CE212 ASSIGNMENT 2 2021

Set by: Mike Sanderson

Credit: 10% of total module mark

Deadline: 11.59.59, Monday 22 March

You should refer to the Undergraduate Students' Handbook for details of the University policy regarding late submission and plagiarism; the work handed in must be entirely your own.

It is expected that marking of the assignments will be completed by the beginning of the summer term.

Introduction

On Moodle you will find in a file **`ass2.zip`** an incomplete implementation of an art shop. This assignment involves the completion of the implementation.

The final version of the deployed code must be submitted in a folder called **`ass2`** in the folder **`tomcat\webapps`**. It is recommended that during development you use a Tomcat local server in IntelliJ. Details of how to create an appropriate deployment folder from the IntelliJ files will be supplied online in week 24.

The shop uses an HSQLDB database; refer to lab 6 (in unit 4 on Moodle) for the hsqldb jar file and instructions for setting it up for IntelliJ. The database files (from the **`database`** folder in the zip file) should be stored directly in the **`tomcat\webapps\ass2`** folder (i.e.not in a sub-folder called **`database`**). If you are using your own PC for development and Tomcat is not in a top-level folder on the C: drive you may need to change the first argument to the `getConnection` function in the `ShopDB` constructor; remember to restore it to the original before submission. In the final submission the **`images`** and **`thumbnails`** folders should be placed in the **`ass2`** folder; during development you will probably need to place copies of these folders in the same folder as your JSP files.

A text file within the supplied zip file contains instructions on how HSQLDB databases may be accessed outside of Java programs.

Subject to the above restrictions you may make any reasonable modifications to the supplied code and database.

I will be using Firefox when testing your submissions so you should make sure that everything works correctly in that browser.

Submission

Any CSS files and JavaScript files you create should be placed in **`tomcat\webapps\ass2`** (in appropriate subfolders if their names in the HTML refer to these). They should have personalised names (e.g. include your initials or student ID within the file name), since I will be testing all of your assignments on my M: drive so versions of other students' files in my browser cache could be picked up instead of your files if they happen to have the same names. (This personalisation of names does not apply to JSP files.)

Copies of all files (i.e. the complete contents of **`tomcat\webapps\ass2`** and all java source files) must be submitted to FASER by the deadline in a single `.zip`, `.7z` or `.rar` file; no other file formats will be accepted. (Do not submit the entire contents of your **`tomcat`** or **`tomcat\webapps`** folders.)

Part 1 [20%]

This part of the assignment involves getting the basics of the shop working so that a user can browser the catalogue of available products, add products to a shopping basket and check out, with the order being recorded in the database.

The supplied code uses JSP to present the shop data to the user in HTML. The JSP is kept reasonably simple by doing most of the work in Java. Java classes have been defined to handle all the database access (**`ShopDB.java`**), to store details of an individual product (**`Product.java`**), and to store a set of products in a basket (**`Basket.java`**).

The JSP pages are as follows:

`products.jsp`:
: Lists the products available in the shop

`viewProduct.jsp`
: Displays the details of a particular product, and allows it to be added to the user's basket

`basket.jsp`
: Displays all the items in the user's basket of each user. Should also display the total cost of all items. Should provide a means for emptying the basket, and for placing an order for all the contents of the basket, given the user's name (which can be input using a text-field on this page).

`order.jsp`
: Responsible for making the order transaction. Having completed the order, it should then empty the basket of all its items.

The following changes need to be made in order to make the shop usable.

1. Fix **`viewProduct.jsp`**

   The product is currently shown, but you are not able to add it to the basket. Fix this by using an appropriate hyperlink to **`basket.jsp`**.

2. Fix the **`getTotal`** and **`getTotalString`** methods of **`Basket.java`**

   The supplied **`getTotal`** method returns a default result instead of the actual total price; you will need to iterate over the contents of the basket to calculate the correct total. The prices in the database are in pence (unlike the example in the lecture slides); the **`getTotal`** method should return a number of pence, but the **`getTotalString`** method must return the price in pounds and pence (with exactly 2 digits being used for the number of pence in all cases e.g. £27.05 or £27.50, not £27.5).

3. Modify **`products.jsp`**

   Product prices are currently shown as pence; they should be shown as pounds and pence.

4. Complete the **`order.jsp`** page and the **`order`** method of **`ShopDB.java`**

   The current JSP page has some parts commented where you need to make the order; these must be completed. Once the order has been made, you then need to empty the basket.

   You should also add some JavaScript to check that the name has been supplied when the form is submitted.

   The private **`order`** method currently does nothing; you must complete this method. In order to prevent any possibility of SQL injection a prepared statement must be used.

Part 2 [25%]

Although you should now have a working shop things can be improved. The next requirement is to improve the display of the basket. A thumbnail image of each item should be shown and the items should be displayed in tabular form.

Additionally if more than one copy of an item is contained in the basket it will currently appear several times. The page should instead show the quantity, the price per item and the total price. To facilitate this, the **`Basket`** class will need to be modified; the simplest way is to use a map instead of a list.

The total cost of the basket contents should also be displayed.

A facility to change remove items from the basket and to change their quantities should be provided.

Make the necessary changes to **`Basket.java`** and **`Basket.jsp`** to meet these requirements. You will also need to modify the public **`order`** method of **`ShopDB.java`** so that quantities other than 1 can be stored in the database.

Part 3 [5%]

There is an additional problem with the basket page. Refreshing the page causes the item to be added again. Attempt to fix this. (This may require some online research.)

Part 4 [20%]

Improvements are needed to the presentation and usability of the shop.

The price of the product and its description should be shown on the **`viewProduct`** page.

There should be consistent page layout with a common header (using **`<jsp:include>`** and CSS).

Better navigation between pages is required (e.g. the basket page should provide an option to return to the product catalogue and the user should be able to view the basket without adding any items to it).

Make appropriate improvements so the shop is more user-friendly. Also add more products to the database with images that are larger than their thumbnails and add some product descriptions in place of the “insert description” currently used in the database.

Part 5 [30%]

Add a facility to allow for the searching for all products by a particular artist and for all products whose titles contain a particular substring.

The artist search should find only exact matches and should be case-significant. The user is expected to type the name that (s)he wishes to search for; you should not use a drop-down menu of artist's names.

The substring supplied for the second search must contain at least two characters; single-character inputs should be rejected. This search should ***not*** be case-significant.

Two separate search options should be provided – do not attempt to combine them. You may perform the searches by using SQL queries with WHERE clauses or by retrieving all products from the database and using Java to determine which match the search criteria. If you use the first approach must use prepared statements to perform the searches.

Appropriate user-friendly messages should be displayed if a search produces no results.