Homework 01 – Grocery Shopping

Author: Srujal Gawali, Eric Dong, Lily Kwok, Helen Chen

Topics: Classes and objects, encapsulation, constructors, visibility modifiers, getters, setters

Problem Description

It's the weekend and that means you need to make a quick run to the grocery! You can now use your knowledge of object-oriented programming to help keep track of all the things you are going to buy. To do this you will create three files: `Dairy.java`, `Bakery.java`, and `ShoppingCart.java` to keep track of your groceries.

Solution Description

Create `Dairy.java` which will represent the things you buy from the Dairy section, `Bakery.java` to represent the things you buy from the Bakery section, and `ShoppingCart.java` which will hold all the things you have bought. You will be creating several fields and methods for each file. Based on the description given for each variable and method, you will have to decide whether the variables and methods should be static, and whether they should be private or public. To make these decisions, you should carefully follow the guidelines on these keywords as covered in lecture. In some cases, your program will still function with an incorrect keyword.

HINT: A lot of the code you will write can be reused. Try to think of what keywords you can use that will help you!

REMEMBER: Homeworks are different from PEs in that they state *REQUIREMENTS*. These requirements may *NOT* always be stated in the *ORDER* in which you should implement them. For example, we may ask you to return a value and report an error, but these two requirements must be handled in the correct order.

Dairy.java

This file defines a `Dairy` object.

Variables:

All variables must **not** be allowed to be **directly modified** outside the class in which they are declared, unless otherwise stated in the description of the variable. The `Dairy` class must have these variables:

- `product` – the name of the dairy product (example: cheese, milk, butter)
- `quantity` – quantity of the product as a whole number
- `cost` – the cost of one item of the product, as a Java default floating-point number (e.g., 4.99)
- `daysToExpiration` – the number of days to expiration as a whole number
- `isConsumable` – whether there are 3 or more days until expiration; If the number of days is less than 3, the product is not consumable. This value will not be passed as a parameter to the constructor.
    - When `isConsumable` is initialized, print out "`Good choice!`" if the product is consumable. Otherwise, print "`Check the expiration date`". This should only be printed when `isConsumable` is first initialized.

Constructor(s):

- A constructor that takes in the `product`, `quantity`, `cost`, and `daysToExpiration`
- A constructor that takes in `product` and `quantity`. In this case, `cost` should be assumed to be `5.59` and `daysToExpiration` should be assumed to be `5`.
- A constructor that takes in no arguments. In this case, `product` should be "`yoghurt`", `quantity` should be `4`, `cost` should be `3.99`, and `daysToExpiration` should be `3`.
- **NOTE:** The constructor parameters should be in the order listed above. Assume that constructor inputs will be valid (i.e., `product` will not be null, `cost` will not be negative).

Methods:

All methods should have the proper visibility to be used where it is specified, they are used.

- Getters and setters ***only as necessary***
- Any helper methods that you may need; ensure that these methods are not accessible outside of this class.

Bakery.java

This file defines a `Bakery` object.

Variables:

All variables must not be allowed to be directly modified outside the class in which they are declared, unless otherwise stated in the description of the variable. The `Bakery` class must have these variables:

- `product` – the name of the bakery product (example: bread, bagel)
- `quantity` – quantity of the product as a whole number
- `cost` – the cost of one item of the product, as a floating-point number (e.g., 4.99)

Constructor(s):

- A constructor that takes in `product`, `quantity`, and `cost`
- **NOTE:** The constructor parameters should be in the order listed above. Assume that constructor inputs will be valid (i.e., `product` will not be null, `cost` will not be negative).

Methods:

All methods should have the proper visibility to be used where it is specified, they are used.

- Use getters and setters ***only as necessary***
- Any helper methods that you may need; ensure that these methods are not accessible outside of this class.

ShoppingCart.java

This file defines a `ShoppingCart` object which contains `Dairy` and `Bakery` objects.

Variables:

All variables must not be allowed to be directly modified outside the class in which they are declared, unless otherwise stated in the description of the variable. The `ShoppingCart` class must have these variables:

- `dairyItems` – a value that represents the `Dairy` objects in your shopping cart, represented by an array of `Dairy` objects
- `bakeryItems` – a value that represents the `Bakery` objects in your shopping cart, represented by an array of `Bakery` objects

Constructor(s):

- A constructor that takes in `dairyItems` and `bakeryItems`.
- A constructor that takes in no parameters and initializes `bakeryItems` to an array of size 2 and with one `Bakery` object at index 0 with the following values `<Bread, 1, 5.99>` and initializes `dairyItems` to an empty array of size 4.
- **NOTE**: The constructor parameters should be in the order listed above. Assume that constructor inputs will be valid (i.e., `dairyItems` will not be null, `bakeryItems` will not be null).

Methods:

All methods should have the proper visibility to be used where they are specified to be used.

- `addDairyItemAtIndex`
  - Given an `index` and a `Dairy` parameter, add that `Dairy` object to the index in the `dairyItems` array and print out the name of the item that was previously at that index in the following format: "`There was {product} here before.`" and return the `Dairy` object that was previously placed there.
  - If there was no item at the index, print out the name of the `Dairy` object added in the following format "`{product} was added here.`" and return `null`.
  - If the index is invalid or if the `Dairy` object passed in is `null`, return `null` and print out "`Cannot add Dairy item!`"
- `removeBakeryItemAtIndex`
  - Given an `index`, remove the `Bakery` item from `bakeryItems` at that index and print out "`{product} was removed from the cart.`" Return the `Bakery` object that was removed.
  - If the index is invalid, print "`Invalid index entered.`" and return `null`.
  - If no object exists at that index, print "`No Bakery item exists at this index!`" and return `null`.
- `updateDairyQuantity`
  - Given the `product` name of the `Dairy` item (case-insensitive) and the new `quantity`, search and update the `Dairy` item's `quantity` with the `quantity` passed in and print out "`Quantity of {product} was updated from {old quantity} to {new quantity}.`"
  - If the `product` name is not found or if the `product` entered was null, print "`No such Dairy item exists in the cart!`"
  - Remember to check if the quantity entered is nonnegative. Else, print "`Invalid quantity entered.`" and do not update the quantity.
  - This method does not need to return anything.

- **NOTE:** You can assume that there are no duplicate items. That is, no two Dairy objects have the same product name.

- `displayItems`
  - This method takes in no parameters, simply print out the `product` name and `quantity`, and `cost` of each item in both `dairyItems` and `bakeryItems`.
  - Print each item's details on a different line.
  - For each Dairy object, print the following on its own line:
    `"Dairy Product: {product} Quantity: {quantity} Cost: {cost}"`
    For each null object, print "null" on its own line. Cost should be displayed to 2 decimals.
  - For each Bakery object, print the following on its own line:
    `"Bakery Product: {product} Quantity: {quantity} Cost: {cost}"`
    For each null object, print "null" on its own line. Cost should be displayed to 2 decimals.
  - Remember you cannot print out objects itself, you can only print out the variables associated with each object. How will you get that data?
  - **NOTE:** the order of the items is important. First, print out all the Dairy items in order and then the Bakery items in order.

When writing your methods:

- Include Getters and Setters ***only as necessary***
- Include any helper methods that you may need, but ensure that these methods are not accessible outside of this class.
- **HINT**: You will likely run into a few exceptions (think of the empty spaces in the array), consider how you can prevent it from occurring using conditionals and consider what it is.

Driver.java

This file is used to test your code. You do not need to turn this in.

Methods:

- `main`
  - Create 3 `Dairy` objects and place them in an array.
  - Create 3 `Bakery` objects and place them in an array.
  - Create a `ShoppingCart` object with the arrays created previously.
  - Create and add 1 `Dairy` object using `addDairyItemAtIndex`
  - Remove 1 `Bakery` object using `removeBakeryItemAtIndex`
  - Update the quantity of a `Dairy` object using `updateDairyQuantity`
  - **TIP:** Print out the contents of `dairyItems` and `BakeryItems` using the `displayItems` method to see the changes expected after adding, removing, or updating objects.
  - **TIP:** Check the sample output given on the Ed clarification thread to make sure your program is running correctly.
  - **NOTE:** This is to help you test your code, but it is not comprehensive. It is suggested you create more test cases. Great testing would include creating objects with each of the constructors, invoking any public methods, and observing that the results are consistent with what you would expect.

Checkstyle

You must run checkstyle on your submission (To learn more about Checkstyle, check out cs1331-style-guide.pdf under CheckStyle Resources in the Modules section of Canvas.) **The Checkstyle deduction cap for this assignment is <u>10 points</u>.** If you do not have Checkstyle yet, download it from Canvas -> Modules -> CheckStyle Resources -> checkstyle-8.28.jar. Place it in the same folder as the files you want to run Checkstyle on. Run checkstyle on your code like so:

```
$ java -jar checkstyle-8.28.jar yourFileName.java
Starting audit...
Audit done.
```

The message above means there were no Checkstyle errors. If you had any errors, they would show up above this message, and the number at the end would be the points we would take off (limited by the checkstyle cap mentioned in the Rubric section). In future homeworks we will be increasing this cap, so get into the habit of fixing these style errors early!

Turn-In Procedure

Submission

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

- `Dairy.java`
- `Bakery.java`
- `ShoppingCart.java`

Make sure you see the message stating the assignment was submitted successfully. From this point, Gradescope will run a basic autograder on your submission as discussed in the next section. **Any autograder test are provided as a courtesy to help "sanity check" your work and you may not see all the test cases used to grade your work.** Autograder tests are **NOT** guaranteed to be released when the assignment is released, so ***<u>YOU</u>*** are responsible for thoroughly testing your submission on your own to ensure you have fulfilled the requirements of this assignment. If you have questions about the requirements given, reach out to a TA or Professor via our class discussion forum for clarification.

You can submit as many times as you want before the deadline, so feel free to resubmit as you make substantial progress on the homework. We will only grade your <u>latest submission</u>. **Be sure to submit every file each time you resubmit**.

Gradescope Autograder

If an autograder is enabled for this assignment, you may be able to see the results of a few basic test cases on your code. Typically, tests will correspond to a rubric item, and the score returned represents the performance of your code on those rubric items only. If you fail a test, you can look at the output to determine what went wrong and resubmit once you have fixed the issue.

The Gradescope tests serve two main purposes:

- Prevent upload mistakes (e.g. non-compiling code)

- Provide basic formatting and usage validation

In other words, the test cases on Gradescope are by no means comprehensive. Be sure to thoroughly test your code by considering edge cases and writing your own test files. You also should avoid using Gradescope to compile, run, or Checkstyle your code; you can do that locally on your machine.

Other portions of your assignment can also be graded by a TA once the submission deadline has passed, so the output on Gradescope may not necessarily reflect your grade for the assignment.

Allowed Imports

To prevent trivialization of the assignment, **you are not allowed to import any classes or packages**.

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`
- `System.arraycopy`

Collaboration

Only discussion of the Homework (HW) at a conceptual high level is allowed. You can discuss course concepts and HW assignments broadly, that is, at a conceptual level to increase your understanding. If you find yourself dropping to a level where specific Java code is being discussed, that is going too far. Those discussions should be reserved for the instructor and TAs. To be clear, you should never exchange code related to an assignment with anyone other than the instructor and TAs.

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 our class forum for a note containing all official clarifications and sample outputs**

It is expected that everyone will follow the Student-Faculty Expectations document, and the Student Code of Conduct. The professor expects a **positive, respectful, and engaged academic environment** inside the classroom, outside the classroom, in all electronic communications, on all file submissions, and on any document submitted throughout the duration of the course. **No inappropriate language is to be used, and any assignment, deemed by the professor, to contain inappropriate, offensive language or threats will get a zero**. You are to use professionalism in your work. Violations of this conduct policy will be turned over to the Office of Student Integrity for misconduct.