Homework 1: The Vending Machine

Authors: Nicolas, Mridul, Michael, Dongjae, Melanie, David

Problem Description

This assignment will test your knowledge on static methods, arrays, and the Math class.

You recently found out that Georgia Tech decided to install a brand-new vending machine filled with your favorite foods. However, they have yet to implement the system that handles the sale and choosing of items. Now they are seeking your help and your handy CS1331 programming skills to finish the project.

Solution Description

You will be creating the `VendingMachine` class. First, you will need to keep track of a list of 8 items along with their prices. To accomplish this, you will need to create an array. Note that the wanted structure of the array should resemble the following (as an example): `[[104, 5], [42, 10],…]`. Each subarray represents an item code along with its price. In this example, in the first subarray, 104 will be the item code and 5 represents its price. No two items should have the same item code. Item codes can be any positive whole number and do not have to be in any particular order. You can assume that the item price will always be positive. Create this array within your main method and ensure that it has the name `items`.

Additionally, you will also need to keep track of a person's balance. Create an integer variable for `balance` within your main method.

Main Method:

- The main method should initialize and populate the items array with 8 different items along with their prices, initialize the `balance` variable (with a non-negative amount), and call all the methods listed below at least once (except for `updateBalance` since it is only meant to help you implement the `buyItem` and `buyRandomItem` methods). This is also a great place to check that your methods work as expected by using print statements after you have finished their implementation.

Methods (note all methods must be static):

Note: For all methods that take in the `items` array, you may assume that the passed in array will always be valid (i.e., not `null`)

- An `updateBalance` method that takes in the price of the purchased item and the user's current balance as arguments and subtracts the price of the purchased item from the tracked balance. This method should return the user's new balance after having purchased the item. Note that if the user does not have enough money to purchase an item, simply return the original balance.
- A `buyItem` method that takes in the `items` array, the user's current balance, and an integer (representing the item code) as arguments. This method must update the user's balance accordingly by subtracting the price associated with the item code. Note that the purchase is unsuccessful if the person has insufficient balance to purchase the item or if the item does not exist in the list. In those events, simply return the original balance. Otherwise, this method should return the user's new balance after having purchased the item. Be sure to reuse code as much as possible and note that calling the `updateBalance` method could come in handy.
- A `buyRandomItem` method that that takes in the `items` array and the user's current balance as arguments. This method randomly generates a **valid index** in the `items` array and attempts to purchase that item. Make sure to update the balance accordingly if the user can buy the item (note that calling the `updateBalance` method could come in handy). This method returns the user's new balance after having purchased the item. Be sure to reuse code as much as possible.

- An `updateItems` method that takes in the `items` array, an integer (representing the new item code), and another integer (representing the price of the new item) as arguments. This method must update the item with the lowest price in the items array (update both the item code and price). Note that if the item code already exists in the array, you should update that item instead to have the new price. This method should not return anything (as food for thought, think about why we don't necessarily need to return the updated array back to the main method).

Submitting

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

- `VendingMachine.java`

Make sure you see the message stating, "Homework 1 submitted successfully." We will only grade your last submission be sure to **submit *every file* each time you resubmit.**

Checkstyle

You must run Checkstyle on your submission. (To learn more about Checkstyle, examine our course's external website Java Resources page under "CS 1331 Style Guide".) The Checkstyle cap for this assignment is **5 points**. If you don't have Checkstyle yet, download it from Canvas → Files → Resources → checkstyle-8.28.jar. You can also find it on the course website under "Java Info." 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 above). The Java source files we provide contain no Checkstyle errors. In future assignments, we will be increasing this cap, so get into the habit of fixing these style errors early!

Import Restrictions

You may only import the Random class (if needed).

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`

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 Ed Discussion 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 "Import Restrictions" and "Feature Restrictions" to avoid losing points

- Check on Ed Discussion for all official clarifications
- Make sure to test your program manually based on the assignment instructions and expected outputs.

The Gradescopeautograder 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.