CS 152 Computer Programming Fundamentals
Project 7: Dish and Menu Objects

Brooke Chenoweth

Spring 2024

To give you some practice creating and using classes, you are going to write two classes for this project: `Dish` and `Menu`. You will have to create the `Dish` class from scratch, but don't worry, I give a detailed description of what I expect below. For the `Menu` class, you'll be given a skeleton file to fill in, but since it will build on your `Dish` class, don't even start it until you are done with your `Dish` implementation.

I have given you a file of testing code in `DishMenuTester.java` and a class containing some constants in the `FoodConstants.java` file. Do not change these files. Please note: these files will not compile without your classes.

1 Dish class

A `Dish` represents an item served in in a restaurant. A dish has a name, a price, a menu section, and some food group classifications.

You will create `Dish.java` from scratch. Don't forget to comment all your public methods and use appropriate access modifiers on your member variables.

1.1 Constructors

`Dish` has three constructors:

- `public Dish()`

  The default (no parameters) constructor, creates a new Dish with unknown name, price, and menu section and with empty food groups.

- `public Dish(String menuSection)`

  Creates a new Dish with the given menu section, unknown name and price, and with empty food groups.

- `public Dish(String menuSection, String name, String foodGroups, double price)`

  Creates a new Dish with all member variables initialized with the values given.

1.2 Methods

A `Dish` has at least the following methods: (You can make more if you feel you need to, for helping your other methods, but I will be calling and testing these.)

- `public void setName(String name)`

  Sets the dish's name. Name must not be empty.

- `public String getName()`

  Returns the String representing the dish's name.

- `public void setPrice(double price)`

  Set the dish's price. The price must not be negative.

- `public double getPrice()`

  Returns the double value representing the dish's price.

- `public void setMenuSection(String section)`

  Set the dish's menu section.

- `public String getMenuSection()`

  Returns the string representing the dish's menu section.

- `public void setFoodGroups(String foodGroups)`

  Set the dish's food groups string. However, we cannot just assign the parameter to the member variable. We only want to include the known food group characters given in the `FoodConstants` class, and ignore any other unknown characters. We also want to ignore any duplicate characters.

  So, if we call this method with an argument of "mdvdxmq", the food groups string should be set to "mdv". (Setting food groups to meat, dairy, and vegetable, ignoring the unknown characters 'x' and 'q' and the duplicate dairy and meat.)

- `public String getFoodGroups()`

  Returns the string representing the dishes food groups.

- `public String toString()`

  Should return a string representation of the dish.

  - If food groups are empty and price is unknown:
    `Carrot Cake, DESSERT`
  - If food groups are empty and price is known:
    `Carrot Cake, DESSERT: $9.99`
  - If food groups are given and price is unknown:
    `Carrot Cake (nvdg), DESSERT`

  - If all fields are known:

    ```
    Carrot Cake (nvdg), DESSERT: $9.99
    ```

- `public String toMenuString()`

  Returns a fancier string representation of the dish, suitable for printing on a menu. The menu section is not included in this string, since it is assumed this string will be printed in the appropriate menu section already. We also will list the food groups (if known) on a second line indented with five spaces and using the words defined in the `FoodConstants` class, rather than just the character abbreviations stored in the food groups member variable string.

  - If price is unknown and food groups are empty:

    ```
    Carrot Cake
    ```

  - If price is known and food groups are empty:

    ```
    Carrot Cake: $9.99
    ```

  - If price is unknown and food groups are known:

    ```
    Carrot Cake
         NUTS, VEGETABLE, DAIRY, GRAINS
    ```

  - If price and food groups are known:

    ```
    Carrot Cake: $9.99
         NUTS, VEGETABLE, DAIRY, GRAINS
    ```

- `public boolean isVegetarian()`

  Returns true if the dish vegetarian, false otherwise. A dish is vegetarian if its food groups do not contain meat.

- `public boolean isVegan()`

  Returns true if the dish vegan, false otherwise. A dish is vegan if its food groups do not contain meat or dairy.[1]

- `public boolean isSame(Dish other)`

  Returns true if the two dishes are the same, false if not.

  Two dishes are the same if they both have unknown names, or if they have the same name, price, menu section, and food groups (in any order).

[1] Yes, I know this is ignoring other animal products like eggs, but we're trying not to overcomplicate this too much. A "real" application could have more detail (full ingredient lists!), which would not necessarily be harder conceptually to implement, but certainly would be more tedious and time consuming

2 Menu class

I am providing a skeleton of `Menu.java` for you to fill in.

The member variables and constructor have been provided for you, but you will have to fill in the following methods to complete the class and pass the tests.

Do not change the member variables provided, just initialize them in the constructor and use them in the methods.

2.1 Member Variables

- The name of the menu is stored in a string.
- An array of Strings holds the names of the menu sections.

  The order of the sections in this array corresponds to the order of the section information stored in the next two arrays.
- An array of integers stores how many dishes are currently in each section.
- An array of array of dishes holds the dishes for each section.

  This is a "ragged" array, where each row in the array may be of a different length. The capacity of each section is specified in the constructor.

2.2 Constructors

`Menu` has one constructor:

- `public Menu (String name, String[] sections, int[] capacity)`

  Initialize an empty menu with the given name and sections, with ability to hold number of dishes in each section as specified by the capacity array.

2.3 Methods

- `public int getMenuSize()`

  Get the total number of dishes in this Menu.
- `public String getName()`

  Get the name of the menu.
- `public void setName(String name)`

  Set the menu's name. Name must not be empty.
- `public String addDish(Dish d)`

  Add a dish to the menu in the appropriate row for its menu section. If the dish is already present, do not add a duplicate. If the dish is new, add it after the existing dishes in its section.

  Returns a string to denote success or failure, one of the following:

```
Section not found
Ignoring known dish
Menu section is full
New dish added
```

- `public String[] addDishes(Dish[] dishes)`

  Add an array of dishes to the menu. Return an array of status strings as described in `addDish` to indicate success or failure for each dish.

- `public Dish[] getDishes()`

  Get an array of all the dishes on the menu.

- `public String toString()`

  Get a string representation of the menu, consisting of the menu name on the first line, followed by all the dishes using their `toString()` format.

  For example:

```
Summer Menu
Pad Thai (nvdmg), MAIN: $17.99
Cashew Chicken and Rice (dngmv), MAIN: $17.99
Vegetable and Nut Pilaf (nvdg), MAIN: $14.99
Pad See Ew (mvg), MAIN: $16.99
Clam Chowder (mvd), SOUP: $14.99
Beef Stew (mvd), SOUP: $8.99
Vegetable Stew (vgd), SOUP: $7.99
Winter Salad (vgn), SIDE: $7.99
Vegetable Fried Rice (vgd), SIDE: $8.99
Southwest Salad (nvdmg), SIDE: $8.99
```

- `public String getFullMenu()`

  Get a string of the entire menu, grouped by menu section and using Dish's `toMenuString()` format, as follows:

```
*** Summer Menu ***
MAIN:
- Pad Thai: $17.99
     NUTS, VEGETABLE, DAIRY, MEAT, GRAINS
- Cashew Chicken and Rice: $17.99
     DAIRY, NUTS, GRAINS, MEAT, VEGETABLE
- Vegetable and Nut Pilaf: $14.99
     NUTS, VEGETABLE, DAIRY, GRAINS
- Pad See Ew: $16.99
     MEAT, VEGETABLE, GRAINS
```

```
SOUP:
- Clam Chowder: $14.99
     MEAT, VEGETABLE, DAIRY
- Beef Stew: $8.99
     MEAT, VEGETABLE, DAIRY
- Vegetable Stew: $7.99
     VEGETABLE, GRAINS, DAIRY
SIDE:
- Winter Salad: $7.99
     VEGETABLE, GRAINS, NUTS
- Vegetable Fried Rice: $8.99
     VEGETABLE, GRAINS, DAIRY
- Southwest Salad: $8.99
     NUTS, VEGETABLE, DAIRY, MEAT, GRAINS
```

- `public String getVegetarianMenu()`

  Create a menu of only the vegetarian dishes in the same format as getFullMenu, but indicate vegan dishes with an asterisk, as follows:

```
*** Summer Menu ***
MAIN:
- Vegetable and Nut Pilaf: $14.99
     NUTS, VEGETABLE, DAIRY, GRAINS
SOUP:
- Vegetable Stew: $7.99
     VEGETABLE, GRAINS, DAIRY
SIDE:
* Winter Salad: $7.99
     VEGETABLE, GRAINS, NUTS
- Vegetable Fried Rice: $8.99
     VEGETABLE, GRAINS, DAIRY
```

- `public boolean removeDish(Dish d)`

  Bonus problem! This method deletes a dish entirely from the menu, if it exists. It must correctly update all relevant member variables to not leave a hole in a menu section or break any other bookkeeping.

  Returns true if dish was removed and false if it was not found.

3 General Notes

3.1 Invalid Values

When trying to run a setter method with an unallowable variable, you should leave the previous value unchanged. Printing error messages when this happens is a good idea, however the text of them is up to you.

3.2 Uninitialized Values

There are a series of constants provided for you in the `FoodConstants` class. You should use these variables for dish properties that have not yet been set.

3.3 Access Modifiers

The member variables for your classes should be private. The methods and constructors listed above should be public. If you make any additional methods, it is up to you which access modifier you use, but do pick one. Consider if anyone outside of your class might need access to your helper method.

4 Testing Your Code

4.1 Writing Your Own Tests

The provided test code will not compile until you have implemented the `Dish` and `Menu` classes. I strongly suggest that you create your own main method in each of these two classes to test your methods as you are developing them. Java will allow you to put a main method into any class you like, so have at it!

4.2 Provided Test Code

The graders will use the `DishMenuTester` class to test your code.

The expected output if you pass all the tests looks like the following. (There may be additional output if you printed some error messages for invalid values, but it will be something like this.)

```
*** TESTING DISH IMPLEMENTATION ***
Attempting constructors:
  - Constructors seem functional: 7/7
  - toString before setters: 2/2
  - isSame before setters: 2/2
Attempting setters and getters:
  - setName & getName: 4/4
  - setMenuSection & getMenuSection: 2/2
  - setFoodGroups & getFoodGroups: 4/4
  - setPrice & getPrice: 4/4
Attempting other methods:
  - toString: 5/5
  - toMenuString: 5/5
  - isSame: 3/3
  - isVegetarian: 3/3
  - isVegan: 4/4
Dish implementation score: 45/45
```

```
*** TESTING MENU IMPLEMENTATION ***
Attempting initialization:
  - Initialization seems functional: 3/3
Attempting setters and getters:
  - getMenuSize: 3/3
  - getDishes: 3/3
  - setName and getName: 2/2
  - adding to empty menu: 2/2
Attempting other methods:
  - addDish: 7/7
  - addDishes: 2/2
  - getFullMenu: 6/6
  - getVegetarianMenu: 6/6
  - toString: 6/6
  - *Bonus* removeDish: 5/5
Menu implementation score: 45/40 (max 45)
```

5 Turning in your assignment

Submit your `Dish.java` and `Menu.java` files to Canvas. Do not attach `.class` files or any other files.

6 Grading Rubric (total of 95 points + 5 bonus)

-5 points File submitted to Canvas was not correctly named.

-5 points The code did not compile without errors or warnings.

10 points The code adheres to the coding standard specified on the course website.

45 points Score from dish test in `DishMenuTester`

40 points (+ possibly 5 bonus) Score from menu test in `DishMenuTester`

Note: you may not get the full points for these tests if examination of your code indicates that you hard-coded answers to these specific test cases or otherwise did not actually implement the class as specified.