Seneca College

Applied Arts & Technology

SCHOOL OF COMPUTER STUDIES

Workshop #4

INSTRUCTIONS

- *This workshop must be completed individually without any outside collaboration. All work must be your own. Copying or reproducing the work done by others (in part or in full) or letting others to copy or reproduce your own work is subject to significant grade reduction or getting no grade at all and/or being treated as academic dishonesty under the College's Academic Dishonesty Policy.*

- *This is in lab workshop part, and you are required to complete this part during the lab time, unless otherwise specified by your instructor.*

- *Your application must compile and run upon download to receive any mark.*

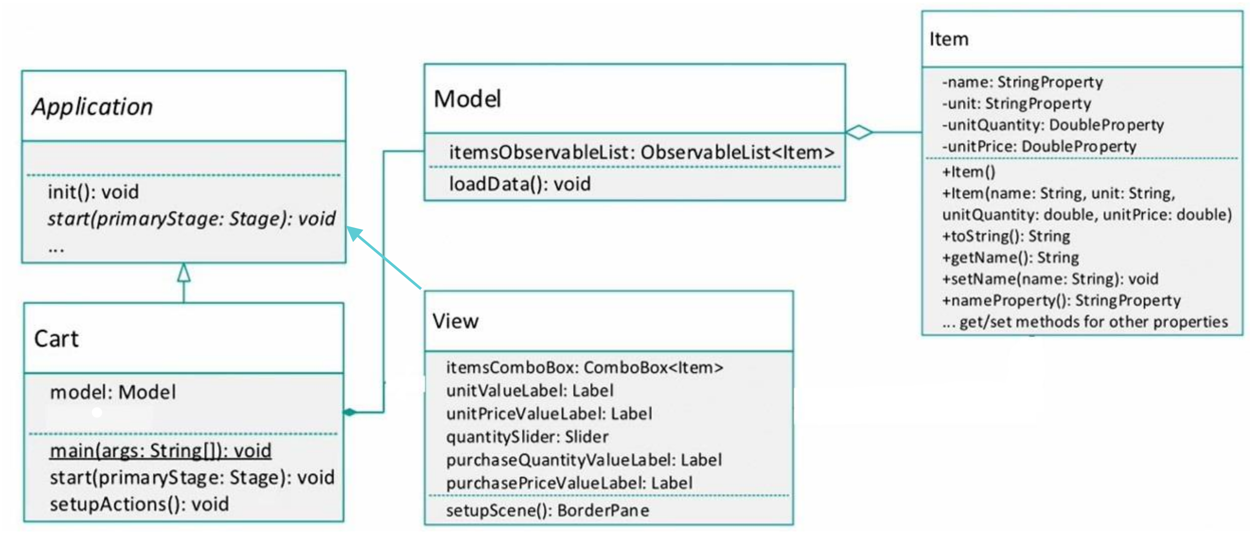
- *To submit the workshop, please follow the Submission Guideline provided at the end of this document.*

- *You must submit your workshop by the due date. Late submissions policy is specified in the Academic Procedures for Evaluations document available through the class plan on Blackboard.*

Task:

You haven been asked to create an application which is going to help a renowned grocery store.

Part - 1


Application
init(): void
start(primaryStage: Stage): void
...
Cart
model: Model
main(args: String[]): void
start(primaryStage: Stage): void
setupActions(): void
Model
itemsObservableList: ObservableList<Item>
loadData(): void
View
itemsComboBox: ComboBox<Item>
unitValueLabel: Label
unitPriceValueLabel: Label
quantitySlider: Slider
purchaseQuantityValueLabel: Label
purchasePriceValueLabel: Label
setupScene(): BorderPane
Item
-name: StringProperty
-unit: StringProperty
-unitQuantity: DoubleProperty
-unitPrice: DoubleProperty
+Item()
+Item(name: String, unit: String, unitQuantity: double, unitPrice: double)
+toString(): String
+getName(): String
+setName(name: String): void
+nameProperty(): StringProperty
... get/set methods for other properties

Item Class:

- This is our bean/ model class.
- All the private properties are denoted with "-" symbol.
- All the public members/ functions/ behaviors are denoted with "+" symbol.

Model Class:

- The class is responsible to hold the items of the beans as an Observable list.
- loadData() method is supposed to load the data form the first column of comma separated sheet called **ItemsMaster.csv (**the file is uploaded on the BB with this workshop).
- To read the data from the file you can use BufferedReader with the combination of FileReader like

```
BufferedReader br = new BufferedReader(new FileReader("ItemsMaser.csv"))
```

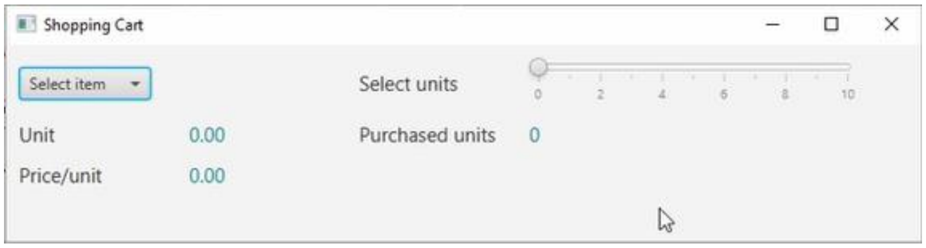
View Class:

- This class is responsible for showing the GUI parts. (For you the view means to use the FXML designed front end).

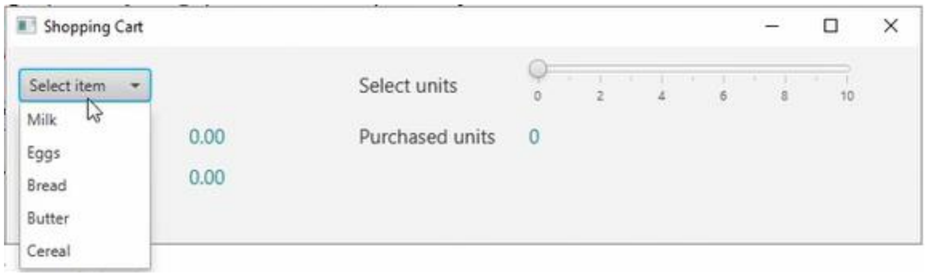
- You are required to use a ComboBox<T> javaFx class and you can learn about it more here.

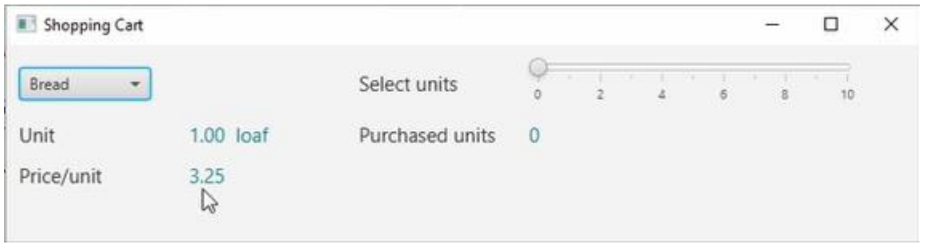
Cart Class:

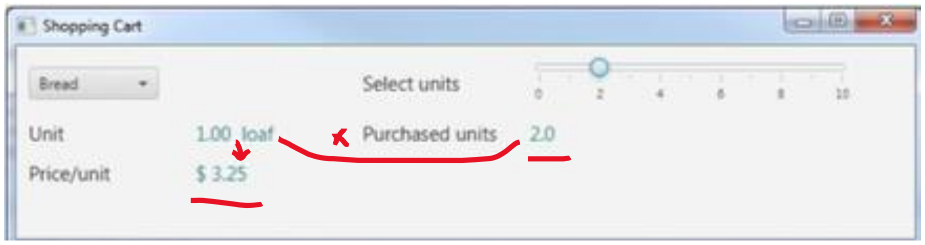
- This class along with Application is responsible to run the project.
- Your cart class
    - **setupAction Method** is responsible for
        - binding `itemsComboBox<T>` with data.
        - Binding purchase units label with slider value using Fluent API
        - Binding `unitValueLabel` and `unitPriceValueLabel` with the respective properties in the item selected in the `itemsComboBox`.
            - In order to achieve the last binding you have to create an ObjectBinding<T>



Once it will run there should not be any item selected in the combo box.







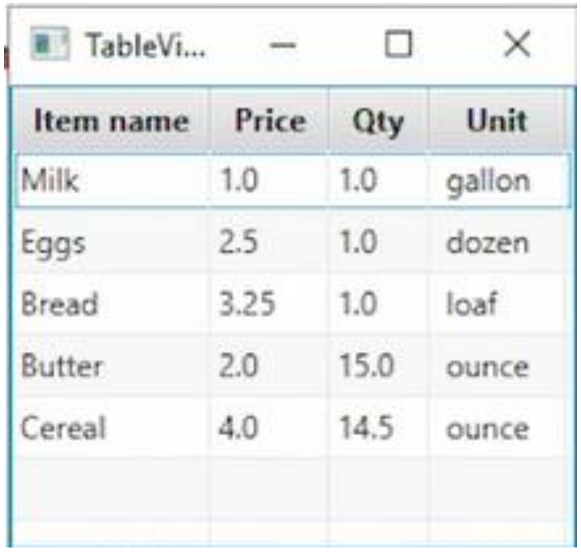
Unit 1.00 loaf

Purchased units 2.0

Price/unit $ 3.25

Part – 2

In this part you need to work with javaFX TableView. Your job is just create a simple design of a TableView and read the data from the **ItemsMaster.csv** file and load them into the respected columns. A sample is provided below.

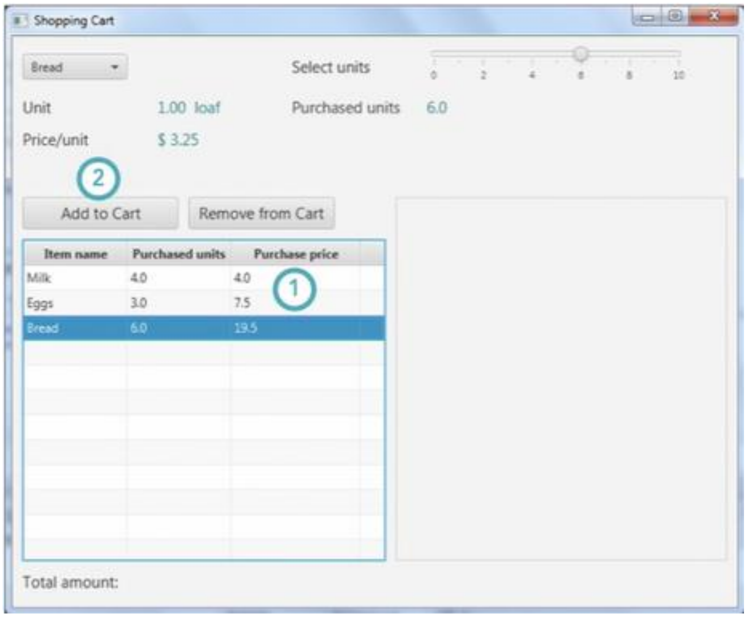


Item name	Price	Qty	Unit
Milk	1.0	1.0	gallon
Eggs	2.5	1.0	dozen
Bread	3.25	1.0	loaf
Butter	2.0	15.0	ounce
Cereal	4.0	14.5	ounce

DIY – Part

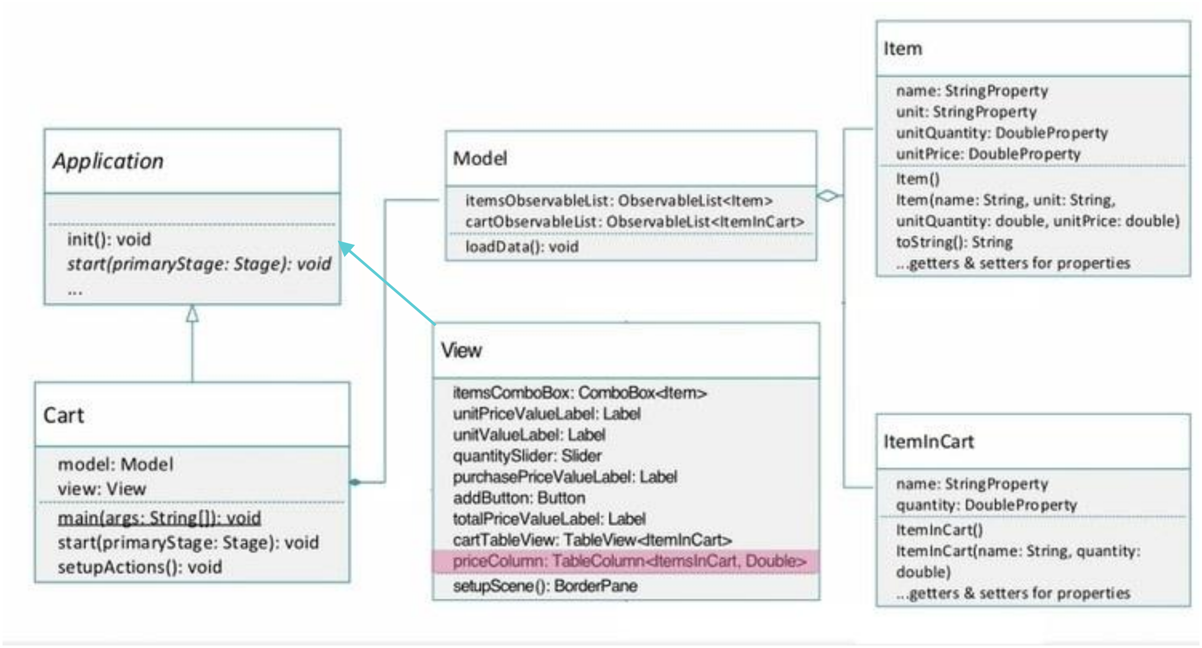
Part – 3

In this part you are going to combine Part - 1 and Part – 2



Item name	Purchased units	Purchase price
Milk	4.0	4.0
Eggs	3.0	7.5
Bread	6.0	19.5

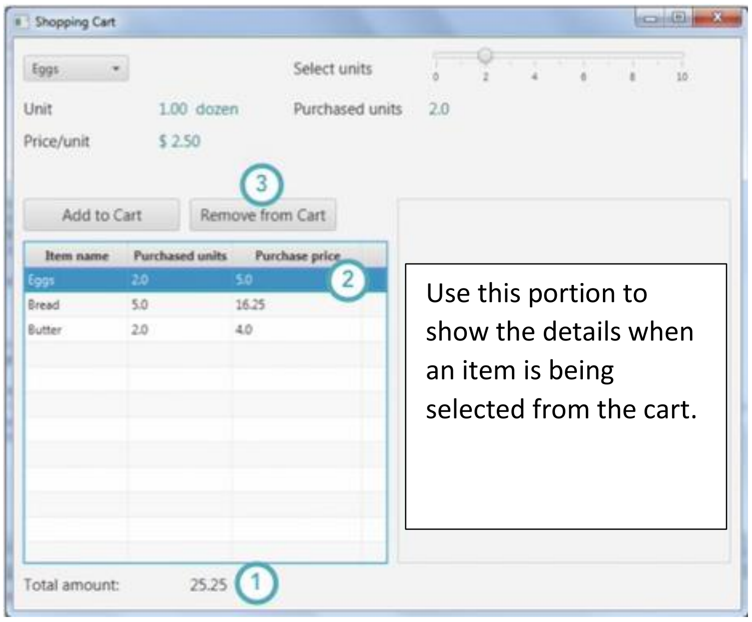
1. TableView with a custom cell value factory for Purchase price column.
2. Event handler Add to Cart button.


Application
init(): void
start(primaryStage: Stage): void
...
Cart
model: Model
view: View
main(args: String[]): void
start(primaryStage: Stage): void
setupActions(): void
Model
itemsObservableList: ObservableList<Item>
cartObservableList: ObservableList<ItemInCart>
loadData(): void
View
itemsComboBox: ComboBox<Item>
unitPriceValueLabel: Label
unitValueLabel: Label
quantitySlider: Slider
purchasePriceValueLabel: Label
addButton: Button
totalPriceValueLabel: Label
cartTableView: TableView<ItemInCart>
priceColumn: TableColumn<ItemsInCart, Double>
setupScene(): BorderPane
Item
name: StringProperty
unit: StringProperty
unitQuantity: DoubleProperty
unitPrice: DoubleProperty
Item()
Item(name: String, unit: String, unitQuantity: double, unitPrice: double)
toString(): String
...getters & setters for properties
ItemInCart
name: StringProperty
quantity: DoubleProperty
ItemInCart()
ItemInCart(name: String, quantity: double)
...getters & setters for properties

Your **`setupAction()`** method/ function is going to need some updates

- Provide cell value factory for Price column in `itemsTableView` by looking up the unit price in `itemsObservableLists’s` item and multiplying it with the `ItemsInCart’s` quantity.
- Set price column’s call-back for factory method
- Event handler for the add button creates new `ItemInCart` with currently selected item’name and purchase qty and adds it to `cartObservableList`.

Part – 4



Item name	Purchased units	Purchase price
Eggs	2.0	5.0
Bread	5.0	16.25
Butter	2.0	4.0

Use this portion to show the details when an item is being selected from the cart.

1. Binding for total amount label with total purchase price
2. `ChangeListener` to change data in the top grid based on the item selected in the `TableView`.
3. Event handler for Remove from Cart button.

Your **setupAction()** method/ function is going to need final updates

- `totalBinding` to add all cell values in the price column of table view.
- bind `totalValueLabel`'s `textProperty` to `totalBinding`.
- listener for item-selection in itemsTableView updates the image, item in combo-box, and qty in quantitySlider it looks for the selected item in the `tableView` in `itemsObservableList`, sets `itemComboBox` to that index, and `quantitySlider` to the `newValue`'s quantity.

- event handler for remove button checks the current selected index if it is >= 0, it removes the item at that index from `cartObservableList`

Workshop Header

/**
Workshop #
Course:*<subject type> - Semester*
Last Name:*<student last name>*
First Name:*<student first name>*
ID:*<student ID>*
Section:*<section name>*
This assignment represents my own work in accordance with Seneca Academic Policy.
Signature
Date:*<submission date>*
**/

Code Submission Criteria:

Please note that you should have:

- Appropriate indentation.
- Proper file structure
- Follow java naming convention.
- Do Not have any debug/ useless code and/ or files in the assignment.

Deliverables and Important Notes:

- Your submission should include:
    - Your codes, your implemented interfaces and the image of the received outputs.

- Late submissions would result in additional 10% penalties for each day or part of it.

Remember that you are encouraged to talk to each other, to the instructor, or to anyone else about any of the workshops, but the final solution may not be copied from any source.

All deliverables are supposed to be uploaded as a combined PDF file on the blackboard once done.