Homework 02 – The Office Warehouse

Authors: Ananay, Vibhav, Nicolas, Chelsea
Topics: Inheritance, constructor chaining, copy constructor, static, wrapper classes

Problem Description

Help Darryl keep track of his warehouse of paper products! Due to the new Computer software poorly written in C and his boss Michael making it impossible for him to work in the warehouse by adding in his golden tickets to random shipments, Darryl is unable to keep track of his stock in the warehouse. Help him by rewriting his software in clean Java!

Solution Description

You will implement 5 classes: PaperProduct.java, PhotoPaper.java, GoldenTicket.java, DiscountedPaper.java, and Warehouse.java

Note:

1. *All variables should be inaccessible from other classes and must require an instance to be accessed through, unless specified otherwise.*
2. *All methods should be accessible from everywhere and must require an instance to be accessed through, unless specified otherwise.*
3. *Use constructor and method chaining whenever possible! Ensure that your constructor chaining helps you reduce code reusage!*
4. *Reuse code when possible and helper methods are optional.*
5. *All floating point values must be printed to only 2 decimal places, with rounding.*
6. *Make sure to add all Javadoc comments to your methods and classes!*

PaperProduct.java

This file will store details about a paper product in general. We will create multiple child classes to specialize these products.

Variables:

- `name` - a `String` variable representing the name of the Product at a given instance. An invalid value should default to "A4". The value is invalid if `name` is an empty String or null. This value *cannot* be changed once set.
- `numberOfSheets` - an `int` representing the number of the sheets of paper present in this product. An invalid value should always be set to 500. The value is invalid if it is negative.
- `weightOfUnitSheet` - a `double` representing the weight, in grams, of one sheet of the paper product at a given instance. An invalid value should always be set to 0.25. The value is invalid if it is negative. This value *cannot* be changed once set.
- `totalProductsToShip` - an `int` representing the total number of `PaperProduct` that can be shipped. This variable should be shared across multiple instances. It must have an initial

value of 10. [Hint: This variable should only decrease when you run your code and should not be negative]

- `COST_PER_GRAM` - a constant representing the cost of any product per gram, with the value 0.025. This variable should be accessible everywhere.

Constructor(s):

- A constructor that takes in `name, numberOfSheets, weightOfUnitSheet`.
- A constructor that takes in `name, numberOfSheets,` and defaults `weightOfUnitSheet` to 0.25.
- A constructor that takes in `name,` and defaults `numberOfSheets` to 500 and `weightOfUnitSheet` to 0.25.
- A copy constructor that deep copies all instance variables of the old object to the new object

Methods:

- `totalWeight()` - returns the total weight of the Paper Product.
- `totalCost()` - returns the total cost of the Paper Product
- `paperString()`
  - Returns a String representing the Object
  - Should return the String in the following format:

    `"<totalWeight>g of <name> for $<totalCost>"`
- `ship()`
  - Takes in a name of the company to ship to as a parameter
  - If `totalProductsToShip` does not equal 0, return the String:

    `"Shipped <totalWeight>g of <name> for $<totalCost> to <Company>."`
  - If `totalProductsToShip` equals 0, return the String:

    `"Cannot ship any items, Warehouse is empty!"`
  - Make sure to update all relevant variables in this method (decrement the number of products to ship by 1)
- Getters for `name`, `numberofSheets`, `weightOfUnitSheet`, and `totalProductsToShip`
- Setter for `numberofSheets`

PhotoPaper.java

This file will store details about the Photo Paper stock in the warehouse. It is a child class of `PaperProduct`.

Variables:

- `glossiness` - a `double` representing the glossiness of the paper product in the range [0, 100]. If an invalid value is entered, this variable should default to 70.

Constructor(s):

- A constructor that takes in `name`, `numberOfSheets`, `weightOfUnitSheet`, `glossiness`.
- A constructor that takes in `name`, `numberOfSheets` and defaults `glossiness` to 70, and all other variables to their default in paper-product.
- A constructor that takes in `name`, and defaults `glossiness` to 70, and all other variables to their default in paper-product.
- A copy constructor that deep copies all instance variables of the old object to the new object

Methods:

- `photoString()`
  - Returns a String representing the Object.
  - Should return the String in the following format:

    `"<glossiness>% glossy and <totalWeight>g of <name> for $<totalCost>"`

- `shipPhoto()`
  - Takes in a name of the company to ship to as a parameter
  - If `totalProductsToShip` does not equal 0, return the String:

    `"Paper is <glossiness>% glossy. Shipped <totalWeight>g of <name> for $<totalCost> to <Company>."`

  - If `totalProductsToShip` is 0, then it should return the String:

    `"Paper is <glossiness>% glossy. Cannot ship any items, Warehouse is empty!"`

  - Make sure to update all relevant variables in this method.
    - Keep in mind that the variables may be private in the super class, but they should be able to be modified through a publicly available method if chained correctly.
- Getter and setter for `glossiness`
- *Reuse code if possible*

GoldenTicket.java

This file will store details about the Golden Tickets that customers can use with the Discounted Stock.

Variables:

- `catchphrase` - a `String` object that represents the catchphrase printed on the ticket! An invalid value should default to "Congrats!". The value is invalid if `catchphrase` is an empty String or null.
- `discount` - a `double` representing the discount offered by the ticket, in percent, in the range (0, 25]. If an invalid value is entered, then this variable should default to 15.0.

Constructor(s):

- A constructor that takes in `catchphrase, discount`.

Methods:

- `ticketString()`
  - Returns a String representing the object, with the given format:

    `"Golden Ticket with a <discount>% discount! <catchphrase>"`
- Getters and setters for `catchphrase` and `discount`

DiscountedPaper.java

This file will store details about the Discounted Paper stock in the warehouse. It is a child class of `PaperProduct`.

Variables:

- `discount` - a `double` representing the discount of the paper product, in percent, in the range (0, 50]. If an invalid value is entered, then this variable should default to 15.0.
- `ticket` - a `GoldenTicket` object representing whether this product has a golden ticket attached to it. It has a default value of `null`.

Constructor(s):

- A constructor that takes in `name, numberOfSheets, weightOfUnitSheet, discount, ticket.`
- A constructor that takes in `name, numberOfSheets,` and defaults `discount` to 15, `ticket` to null, and all other variables to their default in `PaperProduct`.
- A constructor that takes in `name`, and defaults `discount` to 15, `ticket` to null, and all other variables to their default in `PaperProduct`.
- A copy constructor that deep copies all instance variables of the old Object to the new Object

Methods:

- `discountedCost()`
  - Should calculate and return the total cost after the discount.
  - If there is a golden ticket attached, apply the Golden Ticket discount to the discounted price.
- `shipDiscounted()`
  - Takes in a name of the company to ship to as a parameter
  - If `totalProductsToShip` does not equal 0, return the String:

    `"Shipped <totalWeight>g of <name> for $<totalCost> to <Company>. The total cost after the discount is <discountedCost>."`

    - If `totalProductsToShip` is 0, then it should return the String:

      `"Cannot ship any items, Warehouse is empty! The total cost after the discount is <discountedCost>."`

    - Make sure to update all relevant variables in this method.
      - Keep in mind that the variables may be private in the super class, but they should be able to be modified through a publicly available method if chained correctly.
- `botherAccounting()`
    - Returns a String to representing the discounting of this product as an entry for the accounting department.
    - It should return a String in the following format:

      ```
      "Discounted Paper Product:
      Original Cost: <original cost>
      Final Cost: <final cost>
      Original Discount: <original discount>%
      Golden Ticket Discount: <golden ticket discount>%"
      ```

    - If there is no Golden Ticket, then the Golden Ticket discount should be 0.
- Getters and setters for `discount` and `ticket`
- ***Reuse code if possible***

Warehouse.java

This Java file is a driver, meaning it will contain and run all the above classes! Use this to test your code.

- Create at least 2 `PaperProducts`, 2 `DiscountedPaper`, 2 `PhotoPaper`, and 1 `GoldenTicket`.
- Use the copy constructors at least once. Try modifying the objects to see if you have deep copied properly.
- Call the relevant ship methods on each of your `PaperProduct` and its children's objects and print the results.
- Call `botherAccounting()` on all `DiscountedPaper` and print the result.
- Reuse your code when possible. These tests and the ones on Gradescope are by no means comprehensive, so be sure to create your own!

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 cap for this assignment is 15 points.** This means there is a maximum point deduction of 15. If you don't 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!

Additionally, you must Javadoc your code.

Run the following to only check your Javadocs:

```
$ java -jar checkstyle-8.28.jar -j yourFileName.java
```

Run the following to check both Javadocs and Checkstyle:

```
$ java -jar checkstyle-8.28.jar -a yourFileName.java
```

For additional help with Checkstyle see the CS 1331 Style Guide.

Turn-In Procedure

Submission

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

- `PaperProduct.java`
- `PhotoPaper.java`
- `GoldenTicket.java`
- `DiscountedPaper.java`
- `Warehouse.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.** You 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 Piazza 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 latest submission. **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 Ed Discussion 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.