Homework 2 – Let's go to the Supermarket!

Problem Description

This assignment will test your knowledge of classes, instance data, encapsulation, constructors, static variables and methods, and wrapper classes.

Solution Description

In this assignment, you will be designing classes to represent a `Store` and a `Cart` and implementing functionalities for two of the items it carries: `Cheese` and `Turkey`. Below we list the necessary functionalities to implement. We also specify some of the class and method details, but others are intentionally left vague to give you practice with design. Feel free to create your own helper variables and methods to achieve the desired functionality. Your classes should utilize good design practices and they **must use encapsulation**.

`Cheese`

- Every `Cheese` item has a name representing what type of cheese it is and whether it is shredded. Cheese items share a price initially set to `1.50` that can change.
- Cheese should have a constructor that takes in and initializes the name of cheese and whether it is shredded.
- Additionally, it should include 2 chained constructors: One should take in only the name of the cheese and create a `Cheese` object that is shredded and has the given name. The other should have no-args and create an object with the name "Cheddar".
- `Cheese` should have a method to change price.

`Turkey`

- Every `Turkey` item has a floating-point number representing the ounces of turkey in the package. `Turkey` items also share a price per ounce initially set to `1.99` that can change.
- It should have a constructor that takes in and initializes the ounces of turkey.
- `Turkey` should include a method that calculates and returns the total price of a particular Turkey object rounded to 2 decimal places.
- `Turkey` should have a method to change price.

`Store`

The stock available at the store can hold at most 5 `Cheese` objects and 5 `Turkey` objects. The stocks of the two products should be initialized to be empty arrays of length 5 in the `Store` constructor. Your `Store` should also have a name. `Store` should have a constructor that takes in a name and initializes the instance variables appropriately.

- Your `Store` should also include a method which returns the current stock of the store as a `String` in the format where `x` and `y` represent the amount of `Cheese` and `Turkey` in stock at the store:

```
Current Stock of (store name):
Cheese: x
Turkey: y
```

- `Store` should have two overloaded methods to add to stock for `Cheese` and `Turkey` objects. Add the item to the end of the respective item's stock array. If the respective item's stock is full, do nothing.
- `Store` should have two methods to remove a Cheese or Turkey object from stock. If the stock is empty when this method is called, you should refill the entire stock of that type of item with any arbitrary cheese name(s) or ounces of turkey. Then, it will remove and return the object from the end of the respective array.

`Cart`

You will shop at the store using a cart that can hold at most 3 `Cheese` objects and 3 `Turkey` objects, and the items in your cart will be represented through two variables: one representing the `Cheese` objects in the cart, and one representing the `Turkey` objects in your cart.

- A `Cart` belongs to a `Store`. Your constructor should take in a `Store` and initialize values with an empty cart.
- `Cart` should include a method to add to cart, which takes in a `String`. If the String is either "`Cheese`" or "`Turkey`", then you should remove one of that corresponding item from the Store's stock to place it in this cart. If some other `String` is entered, the method should not do anything. Think about the best object-oriented way to do this that respects encapsulation.
- `Cart` should include a method that calculates and returns the total of all the items in the cart. The price of the Cheese should be calculated as the number of cheese objects * the price of a cheese object, and the price of the Turkey should be calculated as the sum of the total prices of all Turkey objects in the cart.
- `Cart` should include a method to check out, where the method takes in the amount of money as a `double`. The method should return their change if they inputted enough money to buy the items in their cart or a `-1.0` if they did not input enough money. If checkout is successful, then the cart should be empty.

`StoreDriver`

Create a driver class to test your classes. You should create at least one store and a cart and call/print your methods. The main method should check the condition where one of the items goes out of stock and must be replenished as well as a check out attempt that succeeds and one that does not succeed (have sufficient money).

Submitting

To submit, upload the files listed below to the corresponding assignment on Gradescope:

- `Cheese.java`
- `Turkey.java`
- `Store.java`
- `Cart.java`
- `StoreDriver.java`

Make sure you see the message stating "Homework 2 submitted successfully." We will only grade your last submission be sure to **submit *every file* each time you resubmit.**

Feature Restrictions

There are a few features and methods in Java that overly simplify the concepts we are trying to teach or break our auto grader. For that reason, do not use any of the following in your final submission:

- `var` (the reserved keyword)
- `System.exit`

Collaboration

No collaboration is allowed on this assignment. See syllabus for more details.
In addition, note that it is not allowed to upload your code to any sort of public repository. This could be considered an Honor Code violation, even if it is after the homework is due. Only post code on Piazza in a **private** post.

Important Notes (Don't Skip)

- Non-compiling files will receive a 0 for all associated rubric items
- Do not submit `.class` files.
- Test your code in addition to the basic checks on Gradescope
- Submit every file each time you resubmit
- Read the "Allowed Imports" and "Restricted Features" to avoid losing points
- Check on Piazza for all official clarifications
- Make sure to test your program manually based on the assignment instructions and expected outputs. The Gradescope autograder visible to you is **NOT** comprehensive. A few test cases will be hidden when you submit. This is done intentionally so that you learn to write your own test cases for your assignments.