C# Partial Class

In c#, a **partial class** is helpful to split the functionality of a particular class into multiple class files, and all these files will be combined into one single class file when the application is compiled.

While working on large-scale projects, multiple developers want to work on the same class (/tutorial/csharp/csharp-classes-and-objects-with-examples) file simultaneously. To solve this problem, c# provides an ability to spread the functionality of a particular class (/tutorial/csharp/csharp-classes-and-objects-with-examples) into multiple class (/tutorial/csharp/csharp-classes-and-objects-with-examples) files using `partial` keyword (/tutorial/csharp/csharp-keywords-reserved-contextual).

In c#, we can use `partial` keyword (/tutorial/csharp/csharp-keywords-reserved-contextual) to split the definition of a particular class (/tutorial/csharp/csharp-classes-and-objects-with-examples), structure (/tutorial/csharp/csharp-structures-structs), interface (/tutorial/csharp/csharp-interface)**,** or method (/tutorial/csharp/csharp-methods-functions-with-examples) over two or more source files.

Following is the example of splitting the definition of **User** class (/tutorial/csharp/csharp-classes-and-objects-with-examples) into two class (/tutorial/csharp/csharp-classes-and-objects-with-examples) files, **User1.cs** and **User2.cs**.

User1.cs

```
public partial class User
{
    private string name;
    private string location;
    public User(string a, string b)
    {
        this.name = a;
        this.location = b;
    }
}
```

If you observe the above code, we created a partial class called **User** in **User1.cs** class file using `partial` keyword (/tutorial/csharp/csharp-keywords-reserved-contextual) with required variables (/tutorial/csharp/csharp-variables-with-examples) and constructor (/tutorial/csharp/csharp-constructors-with-examples).

User2.cs

```
public partial class User
{
     public void GetUserDetails()
     {
        Console.WriteLine("Name: " + name);
        Console.WriteLine("Location: " + location);
     }
}
```

If you observe the above code, we created a partial class called **User** in **User2.cs** class file using `partial` keyword (/tutorial/csharp/csharp-keywords-reserved-contextual) with **GetUserDetails()** method (/tutorial/csharp/csharp-methods-functions-with-examples).

When you execute the above code, the compiler will combine these two partial classes into one **User** class as shown below.

User

```
public class User
{
    private string name;
    private string location;
    public User(string a, string b)
    {
        this.name = a;
        this.location = b;
    }
    public void GetUserDetails()
    {
        Console.WriteLine("Name: " + name);
        Console.WriteLine("Location: " + location);
    }
}
```

This is how the compiler will combine all the partial classes into a single class while executing the application in the c# programming language.

Rules to Implement Partial Class

In c#, we need to follow certain rules to implement a partial class in our applications.

- To split the functionality of class (/tutorial/csharp/csharp-classes-and-objects-with-examples), structure (/tutorial/csharp/csharp-structures-structs), interface (/tutorial/csharp/csharp-interface)**,** or method (/tutorial/csharp/csharp-methods-functions-with-examples) over multiple files, we need to use `partial` keyword (/tutorial/csharp/csharp-keywords-reserved-contextual) and all files must be available at compile time to form the final type.
- The `partial` modifier can only appear immediately before the keywords class (/tutorial/csharp/csharp-classes-and-objects-with-examples), struct (/tutorial/csharp/csharp-structures-structs), or interface (/tutorial/csharp/csharp-interface).
- All parts of partial type definitions must be in the same namespace (/tutorial/csharp/csharp-namespaces-with-examples) or assembly.
- All parts of partial type definitions must have the same accessibility, such as **public**, **private**, etc.
- If any partial part is declared as **abstract**, sealed (/tutorial/csharp/csharp-sealed-keyword)**,** or base (/tutorial/csharp/csharp-base-keyword), then the whole type is considered **abstract** or sealed (/tutorial/csharp/csharp-sealed-keyword)**,** or base (/tutorial/csharp/csharp-base-keyword) based on the defined type.
- As discussed in the inheritance (/tutorial/csharp/csharp-inheritance) concept, in c# a class can have a single base class so the partial classes that we create for a particular class must inherit from the same base (/tutorial/csharp/csharp-base-keyword) class.

- Nested partial types are allowed in partial type definitions.

C# Partial Class Example

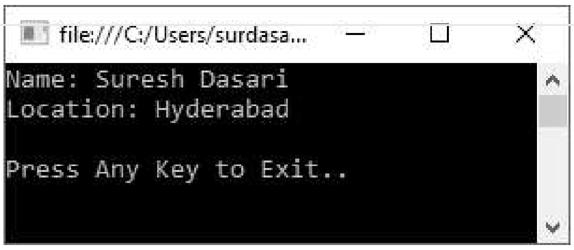
Following is the example of defining partial classes using `partial` keyword (/tutorial/csharp/csharp-keywords-reserved-contextual) in c# programming language.

```
using System;

namespace Tutlane
{
     public partial class User
     {
         private string name;
         private string location;
         public User(string a, string b)
         {
            this.name = a;
            this.location = b;
         }
     }
     public partial class User
     {
         public void GetUserDetails()
         {
            Console.WriteLine("Name: " + name);
            Console.WriteLine("Location: " + location);
         }
     }
     class Program
     {
        static void Main(string[] args)
        {
            User u = new User("Suresh Dasari", "Hyderabad");
            u.GetUserDetails();
            Console.WriteLine("\nPress Enter Key to Exit..");
            Console.ReadLine();
        }
     }
}
```

If you observe the above example, we created a partial class **User** using `partial` keyword (/tutorial/csharp/csharp-keywords-reserved-contextual) and we are able to access all partial classes as a single class to perform required operations.

When you execute the above c# program, we will get the result below.



```
Name: Suresh Dasari
Location: Hyderabad

Press Any Key to Exit..
```

This is how you can split the functionality of class (/tutorial/csharp/csharp-classes-and-objects-with-examples), structure (/tutorial/csharp/csharp-structures-structs), interface (/tutorial/csharp/csharp-interface), or method (/tutorial/csharp/csharp-methods-functions-with-examples) over two or more source files using `partial` modifier based on our requirements.