C# Class

A class is like a blueprint of a specific object. In the real world, every object has some color, shape, and functionalities - for example, the luxury car Ferrari. Ferrari is an object of the luxury car type. The luxury car is a class that indicates some characteristics like speed, color, shape, interior, etc. So any company that makes a car that meets those requirements is an object of the luxury car type. For example, every single car of BMW, Lamborghini, Cadillac are an object of the class called 'Luxury Car'. Here, 'Luxury Car' is a class, and every single physical car is an object of the luxury car class.

Likewise, in object-oriented programming, a class defines some properties, fields, events, methods, etc. A class defines the kinds of data and the functionality their objects will have.

A class enables you to create your custom types by grouping variables of other types, methods, and events.

In C#, a class can be defined by using the class keyword.

Example: C# Class

```
public class MyClass
{
    public string  myField = string.Empty;

    public MyClass()
    {
    }

    public void MyMethod(int parameter1, string parameter2)
    {
        Console.WriteLine("First Parameter {0}, second parameter {1}",
                                                    parameter1, parameter2);
    }

    public int MyAutoImplementedProperty { get; set; }

    private int myPropertyVar;

    public int MyProperty
    {
        get { return myPropertyVar; }
        set { myPropertyVar = value; }
    }
}
```

C# Access Modifiers

Access modifiers are applied to the declaration of the class, method, properties, fields, and other members. They define the accessibility of the class and its members. Public, private, protected, and internal are access modifiers in C#.

C# Field

The field is a class-level variable that holds a value. Generally, field members should have a private access modifier and used with property.

C# Constructor

A class can have parameterized or parameterless constructors. The constructor will be called when you create an instance of a class. Constructors can be defined by using an access modifier and class name:
`<access modifiers> <class name>(){ }`

Example: Constructor in C#

```
class MyClass
{
    public MyClass()
    {

    }
}
```

C# Method

A method can be defined using the following template:

`{access modifier} {return type} MethodName({parameterType parameterName})`

Example: Method in C#

```
public void MyMethod(int parameter1, string parameter2)
{
    // write your method code here..

}
```

Property

A property can be defined using getters and setters, as shown below:

Example: Property in C#

```
private int _myPropertyVar;

public int MyProperty
{
    get { return _myPropertyVar; }
    set { _myPropertyVar = value; }
}
```

Property **encapsulates** a private field. It provides getters (get{}) to retrieve the value of the underlying field and setters (set{}) to set the value of the underlying field. In the above example, _myPropertyVar is a private field that cannot be accessed directly. It will only be accessed via MyProperty. Thus, MyProperty encapsulates _myPropertyVar.

You can also apply some additional logic in get and set, as in the below example.

Example: Property in C#

```
private int _myPropertyVar;

public int MyProperty
{
    get {
        return _myPropertyVar / 2;
    }

    set {
        if (value > 100)
            _myPropertyVar = 100;
        else
            _myPropertyVar = value; ;
    }
}
```

Auto-implemented Property

From C# 3.0 onwards, property declaration has been made easy if you don't want to apply some logic in get or set.

The following is an example of an auto-implemented property:

Example: Auto implemented property in C#

```
public int MyAutoImplementedProperty { get; set; }
```

Notice that there is no private backing field in the above property example. The backing field will be created automatically by the compiler. You can work with an automated property as you would with a normal property of the class. Automated-implemented property is just for easy declaration of the property when no additional logic is required in the property accessors.

Namespace

The namespace is a container for a set of related classes and namespaces. The namespace is also used to give unique names to classes within the namespace name. Namespace and classes are represented using a dot (.).

In C#, namespace can be defined using the namespace keyword.

Example: Namespace

```
namespace CSharpTutorials
{
    class MyClass
    {

    }
}
```

In the above example, the fully qualified class name of `MyClass` is `CSharpTutorials.MyClass`.

A namespace can contain other namespaces. Inner namespaces can be separated using (.).

Example: Namespace

```
namespace CSharpTutorials.Examples
{
    class MyClassExample
    {

    }
}
```