CPRG352 - Web Application Programming

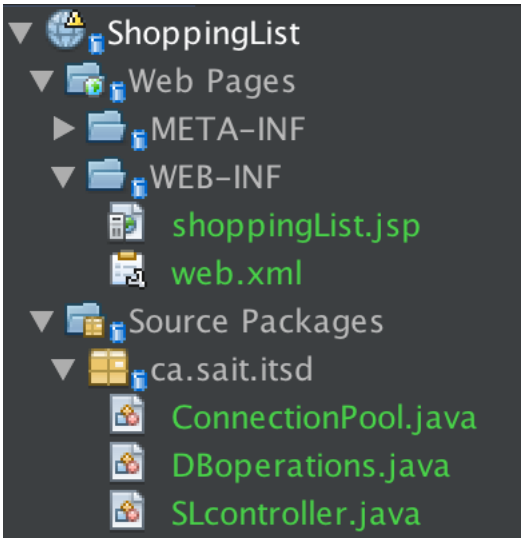
Fall 2021

Topic: Use of JPA (Java Persistence API) for data access

Problem: Revise ShoppingListApplication to use JPA instead of JDBC

For this lab you should make a copy of the solution for the **ShoppingList** application seen in the JDBC module and rename it **ShoppingListJPA**. Your task for this lab is to revise the application so that it uses JPA to retrieve and manipulate data in the ***shoppinglist*** database, rather than JDBC[1].

The structure of the ShoppingList(JPA) application is as follows:



The database operations are handled in the **DBoperations** class. That is where most, but not all, of the changes required to switch from JDBC to JPA for data access will be made.

Additionally, to use JPA you will need to add a ***persistence unit*** (PU) and an ***entity class*** representing the **listitems** data in the database to the application.

Adding an Entity Class and Persistence Unit to the Project

- In NetBeans, right-click on the project name and select **New…**

[1] You can find copies of the required project and SQL file for creating the database under "Sample Code" on BrightSpace.

- Select "**Entity Classes from Database…**"

- In the "**New Entity Classes from Database**" dialog box pull down the "**Database Connection**" list, and select "**New Database Connection…**"

- In the "**New Connection Wizard**" dialog box pull down the "**Driver**" list and select "**MySQL**". Then click on "**Next>**"

- In the next pane of the wizard, enter the database name "**shoppinglist**" in the "**Database**" field, and the MySQL password ("*password*") in the "**Password**" field. Select the "**Remember Password**" check box

- Click on the "**Test Connection**" button. You should see the message "***Connection succeeded***" appear after a couple of seconds. Click on "**Next>**"

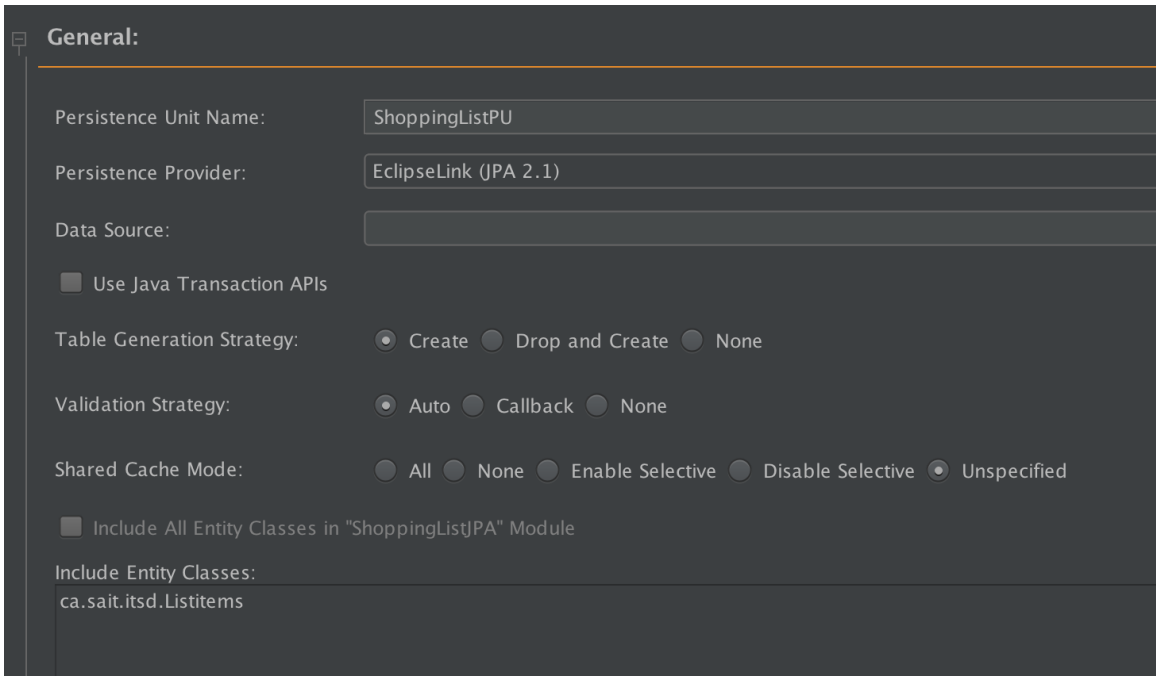
- Click on the "**Finish**" button. You should now see the "**New Entity Classes from Database**" dialog again, but this time the table "**listitems**" should appear in the "**Available Tables**" list box. Select it and click on the "**Add>**" button in the middle of the dialog to move it to the "**Selected Tables**" list box. Click on "**Next>**"

- In the next pane of the wizard make sure that the check box "Create Persistence Unit" is selected (leave the other options at their default values) and click on "**Next>**"

- In the final pane of the wizard change the "**Collection Type**" list to "**java.util.list**" and select "**Fully Qualified Database Table Names**". Then click on the "**Finish**" button

Two additional files have been added to the project:

1. In the Java source code package in the project you should see a file called **Listitems.java**. This is the entity class that represents the ***listitem*** table data in the application (as a **java.util.List** of **Listitem** objects). The persistence unit can populate that list, and also add, remove and update objects in the list, as required
2. In the "**Configuration Files**" folder of the project you should see a file called "**persistence.xml**". This file contains the configuration information for the persistence unit. It should look as follows:



The name of the PU is **ShoppingListPU**. The class managed by the PU is called **ca.sait.itsd.Listitems** (from the source code folder in the project)

If these two files are created properly in the project, then you can continue on and rewrite the **DBoperations** class to use them for data access and manipulation.

The ListItems Entity Class

If you open the **Listitems.java** file you should see that it contains pre-generated code representing the **listitems** table in the database, i.e. the class contains attributes for each column in the **listitems** table, and operations for accessing and setting them, etc. One **Listitem** object in memory will represent one row of data in the database table (that's why we need a List of them, to represent the whole table).

At the top of this file you will see the pre-generated JPQL queries:

```
@NamedQueries({
    @NamedQuery(name = "Listitems.findAll", query = "SELECT l FROM Listitems l")
    , @NamedQuery(name = "Listitems.findByListitemid", query = "SELECT l FROM Listitems l WHERE l.listitemid = :listitemid")
    , @NamedQuery(name = "Listitems.findByListitemdesc", query = "SELECT l FROM Listitems l WHERE l.listitemdesc = :listitemdesc")
    , @NamedQuery(name = "Listitems.findByListitemincart", query = "SELECT l FROM Listitems l WHERE l.listitemincart = :listitemincart")})
```

These named queries show you the syntax for JPQL. You may need to create your own JPQL queries to retrieve data for this application, e.g. one to get only items that *are* in the cart, another to get items that are *not* in the cart, etc.

-

Updating the DBoperations.java File

getItemsInCart:

- Create the JPQL query as a String. The query should only return data for rows in the **listitems** table that have a **listitemincart** column value of "**true**" (you can hard-code values in JPQL queries, so you don't need parameterized arguments like "**:listitemid**", as seen in the named queries in **Listitems.java**)
- Create an **EntityManager** object that gets its data from the **ShoppingListPU** persistence unit
- Create an object of type "**TypedQuery<Listitems>**" by calling **.createQuery()** on the entity manager (look up the arguments required by **.createQuery()**)
- Get a **List** of **Listitems** objects from the database by calling **.getResultList()** on the **TypedQuery<Listitems>** object
- Loop through the list of **Listitems** objects and call the getters to pull out the list item ID, description and "item in cart" field values. Separate these with commas, and end the string with a semi-colon (";") to delimit separate records. This needs to be done for the JSTL loops in **shoppingList.jsp** to be able to parse and show the data (look at that code if you are not sure on the required string formatting)
- Close the entity manager when the loop finishes
- Finally, return the string containing all of the semi-colon delimited row data from the method

getItemsNotInCart:

This is exactly the same as for "**getItemsInCart**", except that in the first step you are retrieving rows with a **listitemincart** column value of "**false**" instead of "**true**".

addListItem:

- Create a **boolean** variable called **result**, initialize it to **false**
- Create an **EntityManager** object that gets its data from the **ShoppingListPU** persistence unit
- Create an **EntityTransaction** object also, by calling **.getTransaction()** on the entity manager object
- Begin a new transaction
- Create a new **Listitem** object and initialize it to hold the description of the new list item
- Call **.persist()** on the entity manager and provide a reference to the new **Listitem** object to tell the persistence unit that you want to save the new object's data to the database
- Commit the transaction
- Set the boolean **result** variable to true to indicate that the addition operation was successful

- If an exception occurs, roll back the transaction and print out a stack trace
- One way or the other, close the entity manager
- Return the value of the **result** variable

deleteListItem:

- Create a **boolean** variable called **result**, initialize it to **false**
- Create an **EntityManager** object that gets its data from the **ShoppingListPU** persistence unit
- Create an **EntityTransaction** object also, by calling **.getTransaction()** on the entity manager object
- Begin a new transaction
- Create a **Listitem** object representing the **listitems** table row you want to delete. You need to search for the row with the specified **itemID** method argument value. To do this:
    - call the **.createNamedQuery()** method on the entity manager. Provide the name of the named query in the **Listitems.java** file you want to use ("***findByListitemid***")
    - Call the **.setParameter()** method on the named query object created in the previous step, and use it to set the value of the ":**listitemid**" query parameter to the **itemID** method argument value
    - Call **.getSingleResult()** to return a **List** containing a single **Listitem** object representing the data in the row in the database with the provided **itemID** value
    - Call **.remove()** on the entity manager, provide a reference to the **Listitem** object so that the row it represents in the database will be deleted
- Commit the transaction
- Set **result** to true
- Rollback if there was an exception during all of the above, and print out a stack trace
- Close the entity manager
- Return the value of **result**

toggleInCartStatus:

- Create a **boolean** variable called **result**, initialize it to **false**
- Create an **EntityManager** object that gets its data from the **ShoppingListPU** persistence unit
- Create an **EntityTransaction** object also, by calling **.getTransaction()** on the entity manager object
- Begin a new transaction
- Use the **findByListitemid** named query to get a **Listitem** object representing the row whose "in cart" status you want to change (see "**deleteListItem**" steps for this)

- Call the .**setListitemincart()** method on the object to set it to the Boolean "***not***" of the current value it has (this is a simple way to toggle the value)
- Call the **.persist()** method on the entity manager, providing a reference to the **Listitems** object with the toggled value
- Commit the transaction
- Set the value of **result** to **true**
- Rollback if there was an exception during all of the above, and print out a stack trace
- Close the entity manager
- Return the value of **result**