

Saif Chowdhury
Tessitore

Tucker Hortman

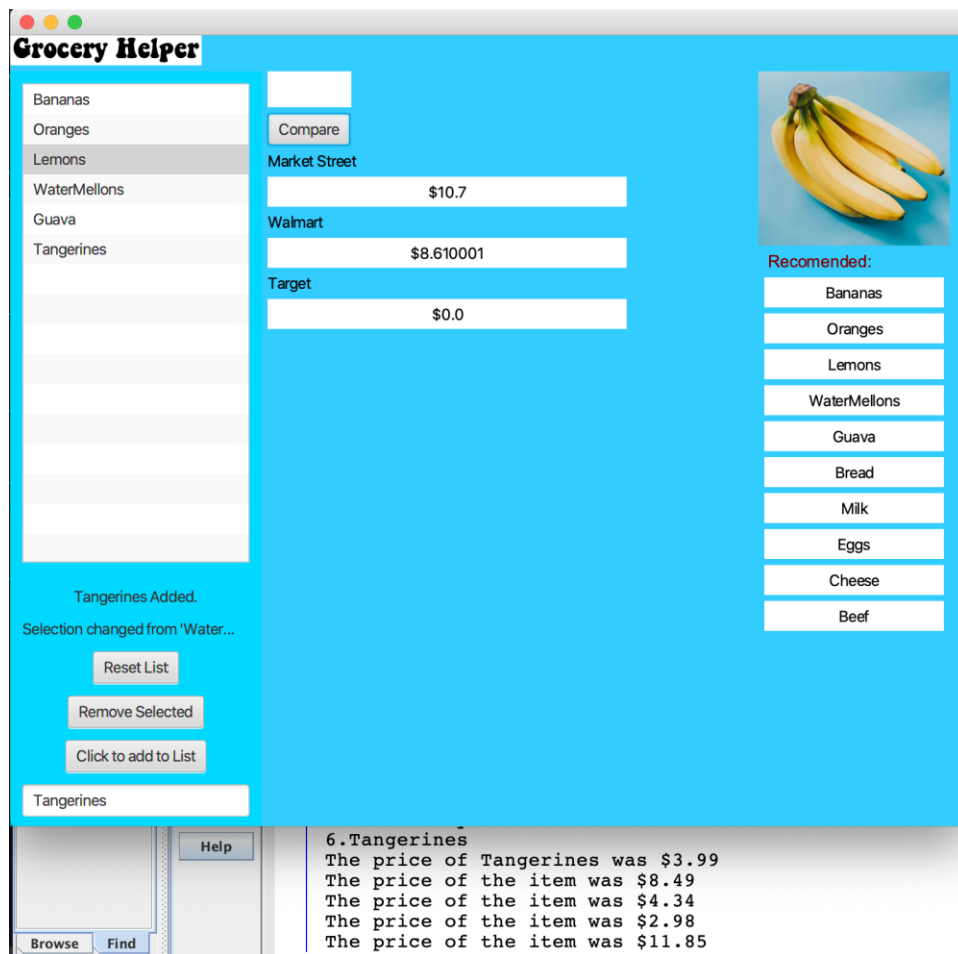
Brian Michelle

Jeff

Shopping Scraper

Hello! This is our Grocery Shopper App!

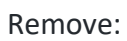
To run this, simply open up Grocery_UI_App in your favorite IDE and hit run. Download this whole folder if you haven't, then run the JavaFX App to use our prototype. If you just want a demonstration of the scrappers, similarly open main.



Currently: You can add, remove, and reset list.

Here's A Picture of These In Action:

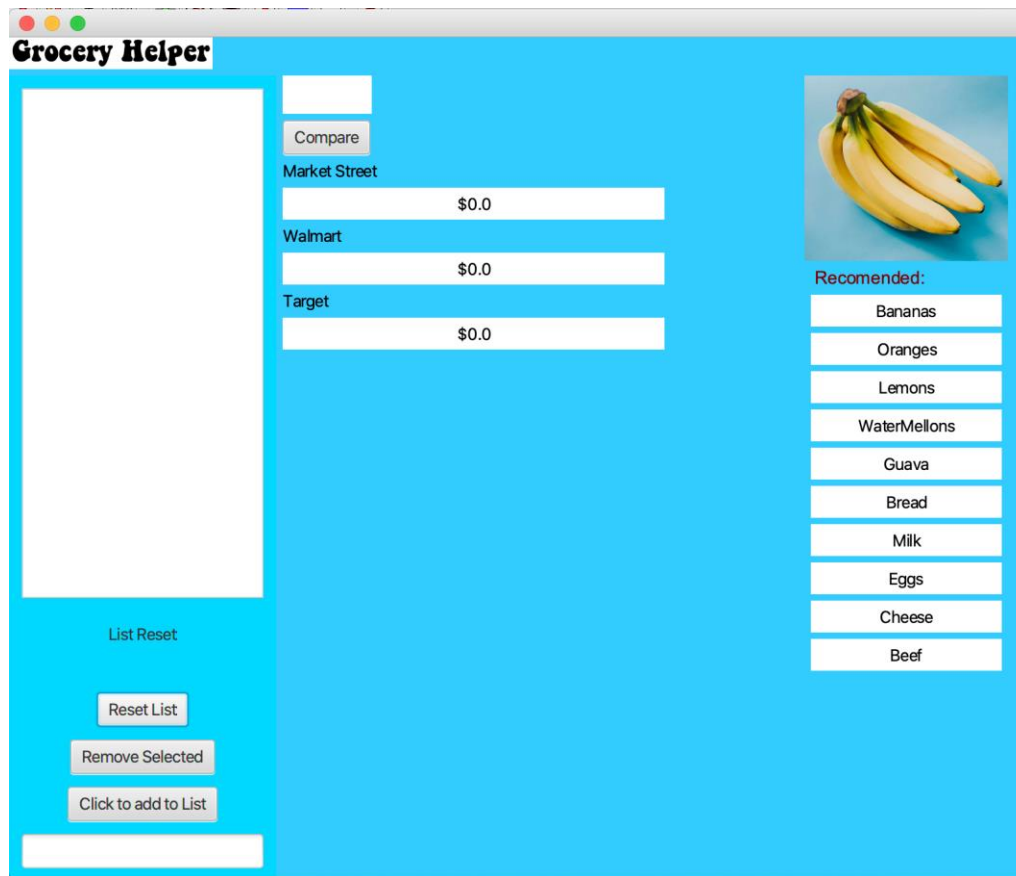
Add:



Remove:

Reset:

Reset:



You can also hit the compare button and our interface will automatically run the scraper programs necessary to find the prices online for MarketStreet and Walmart.

And it will add the totals of each store, if the price was not already found in the database. In which case it will not run the scraper, and will instead add from the database.

Combining these features really improves speed and versatility.

Bellow you will find our HTML Scrapers In Action.

```
1 import java.util.ArrayList;
2
3 public class Main {
4
5     public static void main(String[] args) {
6         Target t_test = new Target("https://www.target.com/p/make-a-size-paper-towels-2-big-rolls-smartly-8482/-/A-75557222");
7         WalmartCrawler w_test = new WalmartCrawler("paper towels");
8         MarketStreet m_test = new MarketStreet("paper towels");
9     }
10
11 }
12
```

Main.java

Compile Messages | jGRASP Messages | Run I/O | Interactions

End

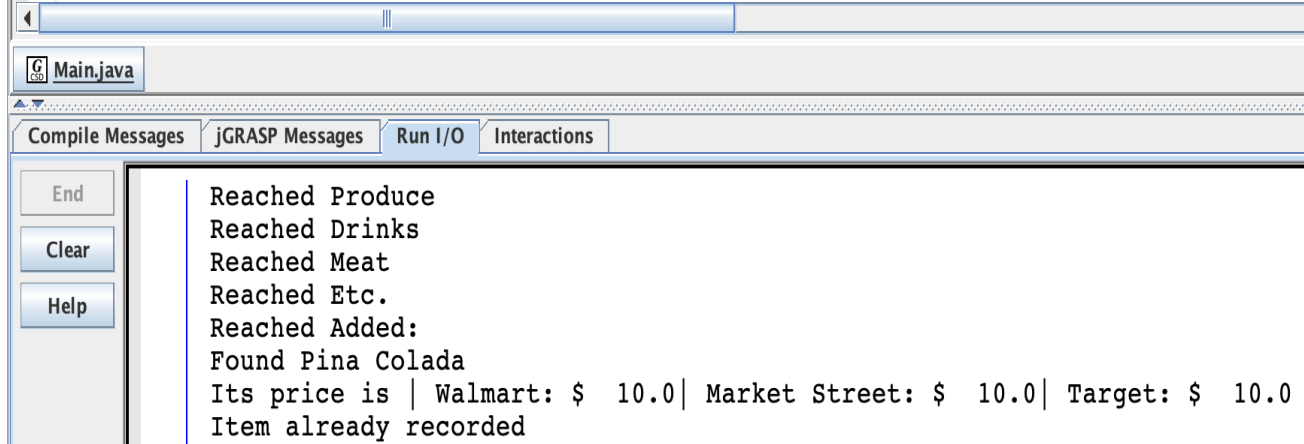
Clear

Help

```
----jGRASP exec: java Main
The price of the item was $0.0
The price of the item was $9.98
The price of the item was $19.88
The price of the item was $14.97
The price of the item was $9.98
The price of the item was $8.42
The price of paper towels was $0.99
```

This is an example of the Database Reader that was later added to store the values of previously searched items:

```
1 import java.util.ArrayList;
2
3 public class Main {
4
5     public static void main(String[] args) {
6         DatabaseReader dbr = new DatabaseReader();
7         dbr.SearchSpecific("Pinapple");
8         DatabaseWriter dbw = new DatabaseWriter("Pina Colada",10.0f,10.0f,10.0f);
9
10    }
11
12 }
13
```



End
Clear
Help

Reached Produce
Reached Drinks
Reached Meat
Reached Etc.
Reached Added:
Found Pina Colada
Its price is | Walmart: \$ 10.0 | Market Street: \$ 10.0 | Target: \$ 10.0
Item already recorded

Heres a Photo of Our Database:

Each of us found one of the 4 Primary lists in this database. We then added the functionality of the growing database in the "Added" Section.

Additionally, at this time, while we get our Target Scraper to work, in order to use Target, a maintainer is required, a second person, who will go into the database.txt, and manually enter in the prices for Target for comparison.

As you can see, the more you use this app, the better it will get. Each time you enter in an item, the software scrapes the internet for its prices. These prices will be recorded, allowing you to have an ever expanding list. Right now the size of the list is limited to an array of size 500.

In the future, we'd hope to expand this by allowing past prices to be updated, and increases the number of stores.

```
database.txt
Produce
Eggs, Walmart, 1.59 , Market, 1.59 ,Target, 1.99
Potatoes, Walmart, 0.41 , Market, 0.69, Target, 2.49
Apples, Walmart, 0.64 , Market, 1.29, Target, 5.39
Bananas, Walmart, 1.39, Market, 0.49, Target, 6.9
Strawberries, Walmart, 3.54, Market, 2.79, Target, 6.9
Blackberries, Walmart, 3.12 , Market, 2.49, Target, 2.39
Lemon, Walmart, 0.48 , Market, 0.33, Target, 0.55
Avocado, Walmart, 0.5, Market, 0.59, Target, 6.9
Raspberries, Walmart, 3.24 , Market, 2.99, Target, 3.19

Drinks
Water, Walmart, 3.98, Market, 6.9, Target, .85
Milk, Walmart, 2.87, Market, 6.9, Target, 2.49
Apple Juice, Walmart, 2.68, Market, 6.9, Target, 2.29
Orange Juice, Walmart, 3.58, Market, 6.9, Target, 3.89
Coke, Walmart, 1.82, Market, 6.9, Target, 1.89
Dr. Pepper, Walmart, 1.82, Market, 6.9, Target, 6.9
Sprite, Walmart, 1.82, Market, 6.9, Target, 6.9
Grape Juice, Walmart, 3.26, Market, 6.9, Target, 6.9

Meat
Top Sirloin, Walmart, 13.95, Market, 6.9, Target, 6.9
Chuck Steak, Walmart, 28.90, Market, 6.9, Target, 6.9
Brisket, Walmart, 37.87, Market, 6.9, Target, 6.9
Pork Chops, Walmart, 6.64, Market, 6.9, Target, 6.9
Chicken leg, Walmart, 3.86, Market, 6.9, Target, 6.9
Chicken Breast, Walmart, 6.55, Market, 6.9, Target, 6.9
Tilapia, Walmart, 5.18, Market, 6.9, Target, 6.9
Salmon, Walmart, 8.48, Market, 6.9, Target, 6.9
Tuna, Walmart, 6.84, Market, 6.9, Target, 6.9
Corned Beef, Walmart, 4.88, Market, 6.9, Target, 6.9

Etc.
Ice Cream, Walmart, 6.9, Market, 6.9, Target, 5.99
Cream Cheese, Walmart, 6.9, Market, 6.9, Target, 6.29
sour Cream, Walmart, 6.9, Market, 6.9, Target, 2.49
Yogurt, Walmart, 6.9, Market, 6.9, Target, 0.5
Vegetable Oil, Walmart, 6.9, Market, 6.9, Target, 2.59
Lays Chips, Walmart, 6.9, Market, 6.9, Target, 4.79
Mayonaise, Walmart, 6.9, Market, 6.9, Target, 3.19
Ramen, Walmart, 0.039, Market, 6.9, Target, 6.9

Added:
Tang, Walmart, 2.99, Market, 5.16, Target, 0.0
Donuts, Walmart, 5.99, Market, 3.67, Target, 0.0
Soup, Walmart, 1.99, Market, 1.68, Target, 0.0
Oranges, Walmart, 1.13, Market, 0.7, Target, 0.0
Lemons, Walmart, 0.33, Market, 0.54, Target, 0.0
WaterMellons, Walmart, 0.79, Market, 2.88, Target, 0.0
Guava, Walmart, 1.99, Market, 2.1, Target, 0.0
Bread, Walmart, 2.78, Market, 2.29, Target, 0.0
Pomegranate, Walmart, 0.0, Market, 13.99, Target, 0.0
Tangerines, Walmart, 2.98, Market, 3.99, Target, 0.0
Pina Colada, Walmart, 10.0, Market, 10.0, Target, 10.0
```

We hope you have enjoyed this demonstration of our Grocery Comparison App!

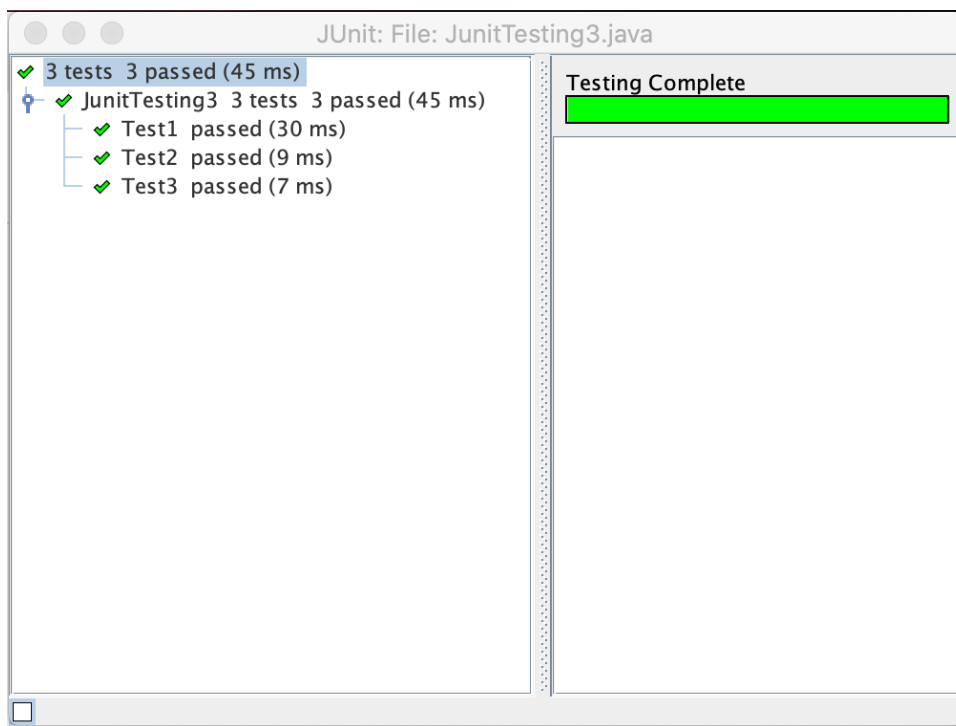
Additionally, here is an implementation of Unit Testing:

```

10
11 @Test public void Test1() {
12     DatabaseReader dbr = new DatabaseReader();
13
14     float a = dbr.getPriceW("Eggs");
15
16     //Test to see if the price exists:
17     assertNotNull(a);
18
19 }
20
21 @Test public void Test2() {
22     DatabaseWriter dbw = new DatabaseWriter("Salami",5.99f,4.28f,6.29f);
23
24     //Check to see if the Database exists
25     assertNotNull(dbw);
26
27 }
28
29 @Test public void Test3() {
30     DatabaseReader dbr = new DatabaseReader();
31
32     float a = dbr.getPriceW("Salami");
33     float b = 5.99f;
34     //Check to see if the Price Written is the One Matched
35     Assert.assertEquals(a,b, 1f); //asserts that a price exists and they are within a range of 1
36
37 }
38
39
40
41

```

As you can see, it passed:



We chose to focus on 3 key areas in the process of making these JUnit Tests.

1) we make sure that there is a number value associated with the String Eggs. Which is done by sending string Eggs through a method to enter in the value to compare against a large number of strings, each matched with a line in that array. Each item in that holds the string that holds the item name. Before it

makes its way back to us, it needs to see if the value Eggs is even present. For if it were not present, that Writer would create an entry.

The Database Reader gets text from a file database.txt. Because in this case, an Entry for eggs has already been entered in, the Test case for asserting a number for Eggs exists for the store Walmart is true, passes.

2) secondly, we chose to covered, the matter of initializing the database writer. For if the writer object, with the constructor properly working, hadn't initialized, then we would not have been able to access it for when we need to add a new set of values into it.

3) our third test covers checking if the price an item returns, is what it needs to return. This is very important to make sure than somewhere along the process of coding these three stores, they didn't get mixed up. It is very easy for one to overlook and swap items mistakenly somewhere along the way. Including a unit test which confirms none of the prices got swapped at any point, really helps us to confirm that the whole thing is working and can be trusted as a credible source of comparing prices.

The Location of Some Important Information:

Sequence Diagrams for Use Cases, An Architectural Design Diagram, A State Chart Diagram of One Object, and Screenshots of me running the JUnit Tests, are all located in

Folder: **Documentation/Part 3/**

In the Folder: **Images/Extra Screenshots**

You will see extra screenshots we took during the development of this program, that we think further highlight the developmental steps we took.

Trello and Github:

We also used this Trello board to organize our ideas in an Agile methodology:

<https://trello.com/b/Uno5SFEE/software-engineering-group-project>

Link to our Github repo, where you can find this same folder present:

<https://github.com/SaifTTU/Shopping-Scraper>