IPL escola superior de tecnologia e gestão instituto politécnico de leiria

DEI Departamento Engenharia Informática

Morro do Lena, Alto Vieiro · Apart. 4163·2401 – 951 Leiria
Tel.: +351·244 820 300 · Fax.: +351·244 820 310
E-mail: estg@estg.ipleiria.pt · http://www.estg.ipleiria.pt

Desenvolvimento de Aplicações Empresariais – 2015-16-1S

Engenharia Informática – 3.º ano – Ramo SI

Worksheet 1

Topics: Java EE first Cup ☺ - Creating a simple Entity, creating EJBs (Singleton and Stateless), managed beans and simple JSF pages.

In the PL classes of DAE we will develop a enterprise application for academic management. In this first worksheet, please, execute the following steps:

1. Create the ACADEMICS database with the *dae* username and password.
2. Create the *AcademicManagement* web application (New Project -> Categories: Java Web -> Projects: Web Application). In the last step of the project creation wizard select the JavaServer Faces framework).
3. Create a persistence unit:
   - New File -> Categories: Persistence ->File Types: Persistence Unit;
   - Persistence Unit Name: AcademicManagementPU;
   - Data Source -> New Data Source -> JNDI Name: academic_management -> Database Connection: jdbc:derby: .../ACADEMICS [dae on DAE];
   - Table Generation Strategy: Drop and Create
4. Create the *Student* entity in the *entities* package. It should have the following attributes: *username* (which is the entity's ID), *password*, *name* and *email*. The entity must have at least a default constructor and a get and set method for each attribute. Also write a constructor that receives and sets all attributes' values.
5. Create the stateless Enterprise Java Bean (EJB) *StudentBean* in the *ejbs* package and write the method *createStudent(...)* that receives as arguments the attribute values needed to create a student, creates and persists a student. Don't forget to declare and use an *EntityManager* in the EJB.
6. Create the singleton EJB *ConfigBean* (use the @Singleton annotation) in the *ejbs* package. It should be instantiated be the server when the application starts up (use the @Startup annotation). Write the method *populateDB()* that creates students and persists them in the

database. You need to call the *createStudent(…)* method of the *EstudentBean* EJB. To do so, you need to declare a *StudentBean* variable in the *ConfigBean* EJB and annotate it with the @EJB annotation. The *populatedDB()* method should be called by the server right after it instantiates the *ConfigBean* EJB (annotate the method with the @PostConstruct annotation).

7. Run the application and confirm that the STUDENT table was created in the ACADEMICS database, as well as the lines corresponding to the students created in the *populateDB()* method.
8. Create the *AdministratorManager* Managed Bean (@ManagedBean annotation) in the *web* package. It should have a session scope (@SessionScoped) and a default constructor. This Managed Bean will be used as the model for the admins_student_create.xhml JavaServer Face (JSF) page (see the steps below). Declare a *StudentBean* EJB and attributes that can save a student's attributes. For each of the student's attributes you should have corresponding getters and setters methods through which the communication with the admins_student_create.xhml JSF will be done (in a transparent way to the programmer). Write the *createStudent()* (without arguments) method. This method calls the *createStudent(…)* method of the *StudentBean* EJB, sending as arguments the values of the attributes mentioned above.
9. Create the admin_students_create.xhml JavaServer Faces (JSF) page.
10. Modify the index.xhtml page so that it presents a link that, when clicked, opens the admins_student_create.xhml page (use the h:form and h:commandLink JSF components). Run the application and test it.
11. Modify the admins_student_create.xhml page so that it presents a form that allows the user to enter a student's data and submit it to the server. Each component for entering data should be preceded by an informative label ("username: ", "password: ", etc.). Use the h:inputText component to allow the user to enter all the data except the password one. For the later use the h:inputSecret component. Both the h:inputText and h:InputSecret components have a *value* attribute. This attribute allows, through the use of the Expression Language (EL) expressions, the interaction between JSF pages and Managed Beans (the *AdministratorManager* Managed Bean, in this case). For example, for writing/reading the username value to/from the Managed Bean attribute that saves the username value, you may write the EL expression #{administratorManager.newStudentUsername} (if the attribute's name is newStudentUsername). After entering the data, the user must click the "Create" button (use the h:commandButton component). In this component's *action* attribute write the EL expression #{administratorManager.createStudent} so that the *createStudent()* method of the Managed Bean is called when the user clicks the button.
12. Use the *required* and *requiredMessage* attributes of the h:inputText and h:inputSecret components so that, respectively, the introduction of values are mandatory to all four student's attributes and to define an error message if the user does not enter any value. Also, right after each h:inputText and h:inputSecret components, put a h:message component so that all the messages for the former ones are shown right after them. For this, you need to

define a value for the *id* attribute of each h:inputText and h:inputSecret component and repeat it in the *for* attribute in the corresponding h:message component. You may do the same for the h:commandButton component.

13. Run and test the application. Confirm that the students created by the user are added to the database.
14. In the previous step you may have noticed that the components in the admins_student_create.xhml page are not aligned. Wrap all the previously mentioned components (h:inputText, h:inputSecret and h:commandButton) in a h:panelGrid component to solve this cosmetic problem, setting the *columns* attribute to 3 (why 3?).
15. You may also have noticed that the values entered for a student appear again when the page is refreshed. This happens because, as mentioned above, the EL expressions are used not only for writing but also for reading the values from the attributes of the Managed Bean they refer to. If you don't "clear" those values before the page is refreshed, they will be read and put in the page components again. For sure, you want those components' values to be empty before the page is refreshed. In order to achieve this, just create a private method *clearNewStudent()* in the Managed Bean that sets to null all the students attributes and call it right after creating the student in the *createStudent()* method.
16. In the admins_student_create.xhml page, add the following f:validateRegex component to the h:inputText component used the read the student email so that the entered email address format is validated:

    ```
    <f:validateRegex pattern="[\w\.-]*[a-zA-Z0-9_]@[\w\.-]*[a-zA-Z0-9]\.[a-zA-Z][a-zA-Z\.]*[a-zA-Z]" />
    ```

    You need to define the xmlns:f="http://java.sun.com/jsf/core" namespace to use the f:validateRegex component.

Note: you may/should consult the Order project that comes with the Java EE Tutorial.

Bibliography

Java EE Tutorial 7 (all of it).