Source: C# Corner (www.c-sharpcorner.com) PRINT

Article

Singleton Design Pattern In C# - Part Three (Static vs Singleton)

By **Akhil Mittal** on **Updated date Jan 11, 2018**

Introduction

In this series on learning singleton pattern, we learned lazy initialization and eager initialization with practical examples. We also learned why it is necessary to make the singleton class sealed with the sealed keyword. In this article, I'll try to explain the differences between static and singleton class. We will also see where to use static class and where to use singleton classes. Following are the links to the previous two articles of the series.

- Singleton Design Pattern In C# - Part One
- Singleton Design Pattern In C# - Part Two

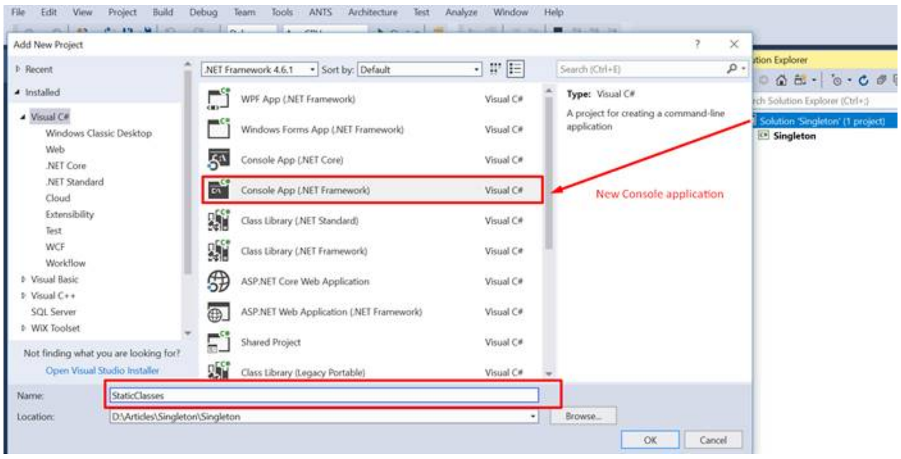
Singleton vs Static class

We'll not discuss in detail what static class is but rather focus more on the differences between singleton and static class. Following are the point to point differences between a static class and singleton pattern.

1. Singleton is an object creational design pattern and is designed following some guidelines to ensure it returns only one object of its class whereas static is a reserved keyword in C# which is used before class or method to make it static.
2. Singleton classes or design pattern can contain static as well as non-static members whereas if a class is marked static, it only should contain static members. For e.g. if a class is static, it should have static methods, static properties and static variables only.
3. Singleton methods could be passed as a reference to other methods or objects as a parameter but static members could not be passed as a reference.
4. Singleton objects could be created in a way where they support disposal; that means they could be disposed of.
5. Singleton objects are stored on heap whereas static objects are stored on the stack.
6. Singleton objects can be cloned.
7. Singleton promotes code re-usability and code-sharing and could implement interfaces as well. Singleton can inherit from other classes and promotes inheritance and have better control of object state whereas a static class cannot inherit its instance members.
8. Singleton class could be designed in a way where it can leverage lazy initialization or asynchronous initialization and is taken into consideration directly by CLR whereas static class is firstly initialized at the time of its first load.
9. Static classes cannot contain instance constructors and cannot be instantiated whereas the singleton classes have private instance constructor.

Static Class Implementation

Let's do some practical implementation to understand in more detail. Open the solution we created in the last article or a new one and add a new project of console application type and name it as StaticClasses.


File Edit View Project Build Debug Team Tools ANTS Architecture Test Analyze Window Help
Add New Project
Recent
Installed
Visual C#
Windows Classic Desktop
Web
.NET Core
.NET Standard
Cloud
Extensibility
Test
WCF
Workflow
Visual Basic
Visual C++
SQL Server
WiX Toolset
Not finding what you are looking for?
Open Visual Studio Installer
WPF App (.NET Framework) Visual C#
Windows Forms App (.NET Framework) Visual C#
Console App (.NET Core) Visual C#
Console App (.NET Framework) Visual C#
Class Library (.NET Standard) Visual C#
Class Library (.NET Framework) Visual C#
ASP.NET Core Web Application Visual C#
ASP.NET Web Application (.NET Framework) Visual C#
Shared Project Visual C#
Class Library (Legacy Portable) Visual C#
Search (Ctrl+E)
New Console application
Solution Explorer
Solution 'Singleton' (1 project)
Singleton
Browse...
OK
Cancel

.NET Framework 4.6.1 Sort by: Default

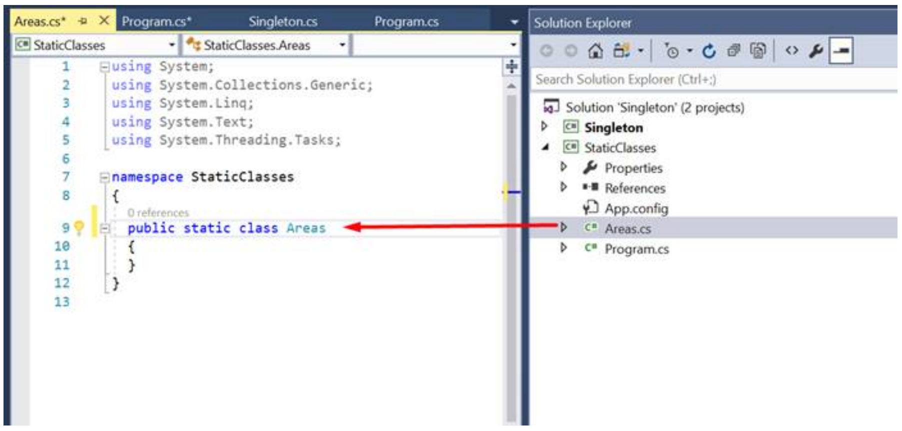
Type: Visual C#

A project for creating a command-line application

Name: StaticClasses

Location: D:\Articles\Singleton\Singleton

Now suppose we have a requirement in our project to create a utility class that holds the area of various shapes; for e.g. let's consider that the class contains the formulas to calculate the area of circle, square, rectangle and triangle. We are very much sure that the formula to calculate the area of the geometrical shapes always remains the same and is based on what input parameters we pass; for example a rectangle is the length of the rectangle multiplied by the width of the rectangle and that of a square is square of the length of the side. Since the area always remains the same, we can achieve this functionality with the help of a static class and static methods. So, create a class named Areas and make it static in the StaticClasses console application.



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace StaticClasses
{
    0 references
    public static class Areas
    {
    }
}
```

Now, let’s create static methods for calculating the area of circle, rectangle, square, and triangle.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using static System.Math;

namespace StaticClasses
{
  public static class Areas
  {
    /// <summary>
    /// Area of Circle is πr2.
    /// value of pi is 3.14159 and r is the radius of the circle.
    /// </summary>
    /// <returns></returns>
    public static double AreaOfCircle(double radius)
    {
      return PI * (radius * radius);
    }

    /// <summary>
    /// Area of Square is side2.
    /// Side * Side
    /// </summary>
    /// <returns></returns>
    public static double AreaOfSquare(double side)
    {
      return side * side;
    }

    /// <summary>
    /// Area of Rectangle is L*W.
    /// L is the length of one side and W is the width of one side
    /// </summary>
    /// <returns></returns>
    public static double AreaOfRectangle(double length, double width)
    {
      return length * width;
    }

    /// <summary>
    /// Area of Traingle is b*h/2.
    /// b is base and h is height of traingle
    /// </summary>
    /// <returns></returns>
    public static double AreaOfTraingle(double baseOfTraingle, double heightOfTraingle)
    {
      return (baseOfTraingle * heightOfTraingle)/2;
    }
  }
}
```

Since our class is static, it should only contain static methods, variables, and properties and so we defined the static methods to calculate the area of Rectangle, Square, Circle, and Triangle which takes input parameters and since formula always remains the same, then do some computation as per the formula and return the result to the caller. Note that I have used PI from static Math class since it is a static property and the value of pi always remains the same, we can use it as is to calculate the area of a circle. Note that I have defined a static class in the usings, this is the new way in which we can define static classes in usings and then can use their members directly in the class without the static class name. I did the same with System.Console class in the main method.

In the main method, we now call the methods by passing parameters to the methods in the following way.

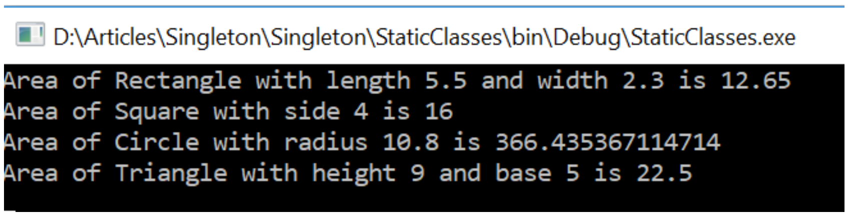
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using static System.Console;

namespace StaticClasses
{
  class Program
  {
    static void Main(string[] args)
    {
      double radiusOfCircle = 10.8;
      double lengthOfRectangle = 5.5;
      double widthOfRectangle = 2.3;
      double sideOfSquare = 4.0;
      double heightOfTriangle = 9.0;
      double baseOfTriangle = 5.0;

      WriteLine("Area of Rectangle with length {0} and width {1} is {2}", lengthOfRectangle, widthOfRectangle, Areas.AreaOfRectangle(lengthOfRectangle
      WriteLine("Area of Square with side {0} is {1}", sideOfSquare, Areas.AreaOfSquare(sideOfSquare));
      WriteLine("Area of Circle with radius {0} is {1}", radiusOfCircle, Areas.AreaOfCircle(radiusOfCircle));
      WriteLine("Area of Triangle with height {0} and base {1} is {2}", heightOfTriangle, baseOfTriangle, Areas.AreaOfTraingle(heightOfTriangle, baseOfTr
      ReadLine();

    }
  }
}
```

In the above mentioned main method, we call the static methods by directly referencing through class name Areas. Note that we do not have to create an object first to call the static methods and they could directly be called via class name. When we run the application, we get the following output.


D:\Articles\Singleton\Singleton\StaticClasses\bin\Debug\StaticClasses.exe

```
Area of Rectangle with length 5.5 and width 2.3 is 12.65
Area of Square with side 4 is 16
Area of Circle with radius 10.8 is 366.435367114714
Area of Triangle with height 9 and base 5 is 22.5
```

So, we can see that we can use static class and static methods working as a helper class to do straightforward tasks and get the result.

The choice between Static and Singleton

Now we know how to implement a static class and how to implement Singleton design pattern and make the class singleton. The question comes, when to use static and when to use a singleton. We can say that singleton gives us more flexibility over the design and more control over the design which we can change as per the demand and need for the requirement. So, the use of static or singleton is only as per your need looking at their pros and cons. In real-world scenarios, singleton could be used in logging, handling database connections, print spooling etc. Just to keep in mind that singleton is an architectural concept whereas static is something pre-available from .net framework. There are potential risks in using static in the cases where we need object disposal as static members do not get disposed until the application is ended and always remain in memory. Moreover, the static variables are global in a way where they are shared between the different applications and using it in web applications or ASP.NET applications could be risky as the variable would be shared to all the users regardless of their sessions because these variables will stay in an application domain. We have discussed a lot about static and singleton to make a choice while in development. I hope these pointers help you to decide when to use what.

<<Click here for previous part