XUnit Practice - Cumulative

Deadline: Monday, September 14th 2020, 09:00AM

GitHub Classroom Repository

GitHub Solution 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 C# Console solution called “XUnitCumulativePractice”.
- Add a new xUnit Test Project (.NET Core) to the solution called “XUnitCumulativePractice_Tests”.
- Add an assembly reference from “XUnitCumulativePractice_Test” to “XUnitCumulativePractice”.
    - Do this by r-clicking on “XUnitCumulativePractice_Test” dependencies, then Add Project Reference to “XUnitCumulativePractice”
- Rename the default file “UnitTest1.cs” in the XUnit project to “Program_Tests.cs”.
- Ensure your “Program” class is public in the XUnitCumulativePractice project.
- Create a method in XUnitCumulativePractice.Program.cs called ValidateOneToOneHundredEven() that will accept a string (“1” or “20” etc.) as a parameter.
    - Validate that the string:
        - Is an integer (numeric, will safely parse to int).
        - Is between 1 and 100.
        - Is even.
    - If it is, return true. Otherwise return false.
- Create test cases in Program_Tests.cs that will test each of the requirements of ValidateOneToOneHundredEven().
    - This is to say, we should be testing (Theories):

Test Data Is: An Integer	Test Data Is: Between 1 and 100	Test Data Is: Even Number	Assertion: ValidateOneToOneHundredEven() Returns:
false	false	false	false
true	false	true	false
true	false	false	false
true	true	false	false
true	true	true	true

    - Don’t forget to include edge cases (right on the border of acceptable, example if we wanted between 1 and 5, we would test at least -1, 0, 1, 5 and 6).

    - Please include at least 3 sets of test data per row.
- Create a class called “Item” with the following properties:
    - “Name” (string).
    - A default and greedy constructor.
- Create a class called “Storage” with the following properties and methods:
    - “Contents” (list of Items).
    - AddItem() will accept an item and add it to the “Contents” list.
    - RemoveItem() will remove the most recent addition to the “Contents” list.
    - A default constructor.
- Add a tests file to the XUnit project called Storage_Tests.cs.
    - Test the addition of items (Fact):
        - (Arrange) Create an Item named “TestItem” and an empty Storage.
        - (Act) Add the Item to the Storage using the AddItem() method.
        - (Assert) Assert that the list has a count of 1
        - (Assert) Assert that the item in Storage.Contents is “TestItem”.
    - Test the removal of items (Fact):
        - (Arrange) Create a Storage prepopulated with 5 Items.
        - (Arrange) Create an object reference variable to the last item in the Contents list.
        - (Act) Run RemoveItem() on the Storage.
        - (Assert) Assert that the count of the Contents has dropped by 1 (is 4)
        - (Assert) Assert that the object reference is no longer in the list.

Challenges

- Convert the Facts for testing the object into Theories, and test multiple runs of each for expected behaviour.
- Make the “Contents” property a polymorphic list of “Items”, derive some classes from “Item” and test for expected behaviour of your choosing.

Reminders

XUnit has two types of tests, a "Fact" which is typically declared on a method with no parameters, and a "Theory" which is typically declared on a method with parameters, and has "InlineData" associated therewith. (XUnit Cheatsheet)

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.

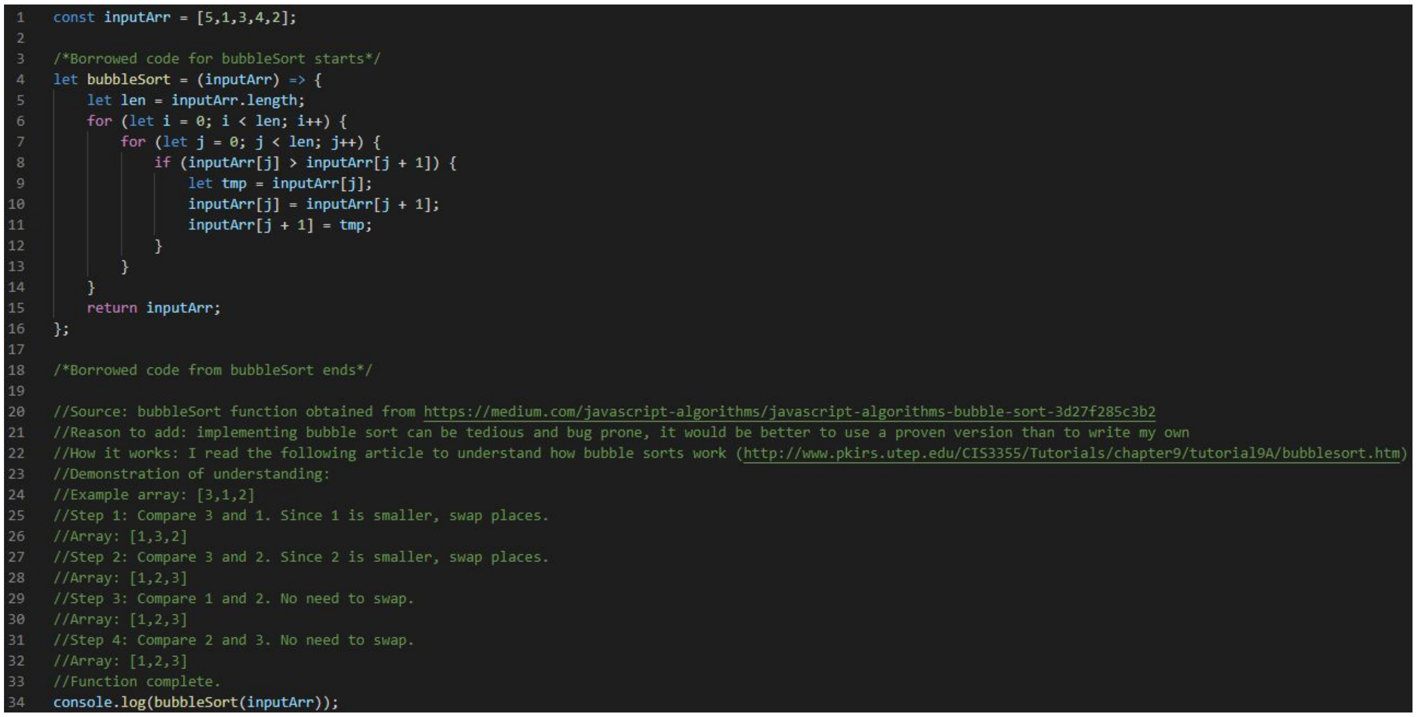
Ensure you are using the #homework-help channel in slack to request for assistance from instructional staff if needed, and that you include (at a minimum) a specific description of the problem and a list of what you have tried.

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));
```