Programming Exercise 05 – Vending Machine

Authors: Matthew, David, Eric
Topics: exceptions, ArrayList

Problem Description

A vending machine contains various items available for purchase. Problems arise when a machine is prompted to vend an item that isn't present within it, or when an attempt is made to purchase an item with insufficient funds. You aim to simulate this behavior with your Java programming skills.

Solution Description

You will create three classes, two of which are exception classes: `OutOfStockException.java`, `NotEnoughMoneyException.java`, and `VendingMachine.java`.

Notes:
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.** Helper methods with appropriate visibility are optional.
5. Make sure to add all Javadoc comments to your methods and classes!

OutOfStockException.java

This class represents a **checked** exception that is thrown when attempting to find an out of stock item.

Constructors:
- `OutOfStockException(String item)`
    - This exception should have the message:

      `"<item> is not in stock."`
    - ***Hint:*** How can we use the super class's constructor here?

NotEnoughMoneyException.java

This class represents an **unchecked** exception that is thrown when attempting to buy an item without sufficient funds.

Constructors:
- `NotEnoughMoneyException()`
    - This exception should have the message:

      `"Item not vended. Insufficient funds."`
    - ***Hint:*** How can we use the super class's constructor here?

VendingMachine.java

This class represents a vending machine.

Variables:

- `ArrayList<String> stock` – list of items that are in stock within the vending machine
- `int cost` – the price in cents per item, identical for all items

Constructors:

- A 2-arg constructor that accepts `stock` and `cost`, in that order.
  - You may shallow copy the ArrayList reference.
  - Assume inputs are valid (`stock` is non-null, `cost` is positive).

Methods:

- `checkStock(ArrayList<String> order)`
  - This method checks whether all items in `order` are in stock.
  - If any item in `order` is not found within `stock`, throw an `OutOfStockException`. The exception message should correspond to the item in question.
    - ***Hint:*** We are required to acknowledge this exception at compile-time. Consider the implications this has for this method's header.
  - If all items in `order` are in stock, print the following on its own line:

    ```
    "All items are in stock!"
    ```
  - If the input is null, throw an `IllegalArgumentException` with a helpful message.
- `buyItem(String item, int money)`
  - An item is purchased from the vending machine if the user has sufficient money (expressed in cents) for this item. If an item is purchased, remove it from `stock`.
  - If the requested item is not in stock, throw an `OutOfStockException`. The exception message should correspond to the item in question.
    - ***Hint:*** We are required to acknowledge this exception at compile-time. Consider the implications this has for this method's header.
  - If the user does not have sufficient money, throw a `NotEnoughMoneyException`.
    - **Note**: If the user does not have sufficient money **and** the item is not in stock, an `OutOfStockException` should be thrown.
  - If the item is successfully vended, print the following on its own line (no curly braces):

    ```
    "{item} successfully vended!"
    ```
  - If `item` is null, throw an `IllegalArgumentException`.
- `availableFavorites(ArrayList<String> favoriteItems)`
  - Create a new ArrayList named `available`.
  - Add items from `favoriteItems` that are currently in stock to `available`.
  - Return the new ArrayList, which should only contain items present in both `stock` and `favoriteItems`.
  - If the input is null, throw an `IllegalArgumentException` with a helpful message.
- `main`
  Note: **You must follow the instructions for this main method as described below**. Think of this main method as the program you will submit instead of a place to test your code. You may still share test cases in a Driver file with a different main method on the course discussion forum, but **no part of this main method should be shared or copied**.

- Create an ArrayList named `vendingItems` and add at least five String elements, each representing the name of an item found within a vending machine.
- Create an ArrayList named `favorites` and add at least three String elements, each representing one of your favorite items.
- Create an instance of `VendingMachine` and pass in `vendingItems` and a positive cost to the constructor.
- In a try/catch block:
  - Call `checkStock` and pass in `vendingItems`.
  - Call `buyItem` and pass in an item and a money amount. Call this method with each of the following input cases:
    - In stock item, sufficient money amount.
    - In stock item, insufficient money amount.
    - Out of stock item, sufficient money amount.
    - Out of stock item, insufficient money amount.
  - Call `availableFavorites` and pass in `favorites`.
    - Iterate through the returned ArrayList and print out each item on its own line.
  - If an `OutOfStockException` is thrown, catch the exception and print out its message to the console on its own line.
  - If a `NotEnoughMoneyException` is thrown, catch the exception and print out its message to the console on its own line.
  - Do not catch any other types of exceptions.
  - Regardless of whether an exception is caught, print the following to the console on its own line:

    ```
    "Took a trip to the vending machine!"
    ```

    - ***Hint***: Which block of a try-statement always runs?
  - **Tip**: Play around with the contents of `vendingItems` and the value of your cost to test the output of your program for different cases.

Checkstyle

You must run Checkstyle on your submission (to learn more about Checkstyle, check out cs1331-style-guide.pdf under the Checkstyle Resources module on Canvas). The Checkstyle cap for this assignment is 35 points. This means there is a maximum point deduction of 35 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 number of points we would take off (limited by the Checkstyle cap). In future assignments 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:

- `OutOfStockException.java`
- `NotEnoughMoneyException.java`
- `VendingMachine.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 tests 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 the class 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 assignment. 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. **We reserve the right to hide any or all test cases, so you should make sure to test your code thoroughly against the assignment's requirements.**

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.

Burden of Testing

You are responsible for thoroughly testing your submission against the written requirements to ensure you have fulfilled the requirements of this assignment.

Be **very careful** to note the way in which text output is formatted and spelled. Minor discrepancies could result in failed autograder cases.

If you have questions about the requirements given, reach out to a TA or Professor via the class forum for clarification.

Allowed Imports

- `java.util.ArrayList`

Feature Restrictions

There are a few features and methods in Java that overly simplify the concepts we are trying to teach or break our autograder. 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 assignment at a conceptual high level is allowed. You can discuss course concepts and 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.

The only code you may share are test cases written in a Driver class. If you choose to share your Driver class, it should be posted to the assignment discussion thread on the course discussion forum. We encourage you to write test cases and share them with your classmates, but we will not verify their correctness (i.e., use them at your own risk).

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. **<u>No inappropriate language is to be used</u>, 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.