**Partial Classes and Methods**

**we will discuss about**  
**1.** What are partial classes?  
**2.** What are the advantages of using partial classes?  
**3.** Where are partial classes used?  
  
**Partial classes allow us to split a class into 2 or more files.**  All these parts are then combined into a single class, when the application is compiled. The partial keyword can also be used to split a struct or an interface over two or more files.  
  
  
  
**Let's understand partial classes with an example.**Create an asp.net web application project. Add a class file, with name **Customer.cs** to the project. Copy and paste the following code in the **customer.cs** file. This is a very simple customer class, with 2 private fields, 2 public properties and a public method.  
public class Customer  
{  
    private string \_firstName;  
    private string \_lastName;  
  
    public string FirstName  
    {  
        get { return \_firstName; }  
        set { \_firstName = value; }  
    }  
  
    public string LastName  
    {  
        get { return \_lastName; }  
        set { \_lastName = value; }  
    }  
  
    public string GetFullName()  
    {  
        return \_firstName + ", " + \_lastName;  
    }  
}  
  
  
  
**Now, let us split this class into 2 files.**One file is going to contain, the private fields and public properties, and the other file is going to contain the public method. Right click on the web application project, and add a class file, with name **PartialCustomerOne.cs**. Notice, that the **PartialCustomer** class is marked with the **partial** keyword and it contains, only, the 2 private fields and the public properties.   
public partial class PartialCustomer  
{  
    private string \_firstName;  
    private string \_lastName;  
  
    public string FirstName  
    {  
        get { return \_firstName; }  
        set { \_firstName = value; }  
    }  
  
    public string LastName  
    {  
        get { return \_lastName; }  
        set { \_lastName = value; }  
    }  
}  
  
**Now, add another class file with name, PartialCustomerTwo.cs**. Notice that, the **PartialCustomer** class, in this file is also marked as a **partial** class, and contains only the public method - **GetFullName()**. We are able to access the private fields, **\_firstName** and **\_lastName**, that are defined in **PartialCustomerOne.cs** file.  
public partial class PartialCustomer  
{  
    public string GetFullName()  
    {  
        return \_firstName + ", " + \_lastName;  
    }  
}  
  
**Copy and paste the following code in the Page\_Load() event of the webform1.** Though, the **PartialCustomer** class is split across 2 files(PartialCustomerOne.cs and PartialCustomerTwo.cs), we are able to use it the same way as the Customer class.  
Customer c1 = new Customer();  
c1.FirstName = "Pragim";  
c1.LastName = "Technologies";  
  
string FullName1 = c1.GetFullName();  
Response.Write("Full Name = " + FullName1 + "<br/>");  
  
PartialCustomer c2 = new PartialCustomer();  
c2.FirstName = "Pragim";  
c2.LastName = "Tech";  
  
string FullName2 = c2.GetFullName();  
Response.Write("Full Name = " + FullName2 + "<br/>");  
  
**Advantages of partial classes**  
**1.** The main advantage is that, visual studio uses partial classes to separate, automatically generated system code from the developer's code. For example, when you add a webform, two .CS files are generated  
**a) WebForm1.aspx.cs -**Contains the developer code  
**b) WebForm1.aspx.designer.cs -** Contains the system generated code. For example, declarations for the controls that you drag and drop on the webform.  
  
**2.** When working on large projects, spreading a class over separate files allows multiple programmers to work on it simultaneously.

**1.**All the parts spread across different files, must use the **partial keyword**. Otherwise a compiler error is raised.   
**Missing partial modifier. Another partial declaration of this type exists**  
  
**2.** All the parts spread across different files, must have the **same access modifiers**. Otherwise a compiler error is raised.   
**Partial declarations have conflicting accessibility modifiers**  
  
**3.** If any of the parts are declared abstract, then the **entire type is considered abstract**.  
  
**4.** If any of the parts are declared sealed, **then the entire type is considered sealed**.   
  
**5.**If any of the parts inherit a class, **then the entire type inherits that class.**  
  
  
  
**6. 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 partial class SamplePartialClass : Employee  
{  
}  
public partial class SamplePartialClass : Customer  
{  
}  
public class Employee  
{  
}  
public class Customer  
{  
}  
  
  
  
**7.** 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, **SamplePartialClass** needs to provide implementation for both **IEmployee**, and **ICustomer** interface methods.  
public partial class SamplePartialClass : IEmployee  
{  
    public void EmployeeMethod()  
    {  
        //Method Implementation  
    }  
}  
public partial class SamplePartialClass : ICustomer  
{  
    public void CustomerMethod()  
    {  
        //Method Implementation  
    }  
}  
public interface IEmployee  
{  
    void EmployeeMethod();  
}  
public interface ICustomer  
{  
    void CustomerMethod();  
}  
  
**8.** Any **members that are declared in a partial definition** are available to all of the other parts of the partial class.

**A partial class or a struct can contain partial methods. A partial method is created using the partial keyword.** Let us understand partial methods with an example. Create a console application. Add a class file, with name **PartialClassFileOne.cs**, to the project. copy and paste the following code.  
  
Notice, that, the **SampleMethod**() definition has the **partial keyword**, and does not have a **body(implementation) only the signature**. The implementation for a partial method is optional. If we don't provide the implementation, the compiler removes the signature and all calls to the method.  
  
**The implementation can be provided in the same physical file, or in another physical file**, that contains the partial class. In this example, the partial SampleMethod() is invoked in the PublicMethod().  
partial class SampleClass  
{  
    // Declaration of the partial method.  
    partial void SampleMethod();  
  
    // A public method calling the partial method  
    public void PublicMethod()  
    {  
        Console.WriteLine("Public Method Invoked");  
        SampleMethod();  
    }  
}  
  
  
  
**Copy and paste the following code in the Main() method of the console application.**When we run the application now, notice that, we don't get a compiler error, in spite of not having an implementation for the partial **SampleMethod**(). Since, the implementation for the partial method is missing, the compiler will remove the signature and all calls to the method.  
SampleClass SC = new SampleClass();  
SC.PublicMethod();  
  
**Now, add a class file**, with name **PartialClassFileTwo.cs**. Copy and paste the following code. The implementation for the partial method is provided here.  
partial class SampleClass  
{  
    // Partial method implemented  
    partial void SampleMethod()  
    {  
        Console.WriteLine("Partial SampleMethod Invoked");  
    }  
}  
  
  
  
**Now, run the console application and notice the output.** The partial method and the public method messages are printed on the console.   
  
**A partial method declaration consists of two parts.**  
**1.** The definition (only the method signature ending with a semi-colon, without method body)  
**2.** The implementation.   
**These may be in separate parts of a partial class, or in the same part.**  
  
**Partial methods are private by default**, and it is a compile time error to include any access modifiers, including private. The following code will raise an error stating - A partial method cannot have access modifiers or the virtual, abstract, override, new, sealed, or extern modifiers.  
partial class SampleClass  
{  
    private partial void SampleMethod();  
}  
  
**It is a compile time error, to include declaration and implementation at the same** time for a partial method. Code below produces a compile time error - No defining declaration found for implementing declaration of partial method 'PartialMethodsDemo.SampleClass.SampleMethod()'  
partial class SampleClass  
{  
    partial void SampleMethod()  
    {  
        Console.WriteLine("SampleMethod Implemented");  
    }  
}  
  
**A partial method return type must be void.** Including any other return type is a compile time error - Partial methods must have a void return type  
partial class SampleClass  
{  
    partial int SampleMethod();  
}  
  
**A partial method must be declared within a partial class or partial struct.** A non partial class or struct cannot include partial methods.  
  
**Signature of the partial method declaration**, must match with the signature of the implementation.  
  
**A partial method can be implemented only once**. Trying to implement a partial method more than once, raises a compile time error - A partial method may not have multiple implementing declarations.