CSIS 10B Lab 1 Java Review

Please work with a partner for all labs. Each partner should submit their completed lab code to Canvas when finished.

The two rules of pair programming are: 1) change seats every 10 minutes (driver/navigator), and 2) talk over the solution as you are working it out.

Part 1 Java Review / Coding Bat Drills

- You may either solve the problems in the Drills.java and LoopDrills.java files in the lab1 download (taken from CSIS10A)
- OR sign in to http://codingbat.com.
    - Working with a partner, alternate solving problems on each person's computer. Do these problems: **makeOutWord**, **firstHalf**, **love6**, **more20**, **commonEnd**, **reverse3**.

Part 2 Working with Java Arrays – Array of String and Array of double

Write a Java program to accept a given number of item names and prices, then output them in reverse order in which they were input. In addition, output the average price if one of the items is named Peas (not case sensitive) otherwise, output "no average output". (The first user input will be the number of items to process.)

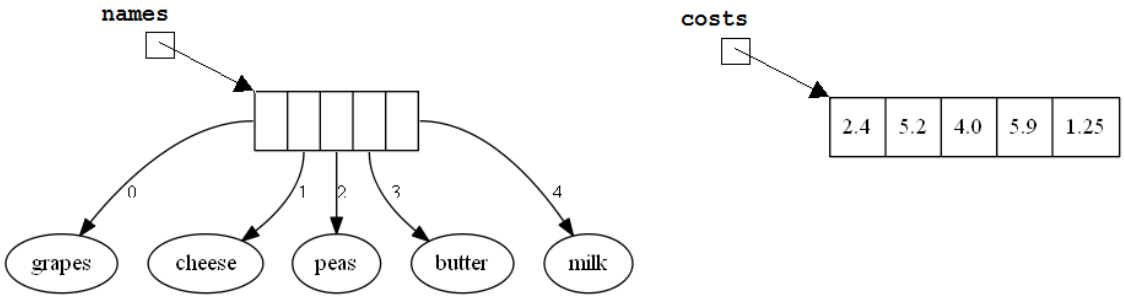
Here is a sample session (user input in bold face):

How many items? 5
enter next item and price: **grapes 2.40**
enter next item and price: **cheese 5.20**
enter next item and price: **peas 4.00**
enter next item and price: **butter 5.90**
enter next item and price: **milk 1.25**

Thankyou. Your items were:
milk 1.25
butter 5.90
peas 4.00
cheese 5.20
grapes 2.40

average price: 3.75

Use a simple array of String for the item names, and and an array of double for the item costs. This is an example of storing multiple data fields in "parallel arrays," because the different parts of data for an item purchased are stored at the same index in two different arrays. Here is what that looks like in memory:



Memory Map for Part 2, using Parallel Arrays

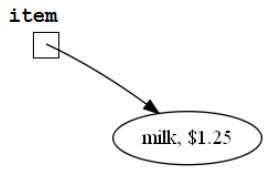
(**Part 2** instructions continue on next page)

1) Add a static method to the program described above that outputs "Welcome to MPC" before the items and prices are output. The main method should invoke this method.
2) Modify your program so the average price is calculated by a separate static method and returned to the invoker. The main method should invoke this method.
3) OPTIONAL: Modify your program so that after it produces its output it asks the user if they would like to process another group of items. The program should terminate when the user enters "no" (not case sensitive). Otherwise, it continues processing names and prices.

Part 3 Making an Item Class

The way we stored the data for a purchased item in the previous example is not perhaps the most effective way. It's a little awkward to split a data's fields across multiple arrays. An approach that better mirror's the situation we are modelling would be to create our own Item class that represents a single item being purchased by storing the name and the cost as separate fields within the same object.

This way a single Item object contains all the necessary information for one item purchased. If we declared an Item using the statement: `Item item = new Item("milk", 1.25);` it would look something like this:

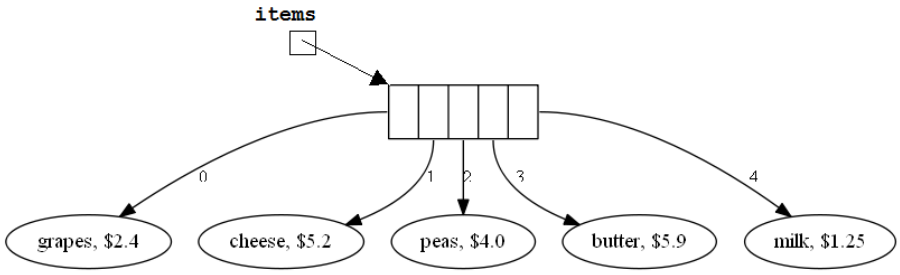


1) FInish developing the Item class, which represents an item's name and cost. Develop and test the constructors, toString, accessors, and modifier methods, and demo code that reads an Item from a Scanner

Part 4 Making an Array of Item objects

Once we have our own Item class, and we've tested it to make sure it works properly, we can then use this class in a revised version of the `GroceryCheckout` class that uses an Array of Item objects to store the data input by the user.

Here is what the new array will look like with all the same data as was input above.



Memory Map for Part 4, using an Array of Item objects

2) Open `GroceryCheckoutWithItemClass` and solve the problem from Part 2 using an array of Item objects.

3) OPTIONAL: If you want more practice writing classes, complete the class definition for Name and test it in a manner similar to what we did for the Item class.