Assignment 3: Architecture patterns and styles

Following up from the architecture patterns class activity where you attempted to design a software architecture for the Cash Register application using different architecture styles. In this assignment the MVC design / architecture pattern will be leveraged for the design of a Cash Register interface.

Submission Instructions

Do all your work in a GitHub repository and submit in Canvas the link to the repository.

Background Info on the MVC Pattern

The MVC pattern has always been a misunderstood architectural pattern that is implemented in 2 basic approaches. Fundamentally the pattern leverages an interaction pattern as shown in Figure 1. The differences are primarily on how the update to the View is implemented in the 2 approaches and their implied dependencies.

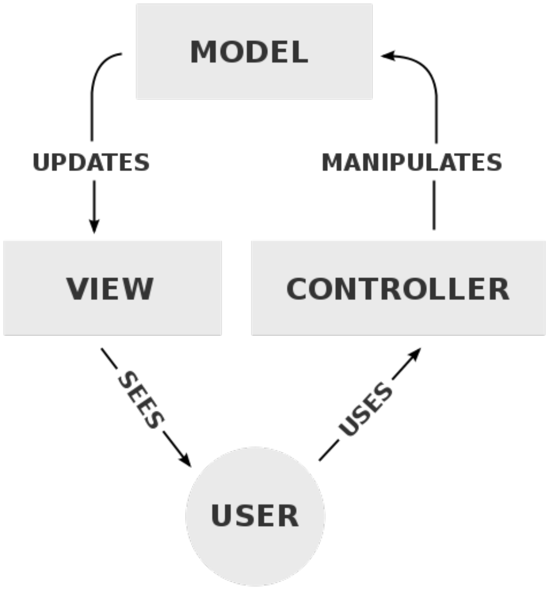

Figure 1. MVC Interaction model

In some examples the updates are direct calls to the View operators (Dependency from Model to View) and on other examples the Observer pattern is implemented creating a stronger dependency from the View to the Model.

Dependencies in the MVC pattern

The ideal dependency that one wants to achieve in the MVC pattern is from the View and Controller to the Model. The Controller already has a natural dependency from the Controller to the model as the Controller leverages the operators in the Model to update the state of the Model.

The Cash Register Application

Going back to the Cash Register Requirements one can define the following significant components: *CashRegister*, *Display*, *Keyboard*, *TicketPrinter*, *Scanner*, and *ProductDB*. The responsibilities and operators for each module are listed below:

CashRegister	
Looks up the price and name of a product based on a UPC	
setCurrentProductUPC (UPCCode)	Sets the UPC code for the current scanned product.
getCurrentProductInfo: Product	Gets the product information for the latest scanned product.

Display	
Is a graphical display of scanned or manually entered item name and price	
displayText (text)	Displays some text on the screen.

TicketPrinter	
Prints on paper the scanned or manually entered item name and price	
displayText (text)	Prints some text on the paper.

Keyboard	
Manual input of a UPC code.	
setUPCCode (UPCCode)	Saves the UPC code entered by cashier

Scanner	
Capture of UPC code using a bar scanner	
scannedUPCCode (UPCCode)	Captures the UPC code read by the scanner.

ProductDB	
Persistence storage of the products in a store	
GetProductInfo (UPCCode): Product	Gets the product information for the product with the UPC code equal to *UPCCode*.

Exercises (For each exercise you should create a separate folder in the GitHub)

1. In this first exercise implement a Cash Register Application using the above components that processes input from the *Keyboard* and/or *Scanner* (Controllers) and outputs the product information on the *Display* and *TicketPrinter* (Views) by interacting with the *CashRegister* (Model). This design should follow a standard interaction pattern where the Controllers depend on the *CashRegister* operators and the *CashRegister* depends on the View operators. In this exercise I will be looking that *CashRegister* calls the operators in *Display* and *TicketPrinter*.
2. Create a View interface that the Display and TicketPrinter will inherent from that contains an operator called *displayProduct(Product)*. Implement this operator in both *Display* and *TicketPrinter* and modify the *CashRegister* component to leverage this operator. This 2nd implementation uses Interfaces to invert the dependency between the Model and View.
3. Modify the original *CashRegister*, *Display* and *TicketPrinter* components from Exercise 1 so that an *Observer* pattern is used where the *Display* and *TicketPrinter* components are Observers to the *CashRegister* components and are notified when the current scanned product's name and price have been updated. This 3rd implementation uses a Subject / Observer pattern to invert the dependency between the Model and View.
4. Comment on the advantages and disadvantages of the 3 approaches.