**What is a Class?**

In Object Oriented Programming, a Class is a blueprint for an object. In fact, classes describe the type of objects, while objects are usable instances of classes. Each Object was built from the same set of blueprints and therefore contains the same components (properties and methods). A class can have any number of properties and methods to access the value of various kinds of methods.

What is a Class?

In real life, similar objects can be grouped based on some criteria. For example: A Ford car and a Toyota car are both Cars, so they can be classified as belonging to the Car class. There may be thousands of other Cars in existence, all of the same make and model. Each Car was built from the same set of blueprints and therefore contains the same components. In object-oriented terms, we can say that your car is an object (instance) of the class known as CAR. You can create different objects using the same class, because a class is just a template, while the objects are concrete instances, based on the template.

C# Class

A class is like a blueprint of 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 specify certain characteristic like speed, color, shape, interior etc. So any company that makes a car that meet 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 certain properties, fields, events, method etc. A class defines the kinds of data and the functionality their objects will have.

A class enables you to create your own custom types by grouping together 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()

{

}

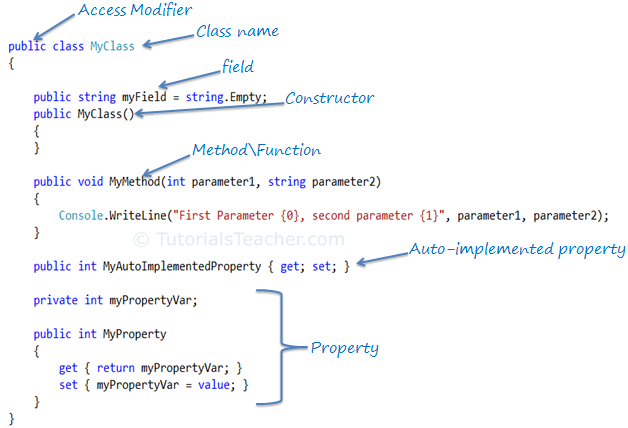
public void MyMethod(int parameter1, string parameter2)

{

Console.WriteLine("First Parameter {0}, second parameter {1}",

parameter1, parameter2);

}



**C# Access Modifiers**

Access modifiers are applied on 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#.

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

**C# Constructor**

A class can have parameterized or parameter less 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** is a block of code which only runs when it is called.You can pass data, known as parameters, into a method.
* Methods are used to perform certain actions, and they are also known as **functions**.
* Why use methods? To reuse code: define the code once, and use it many times.

## Create a Method

A method is defined with the name of the method, followed by parentheses **()**. C# provides some pre-defined methods, which you already are familiar with, such as Main(), but you can also create your own methods to perform certain actions:

### Example

Create a method inside the Program class:

class Program

{

static void MyMethod()

{

// code to be executed

}

}

#### Example Explained

* MyMethod() is the name of the method
* static means that the method belongs to the Program class and not an object of the Program class. You will learn more about objects and how to access methods through objects later in this tutorial.
* void means that this method does not have a return value. You will learn more about return values later in this chapter

## Call a Method

To call (execute) a method, write the method's name followed by two parentheses **()** and a semicolon**;**

In the following example, MyMethod() is used to print a text (the action), when it is called:

static void MyMethod()

{

Console.WriteLine("I just got executed!");

}

static void Main(string[] args)

{

MyMethod();

}

**What is a constructor in C#?**

A special method of the class that is automatically invoked when an instance of the class is created is called a constructor. The main use of constructors is to initialize the private fields of the class while creating an instance for the class. When you have not created a constructor in the class, the compiler will automatically create a default constructor of the class. The default constructor initializes all numeric fields in the class to zero and all string and object fields to null.

### Some of the key points regarding constructor are

* A class can have any number of constructors.
* A constructor doesn't have any return type, not even void.
* A static constructor can not be a parametrized constructor.
* Within a class, you can create one static constructor only.

### In C#, constructors can be divided into 5 types

1. Default Constructor
2. Parameterized Constructor
3. Copy Constructor
4. Static Constructor
5. Private Constructor

Now, let's see each constructor type with the example below.

## Default Constructor in C#

A constructor without any parameters is called a default constructor; in other words, this type of constructor does not take parameters. The drawback of a default constructor is that every instance of the class will be initialized to the same values and it is not possible to initialize each instance of the class with different values. The default constructor initializes:

1. All numeric fields in the class to zero.
2. All string and object fields to null.

**Example**

using System;

namespace DefaultConstractor

{

class addition

{

int a, b;

public addition() //default contructor

{

a = 100;

b = 175;

}

public static void Main()

{

addition obj = new addition(); //an object is created , constructor is called

Console.WriteLine(obj.a);

Console.WriteLine(obj.b);

Console.Read();

}

}

}

**Parameterized Constructor in C#**

A constructor with at least one parameter is called a parameterized constructor. The advantage of a parameterized constructor is that you can initialize each instance of the class with a different value.

using System;

namespace Constructor

{

class paraconstrctor

{

public int a, b;

public paraconstrctor(int x, int y) // decalaring Paremetrized Constructor with ing x,y parameter

{

a = x;

b = y;

}

}

class MainClass

{

static void Main()

{

paraconstrctor v = new paraconstrctor(100, 175);

Console.WriteLine("-----------parameterized constructor example by vithal wadje---------------");

Console.WriteLine("\t");

Console.WriteLine("value of a=" + v.a );

Console.WriteLine("value of b=" + v.b);

Console.Read(); ) )

Copy Constructor in C#

The constructor which creates an object by copying variables from another object is called a copy constructor. The purpose of a copy constructor is to initialize a new instance to the values of an existing instance.

Syntax

public employee(employee emp)

{

name=emp.name;

age=emp.age;

}

The copy constructor is invoked by instantiating an object of type employee and bypassing it the object to be copied.

using System;

namespace copyConstractor

{

class employee

{

private string name;

private int age;

public employee(employee emp) // declaring Copy constructor.

{

name = emp.name;

age = emp.age;

}

public employee(string name, int age) // Instance constructor.

{

this.name = name;

this.age = age;

}

public string Details // Get deatils of employee

{

get

{

return " The age of " + name +" is "+ age.ToString();

}

}

}

class empdetail

{

static void Main()

{

employee emp1 = new employee("Vithal", 23); // Create a new employee object.

employee emp2 = new employee(emp1); // here is emp1 details is copied to emp2.

Console.WriteLine(emp2.Details);

Console.ReadLine();

}

}

}

**Destructor**

A destructor is a special member function of a class that is executed whenever an object of its class goes out of scope.

public Demo()

{ // constructor

Console.WriteLine("Object is being created");

}

~Demo() { //destructor

Console.WriteLine("Object is being deleted");

}

**as exactly the same name as that of the class with a prefixed tilde (~),**

using System;

namespace LineApplication {

class Line {

private double length; // Length of a line

public Line() { // constructor

Console.WriteLine("Object is being created");

}

~Line() { //destructor

Console.WriteLine("Object is being deleted");

}

public void setLength( double len ) {

length = len;

}

public double getLength() {

return length;

}

static void Main(string[] args) {

Line line = new Line();

// set line length

line.setLength(6.0);

Console.WriteLine("Length of line : {0}", line.getLength());

}

}

}