UI/Program – Calls Menu.Start()

UI/ConsoleIO

DisplayAllOrders(List<order>, string date) – calls DisplayOrder foreach in the order list

DisplayOrder(Order order, string date) – maybe just this one and then call it repeatedly for display whole list

UI/Menu - Displays menu, depending on response:

Calls Display Workflow, Add Workflow, Edit Workflow, Remove Workflow

Data/TestOrderRespository (implementation of IOrderRepository)

LoadAllOrders() – Returns sample order in list

LoadOrder() – Returns sample order as single order

Save() - does not write

Remove() – does not remove

Data/ProdOrderRepository (implementation of IOrderRepository)

LoadAll() – Reads all orders from a file for a particular date

Load() – Read one order from prod file

Save() – Save to prod file, need two different ways, append (possibly with file create) and delete/append

Remove() – Removes one order from a file

Data/TestTaxRepository (implementation of ITaxRepository)

Load() – returns sample tax data

Data/ProdTaxRepository (implementation of ITaxRepository)

Load() – read from tax file, returns LoadTaxfileResponse (list of taxes, success, message if fail)

Data/TestProductsRepository (implementation of IProductsRepository)

Load() – returns sample tax data

Data/ProdProductsRepository (implementation of IProductsRepository)

Load() – read from products file, returns LoadProductsFileResponse (list of taxes, success, message if fail)

BLL/OrderManagerFactory – Selects test or prod, returns an OrderManager feeding in either test or prod data via the constructor

BLL/OrderManager – has constructor that assigns the right repository to a variable(\_orderRepository). Uses that variable to call load() and save() from Test or Prod Order Repository.

Has AddOrder(), EditOrder(), DeleteOrder(), DisplayOrder() each of which calls load and save from \_orderRepository, in between creates and calls the appropriate Rule object

UI/AddOrderWorkflow – Execute()

Calls OrderManagerFactory to create an OrderManager object (orderManager)

Create an order object to store user input temporarily

Get order data from user (date, customer name, state, product type, area) – need to calculate rest, saving to new order object

Call ConsoleIO.DisplaySingleOrder (send order object)

Ask for save confirmation

Yes: Call OrderManager.AddOrder (which calls load, AddOrderRules, save)

No: Return to menu

UI/EditOrderWorkflow – Execute()

Calls OrderManagerFactory to create an OrderManager object (orderManager)

Create an order object to store user input temporarily – or just fields?

Call orderManager.LoadOrder to get existing order data

Get order data from user (only customer name, state, product type, area can be changed) – display each and let them enter a change if want to (if just press enter don’t update) need to calculate rest, saving to new order object or just fields

Call ConsoleIO.DisplaySingleOrder to show whole revised order

Ask for confirmation

Yes: Call OrderManager.EditOrder (which calls load, EditOrderRules, save)

No: Return to menu

UI/RemoveOrderWorkflow – Execute()

Calls OrderManagerFactory to create an OrderManager object (orderManager)

Create fields to store date and order number

Get order data from user (date and order number)

Call orderManager.LoadOrder(pass date and order number)

Call ConsoleIO.DisplaySingleOrder to show order to be deleted

Ask for confirmation

Yes: Call OrderManager.RemoveOrder (which calls load, RemoveOrderRules, save)

No: Return to menu

UI/DisplayOrdersWorkflow – Execute()

Calls OrderManagerFactory to create an OrderManager object (orderManager)

Create a string variable store user input temporarily

Call orderManager to read list of all orders for that date and return the list

Get order date from user Call ConsoleIO.DisplayAllOrders (sending list) to show all orders for that date

Create these classes with properties:

Models/Order: all of the fields in the orders file

Models/TaxByState: three fields in the tax file

Models/ProductTypes: three fields in the products file

Models/Interfaces/IOrderRepository – LoadOrder(string orderDate, string orderNumber), LoadAllOrders(string orderDate), SaveOrder(Order order)

Models/Interfaces/IProductRespository – LoadProducts)

Models/Interfaces/ITaxRepository – LoadTaxes()

Models/Responses –

Response with Success bool & String Message

LoadAllOrdersResponse with List<Order>, SingleOrder :Response

ProductsResponse

TaxesResponse

I think these can just be in the workflow since the rules are the same for each workflow:

BLL/Rules/AddOrderRule

NewOrder() – Checks if the new order meets the criteria, if so assigns to order fields (both response and order fields??), returns a response (includes success and other fields) back to OrderManager

BLL/Rules/EditOrderRule

EditOrder() – Checks if the edited order meets the criteria, if so assigns to order fields (both response and order fields??), returns a response (includes success and other fields) back to OrderManager

BLL/Rules/DeleteOrderRule

DelectOrder() – Checks if the order to remove meets the criteria, if so assigns to order fields (both response and order fields??), returns a response (includes success and other fields) back to OrderManager

Display Accounts:

Calls DisplayOrdersWorkflow

Creates an OrderManager by calling ordermanagerfactory.create()

Gets order date from user

Creates a LoadAllOrderResponse and assigns it to orderManager.DisplayOrders(orderdate)

orderManager.DisplayOrders creates a LoadAllOrdersResponse and assigns it to \_ordermanager.loadallorders(orderdate) (this is actually calling the loadallorders in the test respository)

Gets back a LoadAllOrderResponse which includes the list of orders, order date, success

Calls ConsoleIO.DisplayAllOrders()

Deposit (from SGBank - delete):

depositworkflow: reads input from user, calls accountmanager.deposit

accountmanager.deposit: Calls \_accountRepository.LoadAccount to check if valid, then calls DepositRule.deposit

depositRule.Deposit: checks that it meets criteria, assigns info to response, returns response to accountmanager.deposit

accountmanager.deposit: If successful, calls \_accountRepository.SaveAccount(new account), returns response to deposit workflow

depositworkflow: Displays response account info to user