ASSIGNMENT 3 SOFTWARE DESIGN AND ARCHITECTURES

STUDENT NAME	STUDENT NUMBER
Aryan Singh	100748196
Joshua Ramnaraine	100692194
Frederick Tetteh	100569808

Question 4:

Approach 1:
The original MVC approach used the cash register as a model object, the *Display and Ticket Printer* as View objects and the Scanner and keyboard classes as controller objects. The controllers were used to modify and update UPC codes, which were then passed to the model object to update the current scanned UPC code. After the model would the product info with the current UPC code and print the ticket and the name and price of product on the display accordingly. The advantage would be that using the MVC allows a user to be able to support multiple views at once. Furthermore, if in future a new controller needed to be created (other than the keyboard and the scanner), it could be added to the application without the need of changing the model, hence the *Cash Register* class. However the MVC pattern still provides many challenges. Using this approach application to degrade over-time. Frequent updates being made to the model could degrade performance and slow the display of the data to the user.

Approach 2:

In this approach, we created an interface class that contains an operator called *displayProduct(Product)*. This is used to implement the *Display* and *TicketPrinter* components method of outputting product information. An advantage of using this interface is that your view component can access the same method of displaying while being modified individually in each component. For example, in our case the *Display* and *TicketPrinter* components both pass a string to their display text methods. With this approach, we would be using interface components that would become highly dependent on the model components. So in our case, in order to print the scanned items on a ticket, the *displayProduct* has to take in the product and create the string that would then be passed into the *displayText* method in the *TicketPrinter* and then be able to print information on the ticket.

Approach 3:

The Observer pattern is a design pattern in comparison to the MVC which is an architecture style pattern. Both patterns aim to separate computations with the User Interface. One of the main advantages of the observer pattern is that not only are we separating the objects from the model we are trying to simultaneously update multiple objects(views) at once in comparison to the MVC. The MVC aims to show *different kind*s of views for a single model while being modified by a controller, while the observer pattern tries to update *multiple different components* from a single model eliminating the need for a controller. In other words the observer pattern allows multiple dependent views to be updated at once. In the case with the *CashRegister* we see that now we can now update multiple components being the *Display* and the *TicketPrinter* without the need of a controller. One of the main disadvantages of this approach is that observer pattern resulted in a much more complex code than needed because of the added two classes to mitigate use of the controller in order to update all the required components.