CS 101 - Algorithms & Programming I

Fall 2021 - Lab 8

Due: Week of December 6, 2021

*Remember the **honor code** for your programming assignments.*
*For all labs, your solutions must conform to the CS101 style **guidelines**!*
All data and results should be stored in variables (or constants where appropriate) with meaningful names.

The objective of this lab is to learn how to define custom classes and understand object references. As you know an object-oriented software is essentially a collection of interacting objects (class instances) that know each other through these references. Remember that analyzing your problems and designing them on a piece of paper *before* starting implementation/coding is always a best practice. Specifically for this lab, you are to both organize your data and methods working on them.

0. Setup Workspace

Start VSC and open the previously created workspace named `labs_ws`. Now, under the `labs` folder, create a new folder named `lab8`.

In this lab, you are to have four Java classes/files (under `labs/lab8` folder) as described below. We expect you to submit a total of 4 files including the revisions (`User` class, `Food` class, `Order` class, and `FoodBasket` class where you define the `main` method). Do *not* upload other/previous lab solutions in your submission. Outputs of sample runs are shown as **brown**.

In this lab, you are going to implement the system of a food delivery application called *FoodBasket*. FoodBasket maintains the relationships between foods, users, and orders to carry out the basic functionalities of the application. The system is composed of 3 classes (`User` class, `Food` class, `Order` class) and a `FoodBasket` class which contains only the `main` method of the application to drive the functions with test cases. You can find the details of the classes below.

The class details and the revision requirements on the lab day present the minimum conditions that you need to meet for this lab, but you are not limited to these! You are welcome to get creative and add additional features to *your* FoodBasket application to make it better.

Important Note: All of the data members are to have **private** access modifiers, and should be obtained or modified with accessor or mutator methods, respectively.

1. User Class

Create a new/empty file to define a class named `User`. This class defines the details of a user in the application.

Static Data Members:

- `count`: Number of users in the system.

Instance Data Members:

- `ID`: Unique identification number given to each user. (Hint: you can assign this by using `count`)
- `name`: Stores the name of the user.
- `age`: Stores the age of the user.
- `orders`: This is a *String* to store the `ID` of the orders that the user makes. Multiple entries are separated by a single dash (“-”) character in this string.

- `numberOfOrders`: Stores the total number of orders the user has made so far.
- `eatenFoods`: This is a *String* to store the names of the unique foods the user has tried so far. Multiple entries are separated by a single dash ("-") character in this string.

Methods:

- **Constructor**: This constructor takes two parameters for `name` and `age`. It assigns an ID to the user and initializes all of the instance data members either with given parameters or with proper initial values. Lastly, it calls the `addUser()` method; please see below for this method's specification.
- **Accessor and Mutator Methods for** `ID`, `name`, `age`, `orders`, `numberOfOrders`, `eatenFoods`, and `count` (only an accessor is necessary for this particular data member).
- **Other Methods**:
  - `addUser()`: Increments `count` by one.
  - `equals(User user)`: This method takes a User object as the parameter and returns true if the `ID`, `name`, and `age` values of the target user and the user passed as a parameter are the same. It returns false otherwise.
  - `didEat(Food food)`: Returns true if the user has eaten the food given in the parameter. (Hint: you can check the `eatenFoods` data of the user to decide.)
  - `pickOrderbyIndex(int index)`: Returns the order `ID` as a String with the index. You should check that the argument is a valid index. Please assume that the indices start with 1.
  - `addNewOrder(Order order)`: Accepts an Order object as parameter and updates `orders`, `numberOfOrders`, and `eatenFoods` data members of the user.
  - `toString()`: Returns a String representation of a user. Please see the example output below.

Sample run:

```
User user1 = new User("Vedat Milor", 66);
User user2 = new User("John Doe", 32);

System.out.println(user1); // calls toString() method implicitly
System.out.println(user2);

System.out.println(user1.getName() + " made " + user1.getNumberOfOrders() + "
orders.");

System.out.println("user1.equals( user2 ) is " + user1.equals(user2));

************************
User ID: 0, User Name: Vedat Milor, Age: 66
User ID: 1, User Name: John Doe, Age: 32
Vedat Milor made 0 orders.
user1.equals( user2 ) is false
```

2. Food Class

Create a new/empty file to define a class named `Food`. This class defines the details of foods that are registered in the application.

Instance Data Members:

- `name`: *String* value to define the name of the food.
- `ingredients`: *String* value to store the ingredients of the food separated by a single comma (“,”).
- `calories`: *int* value to hold the total calories in one serving of the food.
- `type`: *String* value to define the course of food such as “mainCourse” or “dessert”.
- `price`: *double* value to store the price of one serving of the food.

Methods:

this.name = name //initialized +

- **Constructor**: This constructor takes two parameters for `name` and `price`. It initializes all of the instance data members either with given parameters or with proper initial values.
- **Constructor**: This constructor takes five parameters for `name`, `ingredients`, `calories`, `type`, and `price`. It initializes all of the instance data members with given parameters.
- **Accessor and Mutator Methods for** `name`, `ingredients`, `calories`, `type`, and `price`.
- **Other Methods**:
  - `equals(Food food)`: This method takes a Food object as the parameter and returns true if the values of `name` and `ingredients` in the target food and the food passed as a parameter are the same. It returns false otherwise.
  - `addIngredient(String newIngredient)`: Adds the new ingredient to the `ingredients` member of the food. Each ingredient is separated by a single comma (“,”) in `ingredients`.
  - `doesContain(String ingredient)`: Returns true if the food includes the given ingredient, false otherwise.
  - `toString()`: Returns a *String* representation of a Food object. Please see the example output below.

Sample run:

```
Food food1 = new Food("Manti", 200);
food1.setIngredients("flour, meat or veggies, salt, water");
food1.setCalories(50);
food1.setType("main course");

Food food2 = new Food("Ezogelin Soup", "red lentils", 187, "soup", 20);
food2.addNewIngredients("bulgur");

System.out.println(food1);
System.out.println(food2);

**************************
Manti is a main course dish.
It includes flour, meat or veggies, salt, water.
Single portion contains 50 calories.
Single serving cost = 200.0

Ezogelin Soup is a soup dish.
It includes red lentils, bulgur.
Single portion contains 187 calories.
Single serving cost = 20.0
```

3. Order Class

Create a new/empty file to define a class named `Order`. This class defines the details of an order in the application. Each order contains exactly one food.

Static Data Members:

- `count`: Number of orders in the system.

Instance Data Members:

- `ID`: Unique identification number given to each order. (Hint: you can assign this by using `count`)
- `orderedFood`: *Food* instance of the order.
- `portion`: *double* value to define the serving size of the food in the order.
- `totalPrice`: *double* value to hold the total price of the order. Unless `checkout()` is called, this variable should be equal to 0.0; please see below for this method's specification.

Methods:

- **Constructor**: This constructor takes three parameters for `portion`, `name`, and `price`. It creates the `orderedFood` instance of the order, assigns the `ID`, and initializes other instance data members either with given parameters or with proper initial values. Lastly, it calls the `addOrder()` method (please see below for this method's specification).
- **Constructor**: This constructor takes two parameters for `portion` and `orderedFood`. It assigns the `ID` and initializes other instance data members either with given parameters or with empty values. Lastly, it calls the `addOrder()` method (please see below for this method's specification).
- **Accessor and Mutator Methods for** `ID`, `orderedFood`, `portion`, `totalPrice`, and `count` (only accessor for this particular data member).
- **Other Methods**:
  - `addOrder()`: Increments `count` by one.
  - `equals(Order order)`: This method takes an Order object as the parameter and returns true if the value of `ID` in the target order and the order passed as a parameter are the same. It returns false otherwise.
  - `checkout()`: Calculates and sets the `totalPrice` of the order by multiplying the portion of the order with the price of the food.
  - `increasePortion(int extraPortion)`: Increases the portion by the given amount.
  - `toString()`: Returns a *String* representation of an Order object. If the `totalPrice` is equal to 0.0, in other words, if `checkout()` is not called, the *String* should first include a warning message stating that the order is incomplete. Please see the example output below.

Sample run:

```
Order order1 = new Order(1, "Pizza", 50);
Order order2 = new Order(4, food1);

System.out.print(order1);
System.out.print(order2);
order1.increasePortion(2);

order1.checkout();
System.out.print(order1);
**************************
>> Warning: This order is incomplete.
```

```
Details for Order 0:  // "0" shows the ID
        Pizza(x 1.0)  // format = "name of food (x portion)"
        TOTAL = 0.0   // notice that checkout() has not been called yet

>> Warning: This order is incomplete.
Details for Order 1:
        Manti(x 4.0)
        TOTAL = 0.0   // notice that checkout() has not been called yet

Details for Order 0:
        Pizza(x 3.0)
        TOTAL = 150.0
```

4. Food Basket

FoodBasket class is given to you and it is the driver of your project. It has the main method of your application and includes some test codes together with expected similar outputs. Feel free to add new tests underneath the given codes. Your TA may add new conditions to test this class on lab day for revisions.