2018

JavaFX MVC Shop

DOCUMENTATION

JOSHUA WARBURTON, STEPHEN DUNNE, MATT DIOSSY

BT | DMU

Table of Contents

Application Overview

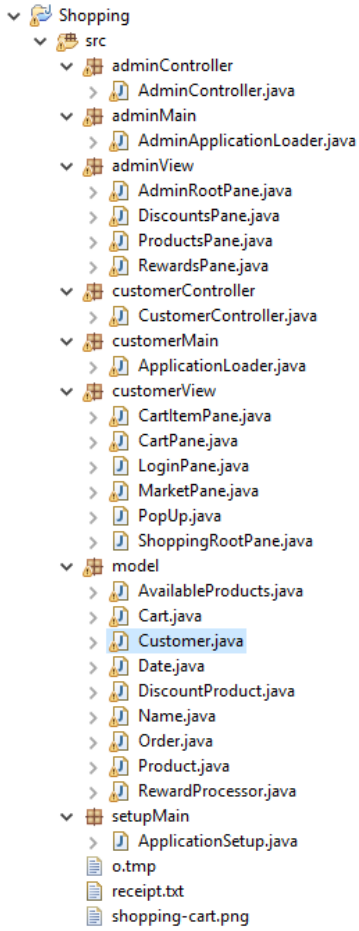


Figure 1- On overview of my Java Project

My two applications are split into several packages. The customer application files are found within “customerController”, “customerMain” and “customerView”, the customer program also interacts with classes within the package “model”.

The admin application files are found within “adminController”, “adminMain” and “adminView”, the admin also interacts with classes with the package “model”.

The customer program and the admin program need an intermediator file for the admin program to add and remove products, or discounted products, from the customers “Market”. The file that the both applications read and write to is “o.tmp” and it is found within the “src” folder.

The source folder also holds a few other files namely “receipt.txt” and “shopping-cart.png”. The receipt file is created by the application if not found upon the user submitting their cart within the customer program. The customer program will then write the contents of their cart as well as other relevant information to this file, the file is overwritten if the customer was to submit their cart again. The png in the src folder is used as an icon within the customer program to quickly draw the users eye to how many products they have in their cart and is a familiar sight that makes the program natural to use without having to understand or work anything out.

Customer program

The customer program launches to a screen asking the user to enter their first and last name, this is added to their customer information. The customer has the option to add items to their basket from the "Market" tab, the customer does so by clicking on a product, or discounted product, and pressing the "Add to Cart" button. Not only does this add the product to the cart, displayed visually in the "Cart" tab, but it also increments the counter in the top right to display how many items the customer already has in the cart.

Once items are in the cart the customer can manipulate how many of each product they would like to order, they do this by pressing either the "+" or "-" button on the respective item in the "Cart" tab. If the customer would like to remove a product completely they need only reduce the number of that product ordered to 0, this removes the product and leaves in its place the message "Removed" to inform the customer that this is the case. Once the customer is happy with their order they can change the delivery date, by default this is set as 3 days from the day that the application was opened. Once the customer is ready to place their order they may press the "Buy Now" button on the cart page, this creates a receipt for the user in the src folder displaying relevant information about their order.

Saving and Loading

It is of note that the customer may close and re-open the customer program at any time and upon navigating to the cart tab will find that their chosen products are still there in the quantities they had chosen before closing the program. The cart is saved to the same file as the available products, discounted products and reward scheme products in o.tmp.

Reward Scheme

The total reward points for a customer will accumulate after orders, the total points can be found in the customers receipt after submitting their cart.

Details

The customer program is run from the file "ApplicationLoader.java" within "customerMain". The application loader initiates the "ShoppingRootPane", the pane responsible for holding all other panes the application uses, and "Customer", a class within the model given to us responsible for holding customer information and using these to create the customers controller "CustomerController.java" within the "customerController" package.

Admin program

The admin program launches to the first tab in the application titled "Add/Remove" here the user can add a product by entering a "Product ID", "Description" and "Price (pennies)" and clicking the "Add Product" button or remove a product (or discounted product) by entering only a "Product ID" and clicking the "Remove (Product ID)" button.

If the admin wishes to apply, or remove, a discount to a product they need to navigate to the "Discounts" tab displayed at the top, here they enter a product ID and the products new discount percentage, this will update the table and discounted products will move to the table on the left to easily differentiate them. To remove a discount the admin needs only set the discount percentage to 0 and the product will move back to the table on the left and lose its discount.

Reward Scheme

The reward scheme tab houses a single list, this is a list of all products that are in the reward shcheme, the admin may add a product to the reward scheme by entering its “Product ID” into the field and clicking the “Add to Scheme” button. This will then show that product on this list.

WARNING

Once a product ID is added to the reward scheme it cannot be removed, this is because in the implementation of the “RewardProcessor” class the list of products are stored in a “Hash Set” this means the list cannot be searched to find a product and remove it. I tried to edit the given classes in the model as little as possible.

Details

The admin application is run from “AdminApplicationLoader.java” within “adminMain”. Similar to the customer’s application loader the admin’s application loader initiates its root pane “AdminRootPane” found in the “adminView” package however it does not pass a customer object as the customer class, found in the “model” package, is not used within the admin program.

Setup program (Optional)

The setup program is used to initiate the “o.tmp” file correctly within src, before I send this project to be submitted the last thing I would have done is run the “ApplicationSetup.java” file within the “setupMain” package. ApplicationSetup can also be used to reset all the products, discounted products, rewarded products and the customers cart if needed as it overwrites the o.tmp file.

This file is optional as both the application and the customer programs will work file if the o.tmp file is included before the application is first run and the user has no use in resetting all shared variables.

MVC Design

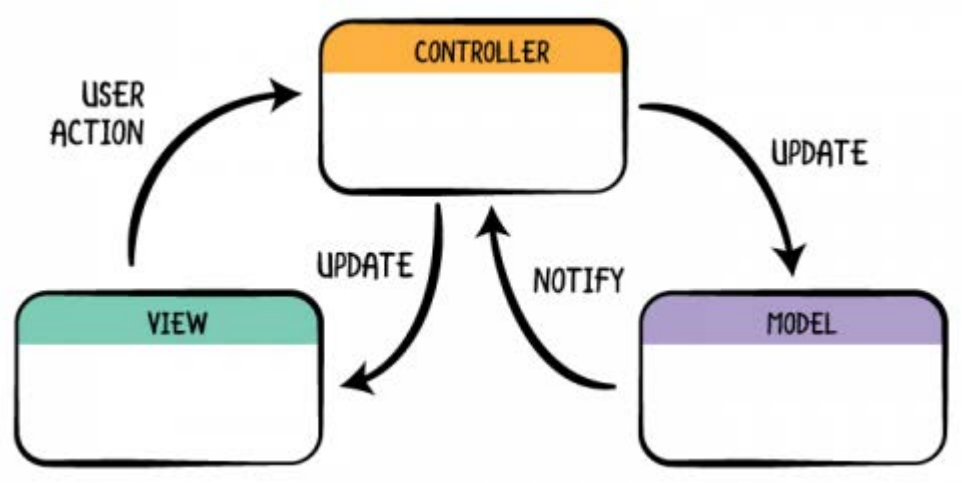


Figure 2- The diagram I used to ensure I was keeping to a MVC design (https://www.raywenderlich.com/132662/mvc-in-ios-a-modern-approach)

Throughout the entirety of my project I strived to use the MVC Design and view Decomposition practice enforced with few exceptions.

From the separation of my classes into view, model and controller packages it can be seen visually what classes are sperate from which and the above diagram shows how these classes interact with each other.
The customer controller (CustomerController) updates the model (all classes within the model package) and customer view (ShoppingRootPane), whilst accepting user action from the view and notifications from the model.
The admin controller (AdminController) updates the model and admin view (AdminRootPane), whilst accepting user action from the view and notifications from the model.

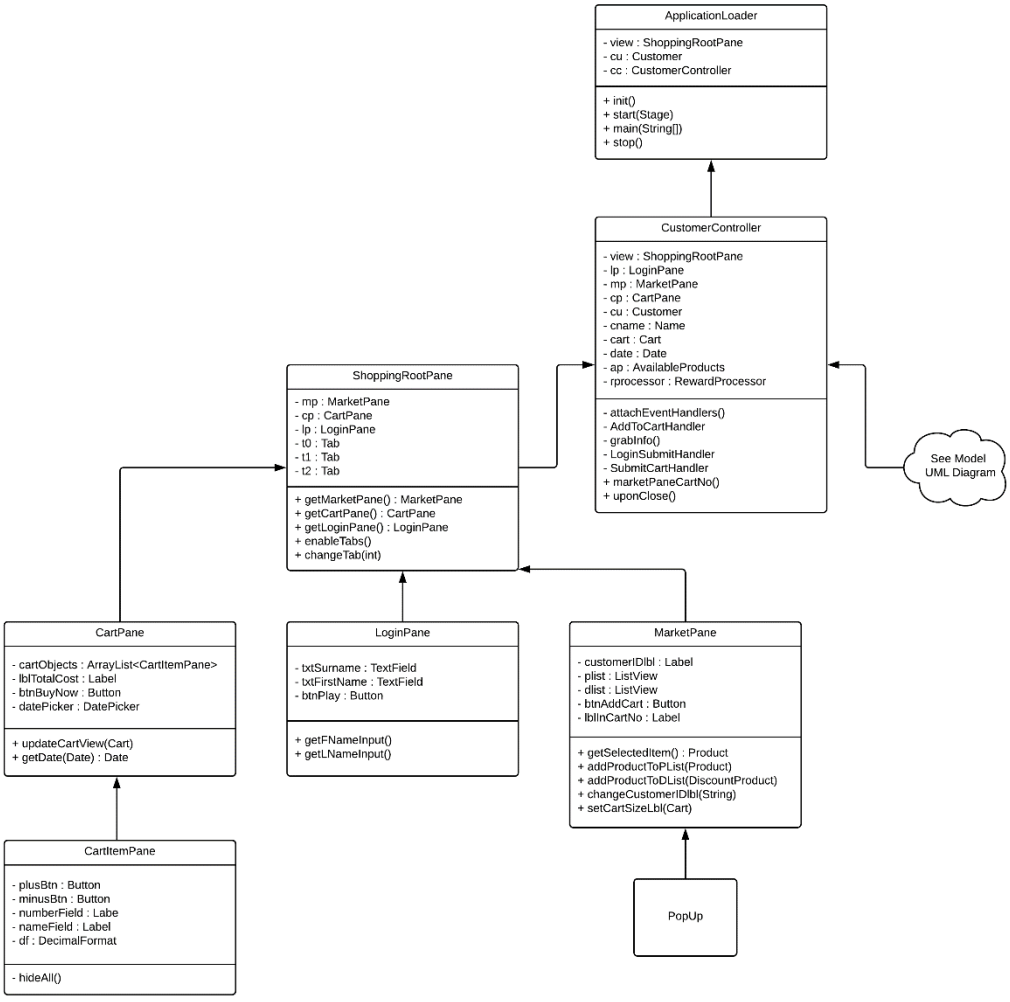
Exceptions

Small exceptions can be found in the “CartItemPane” (Essentially the counters on the Cart Pane) as I wanted these objects to be standalone as they are come and go from view and it only modifies the model in a small and easily understandable way. For this reason, the buttons in the CartItemPane’s event handlers are within itself.

The pop-up window after the customer submits its cart also houses its own handler for the same reasons as above, however it does not have any interaction with the model. The pop-up also appears if the user enters a date that is before the current date.

UML Diagrams (Lucid Chart)

Customer UML Diagram



Admin UML Diagram

o.tmp UML Diagram