Class

Bir class aşağıdaki gibi tanımlanmaktadır.

Example

//Syntax : class name{}

class Class1

{

/\*

Burada class, struct, enum, interface, delegate, event, field, property, indexer

tanımlanabilmektedir.

\*/

}

* İç içe class tanımlanabilir.
* struct içinde class tanımlanabilir.
* Tüm class’lar Object sınıfının mirasçısıdır.
* class’lara takma ad verilebilir.
* class’lar heap’de tutulurlar.
* Erişim belirteçlerine sahip olabilirler. Herhangi bir erişim belirteci kullanılmadığında internal oldukları varsayılır.
* public olarak tanımlanan sınıflara başka projelerden erişilebilmektedir.
* internal olarak tanımlanan sınıfların yaşam alanları proje içiyle sınırlıdır.
* Bir sınıfın private ya da protected olarak işaret edilebilmesi için başka bir sınıfın içerisinde olması gerekmektedir.

protected olarak tanımlanan sınıflar mirasçı sınıflarda kullanılabilmektedir.

Example

class Class1

{

protected class Class2

{

}

}

class Class3 : Class1

{

Class1.Class2 obj = new Class1.Class2();

}

private olarak tanımlanan sınıflar mirasçı sınıflarda kullanılamaz.

Example

class Class1

{

private class Class2

{

}

}

class Class3 : Class1

{

Class1.Class2 obj = new Class1.Class2(); //hata meydana gelir.

}

* Nesne hazırlarken eşitliğin sağ tarafına constructor metodun adını yazmak zorunluluk olmaktan çıkarıldı. (C# 9.0)

Example

Class1 obj = new();

* class’ların örnekleri alınırken class içindeki field ve property’lere aşağıdaki gibi değer ataması yapılabilmektedir.

Example

class Person

{

public string name;

public int age { get; set; }

}

Person obj = new Person { name = "John Doe", age = 30 };

(Inheritance)

Example

class Class1 //base class

{

public int number1;

protected int number2;

public Class1()

{

Console.WriteLine("Class1 constructor.");

}

}

class Class2 : Class1 //derived class veya sub class

{

public int number3;

public Class2()

{

Console.WriteLine("Class2 constructor.");

}

}

class Class3 : Class2

{

public Class3()

{

Console.WriteLine("Class3 constructor.");

}

}

Class3 obj = new Class3();

//Class3'ün örneği alındığında sırasıyla Class1, Class2 ve Class3'ün yapıcı metotları çalışır.

//Class3; Class1 ve Class2’de public olarak tanımlanan üyelere sahip olur.

//Class1 ve Class2’de private olarak tanımlanan üyeler Class3’e aktarılmaz.

//Class3; Class1 ve Class2’de protected olarak tanımlanan üyelere sahip olur. protected olarak tanımlanan üyeler sanki Class3 içerisinde private olarak tanımlanmış gibi kullanılır. Obje üzerinden bu üyelere erişilemez.

* Bir sınıf en fazla bir sınıfı miras alabilir.
* sub class’ın örneği alındığında base class’ın yapıcı metodu otomatikmen işletilmektedir.
* Sub class ile base class’ın constructor’ları aynı parametre yapısına sahip olabilir. Bu durumda base anahtar kelimesi kullanılarak parametreler düzenlenir.

Example

class Class1

{

public int number1;

public int number2;

public Class1(int a, int b)

{

number1 = a;

number2 = b;

}

}

class Class2 : Class1

{

public int number3;

public int number4;

public Class2(int a, int b, int c, int d) : base(c, d)

{

number3 = a;

number4 = b;

}

}

Class2 obj = new Class2(1, 2, 3, 4);

Console.WriteLine(obj.number1); //3

Console.WriteLine(obj.number2); //4

Console.WriteLine(obj.number3); //1

Console.WriteLine(obj.number4); //2

* Base class’da bulunan bir metod sub class’da aynı isimde tanımlanabilmektedir. Bunun için new anahtar kelimesi kullanılmaktadır.

Example

class Class1

{

public void Process()

{

Console.WriteLine("Class1 process.");

}

}

class Class2 : Class1

{

public new void Process()

{

Console.WriteLine("Class2 process.");

}

}

Class2 obj = new Class2();

obj.Process(); //Class2 process.

Eğer base class’daki Process metodunu çağırmak istiyorsanız polymorphism ilkesinden yararlanabilirsiniz.

Example

Class1 obj = new Class2();

obj.Process(); //Class1 process.

Bu sayede bir sınıfın başka bir sınıf kaynaklı metod ve özelliklerini kullanabilirsiniz.

* Base class hazırlayan bir programcı yazdığı metotları subclass’larda ihtiyaçları kaşılamama ihtimaline karşı virtual olarak tanımlayabilir. Bu sayede ilgili metotlar daha sonra subclass’larda override anahtar kelimesi ile yeniden düzenlenebilir.

Example

class Class1

{

public virtual void Process()

{

}

}

class Class2 : Class1

{

public override void Process()

{

}

}

Example

class Class1

{

public virtual void Process()

{

Console.WriteLine("Class1 Process.");

}

}

class Class2 : Class1

{

public override void Process()

{

Console.WriteLine("Class2 Process.");

}

}

//polymorphism

Class1 obj = new Class2();

obj.Process(); //Class2 Process.

(sealed class)

Example

sealed class Class1

{

}

* sealed class’lar kalıtımı desteklemez.

(partial class)

partial anahtar kelimesiyle class’lar parçalı şekilde tanımlanabilmektedir.

Example

partial class Class1

{

}

partial class Class1

{

}

partial class Class1

{

}

* Parçalar farklı kod dosyalarında olabilmektedir.

(static class)

Example

static class Class1

{

}

* Uygulama içerisinde sıklıkla kullanılan metod, field, property vb. elemanlar statik class’larda toplanabilir.
* Tüm elemanları statik olmak zorundadır.
* Statik sınıflardan nesne oluşturulamaz.
* Kalıtımı desteklemezler.
* Yapıcı metod içeremezler.
* İçerisinde statik ya da normal class barındırabilir.

(abstract class)

abstract sınıflar sınıf hiyerarşisi oluşturulurken diğer sınıflara taban olması için tanımlanırlar.

Example

abstract class Class1

{

//abstract metod deklare ettik. public veya protected olabilir. Mirasçı sınıfta override

//edilmelidir.

abstract public int AddNumbers(int x, int y);

//abstract property. public veya protected olabilir. Mirasçı sınıfta override edilmelidir.

abstract public int number { get; set; }

//Normal metod deklare ettik. public, protected veya private olabilir. Mirasçı sınıfta new

//anahtar kelimesi ile yeniden tanımlanabilir.

public void Message(string message)

{

Console.WriteLine("Class1 message : " + message);

}

//Normal property. public, protected veya private olabilir. Mirasçı sınıfta new anahtar

//kelimesi ile yeniden tanımlanabilir.

public string str { get; set; }

//Normal field. public, protected veya private olabilir. Mirasçı sınıfta new anahtar

//kelimesi ile yeniden tanımlanabilir.

public object data;

}

* Abstract sınıfların örneği alınamaz ama yapıcı metot içerebilirler.
* Abstract sınıfta deklare edilen abstract üyeler public veya protected olmalıdır. private olamazlar.
* Abstract sınıfta deklare edilen normal üyeler public, protected veya private olabilirler.
* Abstract üyeler mirasçı sınıflarda override edilmelidir.
* Normal üyeler mirasçı sınıflarda direk kullanılabilir ya da new anahtar kelimesi ile yeniden düzenlenebilirler.
* Bir abstract sınıf başka bir abstract sınıfı miras alabilir.
* Abstract sınıflara interface’ler uygulanabilir.

Example

interface Interface1

{

int AddNumbers(int x, int y);

}

abstract class Class1 : Interface1

{

public int AddNumbers(int x, int y)

{

return x + y;

}

}

|  |
| --- |
| **NOT**  abstract sınıflar interface’lere benzerler ama aralarında bazı farklar bulunmaktadır.   * abstract sınıflar normal üyelere sahip olabilirler. interface’lerde ise metod, property vb. elemanların sadece imzaları tanımlanmaktadır. * interface içindeki tüm öğeler public olmalıdır ama abstract sınıflarda böyle bir zorunluluk yoktur. * Bir sınıfa birden fazla interface uygulanabilirken sadece bir abstract class uygulanabilir. |

(anonymous types)

İsimsiz tipler sayesinde basit ve tek kullanımlık tipler için yeni bir class ya da struct oluşturmak zorunda kalmıyoruz.

Example

var employee = new

{

name = "Eren",

age = 28

};

Console.WriteLine(employee.name); //Eren

Console.WriteLine(employee.age); //28

Derleyici süslü parantezlerin arasına yazılanlara bakıp uygun tipleri derleme sırasında oluşturmaktadır.

* İsimsiz tipler hazırlanırken var keyword kullanılmaktadır.
* Süslü parantezler arasına yazılan anahtar-değer çiftleri field olarak değerlendirilmektedir.
* İsimsiz tipler sadece field içerebilirler. Metod içeremezler.
* İsimsiz tiplerin yaşam alanları tanımlandığı metodun içiyle sınırlıdır.
* İsimsiz tipler başka isimsiz tipleri içerebilirler.

Example

var employee = new

{

name = "Eren",

age = 28,

company = new

{

name = "Turuncu Web Solutions"

}

};