Grocery Store

1 Introduction

This assignment is meant to illustrate the Object-Oriented concept of polymorphism while drawing on your knowledge of inheritance, interfaces, and arrays.

2 Problem Description

As part of a worldwide effort to eliminate any sort of human-to-human contact, your grocery store has been selected to test a batch of shiny new grocery shopping robots, designed to aid customers in the arduous task of picking out groceries and purchasing them. The robots will fetch items and manage the store's inventory, but for this assignment, your focus will be managing a particular customer's "shopping cart" and handling the checkout process for goods with different pricing schemes.

3 Solution Description

Provided files (Do not modify these):

- `Buyable`: an interface that describes an item that can be bought. All `GroceryItems` must implement this interface's methods.
- `GroceryItem`: a specific type of `Buyable` for items you can buy in a Grocery store. It is the parent class of your `ProduceItem`, `CannedGood`, `DryGood`, and any other classes you choose to write for other product categories (optional)

You are required to write the following classes:

- `ProduceItem` an item priced per pound (fruits, vegetables, and such)
- `CannedGood` an item priced per can, but always on sale, "Buy 2 get 1 free."
  - take the quantity the customer specifies literally (e.g. if they request 2 cans, do not give them a 3rd free. They must ask for 3 cans if they want to take advantage of the sale)
- `DryGood` an item priced per unit
- `BoxedGood` a specific type of `DryGood` that is sold in boxes

- `ShoppingCartItem` represents a grocery item that has been added to a shopping cart in some quantity
- `ShoppingCart` wrapper for a resizable array that holds the customer's selected `ShoppingCartItems`
- `GroceryBot` the driver program that will get user input, manage their shopping cart, and perform checkout, providing a "receipt.txt" file

Class Specifications:

- `ProduceItem`, `CannedGood`, `DryGood`, and their children represent items as stocked on the grocery store shelves. Notes about implemented methods:
  - the `toString` method should return the name of the product, the price (in US Dollars) and the pricing scheme (ie "per pound").
  - the `getQuantityQuery` method should return a String that can prompt the user for the appropriate unit amount of each product (ie "How many pounds..." or "How many boxes...").
  - the `getCostOf` returns the cost of a hypothetical quantity of this `GroceryItem`, so your calculations should be different depending on the pricing schemes outlined above.
- `ShoppingCartItem` should have one constructor that takes in a `Buyable`, and each should have the following two fields:
  - a `Buyable`
  - an integer quantity of that item to be added to the `ShoppingCart` (don't worry about non-integer quantities)

  and must have the following methods:
  - `setQuantity`, takes in an integer value and assigns it to the quantity field
  - `calculateCost`, returns the cost of purchasing this `ShoppingCartItem` (type double), takes in no arguments
  - override `toString` from the `Object` class. Change the String representation of the `ShoppingCartItem` to be that of the `Buyable` it holds as well as a second line that includes the quantity and the total cost (in US Dollars) (see example output).
  - override `equals` from the `Object` class. Two `ShoppingCartItems` are equal if the names of their `GroceryItems` are equal, and the quantities are equal (ie two different quantities of the same `GroceryItem` are not equivalent `ShoppingCartItems`).
- `ShoppingCart` should hold a backing array of `ShoppingCartItems` initialized to size 10. It should have the following methods:
  - `add` takes in a `ShoppingCartItem` and adds it to the backing array. If the array is full, copy its contents to a new backing array with twice the size, and then add the new item.

  - `remove` takes in a specific `ShoppingCartItem`, searches through the array for a match, and returns the removed entry or `null` if no match is found. If there is a match, remove it and shift any subsequent items in the array so that any `null` entries are after those that are `!null`
  - override `toString` to iterate through the array of `ShoppingCartItems` and call `toString` for each of them, adding the result of each to the return String

- `GroceryBot` should hold the store's "inventory." Assume that the inventory is continually replenished. After the user exits the program write their receipt to a file called "Receipt.txt" (see below for example).

Running the program should look something like this:

```
$ java GroceryBot
Welcome to the Grocery Store valued customer!

What would you like to do?
1: Add item to cart
2: Remove item from cart
3: List cart contents
4: Checkout and Exit
Enter option numer: 1

We carry the following products:
1: Red Apples @ $2.00 per pound
2: Admiral Crunch Cereal @ $4.00 per box
3: Some Soup @ $5.00 per can, buy 2 get 1 free
Enter the number of the product you'd like to add: 1

How many pounds of Red Apples would you like? 3
Added item:
Red Apples @ $2.00 per pound
        x3 lbs = $6.00

What would you like to do?
1: Add item to cart
2: Remove item from cart
3: List cart contents
4: Checkout and Exit
Enter option number: 1

We carry the following products:
1: Red Apples @ $2.00 per pound
2: Admiral Crunch Cereal @ $4.00 per box
3: Some Soup @ $5.00 per can, buy 2 get 1 free
Enter the number of the product you'd like to add: 2

How many boxes of Admiral Crunch Cereal would you like? 2
Added item:
Admiral Crunch Cereal @ 4.00 per box
        x2 boxes = $8.00

What would you like to do?
1: Add item to cart
2: Remove item from cart
3: List cart contents
4: Checkout and Exit
Enter option numer: 1

We carry the following products:
1: Red Apples @ $2.00 per pound
2: Admiral Crunch Cereal @ $4.00 per box
3: Some Soup @ $5.00 per can, buy 2 get 1 free
```

```
Enter the number of the product you'd like to add: 1

How many pounds of Red Apples would you like? 1
Added item:
Red Apples @ $2.00 per pound
        x1 lbs = $2.00

What would you like to do?
1: Add item to cart
2: Remove item from cart
3: List cart contents
4: Checkout and Exit
Enter option number: 2

Here are your cart contents:
Red Apples @ $2.00 per pound
        x3 lbs = $6.00
Admiral Crunch Cereal @ $4.00 per box
        x2 boxes = $8.00
Red Apples @ $2.00 per pound
        x1 lbs = $2.00

Enter the name of the product you want to remove:
Red Apples

Enter the quantity you want to remove:
3

Here are your cart contents:
Admiral Crunch Cereal @ $4.00 per box
        x2 boxes = $8.00
Red Apples @ $2.00 per pound
        x1 lbs = $2.00

What would you like to do?
1: Add item to cart
2: Remove item from cart
3: List cart contents
4: Checkout and Exit
Enter option number: 2

Enter the name of the product you want to remove:
Let's Potato Chips

I'm sorry, I'm afraid I can't do that. No such item found...

What would you like to do?
1: Add item to cart
2: Remove item from cart
3: List cart contents
4: Checkout and Exit
Enter option number: 1

We carry the following products:
1: Red Apples @ $2.00 per pound
2: Admiral Crunch Cereal @ $4.00 per box
3: Some Soup @ $5.00 per can, buy 2 get 1 free
Enter the number of the product you'd like to add, or -1 to cancel: 3

How many cans of Some Soup would you like? Buy 2, get 1 free!
3

Added item:
Some Soup @ $5.00 per can
        x3 cans = $10.00

What would you like to do?
1: Add item to cart
```

```
2: Remove item from cart
3: List cart contents
4: Checkout and Exit
Enter option number: 3

Here are your cart contents:
Admiral Crunch Cereal @ $4.00 per box
        x2 boxes = $8.00
Red Apples @ $2.00 per pound
        x1 lbs = $2.00
Some Soup @ $5.00 per can, buy 2 get 1 free
        x3 cans = $10.00

What would you like to do?
1: Add item to cart
2: Remove item from cart
3: List cart contents
4: Checkout and Exit
Enter option number: 4

Thank you! Please take your receipt (see Receipt.txt)
$
```

The ”Receipt.txt” should look something like this:

```
Grocery Store Receipt

Here are your purchases:
Admiral Crunch Cereal @ $4.00 per box
        x2 boxes = $8.00
Red Apples @ $2.00 per pound
        x1 lbs = $2.00
Some Soup @ $5.00 per can, buy 2 get 1 free
        x3 cans = $10.00

Total = $20.00

Thank you!
```

4 Checkstyle

Review the CS 1331 Code Conventions and download the Checkstyle jar file and the configuration file to the directory that contains your Java source files. Run Checkstyle on your code like this (in the directory containing all your Java source files):

```
$ java -jar checkstyle-5.6-all.jar -c cs1331-checkstyle.xml *.java
Starting audit...
Audit done.
```

The message above means there were no Checkstyle errors. You can easily count the errors by piping the output of Checkstyle through `grep`.

```
$ java -jar checkstyle-5.6-all.jar -c cs1331-checkstyle.xml *.java | grep -cEv "(Starting
    audit...|Audit done)"
0
$
```

The `-c` option tells `grep` to count matching lines instead of printing them, `E` means use `egrep` (extended `grep`) syntax, and `v` means invert the match. Here we use an inverted match to discard the two non-error lines of Checkstyle's output. If you have `egrep` you could leave off the `-E` and just use `egrep`.

```
$ java -jar checkstyle-5.6-all.jar -c cs1331-checkstyle.xml *.java | egrep -cv "(Starting
    audit...|Audit done)"
0
$
```

Alternatively, if you are Windows, you can use this command to count the number of style errors by piping output through the `findstr` program.

```
C:/> java -jar checkstyle-5.6-all.jar -c cs1331-checkstyle.xml *.java |findstr /v "Starting
    audit..." | findstr /v "Audit done" | find /c /v "!!!!!"
```

The Java source files we provide contain no Checkstyle errors. You are responsible for any Checkstyle errors you introduce when modifying these files. For this assignment, there will be a maximum of 50 points lost due to Checkstyle errors. In future homeworks we will be increasing this cap, so get into the habit of fixing these style errors early!

5 Javadoc

Add Javadoc comments to all classes and methods except `main`. The tags we are looking for are: **@author, @version, @param, @return**.

6 Tips

- **Do** use polymorphism to print the correct `Strings` to the console.
- **Don't** use conditionals and prewritten `Strings`. Your code must work for an arbitrary inventory of `Buyables`, including ones you don't know about. When testing your code, we may even include Buyables of our own, so you additionally should be casting as little as possible and relying on Polymorphism to the do heavy lifting for you.
- When writing your Receipt.txt to the hard drive, you may find it useful to use a `BufferedWriter` or `PrintWriter` classes useful. Look up how to use them via Google or the API.
- You are not allowed to use `ArrayLists` or `System.ArrayCopy` or other similar things for this homework. If you have a concern about the validity of your method, ask a TA.
- You shouldn't be making concrete Apple or Candy subclasses - you'll be doing something akin to `Buyable apple = new GroceryItem("apple", "pounds", "2");`
- Take a step back from the problem! While it may seem overwhelming at first, try drawing out a diagram of what the class relationships look like. You'll find that understand what extends from what and what classes use what other classes **before you begin coding** will help quite a bit. Start from the simplest ones (ones at the top of the heirarchy) and move down. You'll find that each class you write for this homework will be very small, some as little as 10 lines of code.

7 Turn-in Procedure

Submit all of the Java source files you create to T-Square. Do not submit the classes provided to you, any compiled bytecode (`.class` files), the Checkstyle jar file, or the `cs1331-checkstyle.xml` file. When you're ready, double-check that you have submitted and not just saved a draft. Theses files include, but are not limited to:

- `ProduceItem`
- `CannedGood`
- `DryGood`
- `BoxedGood`
- `ShoppingCartItem`
- `ShoppingCart`
- `GroceryBot`

8 Verify the Success of Your Submission to T-Square

Practice safe submission! Verify that your HW files were truly submitted correctly, the upload was successful, and that the files compile and run. It is solely your responsibility to turn in your homework and practice this safe submission safeguard.

1. After uploading the files to T-Square you should receive an email from T-Square listing the names of the files that were uploaded and received. If you do not get the confirmation email almost immediately, something is wrong with your HW submission and/or your email. Even receiving the email does not guarantee that you turned in exactly what you intended.

2. After submitting the files to T-Square, return to the Assignment menu option and this homework. It should show the submitted files.

3. Download copies of your submitted files from the T-Square Assignment page placing them in a new folder.

4. Recompile and test those exact files.

5. This helps guard against a few things.

   (a) It helps ensure that you turn in the correct files.

   (b) It helps you realize if you omit a file or files.[1] (If you do discover that you omitted a file, submit all of your files again, not just the missing one.)

   (c) Helps find last minute causes of files not compiling and/or running.

[1]Missing files will not be given any credit, and non-compiling homework solutions will receive few to zero points. Also recall that late homework will not be accepted regardless of excuse. Treat the due date with respect. The real due date is midnight Saturday. Do not wait until the last minute!