Name: Senelli Jinadasa

This program solves a problem in the consumer world. The problem that this program solve is ordering from a menu in a Tea Shop; this program walks the user through the process of ordering from the menu items. This program welcomes the user and provides a menu of a Tea Shop. This program is an ordering process for the Tea Shop. Once the menu is provided for the user, it will prompt the user to select a number of an item on the menu. After user enters a choice of tea, it will ask user is he/she wants to order more items. Once the user selects an item and he/she does not want to add more items to the order, it will display the items that the user ordered on to the screen, and it will write a file with the customer receipt and append the order to another file that keeps record of the sales made so far. A list of dictionaries with the menu items and their prices is declared in the beginning of the program. A model view controller model was implemented to take the input of the user, create an order list, and display the order list to the user. A model object and a view object are initialized inside of the controller class. The controller takes in user input (selected menu item) from the user inside a while loop while the user wants to continue on ordering more items (while user enters ‘y’). The user selection is passed into a function in the model function where it is appended on to a list. This list is returned back to the control class. Then the list is passed to a function in the view class to be displayed to the user to show what they have ordered. List of filenames are declared inside of the control function. Then threads are created, and the thread is initialized with the function and order list and the filename. The function the thread was initialized with takes the order list as a parameter, and the function writes the order list on to a file. In the main program, reactive programming that incorporates asynchronization is used to display messages to the user about certain items in the list. To do this a function is called to check which items are above a certain price, and it is displayed on to the user (in this case any item that contains milk is above $3.99, so the items containing milk are displayed to give a warning to the user). The objective of this program is to get input from the user to create an order list out of the standard menu item, and then display the order list back to the user. Also, generate a file with a list of items ordered by the user into a text file and also append the list into another file to keep track of all the items sold. The user interacts with the program to let the program know which items they are ordering and this is how their order list is generated. Concurrency is applied to this program to write two files at the same time. The file with the customer copy of the list of items that they ordered and the file containing all sales records are written at the same time. The customer copy file’s text is overwritten each time the user uses the program while the order lists are appended to the end of the sales record text file. The rx and the threading module are incorporated into the program. The threading module is incorporated into the program to write two files at the same time by separating the tasks into two different threads; they are both started together and ended together. The rx module is used to display messages to the user regarding the items in the menu list. One of the limitations of this program is exception handling. When a user inputs an invalid input, the program displays a message saying the user input in an invalid response and the program stops. To improve on this limitation, there should be better code for checking user input and handling exceptions.

IMPORT Thread from threading module

IMPORT Observable from rx module

IMPORT create from rx module

INITIALIZE list of dictionaries items = [{'name': 'Milk Tea', 'price': 4.00},

{'name': 'Thai Tea', 'price': 4.50}, {'name': 'Jasmine Tea', 'price': 3.00},

{'name': 'Oolong Tea', 'price': 3.50}, {'name': 'Green Tea', 'price': 3.00},

{'name': 'Black Tea', 'price': 3.00}]

class QuoteModel:

overloaded constructor with order\_items passed in as a parameter:

initialize order\_items

FUNCTION get\_items():

returns order\_items

FUNCTION addToList with parameters order\_items, index:

append item from items at the index to order\_items

return order\_items

FUNCTION set\_items with parameter order\_items:

set order\_items = order\_items

class QuoteTerminalView:

FUNCTION show with parameter item:

ITERATE through item:

print items in what customer ordered

FUNCTION show\_items(self):

ITERATE through items:

print items in the menu

FUNCTION selected\_items with parameter item:

return item

class QuoteTerminalController:

Constructor:

Initialize order\_items = []

Initialize model object calling model class QuoteModel(order\_items)

Initialize view object calling view class QuoteTerminalView()

FUNCTION run:

INITIALIZE order\_items = []

INITIALIZE selection = 'y';

WHILE user selection is ‘y’:

GET item from menu user wants to order

IF user selects 1:

set index = 0

ELSE IF user selects 2:

set index = 1

ELSE IF user selects 3:

set index = 2

ELSE IF user selects 4:

set index = 3

ELSE IF user selects 5:

set index = 4

ELSE IF user selects 6:

set index = 5

ELSE:

PRINT 'Invalid choice'

GET selection from user if they want to add another item

INITIALIZE item with model object and call addToList with parameters order\_items,

index

Call show function of view object and pass item as parameter

INITIALIZE a list of filenames = ['customer.txt', 'salesrecord.txt',]

create threads

CREATE threads list with target as writeToFile, args as filename and order functions FOR

filename in filenames

start the threads

join the threads

FUNCTION buy\_teas with parameters observer, scheduler:

ITERATE through the items dictionary:

IF item in the list items has a ‘price’ > 3.99:

Call observer.on\_next function with the parameter item['name']

ELIF item with the 'price' <= 0):

Call observer.on\_error function with the parameter item['name']

Call observer.on\_completed function

FUNCTION writeToFile with parameters filename, order\_items:

display message to the user that items being written to the fime

IF the name of the filename is 'customer.txt':

open file in write mode:

write the order\_items to the file

else:

open file in append mode:

write/append order\_list to the end of the file

FUNCTION main():

INITIALIZE order\_items list

create view object by calling QuoteTerminalView()

call show() function for view object

source = create(buy\_teas)

show the message ‘Following Drink Contains Diary Products’on subscribe event

create control object by calling QuoteTerminalController()

Call run method of control object

CALL main function