C# OOP Practice – Shape Class

Deadline: Friday, September 4th 2020, 09:00AM

GitHub Classroom Repository

Introduction

This practice is designed to help you familiarize yourself with inheritance, polymorphism, encapsulation and abstraction.

Requirements

- Create a folder called “Shapes”. All classes therein should have the “.Shapes” namespace extension applied to them.
- Create a “Shape” class with an Area property (public getter only) that **must** be overwritten by derived classes.
- Create a “Circle” class that inherits from Shape with the following properties:
    - Circumference (public getter only)
    - Area (public getter only)
    - Radius (private getter and setter)
- Create a “Rectangle” class that inherits from Shape with the following properties:
    - Perimeter (public getter only)
    - Area (public getter only)
    - Length (private getter and setter)
    - Width (private getter and setter)
- Create a “Triangle” class that inherits from Shape with the following properties:
    - Perimeter (public getter only)
    - Area (public getter only)
    - Width (private getter and setter)
    - Height (private getter and setter)
- Create constructors for all three shapes that require their private properties as arguments.
- Create a “Drawing” class in the root assembly (not “.Shapes”) that has the following properties and methods:
    - SpaceCovered (public getter only), which will get the total area of all shapes drawn.
    - LinesDrawn (public getter only), which will get the total perimeter/circumference of all shapes drawn.
    - Shapes (private getter and setter), polymorphic list
    - Draw(Shape), which will accept a polymorphic argument and add that to the Shapes list.
    - A ToString() override, which will output “A drawing consisting of X shapes.” where X is the number of shapes in the polymorphic list.
- Add comments to specify at least one area of the program that demonstrates each pillar of OOP (see Introduction).

Challenges

- Research and implement getter arrow notation on the Perimeter / Area methods.

- Add a Colour enumeration to Shape - Red, Blue or Green, and require it in the constructors of each shape. In the ToString of drawing, output what percentage of the drawing is each colour, to the nearest hundredth of a percent. Example: “A drawing consisting of X shapes that is AA.AA% red, BB.BB% green and CC.CC% blue."

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.

```
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));
```