Using static keyword

https://csci-1301.github.io/about#authors

July 5, 2021 (02:53:58 PM)

Contents

1 Static classes

One use case for static classes is creating utility classes (or "helper class") that contain related and frequently-used methods; making those methods easily callable anywhere in the program. Some examples of static classes in C# are `Math` and `Console` class.

Pay attention to how these classes are used:

- `Console` object is never instantiated before use
- `WriteLine` method is called referring to the *name of the class* (not object identifier)

```
using System;

class Program {
    static void Main() {
        Console.WriteLine("calling a static method");
    }
}
```

Using your IDE, check what happens if you do the following:

```
using System;

class Program {
    static void Main() {
        Console test = new Console();
    }
}
```

Indeed, it is *not possible* to instantiate an object when a class is declared `static`. Further, if a class is declared static, all its members (attributes, methods, constructors, etc.) must also be declared `static`.

1.1 static Calculator

In your IDE create a new project. Then add a new class file called Calculator.cs

In Calculator.cs:

1. Declare a `static` class and name it `Calculator`.
2. Next add 5 `public` methods to `Calculator` class. Each method takes 2 arguments, `x` and `y`, of type `double`:

   a) `Add` method that returns result of `x + y`.
   b) `Subtract` method that returns result of `x - y`.
   c) `Multiply` method that returns result of `x * y`.
   d) `Divide` method that returns result of `x / y`.
   e) `Modulo` method that returns result of `x % y`.

After implementing `Calculator`,

1. Open the file that contains program's `Main` method
2. Paste the following code inside `Main` method:

   ```
   double x = 10d, y = 2d;

   Console.WriteLine($"{x} + {y} = {Calculator.Add(x, y)}");
   Console.WriteLine($"{x} - {y} = {Calculator.Subtract(x, y)}");
   Console.WriteLine($"{x} * {y} = {Calculator.Multiply(x, y)}");
   Console.WriteLine($"{x} / {y} = {Calculator.Divide(x, y)}");
   Console.WriteLine($"{x} % {y} = {Calculator.Modulo(x, y)}");
   ```

   Again, notice how

   - no instance of `Calculator` is created before use, and
   - each `Calculator` method is called referring to the *name of the class*.

3. Execute the program

   - If your implementation of `Calculator` class matches the instructions, you will see meaningful output after executing the program.
   - Otherwise review the instructions again and retrace your implementation steps to resolve any issues.

Review SimpleCalculator[1] for a sample solution.

2 Static members in non-static class

A non-static class can contain both static or non-static class members.

Study the following program implementation but *do not* execute it. After reading through the implementation, answer the questions below.

"Student.cs"

[1] SimpleCalculator.zip

```
using System;

class Student{

    private int id;
    private string name;
    private static string universityName = "Augusta University";
    private static int studentCount = 0;

    public Student(int id, string name){
        this.id = id;
        this.name = name;
        studentCount++;
    }

    public static void DisplayStudentCount(){
        Console.WriteLine($"Number of students: {studentCount}");
    }

    public override string ToString(){
        return $"id: {id}\n"+
               $"name: {name}\n"+
               $"university: {universityName}";
    }
}
```

"Program.cs"

```
using System;

class Program {
    static void Main() {

        Student alice = new Student(1111, "Alice");
        Console.WriteLine(alice);

        Student.DisplayStudentCount(); // first time

        Student bob = new Student(1112, "Bob");
        Console.WriteLine(bob);

        Student.DisplayStudentCount(); // second time
    }
}
```

1. How many non-static attributes does `Student` class have?
2. How many static attributes does `Student` class have?
3. How many non-static methods does `Student` class have?
4. How many static methods does `Student` class have?
5. What is the output of each of the following lines in "Program.cs"

    a) `Console.WriteLine(alice);`
    b) `Student.DisplayStudentCount(); // first time`

c) `Console.WriteLine(bob);`
d) `Student.DisplayStudentCount(); // second time`

6. if the `studentCount` attribute was *not* `static`, what would be the output of:

   a) `Student.DisplayStudentCount(); // first time`
   b) `Student.DisplayStudentCount(); // second time`

7. When a class contains both static and non-static members, is it possible to refer to non-static members inside a static method? For example, if we try to refer to `name` attribute inside `DisplayStudentCount`, will it work? Why or why not?

Check your answers by downloading and extracting StudentProgram[2] and executing it. To check the last question, uncomment line 16 in downloaded version of "Student.cs".

[2]StudentProgram.zip