Partial Classes and Partial Methods in C#

What are Partial Classes in C#?
allows us to define a class on multiple files. we can physically split the content of the class into different files but even physically they are divided but logically it is one single unit only. A class in which code can be written in two or more files is known as a partial class. To make any class partial we need to use the keyword partial.
Partial classes allow us to split a class definition into 2 or more files. It is also possible to split the definition of a struct or an interface over two or more source files. Each source file will contain a section of the class definition, and all parts are combined into a single class when the application is compiled.

Rules to follow when working with Partial Classes in C#:
All the parts spread across different class files, must use the **partial** keyword. Otherwise, a compiler error is raised. **Missing partial modifier. Another partial declaration of this type exists.**

All the parts spread across different files, must have the **same access specifiers**. Otherwise, a compiler error is raised. **Partial declarations have conflicting accessibility modifiers.**

If any of the parts are declared as abstract, then the **entire type is considered as abstract** or if any of the parts are declared as sealed, **then the entire type is considered as sealed** or if any of the parts inherit a class, **then the entire type inherits that class.**

C# does not support multiple class inheritance. Different parts of the partial class must not specify different base classes. The following code will raise a compiler error stating – **Partial declarations must not specify different base classes.**

```
public class Employee
{
}
public class Customer
{
}
public partial class PartialClass : Employee
{
}
public partial class PartialClass : Customer
{
}
```

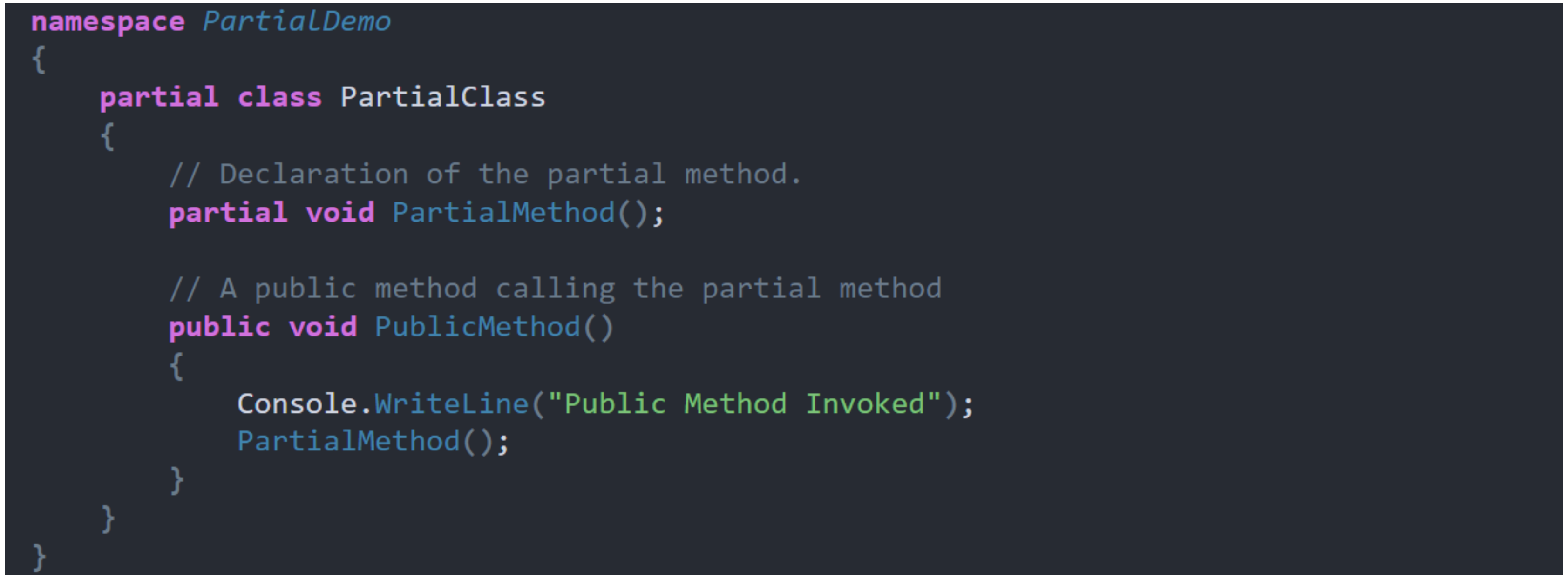
Different parts of the partial class can specify different base interfaces and the final type **implements all of the interfaces listed by all of the partial declarations.** In the example below **PartialClass** needs to provide the implementation for both **IEmployee** and **ICustomer** interface methods.

```
public interface IEmployee
{
    void EmployeeMethod();
}
public interface ICustomer
{
    void CustomerMethod();
}

public partial class PartialClass : IEmployee
{
    public void EmployeeMethod()
    {
        //Method Implementation
    }
}
public partial class PartialClass : ICustomer
{
    public void CustomerMethod()
    {
        //Method Implementation
    }
}
```

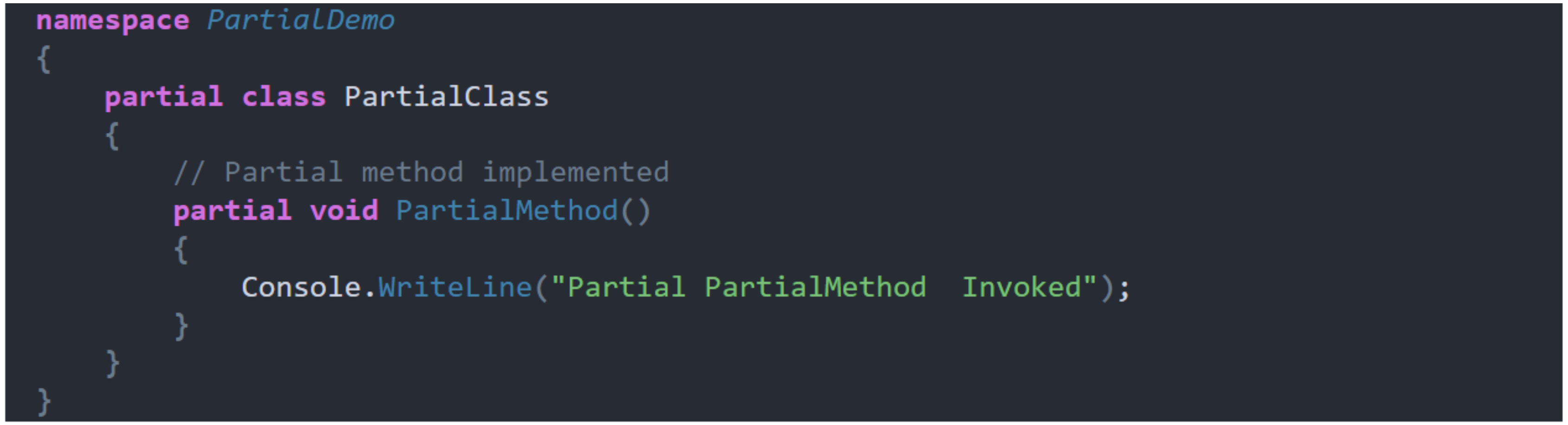
What are Partial Methods in C#?

A partial class or a struct can contain partial methods. A partial method is created using the same **partial** keyword.



```
namespace PartialDemo
{
    partial class PartialClass
    {
        // Declaration of the partial method.
        partial void PartialMethod();

        // A public method calling the partial method
        public void PublicMethod()
        {
            Console.WriteLine("Public Method Invoked");
            PartialMethod();
        }
    }
}
```



```
namespace PartialDemo
{
    partial class PartialClass
    {
        // Partial method implemented
        partial void PartialMethod()
        {
            Console.WriteLine("Partial PartialMethod  Invoked");
        }
    }
}
```

Sealed Class in C#

A class from which it is not possible to create/derive a new class is known as a sealed class. In simple words, we can also define the class that is declared using the sealed modifier is known as the sealed class and a sealed class cannot be inherited by any other class. For example:

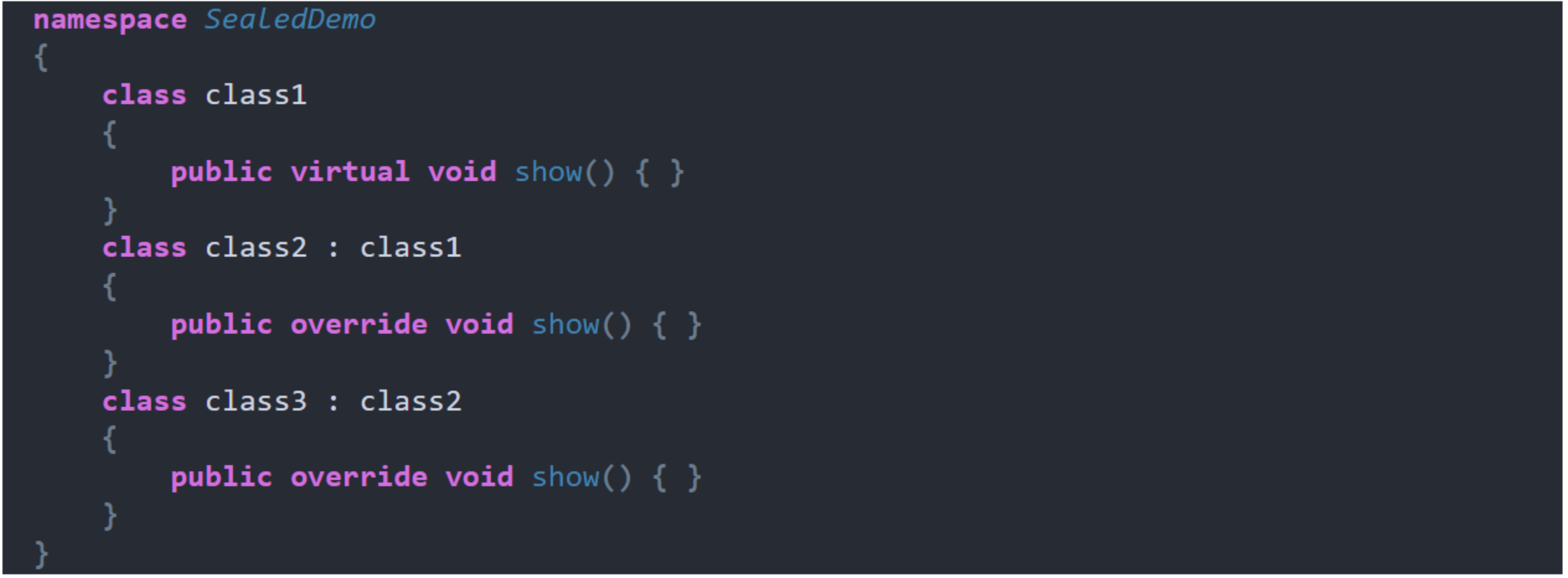
```
sealed class Class1 {}
class class2 : Class1{} //invalid
```

Points to Remember while working with Sealed Class

1. A sealed class is completely opposite to an abstract class.
2. This sealed class cannot contain abstract methods.
3. It should be the bottom-most class within the inheritance hierarchy.
4. A sealed class can never be used as a base class.
5. The sealed class is specially used to avoid further inheritance.
6. The keyword sealed can be used with classes, instance methods, and properties.

Sealed Methods in C#

The method that is defined in a parent class, if that method cannot be overridden under a child class, we call it a sealed method. By default, every method is a sealed method because overriding is not possible unless the method is not declared as virtual in the parent class. If a method is declared as virtual in a class, any child class of it can have the right to override that method.



```
namespace SealedDemo
{
    class class1
    {
        public virtual void show() { }
    }
    class class2 : class1
    {
        public override void show() { }
    }
    class class3 : class2
    {
        public override void show() { }
    }
}
```

What are Extension Methods in C#?

It is a new feature that has been added in C# 3.0 which allows us to add new methods into a class without editing the source code of the class i.e. if a class consists of a set of members in it and in the future if you want to add new methods into the class, you can add those methods without making any changes to the source code of the class.

Extension methods can be used as an approach to extending the functionality of a class in the future if the source code of the class is not available or we don’t have any permission in making changes to the class.

Before extension methods, inheritance is an approach that used for extending the functionality of a class i.e. if we want to add any new members into an existing class without making a modification to the class, we will define a child class to that existing class and then we add new members in the child class.

In the case of an extension method, we will extend the functionality of a class by defining the methods, we want to add into the class in a new class and then bind them to an existing class.

Both these approaches can be used for extending the functionalities of an existing class whereas, in inheritance, we call the method defined in the old and new classes by using object of the new class whereas, in the case of extension methods, we call the old and new methods by using object of the old class.

Points to Remember while working with C# Extension methods:

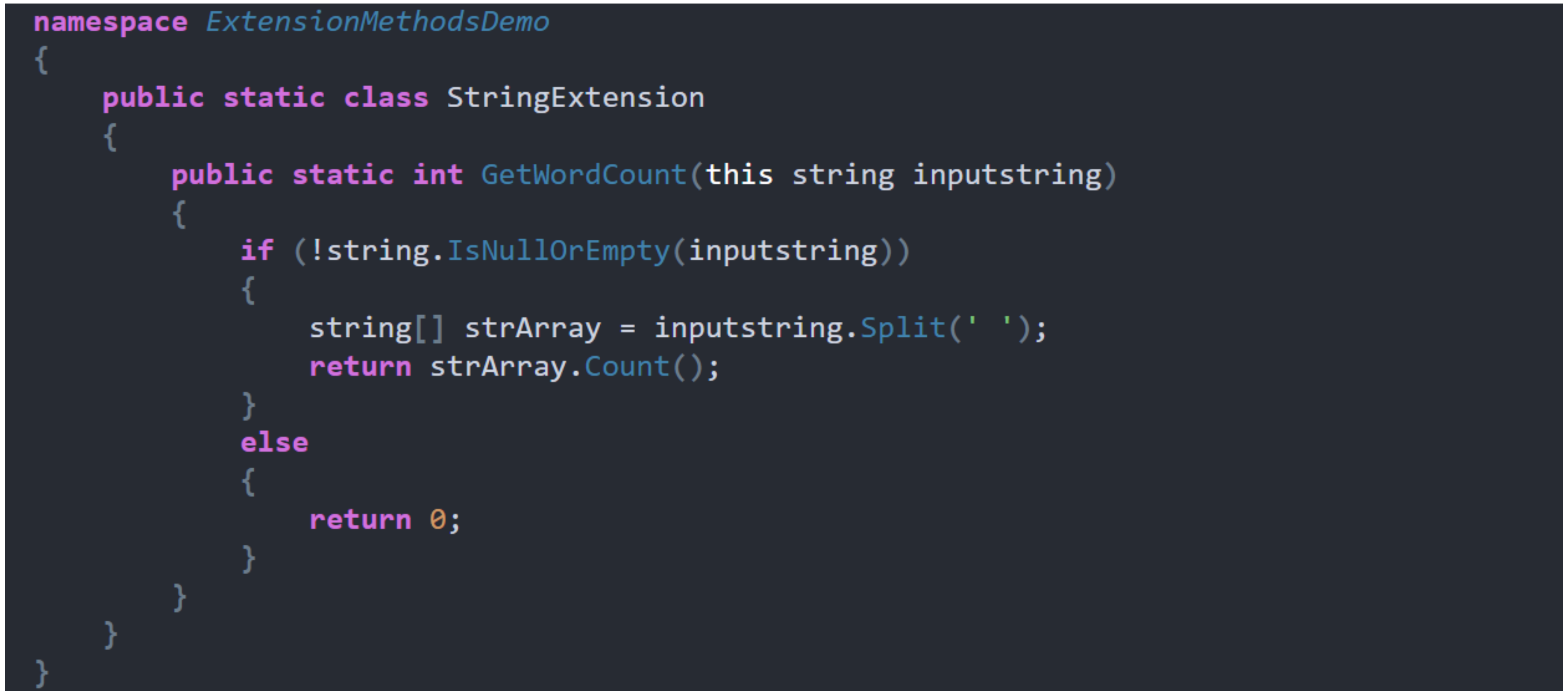
1.Extension methods must be defined only under the **static class.**

2.As an extension method is defined under a static class, compulsory that the method should be defined as static whereas once the method is bound with another class, the method changes into non-static.

3.The first parameter of an extension method is known as the binding parameter which should be the name of the class to which the method has to be bound and the binding parameter should be prefixed with **this** keyword.

4.An extension method can have only one binding parameter and that should be defined in the first place of the parameter list.

5.If required, an extension method can be defined with a normal parameter also starting from the second place of the parameter list.



```
namespace ExtensionMethodsDemo
{
    public static class StringExtension
    {
        public static int GetWordCount(this string inputstring)
        {
            if (!string.IsNullOrEmpty(inputstring))
            {
                string[] strArray = inputstring.Split(' ');
                return strArray.Count();
            }
            else
            {
                return 0;
            }
        }
    }
}
```