(21F) CST8277 Enterprise Application Programming

Assignment 1

Please read this document carefully – if your submission does not meet the requirements as stated here, you may lose marks even though your program runs. Additionally, this assignment is also a teaching opportunity – **material presented in this document may be on the Midterm and/or the Final Exam!**

Assignment 1's submission is to be uploaded to the Assignment1 Brightspace submission folder. It does not matter if it is a zip-file containing a zip-file containing... many-levels-of-nested-zip-files or if you upload the required artifacts individually, Brightspace will handle things. If you have a 100% perfect solution but it is not in the correct folder, you will receive a mark of zero ☹ If you know that you made mistake, please upload it to the correct folder ASAP.

Theme for Assignment 1

After completing this assignment, you will have achieved the following:

1. Created a Managed Bean class to manipulate the Model class
2. Created a DAO class to manipulate the database
3. Created a JSF Application that supports **C**reate – **R**ead – **U**pdate – **D**elete lifecycle for instances of the Model class

Starting Assignment 1

You **must** use the file from Brightspace ***DataBank-JSF-JDBC-Skeleton.zip*** to start your solution – extract its contents to some work folder on your hard drive. ***Before*** you import the project into Eclipse, please change the **name** of the project to include your Student name and number (this is done in two places).

In the folder src/main/resource, there is a file called **Bundle.properties** which contains constants used in the UI – please change the default values for:

```
footer.labsection=Section 3nn
footer.studentnumber=040123456
footer.studentname=StudentName
```

(Your name should be as it appears in ACSIS).

A bundle is ***bound*** to a variable in JSF, declared in the **faces-config.xml** file:

```
<?xml version="1.0" encoding="UTF-8"?>
<faces-config
  xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
    http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2_3.xsd"
  version="2.3">
  <application>
    <resource-bundle>
      <base-name>Bundle</base-name>
      <var>uiconsts</var>
    </resource-bundle>
  </application>
</faces-config>
```

The starter project is an Eclipse Maven project. There is a screencast on Brightspace that shows how to ***import*** an existing Maven project into Eclipse (Note: it is from an earlier version of the course but it is still relevant) – choose the 'Existing Maven Projects' import wizard

Submitting Assignment 1

Your submission must include:

- **Code** – completed project that compiles, has good indentation, not overly complicated and is efficient
    - the code must demonstrate your understanding of how a JSF Application works
- Eclipse has an 'export' function to create an external 'archive' – i.e. a zip file – of your project.

Running Assignment 1

You may notice if you try to run the starter project un-changed, it does not show any people; instead there is an Internal Server Error:

This is *done on purpose*. You must find and fix the problems as well ***extend*** the project to provide full C-R-U-D capabilities.

Data Access Object (DAO) Pattern

(Java EE) There is a 'pattern' for Data Access Objects (DAO) classes. The following reference is a great introduction: http://ramj2ee.blogspot.in/2013/08/data-access-object-design-pattern- or.html

To connect the Controller class with the DAO class, we can **inject** a reference into the Controller class' member field:

```
@Inject
protected PersonDao personDao;
```

Note: for the DAO, we inject the Interface, not the implementation!

JSF Tips N' Tricks

In the first JSF Presentation view XHMTL file **list-people.xhtml**, you must update the <h:viewAction> widget with the appropriate information so that when the Application is about to *update its model classes* (hint, hint – JSF phases), all the people are loaded from the database.

The controller class should be scoped to the HTTP Session so that there is only one instance for **http://localhost:8080/people/list-people.xhtml** (and for navigating back-&-forth to the add/edit pages). If another Web browser (or 2nd tab of the same browser) connects, a second controller will be instantiated. **Q: What is the appropriate scope for the DAOImpl class? for the Model class?**

The Presentation view XHTML file(s) responsible for editing or updating a Model object selected from the list needs a way to communicate between pages. JSF maintains a 'sessionMap' that can hold (almost!) any object – **Q: how is the sessionMap injected?** A <h:commandLink> button's action attribute can invoke a method on the controller class which can then put something 'special' in the map:

list-people.xhtml

```
...
  <h:commandLink
      action="#{personController.doSomething(pers.id)}">
  </h:commandLink
```

PersonController.java

```
...
    public String doSomething(int id) {
        PersonPojo c1 = personDao.readPersonById(id);
        sessionMap.put("editPerson", c1);
```

list-people.xhtml

```
...
   <h:panelGrid columns="2">
     <h:outputLabel value="#{uiconsts['columnLabel_Firstname']}" />
     <h:inputText value="#{editPerson.firstName}" id="firstName" />
...
```

This way, when the edit XHTML form is presented, the existing values are populated into the fields.

The add XHTML form requires the same sort of thing except that we do not wish to populate the add form with the information of the last selected entity: in that case, we put a entirely new empty person Model object into the session map:

```
        sessionMap.put("newPerson", new PersonPojo());
```