Programming Exercise 04 – Dining at Tech

Authors: David, Eric
Topics: enums, static methods, arrays

Problem Description

Having recently begun your out-of-state education at Georgia Tech, you have become increasingly aware of your spending habits. As a result, you decide to write a program to evaluate the different food options around campus. With this tool, you will hopefully make the frugal choice!

Solution Description

Write `Food.java` to represent the different food items that a meal may consist of, and `TechDining.java` to create orders. In this assignment, we use the following definitions:

- "meal" refers to a list of food items represented as an array of `Food` element(s)
- "order" refers to a list of meals represented as an array of `Food` array(s)

Notes:

- You can invoke the `values` method on an enum type to retrieve an array containing the constant values of that enum type (e.g., `Food.values()`).
- Reuse code as much as possible (i.e., if you've already written a method to perform some behavior, you should call that method instead of rewriting the implementation).

Food.java

This public `enum` defines the possible values of `Food` that a meal may consist of.

- Enumerate the possible values of `Food` as follows, maintaining the order as listed:
  - RAT_ON_A_STICK
  - WILLAGE_BURNT_RICE
  - NAVE_RAW_CHICKEN
  - BRITTAIN_FRIED_SQUIRREL
  - TWISTED_TACO
  - PANDA_EXPRESS

TechDining.java

This public class defines various **static methods** to create and calculate the cost of meals and orders.

Methods:

- `createMeal`
  - Takes in an `int` representing the length of the meal.
    - You may assume this value will not be negative.
  - Return a `Food` array of the specified length representing the meal.
  - Populate each slot in this `Food` array such that each kind of food has an equally likely chance of being selected.
    - **Hint:** Generate a random integer in the appropriate range!
- `createOrder`
  - Takes in an `int` representing the number of meals in the order. This parameter also represents the length of the largest meal.
    - You may assume this value will not be negative.
  - Return an array of `Food` arrays representing the order.
  - The largest meal should occupy index 0 in the returned array. Subsequent meals in the array should have one less item than the meal at the previous index.
    - E.g., If the length passed in is 3, then this method should return an array containing a meal of length 3 at index 0, a meal of length 2 at index 1, and a meal of length 1 at index 2.
  - Each meal in the order should be populated with `Food` items such that each kind of food has an equally likely chance of being selected.
    - **Hint:** Reuse code that generates a random meal of a particular length!
- `mealCost`
  - Takes in a `Food` array representing the meal of which to calculate the cost.
    - You may assume that the reference to the array passed in will be non-null, and the array it references will not contain any null elements.
  - Return an `int` representing the cost of the meal.
  - Calculate the cost of the meal, where the price of a given `Food` element is calculated as the product of its index in the meal and the index of that food as it was enumerated in the `Food` enum.
    - **Note:** Do **NOT** hardcode the index where a food is enumerated in the `Food` enum.
    - **Hint:** You can invoke the ordinal method on an enum constant to get the index at which it is enumerated in the enum.
- `orderCost`
  - Takes in an array of `Food` arrays representing the order of which to calculate the cost.
    - You may assume that the reference to the array passed in will be non-null. You may also assume that any meals (i.e., `Food` arrays) it contains will not be null.
    - In addition, you may assume that the food in the meal arrays within the order will not contain any null elements.
    - In summary, you may assume that a reference in any element in any array is non-null.
  - Return an `int` representing the cost of the order.
  - The cost of the order is the sum of the cost of each meal in the order.
    - **Hint:** Reuse code that calculates the cost of a meal!

Driver.java

This public class serves as a tool to help you test your code. We've suggested a few test cases that you should write at a minimum to get you started, but you should also write your own test cases to verify the correctness of your code against a wide variety of inputs and scenarios.

Methods:

- `main`
  - Create at least two meals.
    - Each may have any length of your choice.
  - Determine the cost of the two meals.
  - Create at least two orders.
    - Each may have any length of your choice.
  - Determine the cost of the two orders.

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 5 points.** This means there is a maximum point deduction of 5. 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!

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:

- `Food.java`
- `TechDining.java`
- `Driver.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.Random`

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).

You **MAY NOT** use code generation tools to complete this assignment. This includes generative AI tools such as ChatGPT.

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.