C Sharp Programlama Dili/Metotlar

< C Sharp Programlama Dili
Gezinti kısmına atla Arama kısmına atla
Ders 10. Metotlar

Şu ana kadar yaptığımız örneklerde hep önceden hazırlanmış `ReadLine()`, `WriteLine()` gibi metotları kullandık. Artık kendi metotlarımızı yapmanın zamanı geldi. Bilmem farkında mısınız, ama aslında bütün örneklerimizde birer metot oluşturmuştuk. O da çalışabilir her programda bulunması gereken `Main` metoduydu. Artık Main metodu gibi başka metotlar derleyip programımızın içinde kullanabileceğiz. Metotlar oluşturarak programımızı parçalara böler ve programımızın karmaşıklığını azaltırız. Ayrıca bazı kodları bir metot içine alıp aynı kodlara ihtiyacımız olduğunda bu metodu çağırabiliriz. Bu sayede de kod hamallığı yapmaktan kurtuluruz.

Metot Yapımı ve Kullanımı

```
int MetotAdi(int a,int b)
{
   return a+b;
}
```

Bu metot, iki tane int türünden girdi alır ve bu girdilerin toplamını int türünden tutar. Ancak bu metodu programımız içinde kullanabilmemiz için bu metodun içinde bulunduğu sınıf türünden bir nesne belirleyip "." operatörüyle bu nesne üzerinden metodumuza erişmeliyiz. Örnek:

```
using System;
class Metotlar
{
   int Topla(int a,int b)
   {
      return a+b;
   }
   static void Main()
   {
      Metotlar nesne=new Metotlar();
      int a=nesne.Topla(2,5);
      Console.Write(a);
   }
}
```

> static olarak tanımlanan metotlara erişmek için metodun içinde bulunduğu sınıf türünden bir nesne belirtmeye gerek yoktur. static olarak tanımlanan metotlara sadece metodun adını yazarak erişilebilir. Örnek:

```
using System;
class Metotlar
{
   static int Topla(int a,int b)
   {
      return a+b;
   }
   static void Main()
   {
      int a=Topla(2,5);
      Console.Write(a);
   }
}
```

- Bütün programlarda önce Main metodu çalışır. Diğer metotlar Main metodunun içinden çağrılmadıkça çalışmaz.
- Eğer metot, içinde bulunduğumuz sınıfta değil de, başka bir sınıf içinde oluşturulmuşsa o metodu kullanabilmek için önce sınıfı yazmamız gerekir. Örnek:

```
using System;
class Metotlar1
{
   public static int Topla(int a,int b)
   {
      return a+b;
   }
}
class Metotlar2
{
   static void Main()
   {
      int a=Metotlar1.Topla(2,5);
      Console.Write(a);
   }
}
```

Dikkat ettiyseniz bu metodun yazıldığı satırın (4. satır) başına `public` anahtar sözcüğü konmuş. `public` sözcüğüyle derleyiciye bu metoda her sınıftan erişilebileceğini söylüyoruz. Eğer public sözcüğü yazılmamış olsaydı bu metoda sadece `Metotlar1` sınıfından erişilebilirdi.

Şimdi de static olmayan bir metodu başka bir sınıf içinde yazalım ve kullanalım:

```
using System;
class Metotlar1
{
   public int Topla(int a,int b)
   {
      return a+b;
   }
}
class Metotlar2
{
   static void Main()
   {
      Metotlar1 nesne=new Metotlar1();
      int a=nesne.Topla(3,9);
      Console.Write(a);
   }
}
```

- Bütün değer tutan metotlar bir değermiş gibi kullanılabilir, ancak değişkenmiş gibi kullanılamaz.
- Herhangi bir değer tutmayan (`WriteLine` gibi) metotları `void` anahtar sözcüğüyle üretiriz. Örnek:

```
using System;
class Metotlar1
{
   static void Yaz(object a,int b)
   {
      for(;b>0;b--)
      {
         Console.Write(a);
      }
   }
   static void Main()
   {
      Yaz("deneme",5);
   }
}
```

Burada oluşturduğumuz `Yaz` metodu aldığı ilk parametreyi ikinci parametre kere ekrana yazar. Örneğin programımızda "deneme" stringi ekrana 5 kez yazdırılıyor.

> Herhangi bir değer tutmayan metotlarda `return;` komutu, yanına herhangi bir ifade olmadan kullanılabilir. Aslında `return;` komutunun asıl görevi metottan çıkmaktır, ancak yanına bazı ifadeler koyularak metodun tuttuğu değeri belirtme vazifesi de görür. Örneğin bir `if` koşulu belirtip, eğer koşul sağlanırsa metottan çıkılmasını, koşul sağlanmazsa başka komutlar da çalıştırılmasını sağlayabiliriz. Ancak doğal olarak bir metodun son satırında `return;` komutunun kullanılması gereksizdir. Örnek kullanım:

```
using System;
class Metotlar1
{
   static void Yaz(object a,int b)
   {
       if(b>10)
       {
         return;
       }
       for(;b>0;b--)
       {
         Console.Write(a);
       }
   }
   static void Main()
   {
      Yaz('n',10);
   }
}
```

Bu programda eğer metoda verilen ikinci parametre 10'dan büyükse metottan hiçbir şey yapılmadan çıkılıyor.

Metotlarla ilgili önemli özellikler

- Metotları kullanırken parametrelerini doğru sayıda, doğru sırada ve doğru türde vermeliyiz.
- Değer tutan metotlarda `return` satırıyla belirtilen ifade, metodu yazarken verilen türle uyumlu olmalıdır.
- Değer tutmayan (`void` ile belirtilmiş) metotlarda `return` komutunun herhangi bir ifadeyle kullanılması yasaktır.
- Değer tutmayan metotların bir değermiş gibi kullanılması yasaktır.
- Yapıcı metotların erişim belirleyicisi olarak public belirtilmelidir.
- Metotlar değer tutmayabileceği gibi, parametre de almayabilirler. Örnek program:

```
using System;
class Metotlar1
{
   static void Yaz()
   {
         Console.Write("deneme");
   }
   static void Main()
   {
      Yaz();
   }
}
```

Buradaki ekrana "deneme" yazan metot herhangi bir parametre almaz. Dolayısıyla da programda kullanırken de parantezlerin içine hiçbir şey yazılmaz.

Bir metodun içinde başka bir metot derlenemez. Örneğin aşağıdaki gibi bir kullanım hatalıdır:

```
using System;
class Metotlar1
{
   static void Yaz()
   {
         Console.Write("deneme");
         static void Ciz()
         {
            Console.Write("\n");
         }
   }
   static void Main()
   {
      Yaz();
   }
}
```

- Metot oluşturulurken metot parantezinde " `static void Yaz(object a,int b`)" veya metot küme parantezlerinin içinde tanımlanan değişkenler metottan çıkıldığında bellekten silinirler. Eğer aynı metot tekrar çağrılırsa söz konusu değişkenler tekrar tanımlanıp tekrar değerler atanır.
- Metot oluşturulurken metot parantezindeki değişkenlerin türleri tek tek belirtilmelidir. Virgül ile ortak tür belirtimi yapılamaz. Yani `static void Ornek(int a,int b)` yerine `static void Ornek(int a, b)` yazılamaz.

Metot parametresi olarak diziler

Örnek:

```
using System;
class Metotlar1
{
   static void Yaz(int[] dizi)
   {
         foreach(int i in dizi)
            Console.WriteLine(i);
   }
   static void Main()
   {
         int[] dizi={1,2,4,7,9};
         Yaz(dizi);
   }
}
```

Buradaki `Yaz` metodu kendisine parametre olarak verilen dizinin elemanlarını alt alta yazdı. Eğer yalnızca int[] türündeki değil bütün türlerdeki dizileri ekrana yazan bir metot yazmak istiyorsak:

```
using System;
class Metotlar1
{
   static void Yaz(Array dizi)
   {
         foreach(object i in dizi)
            Console.WriteLine(i);
   }
   static void Main()
   {
         int[] dizi={1,2,4,7,9};
         Yaz(dizi);
   }
}
```

Bu kullanım doğrudur. Ancak aşağıdaki kullanım hatalıdır:

```
using System;
class Metotlar1
{
   static void Yaz(object[] dizi)
   {
         foreach(object i in dizi)
            Console.WriteLine(i);
   }
   static void Main()
   {
         int[] dizi={1,2,4,7,9};
         Yaz(dizi);
   }
}
```

Çünkü dizilerde değişkenlerdeki gibi bir bilinçsiz tür dönüşümünden bahsetmek imkansızdır.

Dizi ve değişken parametreler arasındaki fark

Aşağıdaki iki programı karşılaştırın:

```
using System;
class Metotlar1
{
  static void Degistir(int[] dizi)
  {
        for(int i=0;i<5;i++)
        {
            dizi[i]=10;
        }
  }
  static void Yaz(Array dizi)
  {
        foreach(object a in dizi)
            Console.WriteLine(a);
  }
  static void Main()
  {
        int[] dizi={1,2,4,7,8};
        Degistir(dizi);
        Yaz(dizi);
  }
}
```

```
using System;
class Metotlar1
{
  static void Degistir(int sayi)
  {
        sayi=10;
  }
  static void Yaz(int sayi)
  {
        Console.WriteLine(sayi);
  }
  static void Main()
  {
        int sayi=1;
        Degistir(sayi);
        Yaz(sayi);
  }
}
```

Bu iki programı çalıştırdığınızda göreceksiniz ki metoda parametre olarak aktarılan dizinin metot içinde bir elemanının değiştirilmesi esas diziyi etkiliyor. Ancak metoda parametre olarak aktarılan değişkenin metot içinde değiştirilmesi esas değişkeni etkilemiyor. Çünkü bir metoda parametre olarak bir dizi verildiğinde derleyici metoda dizinin bellekteki adresini verir; metot o adresteki verilerle çalışır. Dolayısıyla da dizinin herhangi bir elemanındaki değişiklik esas diziyi etkileyecektir. Çünkü gerek esas program, gerekse de metot aynı adresteki verilere erişir. Halbuki bir metoda parametre olarak bir değişken verdiğimizde metot için değişkenin bellekteki adresi önemli değildir, metot için önemli olan değişkenin değeridir. Metot, değişkeni kullanabilmek için ram üzerinde geçici

bir bellek bölgesi tutar ve parametre olarak aldığı değişkenin değerini bu geçici bellek bölgesine kopyalar ve o geçici bellek bölgesiyle çalışır. Metottan çıkıldığında da o geçici bellek bölgesi silinir.

Peki bir metoda aktarılan bir değişkende yapılan bir değişikliğin tıpkı dizilerdeki gibi esas değişkeni etkilemesini ister miydiniz? İşte bunun için C#'ın `ref` ve `out` olmak üzere iki anahtar sözcüğü vardır.

ref anahtar sözcüğü

Örnek:

```
using System;
class Metotlar1
{
  static void Degistir(ref int sayi)
  {
        sayi=10;
  }
  static void Yaz(int sayi)
  {
        Console.WriteLine(sayi);
  }
  static void Main()
  {
        int sayi=1;
        Degistir(ref sayi);
        Yaz(sayi);
  }
}
```

`ref` anahtar sözcüğü değişkenlerin metotlara *adres gösterme* yoluyla aktarılmasını sağlar. Gördüğünüz gibi `ref` sözcüğünün hem metodu çağırırken , hem de metodu oluştururken değişkenden önce yazılması gerekiyor. Bu program ekrana 10 yazacaktır. Ayrıca `ref` sözcüğüyle bir değişkenin metoda adres gösterme yoluyla aktarılabilmesi için esas programda değişkene bir ilk değer verilmelidir. Yoksa program hata verir. Örneğin aşağıdaki program hata verir:

```
using System;
class Metotlar1
{
  static void Degistir(ref int sayi)
  {
        sayi=10;
  }
  static void Yaz(int sayi)
  {
        Console.WriteLine(sayi);
  }
  static void Main()
  {
        int sayi;
        Degistir(ref sayi);
        Yaz(sayi);
  }
}
```

out anahtar sözcüğü

Kullanımı `ref` anahtar sözcüğüyle tamamen aynıdır. Tek farkı `out` ile belirtilen değişkenlere esas programda bir ilk değer verilmesinin zorunlu olmamasıdır. Örneğin aşağıdaki kullanım tamamen doğrudur.

```
using System;
class Metotlar
{
  static void Degistir(out int sayi)
  {
        sayi=10;
  }
  static void Yaz(int sayi)
  {
        Console.WriteLine(sayi);
  }
  static void Main()
  {
        int sayi;
        Degistir(out sayi);
        Yaz(sayi);
  }
}
```

NOT: `ref` sözcüğünün dizilerle kullanımı gereksiz olmasına rağmen C# bunu kısıtlamamıştır. Ancak `out` sözcüğü dizilerle kullanılamaz.

Metotların aşırı yüklenmesi

C#'ta parametre sayısı ve/veya parametrelerin türleri farklı olmak şartıyla aynı isimli birden fazla metot yazılabilir. Buna metotların aşırı yüklenmesi denir.

C#, bir metot çağrıldığında ve çağrılanla aynı isimli birden fazla metot bulunduğunda metodun çağrılış biçimine bakar. Yani ana programdaki metoda girilen parametrelerle yeni olan metotların parametrelerini kıyaslar. Önce parametre sayısına bakar. Eğer aynı isimli ve aynı sayıda parametreli birden fazla metot varsa bu sefer parametre türlerinde tam uyumluluk arar, parametre türlerinin tam uyumlu olduğu bir metot bulamazsa bilinçsiz tür dönüşümünün mümkün olduğu bir metot arar, onu da bulamazsa programımız hata verir. Örnekler:

```
using System;
class Metotlar
{
  static void Metot1(int x,int y)
  {
        Console.WriteLine("1. metot çağrıldı.");
  }
  static void Metot1(float x,float y)
  {
        Console.WriteLine("2. metot çağrıldı.");
  }
  static void Metot1(string x,string y)
  {
        Console.WriteLine("3. metot çağrıldı.");
  }
  static void Main()
  {
        Metot1("deneme","deneme");
        Metot1(5,6);
        Metot1(10f,56f);
  }
}
```

Bu programda üç metot da aynı sayıda parametre almış. Bu durumda parametrelerin türlerine bakılır. Ana programdaki `Metot1("deneme","deneme");` satırıyla üçüncü metot çağrılır. `Metot1(5,6);` metot çağrımının parametre türlerinin tam uyumlu olduğu metot birinci metottur, o yüzden birinci metot çağrılır. Eğer birinci metot olmasaydı ikinci metot çağrılacaktı. Son olarak `Metot1(10f,56f);` satırıyla da ikinci metot çağrılır. Başka bir örnek:

```
using System;
class Metotlar
{
  static void Metot1(float x,float y)
  {
        Console.WriteLine("1. metot çağrıldı.");
  }
  static void Metot1(double x,double y)
  {
        Console.WriteLine("2. metot çağrıldı.");
  }
  static void Main()
  {
        Metot1(5,6);
  }
}
```

Bu programda iki metodun da parametre sayısı eşit, iki metotta da tam tür uyumu yok ve iki metotta da bilinçsiz tür dönüşümü mümkün. Bu durumda en az kapasiteli türlü metot çağrılır. Yani bu programda birinci metot çağrılır. Başka bir örnek:

```
using System;
class Metotlar
{
  static void Metot1(float x,float y)
  {
        Console.WriteLine("1. metot çağrıldı.");
  }
  static void Metot1(int x,int y)
  {
        Console.WriteLine("2. metot çağrıldı.");
  }
  static void Main()
  {
        Metot1(5,6.4f);
  }
}
```

Bu durumda birinci metot çağrılır. Başka bir örnek:

```
using System;
class Metotlar
{
  static void Metot1(float x,float y)
  {
        Console.WriteLine("1. metot çağrıldı.");
  }
  static void Metot1(int x,int y)
  {
        Console.WriteLine("2. metot çağrıldı.");
  }
  static void Main()
  {
        Metot1('f','g');
  }
}
```

Bu durumda ikinci metot çağrılır. Çünkü char hem inte hem de floata bilinçsiz olarak dönüşebilir. Ancak int daha az kapasitelidir.

NOT: Metotların geri dönüş tipi (tuttuğu değerin tipi) faydalanılabilecek ayırt edici özelliklerden değildir. Yani iki metodun parametre sayısı ve parametre türleri aynı ise tuttuğu değer tipleri farklı olsa bile bunların herhangi biri çağrılmak istendiğinde programımız derlenmeyecektir.

```
using System;
class Metotlar
{
  static void Metot1(int x,int y,int z)
  {
        Console.WriteLine("1. metot çağrıldı.");
  }
  static void Metot1(int x,int y)
  {
        Console.WriteLine("2. metot çağrıldı.");
  }
  static void Metot1(float x,int y)
  {
        Console.WriteLine("3. metot çağrıldı.");
  }
  static void Main()
  {
        Metot1(3,3,6);
        Metot1(3.4f,3);
        Metot1(1,'h');
  }
}
```

Burada sırasıyla 1., 3. ve 2. metotlar çağrılacaktır.

Değişken sayıda parametre alan metotlar

Şimdiye kadar metotlarımıza gönderdiğimiz parametre sayısı belliydi ve bu parametre sayısından farklı bir miktarda parametre girersek programımız hata veriyordu. Artık istediğimiz sayıda parametre girebileceğimiz metotlar yapmasını göreceğiz. Örnek:

```
using System;
class Metotlar
{
   static int Toplam(params int[] sayilar)
   {
      if(sayilar.Length==0)
      {
         return 0;
      }
      int toplam=0;
      foreach(int i in sayilar)
      {
         toplam+=i;
      }
      return toplam;
   }
   static void Main()
   {
      Console.WriteLine(Toplam());
      Console.WriteLine(Toplam(5));
      Console.WriteLine(Toplam(5,10));
      Console.WriteLine(Toplam(2,9,12));
      Console.WriteLine(Toplam(7,12,45));
      Console.WriteLine(Toplam(123,12,5,7,9,4,12));
   }
}
```

Burada aldığı parametrelerin toplamını tutan bir metot yazdık. Eğer metoda hiç parametre girilmemişse 0 değerini döndürmektedir. Gördüğünüz gibi `params` anahtar sözcüğüyle girilen tüm parametreleri int[] türündeki sayilar dizisine aktardık ve bu diziyi metotta gönlümüzce kullandık. Bir de bu programımızda `Length` özelliğini gördünüz. Özellikler metotlara benzerler, metotlardan tek farkları "()" kısımları olmamasıdır. Dolayısıyla parametre almazlar. Örneğimizde `sayilar.Length` satırı sayilar dizisinin eleman sayısını int türünden tutar. Başka bir örnek:

```
using System;
class Metotlar
{
   static int Islem(string a,params int[] sayilar)
   {
      if(a=="carp")
      {
         if(sayilar.Length==0)
         {
            return 1;
         }
         int carpim=1;
         foreach(int i in sayilar)
         {
            carpim*=i;
         }
         return carpim;
      }
      else if(a=="topla")
      {
         if(sayilar.Length==0)
         {
            return 0;
         }
         int toplam=0;
         foreach(int i in sayilar)
         {
            toplam+=i;
         }
         return toplam;
      }
      else
         return 0;
   }
   static void Main()
   {
      Console.WriteLine(Islem("topla",3,4,7,8));
      Console.WriteLine(Islem("carp",5,23,6));
   }
}
```

Bu programdaki gibi metodumuzda değişken parametre yanında bir ya da daha fazla normal sabit parametre de olabilir. Ancak değişken parametre mutlaka en sonda yazılmalıdır.

NOT: Değer döndüren metotlarımız mutlaka her durumda değer döndürmelidir. Örneğin metodumuzda sondaki `else` kullanılmazsa programımız derlenmez. Çünkü else'i kullanmasaydık birinci parametre yalnızca "carp" veya "topla" olduğunda metodumuz bir değer döndürecekti. Ve bu da C# kurallarına aykırı.

NOT: Değişken sayıda parametre alan metotlar aşırı yüklenmiş metotlar olduğunda değerlendirilmeye alınmaz. Örnek:

```
using System;
class Metotlar
{
  static void Metot1(int x,int y)
  {
        Console.WriteLine("1. metot çağrıldı.");
  }
  static void Metot1(int x,params int[] y)
  {
        Console.WriteLine("2. metot çağrıldı.");
  }
  static void Main()
  {
        Metot1(3,6);
  }
}
```

Burada 1. metot çağrılır. Ancak,

```
using System;
class Metotlar
{
  static void Metot1(int x,int y)
  {
        Console.WriteLine("1. metot çağrıldı.");
  }
  static void Metot1(int x,params int[] y)
  {
        Console.WriteLine("2. metot çağrıldı.");
  }
  static void Main()
  {
        Metot1(3,6,8);
  }
}
```

Burada 2. metot çağrılır.

Kendini çağıran metotlar

C#'ta bir metodun içinde aynı metot çağrılabilir. Örnek:

```
using System;
class Metotlar
{
   static int Faktoriyel(int a)
   {
      if(a==0)
      {
         return 1;
      }
      return a*Faktoriyel(a-1);
   }
   static void Main()
   {
      Console.WriteLine(Faktoriyel(0));
      Console.WriteLine(Faktoriyel(1));
      Console.WriteLine(Faktoriyel(4));
      Console.WriteLine(Faktoriyel(6));
   }
}
```

Programlamadaki metot kavramı aslında matematikteki fonksiyonlar konusunun aynısıdır. Matematikteki fonksiyonlar konusunda öğrendiğiniz bütün kuralları metotlarda uygulayabilirsiniz. Örneğin yukarıdaki örnek, matematikteki fonksiyonları iyi bilen birisi için fazla karmaşık gelmeyecektir. Örneğin:

> Matematikteki fonksiyonlar konusunu bilen birisi bilir ki bir fonksiyona parametre olarak o fonksiyonun tersini verirsek sonuç x çıkacaktır. Örneğin f(x)=2x+5 olsun. f(x) fonksiyonunun tersi $f^{-1}(x)=(x-5)/2$'dir. Şimdi f(x) fonksiyonuna parametre olarak (x-5)/2 verirsek sonuç x olacaktır. Yani $f(x)\circ f^{-1}(x)=x$'tir. Şimdi bu kuralı bir metotla doğrulayalım.

```
using System;
class Metotlar
{
   static float Fonksiyon(float x)
   {
      return 2*x+5;
   }
   static float TersFonksiyon(float x)
   {
      return (x-5)/2;
   }
   static void Main()
   {
      float x=10;
      Console.WriteLine(Fonksiyon(x));
      Console.WriteLine(TersFonksiyon(x));
      Console.WriteLine(Fonksiyon(TersFonksiyon(x)));
   }
}
```

Bu program ekrana sırasıyla 25, 2.5 ve 10 yazacaktır. Eğer ortada bir bölme işlemi varsa ve matematiksel işlemler yapmak istiyorsak değişkenler int türünden değil, float ya da double türünden olmalıdır. Çünkü int türü ondalık kısmı almaz, dolayısıyla da int türüyle yapılan bölme işlemlerinde hatalı sonuçlar çıkabilir. Çünkü örneğin 2 sayısı 2.5 sayısına eşit değildir.

Opsiyonel parametreler

İstersek bir metodun bir veya daha fazla parametresinin seçimli (opsiyonel) olmasını sağlayabiliriz. Bunu, metodun parametrelerini yazarken bir veya daha fazla parametreye değer atayarak yaparız. Bu durumda metot çağrılırken ilgili parametre kullanılmazsa metot tanımında o parametreye atanmış değer kullanılır. Örnek:

```
static void Metot(string s1, string s2="varsayilan", int i=0)
{
//...
}
Metot("deneme"); //s1="deneme" s2="varsayilan" i=0
Metot("deneme1", "deneme2"); //s1="deneme1" s2="deneme2" i=0
Metot("deneme1", "deneme2", 2); //s1="deneme1" s2="deneme2" i=2
Metot("deneme",2); //hatalı
```

Birden fazla opsiyonel parametrenin yan yana olması durumunda sonraki opsiyonel parametreye bir değer atayabilmemiz için önceki parametreye/parametrelere de değer atamamız gerekir. Örneğin yukarıdaki örnekteki son metot çağrısı hatalıdır.

Metot aşırı yüklemede opsiyonel parametreler (tıpkı params parametrelerde olduğu gibi) ikinci plandadır. Yani isimleri, parametre sayıları ve parametre tipleri aynı olan iki metottan opsiyonel parametre kullanmayan seçilir. Örnek:

```
static void Metot(int a,int b) {//1. metot}
static void Metot(int a,int b=0) {//2. metot}
Metot(1,2);
```

Burada 1. metot çağrılır. Ancak opsiyonel parametre ayrımı sadece son aşamada kullanılır. Daha öncesinde parametre sayısı ve parametre tipi (tam uyumluluk) şartlarının sağlanması gerekir. Opsiyonel parametre ile params parametrenin karşı karşıya gelmesi durumunda üstünlük opsiyonel parametrededir. Sonuç olarak şimdiye kadar öğrendiğimiz metot aşırı yükleme kurallarına göre metotların üstünlük sırası şöyledir:

1. İsim, parametre sayısı ve parametre tipi (bilinçsiz dönüşüm dahil)
2. Eğer birinci koşul ayrım yapmaya yetmiyorsa parametre tipinin tam uyumlu olduğu metot seçilir.
3. Eğer ilk iki koşul ayrım yapmaya yetmiyorsa opsiyonel/params parametreli olmayan tercih edilir.
4. Eğer ilk üç koşul ayrım yapmaya yetmiyorsa opsiyonel parametreli olan seçilir.
5. Eğer önceki koşullar ayrım yapmaya yetmiyorsa hata oluşur.

İsimlendirilmiş parametreler

Şimdiye kadar bir metodun hangi parametresine değer gönderdiğimizi parametrenin sırasından ayarlıyorduk. Ancak istersek hangi parametreye değer gönderdiğimizi açık olarak ismiyle belirtebiliriz. Bu sayede parametreleri istediğimiz sırada belirtebiliriz. Örnek:

```
static void Metot(string s1,string s2="varsayilan",int i=0)
{
//...
}
Metot("deneme", i:5, s2:"deneme2");
Metot(s2:"deneme2", s1:"deneme1", i:5);
```

Bir isimlendirilmiş parametre kullandıktan sonraki bütün parametreler de isimlendirilmiş olmalıdır. İsimlendirilmiş bir parametreden sonra isimsiz bir parametre kullanamayız. Aynı parametreye hem isimli hem de isimsiz atama yapamayız. Opsiyonel veya params olmayan her parametreye -isimli veya isimsiz- mutlaka atama yapmalıyız.

params parametrelerin sadece bir tane parametresi olması durumunda bu parametreye isimli atama yapabiliriz. Aksi halde isimsiz atama yapmamız gerekir. Ancak illaki birden fazla parametreden oluşan isimli params parametre istersek;

```
static void Metot(string s1, params int[] i){}
Metot(s1:"deneme", i: new int[]{2,3,5,7});
```

şeklinde bir kullanım da mümkündür.

Main metodu

Bildiğiniz gibi çalışabilir her programda Main metodunun bulunması gerekiyor. Bütün programlarda önce Main metodu çalıştırılır. Diğer metotlar Main metodunun içinden çağrılmadıkça çalışmaz. Bu özellikler dışında Main metodunun diğer metotlardan başka hiçbir farkı yoktur. Ancak şimdiye kadar Main metodunu yalnızca `static void Main()` şeklinde yaptık. Yani herhangi bir parametre almadı ve herhangi bir değer tutmadı. Ancak Main metodunun bir değer tutmasını veya parametre almasını sağlayabiliriz.

Main metodunun değer tutması

Main metodunu `static int Main()` şeklinde de yazabiliriz. Peki bu ne işimize yarayacak? Buradaki kritik soru "Main metodunu kim çağırıyor?"dur. Biraz düşününce bu sorunun cevabının işletim sistemi olduğunu göreceksiniz. Peki işletim sistemi Main metodunun tuttuğu değeri ne yapacak? Programlar çeşitli şekillerde sonlanabilirler. Örneğin Main metodu 0 değerini döndürürse işletim sistemi programın düzgün şekilde sonlandırıldığını, 1 değerini döndürürse de hatalı sonlandırıldığını anlayabilecektir. Ancak şimdilik bunları kullanmamıza gerek yok.

Main metodunun parametre alması

Tahmin edeceğiniz gibi Main metoduna parametreler işletim sisteminden verilir. Main metodu, komut satırından girilen argümanları string türünden bir diziye atayıp programımızda gönlümüzce kullanmamıza izin verir.

```
static void Main(string[] args)
```

Burada komut satırından girilen argümanlar string[] türündeki args dizisine aktarılıyor. Bu diziyi programımızda gönlümüzce kullanabiliriz. Örneğin programımız deneme.exe olsun. Komut satırından

girilirse ilk sözcük olan `deneme` ile programımız çalıştırılır ve sözcükleri de string[] türündeki args dizisine aktarılır. Örnek bir program:

```
using System;
 class Metotlar
 {
    static void Main(string[] args)
    {
       Console.WriteLine("Komut satırından şunları girdiniz: ");
       foreach(string i in args)
       {
          Console.WriteLine(i);
       }
    }
 }
```

Bu program komut satırından girilen argümanları ekrana alt alta yazacaktır. Komut satırında program adından sonra girilen ilk sözcük args dizisinin 0. indeksine, ikinci sözcük 1. indeksine, üçüncü sözcük 2. indeksine vs. aktarılır.

System.Math sınıfı ve metotları

.Net sınıf kütüphanesinde belirli matematiksel işlemleri yapan birçok metot ve iki tane de özellik vardır. System.Math sınıfındaki metotlar static oldukları için bu metotları kullanabilmek için içinde bulundukları sınıf türünden bir nesne oluşturmaya gerek yoktur. System.Math sınıfındaki iki özellik matematikteki pi ve e sayılarıdır. Şimdi bu iki özelliği örnek bir programda görelim:

```
 using System;
 class Metotlar
 {
    static void Main()
    {
       double e=Math.E;
       double pi=Math.PI;
       Console.Write("e->"+e+" pi->"+pi);
    }
 }
```

PI ve E özellikleri double türünden değer tutarlar. Şimdi System.Math sınıfındaki bütün metotları bir tablo hâlinde verelim:

Metot	Açıklama
Abs(x)	Bir sayının mutlak değerini tutar.
Cos(x)	Bir sayının kosisünü tutar.
Sin(x)	Bir sayının sinüsünü tutar.
Tan(x)	Bir sayının tanjantını tutar.
Ceiling(x)	x sayısından büyük en küçük tam sayıyı tutar (yukarı yuvarlama).
Floor(x)	x sayısından küçük en büyük tam sayıyı tutar (aşağı yuvarlama).
Max(x,y)	x ve y sayılarının en büyüğünü tutar.
Min(x,y)	x ve y sayılarının en küçüğünü tutar.
Pow(x,y)	x üzeri y'yi tutar.
Sqrt(x)	x'in karekökünü tutar.
Log(x)	x sayısının e tabanında logaritmasını tutar.
Exp(x)	e üzeri x'in değerini tutar.
Log10(x)	x sayısının 10 tabanındaki logaritmasını tutar.

Şimdi bir örnek verelim:

```
using System;
class Metotlar
{
   static void Main()
   {
      int a=Math.Max(10,34);
      int b=Math.Abs(-3);
      double c=Math.Ceiling(12.67);
      Console.Write("Max:"+a+" Abs"+b+" Ceiling:"+c);
   }
}
```

Bu metotlar çeşitli türlerden değer döndürebilirler. Örneğin Ceiling metodu double türünden değer döndürürken başka bir metot int türünden değer döndürebilir. Bunları deneyerek bulabilirsiniz. Açıklama yapma gereksinimi görmüyorum.

: