XUnit Practice - OOP

Deadline: Thursday, September 10th 2020, 09:00

GitHub Classroom Repository

Introduction

This practice is designed to help you familiarize yourself with the fundamentals of XUnit in an object-oriented implementation.

Requirements

- Create a new solution with an XUnitOOPPractice console application.
- Add a second XUnit Test project to the solution called XUnitOOPPractice_Tests.
- Add an assembly reference from the XUnit project to the console application.
- Create a public abstract class called MenuItem with the following properties:
  - Price (double, abstract)
- Create inherited classes from MenuItem called “FoodItem” and “Combo”
  - Combo should have a list property of FoodItems called “ComboItems”.
  - Combo’s Price should be a sum of all of the combo Items prices, minus 20%.
  - FoodItem’s Price should be a normal double.
  - FoodItem should have a “Type” enum and property thereof:
    - FrenchFries
    - ChickenStrips
    - Drink
- Create a public class called Order with the following properties / methods:
  - Items (polymorphic list)
  - ItemCount (int), a getter only property of the count of the items.
  - AddItem()
    - Accepts polymorphic argument and adds it to the list.
    - If an item is added, and because of that addition there is an instance of each food item (Fries, Chicken Strips and Drink), then add all three to a new Combo. Remove the individual FoodItems from the Order’s Items list, and add the Combo to the Items list.
  - RemoveItem()
    - Removes the most recent addition to the Order’s Items list, if it’s a Combo, remove the whole combo.
  - Total (double), a function of the sum of all of the prices in Items.
- Create unit tests to test the following behaviour:
  - Adding a single item of each type to the order (3 facts).
  - Add a Combo to the order (fact).
  - Remove a single item from the order (fact).

    - Remove a combo from the order (fact).
    - The Combo's Price reflects the sum of the items minus 20% (theory).

Challenges

- Add full unit tests for all classes, testing teach property and method with all edge cases and use cases.
- Support additional food items with multiple types of combos (main item, supplement item, drink).
    - Example of main item: Burger, chicken nuggets, wrap
    - Example of side item: fries, onion rings
- Support multiple sizes of items, use enums (small, medium, large for drinks / fries vs 5, 10 and 20 pieces for chicken nuggets).
- Support coupon items that will have a type of item they apply to that must be in the order in order to add the coupon item, and it will add a negative price to the total (to apply the discount).
    - The RemoveItem() will not remove the coupon(s), unless the item it is tied to is removed.

Reminders

Ensure you are tracking your time and that your timesheet is in the appropriate folder for viewing and marking.

Ensure you are committing frequently. It is advised to commit once per successful feature implementation at a minimum.

Ensure you are pushing your repository to the GitHub Classroom repository, and not a personal, private repository.

Ensure your repository has a readme with at a minimum your name, the name of the project, the project's purpose and a link to your Trello board.

Ensure you are using Trello appropriately to keep track of outstanding features, and that it is linked in the README.md file.

Ensure you are using the #class channel to request clarifications on assignment specifications.

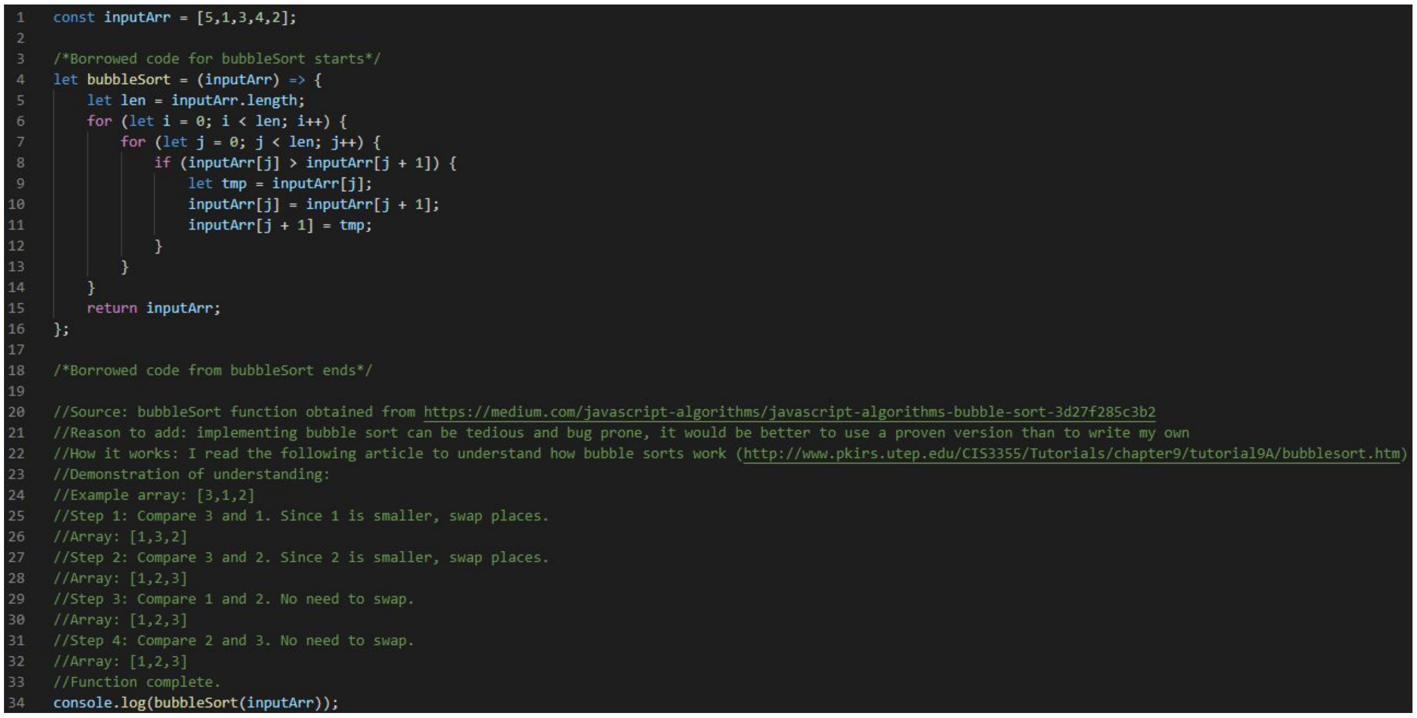
Ensure you are using the #homework-help channel in slack to request for assistance from instructional staff if needed, and that you include (at a minimum) a specific description of the problem and a list of what you have tried.

Feel free to reach out on #peer-support if the #homework-help queue is lengthy. If someone helps you out, give them a shoutout using our handy-dandy form!

Citation Guide for Borrowed Code

Whenever you borrow code, the following information must be included:

- Comments to indicate both where the borrowed code begins and ends.
- A source linking to where you found the code (URL, book, example, etc.).
- Your reason for adding the code to your assignment or project instead of writing it out yourself.
- Explain to us how the code is supposed to work, include links to documentation and articles you read to help you understand.
- A small demonstration to prove you understand how the code works.



```javascript
const inputArr = [5,1,3,4,2];

/*Borrowed code for bubbleSort starts*/
let bubbleSort = (inputArr) => {
    let len = inputArr.length;
    for (let i = 0; i < len; i++) {
        for (let j = 0; j < len; j++) {
            if (inputArr[j] > inputArr[j + 1]) {
                let tmp = inputArr[j];
                inputArr[j] = inputArr[j + 1];
                inputArr[j + 1] = tmp;
            }
        }
    }
    return inputArr;
};

/*Borrowed code from bubbleSort ends*/

//Source: bubbleSort function obtained from https://medium.com/javascript-algorithms/javascript-algorithms-bubble-sort-3d27f285c3b2
//Reason to add: implementing bubble sort can be tedious and bug prone, it would be better to use a proven version than to write my own
//How it works: I read the following article to understand how bubble sorts work (http://www.pkirs.utep.edu/CIS3355/Tutorials/chapter9/tutorial9A/bubblesort.htm)
//Demonstration of understanding:
//Example array: [3,1,2]
//Step 1: Compare 3 and 1. Since 1 is smaller, swap places.
//Array: [1,3,2]
//Step 2: Compare 3 and 2. Since 2 is smaller, swap places.
//Array: [1,2,3]
//Step 3: Compare 1 and 2. No need to swap.
//Array: [1,2,3]
//Step 4: Compare 2 and 3. No need to swap.
//Array: [1,2,3]
//Function complete.
console.log(bubbleSort(inputArr));
```