C# - Partial Classes and Methods

In C#, you can split the implementation of a class, a struct, a method, or an interface in multiple `.cs` files using the `partial` keyword. The compiler will combine all the implementation from multiple `.cs` files when the program is compiled.

Consider the following `EmployeeProps.cs` and `EmployeeMethods.cs` files that contain the `Employee` class.

Example: EmployeeProps.cs

```
public partial class Employee
{
    public int EmpId { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public int Age { get; set; }
}
```

Example: EmployeeMethods.cs

```
public partial class MyPartialClass
{
    public Employee(int Id, string name)
    {
        this.EmpId = Id;
        this.Name = name;
    }

    public void DisplayEmployeeInfo()
    {
        Console.WriteLine(this.EmpId + " " this.FirstName + " " + this.LastName);
    }

    public void Save(int id, string firstName, string lastName)
    {
        Console.WriteLine("Saved!");
    }
}
```

Above, `EmployeeProps.cs` contains properties of the `Employee` class, and `EmployeeMethods.cs` contains all the methods of the `Employee` class. These will be compiled as one `Employee` class.

Example: Partial Class

```
public class Employee
{
    public int EmpId { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public int Age { get; set; }

    public Employee(int Id, string name)
```

```
    {
        this.EmpId = Id;
        this.Name = name;
    }

    public void DisplayEmployeeInfo()
    {
        Console.WriteLine(this.EmpId + " " this.FirstName + " " + this.LastName);
    }

    public void Save(int id, string firstName, string lastName)
    {
        Console.WriteLine("Saved!");
    }
}
```

Rules for Partial Classes

- All the partial class definitions must be in the same assembly and namespace.
- All the parts must have the same accessibility like public or private, etc.
- If any part is declared abstract, sealed or base type then the whole class is declared of the same type.
- Different parts can have different base types and so the final class will inherit all the base types.
- The Partial modifier can only appear immediately before the keywords class, struct, or interface.
- Nested partial types are allowed.

Partial Methods

Partial classes or structs can contain a method that split into two separate `.cs` files of the partial class or struct. One of the two `.cs` files must contain a signature of the method, and other file can contain an optional implementation of the partial method. Both declaration and implementation of a method must have the `partial` keyword.

Example: EmployeeProps.cs

```
public partial class Employee
{
    public int EmpId { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public int Age { get; set; }

    partial void GenerateEmployeeId();

}
```

Example: EmployeeMethods.cs

```
public partial class MyPartialClass
{
    partial void GenerateEmployeeId()
    {
        this.EmpId = random();
    }
```

```
}
```

Above, `EmployeeProps.cs` contains the signature of the `DisplayEmployeeInfo` method, and `EmployeeMethods.cs` contains the implementation of it. The compiler will combine all parts into one at compile-time.

It is required to include a signature of the partial method, but it is not required to provide the implementation. There will be no compile-time or run-time errors if the method is called but not implemented.

Rules for Partial Methods

- Partial methods must use the `partial` keyword and must return `void`.
- Partial methods can have `in` or `ref` but not `out` parameters.
- Partial methods are implicitly private methods, so cannot be virtual.
- Partial methods can be static methods.
- Partial methods can be generic.