Using static keyword

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

May 22, 2024 (01:30:41 PM)

Contents

This lab serves multiple goals:

- To teach you how a static class differs from a non-static one,
- To illustrate the usefulness of static classes,
- To teach you how a non-static class can manipulate static fields.

1 Static Classes – Warm-Up

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

Pay attention to how these classes are used:

- A `Console` object is never instantiated before use.

- The `WriteLine` method is called referring to the *name of the class* (not an object identifier):

  ```
  Console.WriteLine("calling a static method");
  ```

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

  ```
  Console test = new Console();
  ```

Solution:

Indeed, it is *not possible* to instantiate an object when a class is declared **static**. Furthermore, if a class is declared static, all its members (e.g., 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. Add 5 **public** methods to the `Calculator` class. Each method takes 2 arguments x and y of type `double`:

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

After implementing `Calculator`,

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

2 Static Members in a Non-static Class

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

Download, extract, and study this project[1] implementation, but *do not* execute it. After reading through the implementation, answer the questions below.

1. How many non-static attributes does the `Student` class have?

2. How many static attributes does the `Student` class have?

3. How many non-static methods does the `Student` class have?

4. How many static methods does the `Student` class have?

---

[1] labs/Static/Student.zip

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 the `name` attribute inside `DisplayStudentCount`, will it work? Why or why not?

Check your answers by creating a matching program in your IDE and executing it.

To check the last question, in Student.cs, uncomment the following line and verify its behavior matches your answer:

```
// Console.WriteLine(name);
```