Methods

C#’da methodlar aşağıdaki gibi tanımlanmaktadır.

Example

*//Geriye değer döndürmeyen method.*

void Method1(){

}

*//Geriye değer döndüren method.*

bool Method2(){

}

* class ve struct içinde method tanımlanabilir.
* Methodlar erişim belirteçlerine sahip olabilirler. Herhangi bir erişim belirteci kullanılmadığında private oldukları varsayılır.
* Methodlara istediğiniz sayıda parametre tanımlayabilirsiniz.
* Parametreler method içinde tanımlanan değişkenler gibi çalışırlar. Yani parametreler aslında derleyici tarafından method içinde değişken olarak tanımlanmaktadır. Örneğin :

Example

public void Message(string message){

message = "Message2";

Console.Write(message); *//Message2*

}

Message("Message1");

Method içinde **message** adında bir değişken olmamasına rağmen message = “Message2” satırı hataya neden olmaz. Çünkü string message = “Message1” şeklindeki değişken derleyici tarafından otomatik olarak tanımlanmaktadır ve daha sonra değişkenin değeri “Message2” olarak değiştirilmektedir.

* Methodlar her türde parametreye sahip olabilirler.
* Geriye her türde değer döndürebilirler.
* Kendi kendilerini çağırabilirler.
* Methodların parametrelerine default değerler verilebilmektedir.

Example

public void ShowInfo(string name, int age = 0, string city = "İstanbul"){

Console.Write("Name : {0}, Age : {1}, City : {2}", name, age, city);

}

ShowInfo("Camilla"); *//Name : Camilla, Age : 0, City : İstanbul*

ShowInfo("Camilla", 20); *//Name : Camilla, Age : 20, City : İstanbul*

ShowInfo("Camilla", 20, "Amsterdam"); *//Name : Camilla, Age : 20, City : Amsterdam*

ShowInfo("Camilla", city:"Amsterdam"); *//Name : Camilla, Age : 0, City : Amsterdam*

* İsimleri aynı parametre yapıları farklı methodlar tanımlanabilmektedir. Bu işleme methodların aşırı yüklenmesi(**overloading**) adı verilmektedir.

Example

public void AddNumbers(int x, int y){

Console.Write(x + y);

}

public void AddNumbers(int x, int y, int z){

Console.Write(x + y + z);

}

public void AddNumbers(double x, double y){

Console.Write(x + y);

}

* Hazırladığınız metodun geriye birden fazla değer göndermesini isteyebilirsiniz. Bu durumda dizilerden ya da nesnelerden yararlanma yoluna gidebilirsiniz ancak Tuple sınıfı ile bu işlem son derece kolaydır.

Example

static Tuple<int,int> Method1(int x, int y)

{

int kare = x \* x;

int kup = x \* x \* x;

Tuple<int, int> deste = new Tuple<int, int>(kare, kup);

return deste;

}

//Tuple generic bir sınıftır.

//<> diamond operatörleri arasına geriye dönecek bilgilerin tipleri yazılmaktadır.

//diamond operatörleri arasına en fazla 8 adet veri tipi yazılabilmektedir.

//Geriye dönecek bilgiler Tuple sınıfının yapıcı metoduna parametre olarak verilmektedir.

Tuple<int, int> t = Method1(2);

Console.WriteLine(t.Item1); //4

Console.WriteLine(t.Item2); //8

//Tuple nesnesi 2 bilgi içerdiği için ilk özelliğin adı Item1 ikinci özelliğin adı Item2 olur.

Tuple nesnesi hazırlarken Tuple sınıfının statik Create() metodunu kullanabilirsiniz.

Example

static Tuple<int,int> Method1(int x)

{

int kare = x \* x;

int kup = x \* x \* x;

Tuple<int, int> deste = Tuple.Create(kare, kup);

return deste;

}

(ref ve out Parametreleri)

Bir değişken bir methoda parametre olarak gönderildiğinde bu değişkenin bellek adresi değil içeriğinin bir kopyası gönderilir. Örneğin :

Example

public void ChangeValue(int x){

x = 2;

}

int number = 1;

ChangeValue(number);

Console.Write(number); *//1*

Methoda parametre olarak değişkenin referansını göndermek istiyorsanız ref parametresini kullanabilirsiniz.

Example

public void ChangeValue(ref int x){

x = 2;

}

int number = 1;

ChangeValue(ref number);

Console.Write(number); *//2*

Bu örnekte number değişkeni herhangi bir değer içermeseydi hata meydana gelirdi. Bu gibi durumlarda ref yerine out parametresi kullanılır.

Example

public void ChangeValue(out int x){

x = 2;

}

int number = 1;

ChangeValue(out number);

Console.Write(number); *//2*

(Constructor – Yapıcı Method)

Example

class Class1{

public Class1(){ *//Constructor*

}

}

* class ve struct içindeki alanlara(field) ilk değerleri genellikle constructor içerisinden aktarılmaktadır.
* class ve structların örnekleri alındığında yapıcı metotları çalışmaktadır ve geriye ilgili class’dan(veya struct) yola çıkılarak hazırlanan bir nesne gönderirler. Bu yüzden yapıcı metotlar void olarak tanımlanmazlar ve geriye değer gönderemezler. Çünkü zaten geriye bir nesne göndermektedirler.
* class(veya struct) ile aynı ada sahip olurlar.
* Aşırı yüklenebilirler.
* class(veya struct) yapıcı metot içermiyorsa C# derleyicisi otomatikmen bir yapıcı metot oluşturmaktadır.
* Derleyici tarafından oluşturulan constructor class(veya struct) içindeki int türündeki alanlara 0, bool türündeki alanlara false, string türündeki alanlara ise boşluk değeri aktarmaktadır.(başlangıç değeri)
* static constructor tanımlanabilmektedir.

Example

class Class1{

static Class1(){

}

*//Parametresi olamaz.*

*//public olarak tanımlanmazlar. Erişim belirteçlerine sahip olamazlar.*

*//Sadece class'ın ilk instance'ı için çalışır.*

*//Hem static hem de normal constructor tanımlanabilir. Bu durumda static*

*//constructor daha önce çalışır.*

}

(Destructor – Yıkıcı Method)

class’lar garbage collector adı verilen çöp toplama sistemi tarafından bellekten silinmeden önce varsa yıkıcı metotları çalıştırılır.

Example

class Class1{

~Class1(){

}

}

* Parametresi olamaz.
* public olarak tanımlanmazlar.

(Yerel Methodlar)

C# 7.0 ile beraber gelen bir yeniliktir. Artık method içinde method tanımlayabiliyoruz.

Example

static void Main(){

int AddNumbers(int x, int y){

return x + y;

  }

Console.Write(AddNumbers(1, 2));

Console.ReadLine();

}

* Yerel methodlar sadece kapsayıcı methodun içinde yaşarlar.