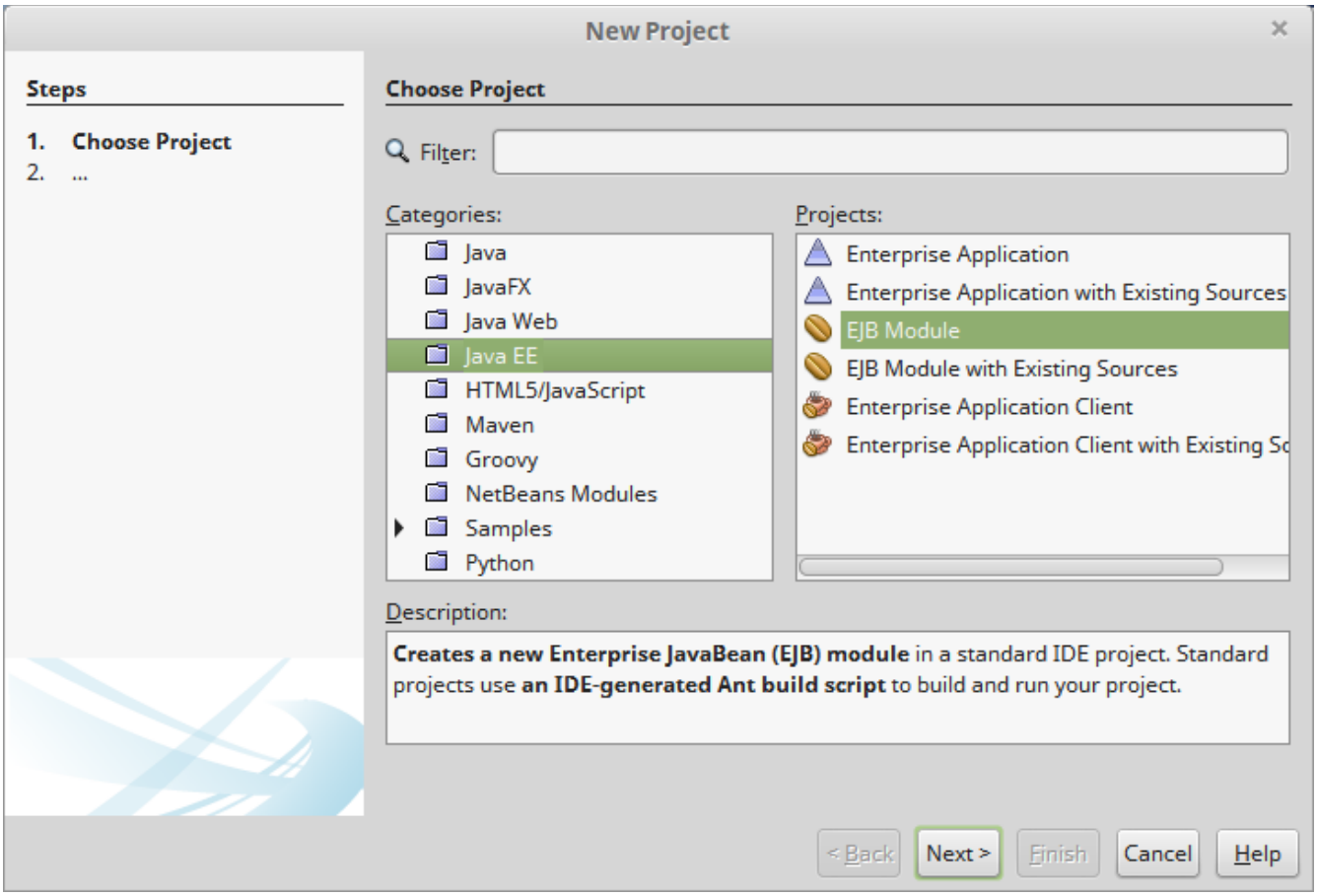
In this exercise, we will develop an EJB taking advantage of the EJB timer service.

Create the project

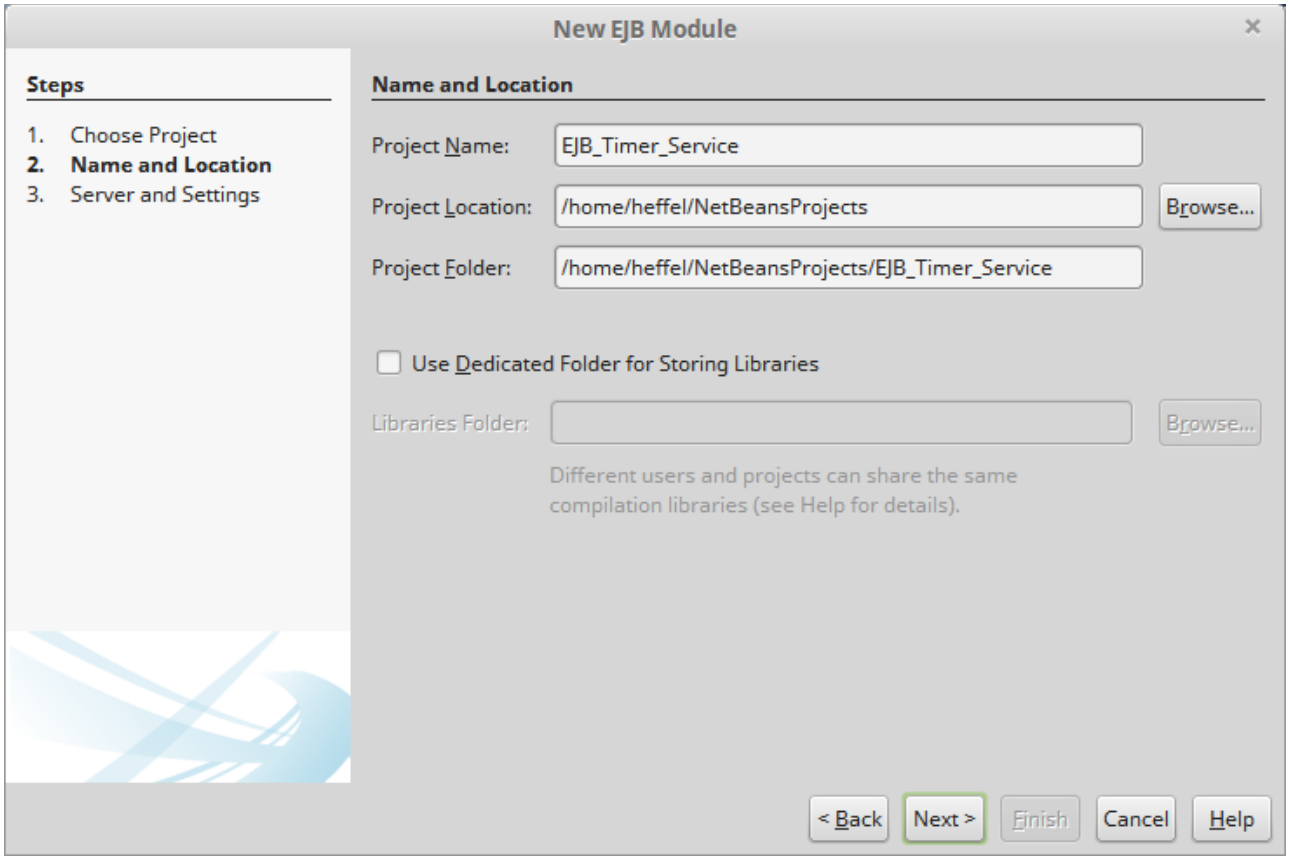
1. Create a new EJB Module Project (File | New Project)

2. Under “Categories”, select “Java EE”

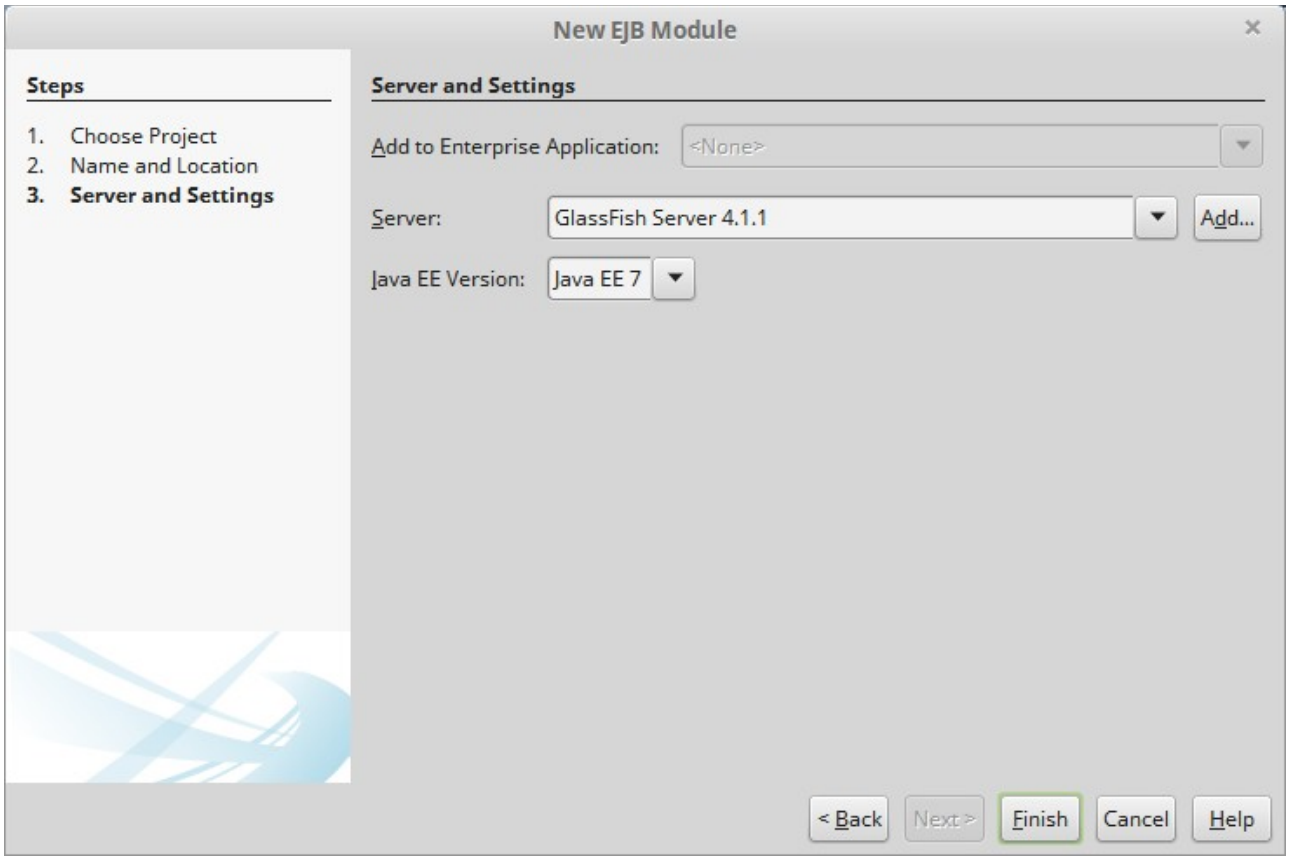
3. Under “Projects” select “EJB Module”



4. Click “Next>”

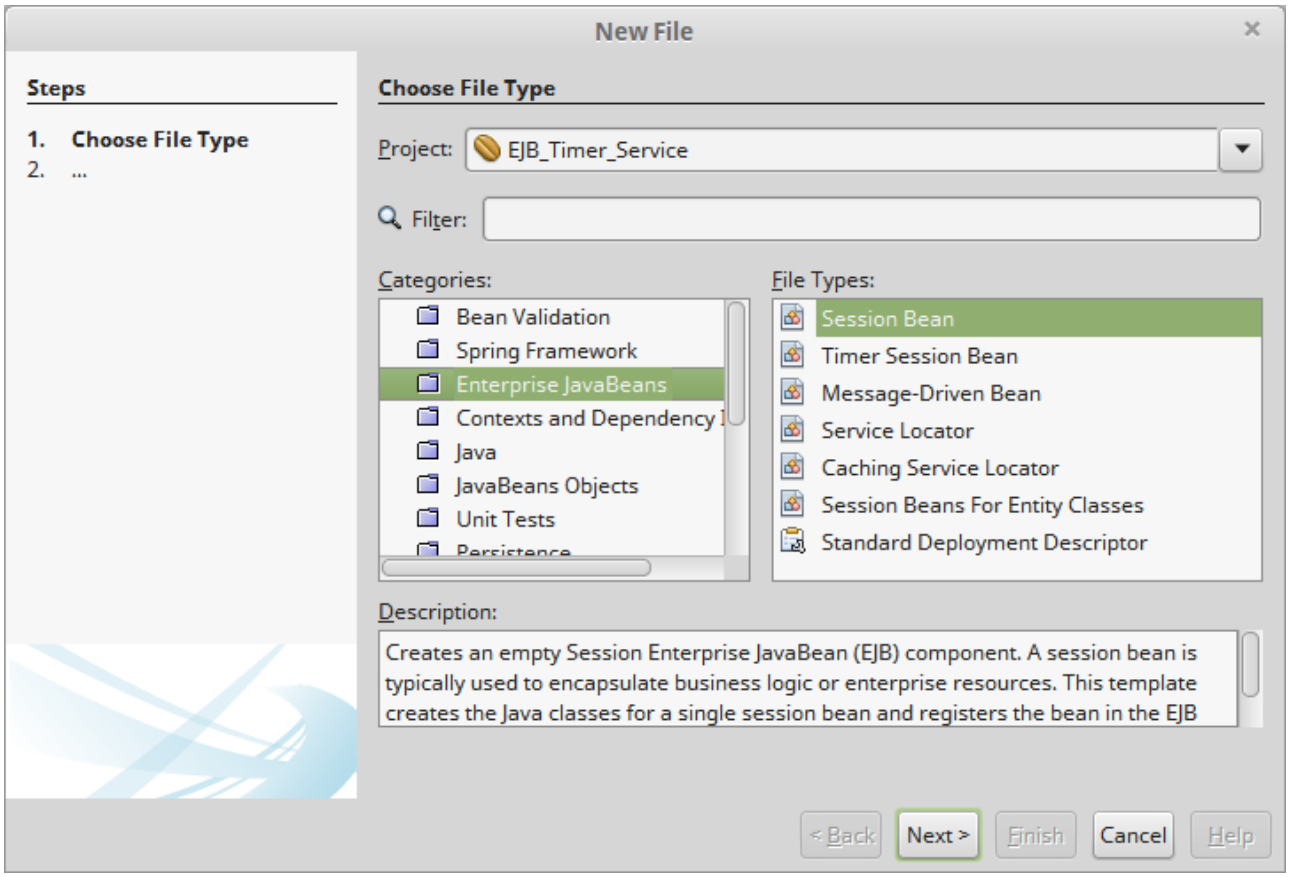


5. Name your project “EJB_Timer_Service”, click “Next>”

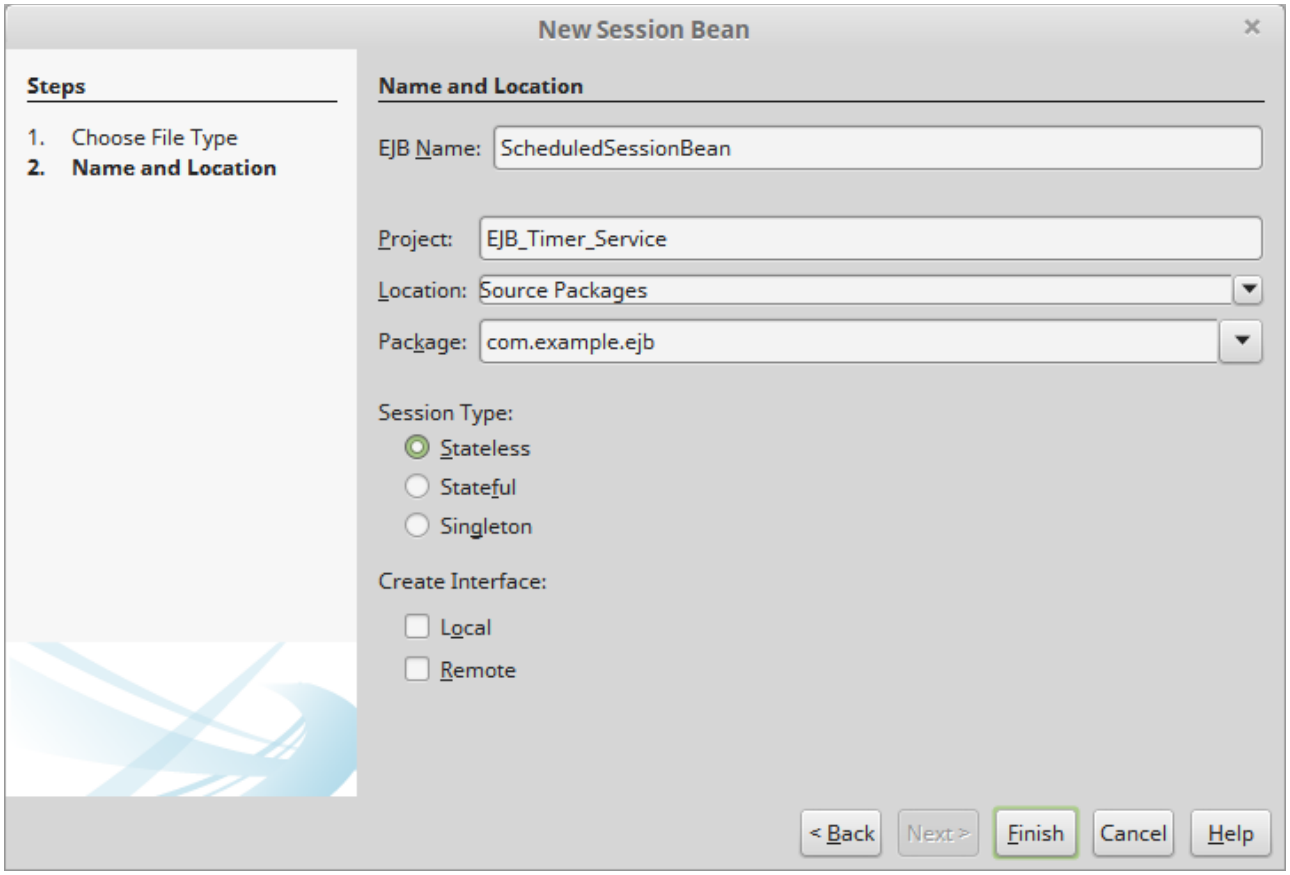


6. Accept the defaults and click “Finish”.

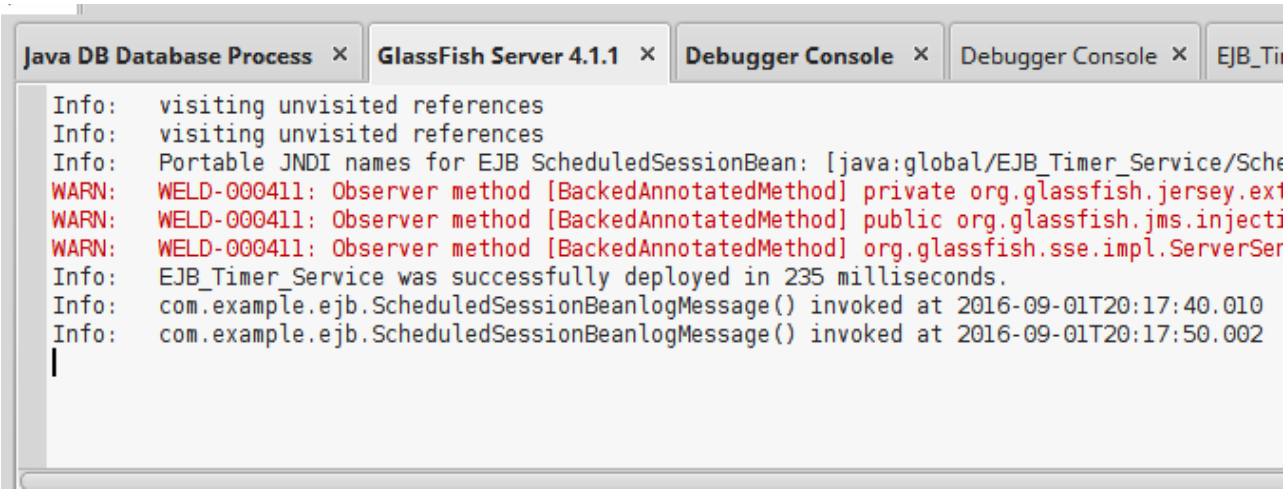
Develop a Session Bean



1. Click on “File | New File...”, select the “Enterprise JavaBeans” category, and “Session Bean” file type, then click “Next>”



2. Name your EJB “ScheduledSessionBean”, put it in pakage “com.example.ejb”, accept all other defaults and click “Finish”.
3. Add a private static final variable of type java.util.Logger to the generated EJB (hint: alt+ins, then select “Logger” to generate this variable). Name your variable “LOG”.
4. Add a public void method named logMessage(), enter the following code in the body of the method:
   ```
   LOG.log(Level.INFO, this.getClass().getCanonicalName() +
   "logMessage() invoked at " + LocalDateTime.now());
   ```
   Hit ctrl+shift+I to add missing imports.
5. Annotate your logMessage() method with the @Schedule annotation as follows:
   ```
   @Schedule(hour = "*", minute = "*", second = "*/10")
   ```
   This annotation will tell the application server to invoke our logMessage() method every 10 seconds.
6. Deploy your project (right click on the project, select “Run”.



7. Watch the server output and make sure you see the output from our logMessage() method every 10 seconds.