

Anshita Khare

Assignment 1: Shopping Application I

General Documentation

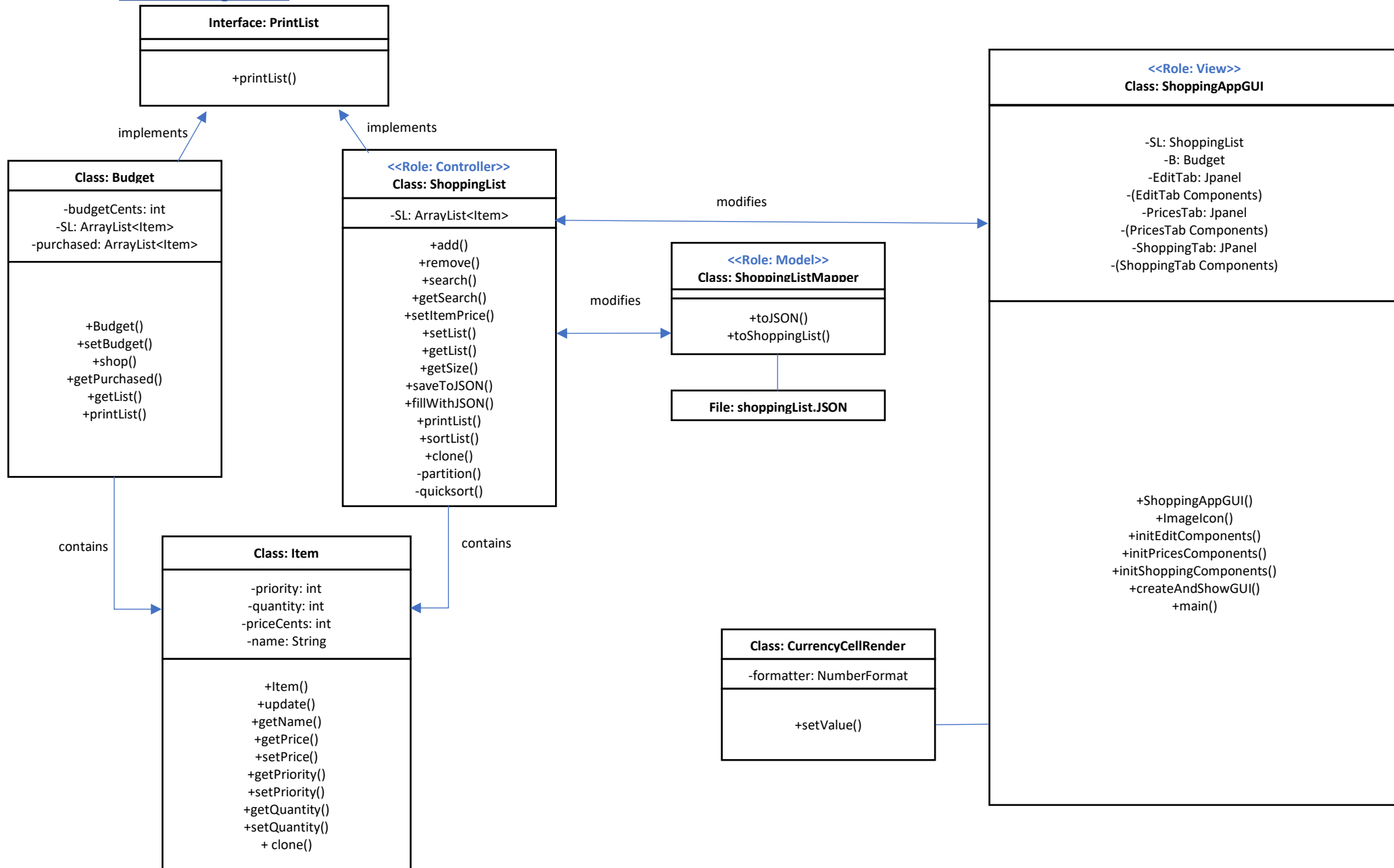
My shopping application uses Java Swing to create a 3-tab window that allows users multiple functionalities. The following tabs enable users to:

- 1.) Edit Items Tab
 - a. Add item by item name, quantity(1-100), a priority(1-5, with 1 being the highest priority)
 - b. Remove selected item
 - c. Save list to .json file named shoppingList.json
 - d. Load saved shopping list from shoppingList.json
- 2.) Set Prices Tab
 - a. Determine the price of each item in the shopping list by dollar and cents by double clicking the price cells for each item in the table
 - i. The input model rejects non numeric characters and appropriately converts excess cents to dollars when needed
 - b. Save prices to shopping list and write that to .json file
- 3.) Go Shopping Tab
 - a. Determine the shopping budget by dollars and cents and go shopping by priority
 - b. Display purchased items
 - c. Display remaining items
 - d. Allows users option to update their shopping list and save the remaining items

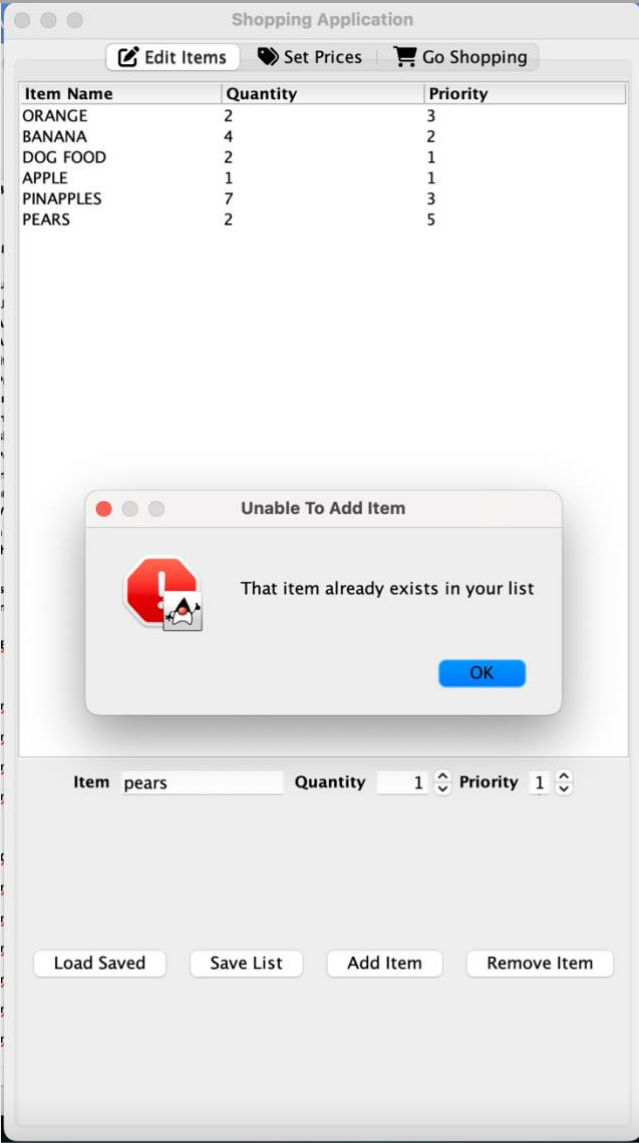
To launch the app; run ShoopingAppGUI.java

Anshita Khare
Assignment 1: Shopping Application I

UML Diagrams

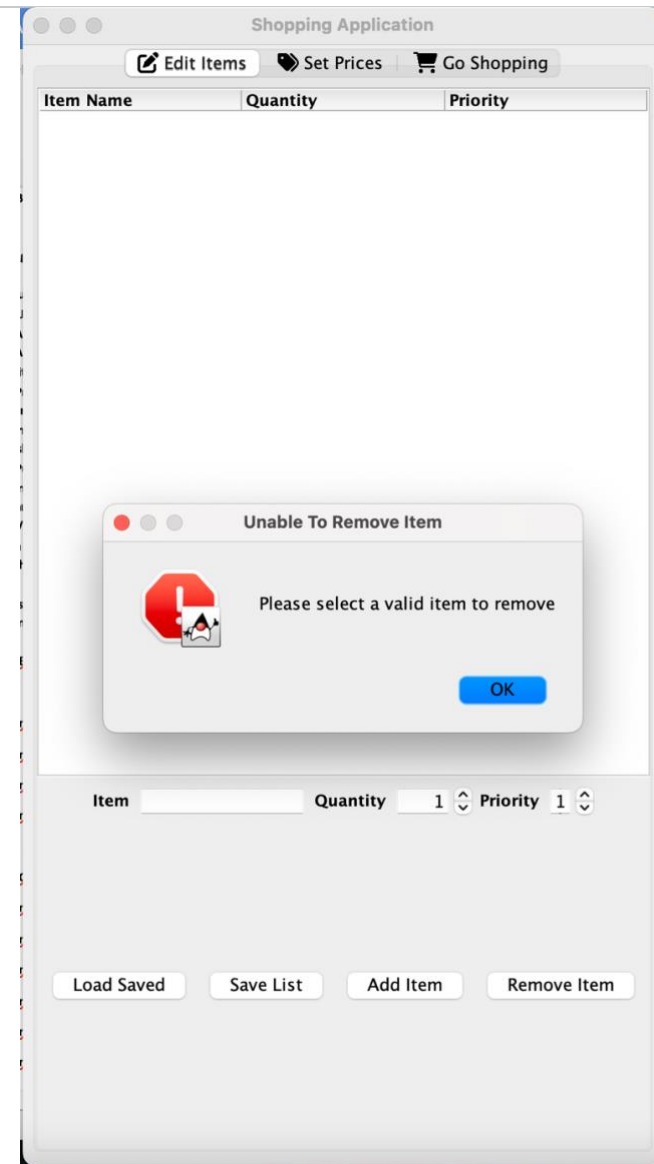


Test Plans

Test Case	Solution																					
<p>Adding Item that already exists</p>	 <p>The screenshot shows a 'Shopping Application' window with three tabs: 'Edit Items', 'Set Prices', and 'Go Shopping'. The 'Edit Items' tab is active, displaying a table with the following data:</p> <table border="1"><thead><tr><th>Item Name</th><th>Quantity</th><th>Priority</th></tr></thead><tbody><tr><td>ORANGE</td><td>2</td><td>3</td></tr><tr><td>BANANA</td><td>4</td><td>2</td></tr><tr><td>DOG FOOD</td><td>2</td><td>1</td></tr><tr><td>APPLE</td><td>1</td><td>1</td></tr><tr><td>PINAPPLES</td><td>7</td><td>3</td></tr><tr><td>PEARS</td><td>2</td><td>5</td></tr></tbody></table> <p>Below the table, there is a modal dialog titled 'Unable To Add Item' with a red stop sign icon and the message 'That item already exists in your list'. An 'OK' button is at the bottom right of the dialog. Below the dialog, there are input fields for 'Item' (containing 'pears'), 'Quantity' (set to 1), and 'Priority' (set to 1). At the bottom of the application window, there are four buttons: 'Load Saved', 'Save List', 'Add Item', and 'Remove Item'.</p>	Item Name	Quantity	Priority	ORANGE	2	3	BANANA	4	2	DOG FOOD	2	1	APPLE	1	1	PINAPPLES	7	3	PEARS	2	5
Item Name	Quantity	Priority																				
ORANGE	2	3																				
BANANA	4	2																				
DOG FOOD	2	1																				
APPLE	1	1																				
PINAPPLES	7	3																				
PEARS	2	5																				

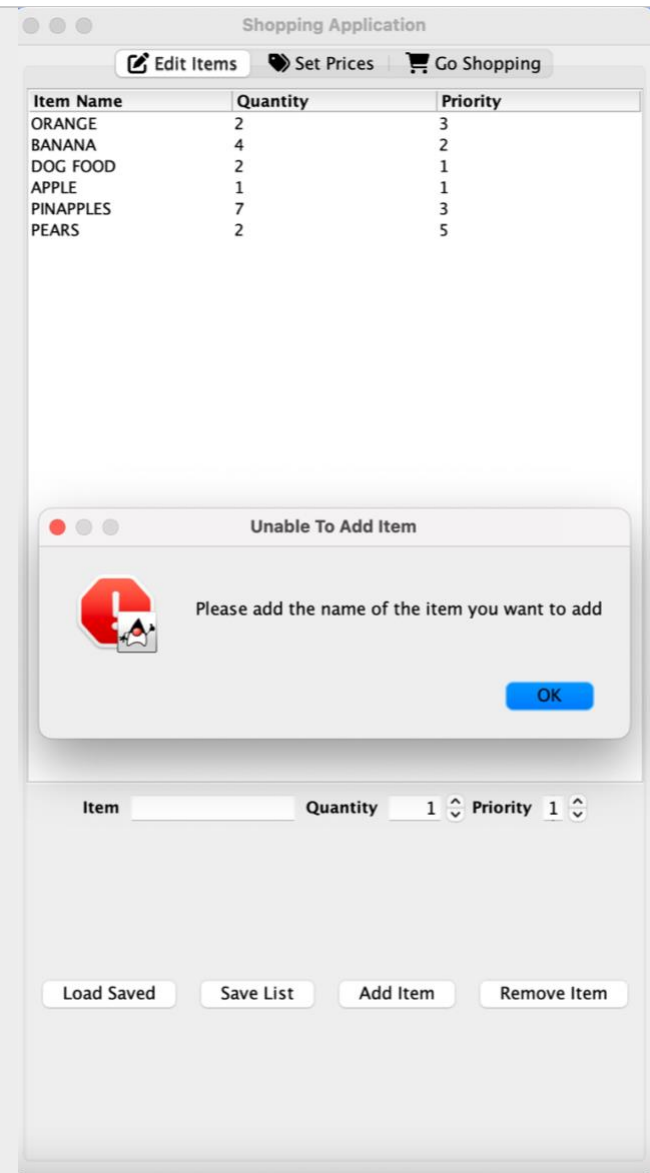
Anshita Khare
Assignment 1: Shopping Application I

Deleting from empty list



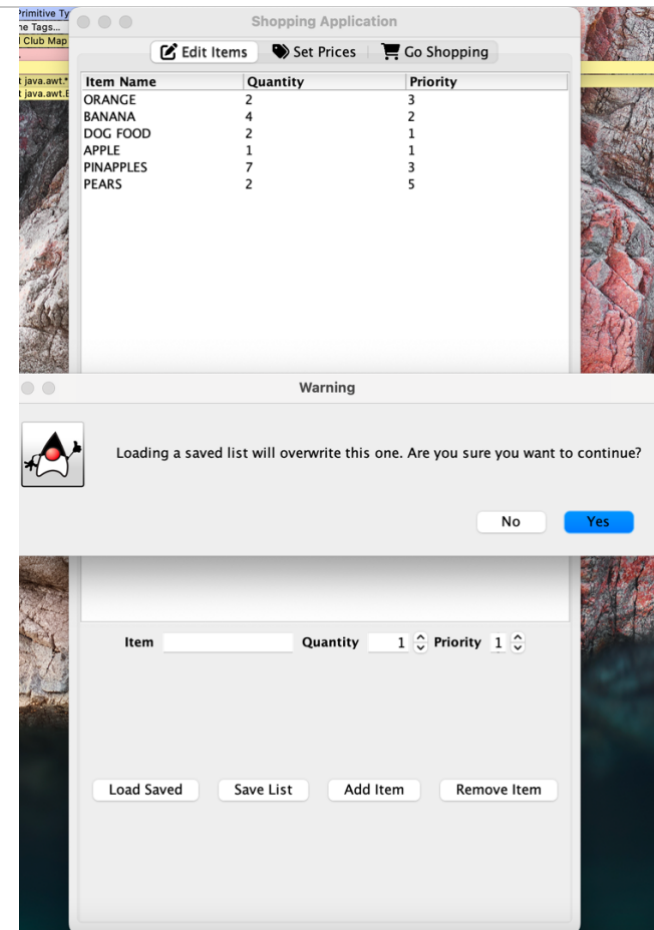
Anshita Khare
Assignment 1: Shopping Application I

Trying to add to shopping list without entering the name of the item (quantity and priority have a default value)



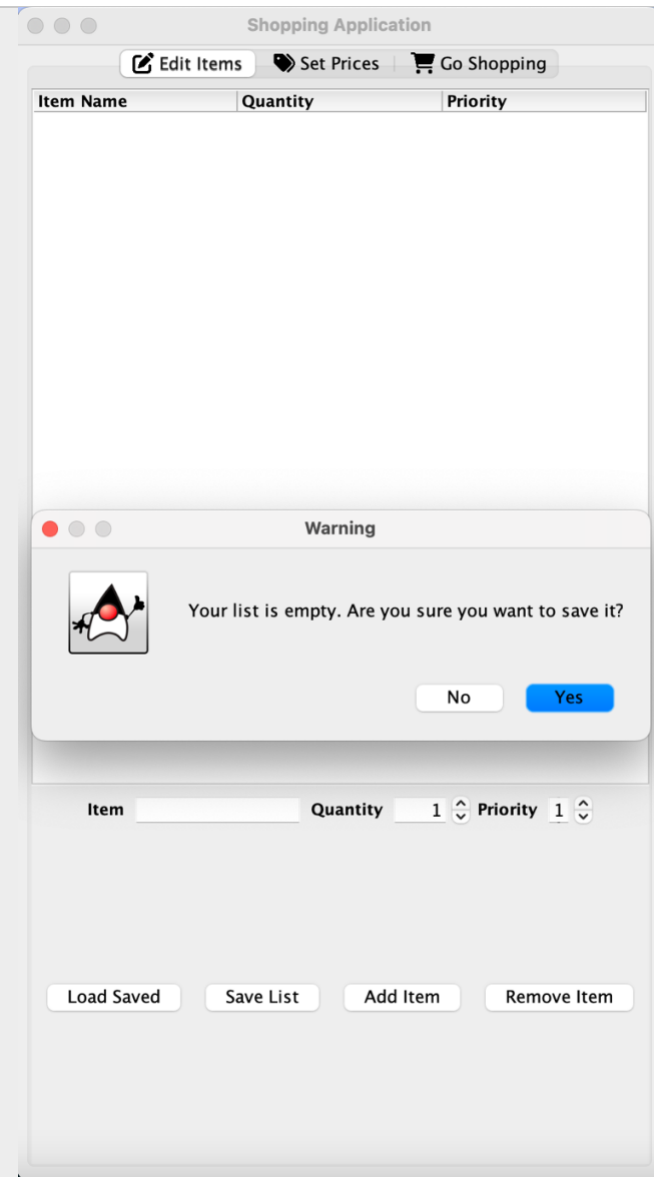
Anshita Khare
Assignment 1: Shopping Application I

When loading multiple times or loading after starting new shopping list, previous information must not be visible after new load & warning must be issued before loading



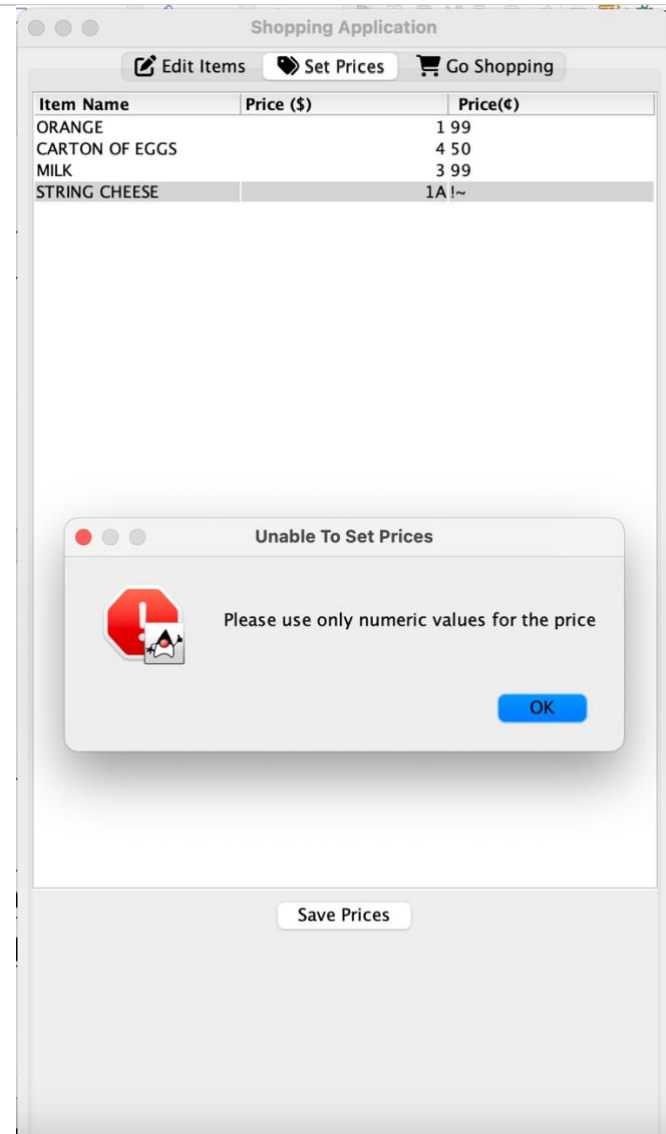
Anshita Khare
Assignment 1: Shopping Application I

Saving an empty list warning



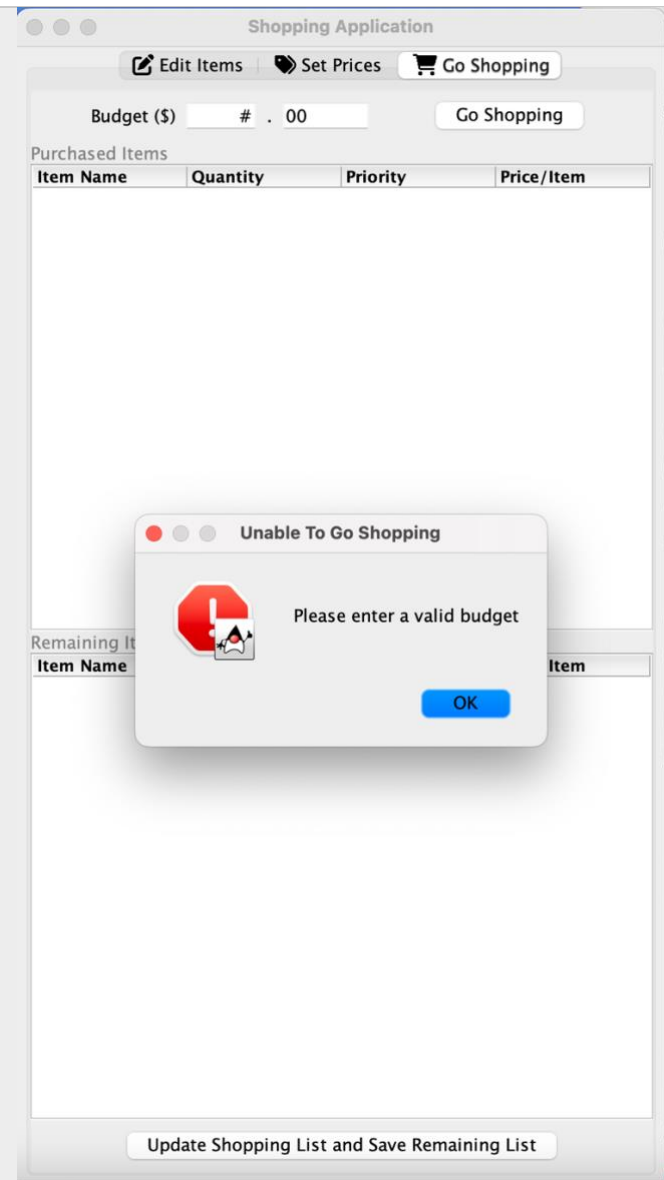
Anshita Khare
Assignment 1: Shopping Application I

Set price with invalid character(abc!#~)



Anshita Khare
Assignment 1: Shopping Application I

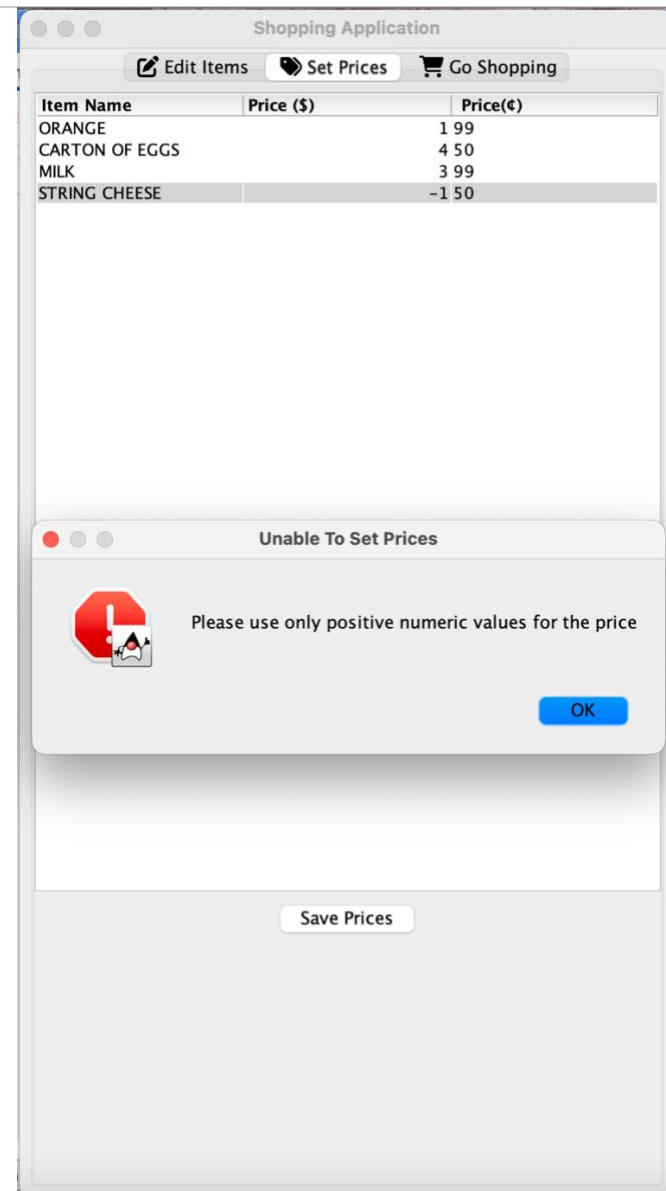
Set budget with invalid characters(abc!#~)



Anshita Khare

Assignment 1: Shopping Application I

**Negative prices, priority, or quantity
priority and quantity use their default values when
invalid entry is inputted**



Pseudo Code for Algorithms

- 1.) Sort the items in the list by order of priority
 - a. Assumptions: 1 is highest priority.

The following algorithm will put the items with the highest priority value at the end of the list. This is because the list will be consumed from the bottom up by the Budget's shop function later.

```
public void sortList() {  
    //calls internal quicksort method  
}  
  
/** A function that partitions the given array list for quick sort  
 * @param currList the sublist that the function works with  
 * @param left the index of the first item in the sublist  
 * @param right the index of the last item in the sublist*/  
private int partition(ArrayList<Item> currList, int left, int right) {  
    //The function creates two variables called biggestIndex and smallestIndex and sets them to left and right, respectively.  
    //Creates a variable called temp to temporarily hold an Item object during the swapping process.  
  
    //The function sets the pivot variable to the priority value of the Item object that is at the midpoint of the sublist represented by  
    left and right.  
  
    //The function enters a loop that will continue until biggestIndex is greater than smallestIndex.
```

Anshita Khare

Assignment 1: Shopping Application I

//Checks whether the priority value of the Item object at biggestIndex is greater than the pivot value.

//If it is, the function increments biggestIndex until it reaches an Item object with a priority value less than or equal to pivot.

//Checks whether the priority value of the Item object at smallestIndex is less than the pivot value.

//If it is, the function decrements smallestIndex until it reaches an Item object with a priority value greater than or equal to pivot.

//If biggestIndex is less than or equal to smallestIndex, the function swaps the Item objects at biggestIndex and smallestIndex in currList, updates biggestIndex and smallestIndex, and continues the loop.

// By the time the loop is finished, the function has moved all Item objects with priority values less than or equal to pivot to the left side of the sublist and all Item objects with priority values greater than pivot to the right side of the sublist.

//The function returns the index of the last item that has a priority value less than or equal to pivot that can be used to further sort the list

/**A recursive quick sort method

* @param currList the list or sublist that is being sorted

* @param lowIndex the index of the first item in the sublist

Anshita Khare

Assignment 1: Shopping Application I

* @param highIndex the index of the last item in the sublist*/

```
private void quickSort(ArrayList<Item> currList, int lowIndex, int highIndex) {
```

```
//Calls a partition function to select a pivot element around which the list is divided
```

```
//Recursively sorts the two resulting sub-lists, the one to the left of the pivot and the one to the right, by calling itself again for each side
```

- 2.) Go shopping, purchase as many items on the list by priority with the given budget; Purchase the highest priority items first then go to the lower priority items and purchase them as needed, minimizing the amount of money left over.
 - a. Assumptions: The list given to the below function has already been sorted and the items with the greatest priority are at the bottom of the list

/**A method that allows for items within the budget to be purchased*/

```
public void shop(){
```

```
//create an empty list called purchased
```

Anshita Khare

Assignment 1: Shopping Application I

//starting from the last item in a list called SL and working backwards so that deletions do not change the index of subsequent items

//if the budget is ever 0, break since we cannot buy any more items

//get the details of the item at the current index i in SL(name, quantity, priority, price)

//calculate the total cost of buying the entire quantity of the item at its price.

//if the budget is greater than or equal to the total cost, buy all of the item's quantity,

//subtract the total cost from the budget, and remove the item from SL.

//if the budget is less than the total cost but greater than or equal to the item's price

//buy as much of the item as possible while the budget is not exhausted by repeatedly subtracting the item's price from the budget until the budget can no longer afford the item and updating the quantity of the item purchased accordingly

//if you ended up purchasing something, create a new item with the same name, priority, and price as the original item but with the quantity set to the quantity purchased, and add it to the purchased list.