



Sir Syed University of Engineering & Technology, Karachi

Introduction to Classes, Objects & Methods

Course Code : CS-127

Course Title : Object Oriented Programming

Semester : 2nd

C# Object and Class

- Since C# is an object-oriented language, program is designed using objects and classes in C#.

C# Object

- In C#, Object is a real world entity, for example, chair, car, pen, mobile, laptop etc.
- In other words, object is an entity that has state and behaviour. Here, state means data and behaviour means functionality.

C# Object

- ✓ Object is a runtime entity, it is created at runtime.
- ✓ Object is an instance of a class. All the members of the class can be accessed through object.
- ✓ Let's see an example to create object using new keyword.

```
Student s1 = new Student();
```

```
//creating an object of Student
```

In this example, Student is the type and s1 is the reference variable that refers to the instance of Student class. The new keyword allocates memory at runtime.

C# Class

- In C#, class is a group of similar objects. It is a template from which objects are created.



Another example:



C# Class

- Let's see an example of C# class that has two fields only.

1. *class Student*

2. *{*

3. *int id;//field or data member*

4. *String name;//field or data member*

5. *}*

C# Object and Class Example

- Let's see an example of class that has two fields: id and name. It creates instance of the class, initializes the object and prints the object value.

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ConsoleApplication36
{
    class student
    {
        int id;
        string name;

        static void Main(string[] args)
        {
            student s = new student();
            s.id = 123;
            s.name = " Ali Ahmed";
            Console.WriteLine(s.id);
            Console.WriteLine(s.name);
            Console.Read();

        }
    }
}
```

C# Class Example 2: Having Main() in another class

- Let's see another example of class where we are having Main() method in another class. In such case, class must be public.


```

using System;
using System.Collections.Generic;
using System.Text;

namespace ConsoleApplication36
{
    class Employee
    {
        int id;
        string name;
        int salary;
        int tax_am;
        public void insertvalue(int i, string n, int sal)
        {
            id = i;
            name = n;
            salary = sal;
        }
        public void tax()
        {
            tax_am = salary / 100 * 25;
        }
        public void display()
        {
            Console.WriteLine("ID : "+id+ " Name : "+name+" Salary : "+salary+" Tax : "+tax_am);
        }
    }
}

```

```
class TestEMPLOYEE
{

    static void Main(string[] args)
    {

        Employee e1 = new Employee();
        e1.insertvalue(11, "Ali Raza", 10000);
        e1.tax();
        e1.display();
        Employee e2 = new Employee();
        e2.insertvalue(22, "Zain Ahmed", 20000);
        e2.tax();
        e2.display();
        Console.Read();
    }
}
```

Access Modifiers in C#

Access Modifiers specifies the scope of variable and functions in C#. The following are the access modifiers used in C#:

- **Public**
The public modifier sets no restriction on the access of members.
- **Protected**
Access limited to the derived class or class definition.
- **Internal**
The Internal access modifier access within the program that has its declaration.
- **Protected internal**
It has both the access specifiers provided by protected and internal access modifiers.
- **Private**
Limited only inside the class in which it is declared. The members specified as private cannot be accessed outside the class.

Access Modifiers in C#

<div> <div>Accessibility</div> <div>keyword</div> </div>	With in a class	Inheritance (In derived class)	Within same Application (Outside a class)	Anywhere Outside the Application
Private	Yes	No	No	No
Protected	Yes	Yes	No	No
Internal	Yes	Yes	Yes	No
Protected Internal	Yes	Yes	Yes	No
Public	Yes	Yes	Yes	Yes

C# Properties (Get and Set)

Properties and Encapsulation

- Before we start to explain properties, revise the basic understanding of "**Encapsulation**".
 - The meaning of Encapsulation, is to make sure that "sensitive" data is hidden from users.
 - To achieve this, we must:
 - ✓ declare fields/variables as private.
 - ✓ provide public get and set methods, through **properties**, to access and update the value of a private field.

Properties

- private variables can only be accessed within the same class (an outside class has no access to it). However, sometimes we need to access them - and it can be done with ***properties***.
- A property is like a combination of a variable and a method, and it has two methods: a **get** and a **set** method:

Example

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ConsoleApplication41
{
    class Employee
    {
        private int salary = 2000;
        public int salaryaccess //read & write property
        {
            get { return salary; }
            set { salary = value; }
        }
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        int bonus;
        Employee emp = new Employee();
        emp.salaryaccess = 1800;
        bonus = emp.salaryaccess / 100 * 30;
        Console.WriteLine(bonus);
        Console.ReadKey();

    }
}
```


Types Of Properties

- **Read and Write Properties:** When property contains both `get` and `set` methods.
- **Read-Only Properties:** When property contains only `get` method.
- **Write Only Properties:** When property contains only `set` method.