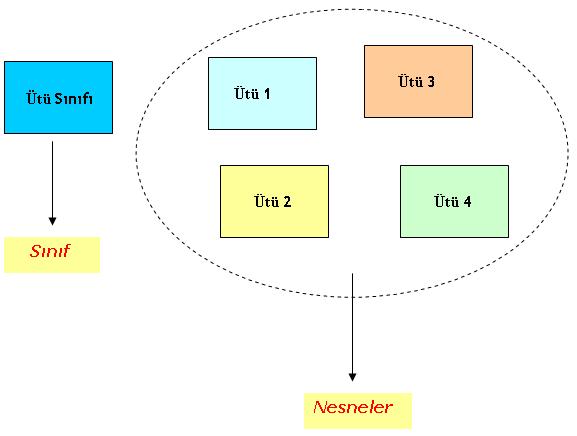
**C# ile Nesne Yönelimli Programlama**

Kimilerine göre geçtiğimiz yüzyılın en önemli buluşu olarak kabul edilen bilgisayar teknolojisi, baş döndürücü bir hızla gelişmektedir. Bilişim sektöründeki değişimler bazen varolan teknolojilere yenilerinin eklenmesi şeklinde olabilir. Diğer taraftan bir kısım yenilikler vardır ki bu alanda büyük değişimlere ve evrimlere yolaçar. Mesela; kişisel bilgisayarların kullanılmaya başlanması veya internetin belli başlı akademik kurumların ve askeri organizasyonların tekelinden alınıp tüm insanlığın hizmetine sunulması gibi.  
  
Hepimizin bildiği gibi bir bilgisayar sistemi iki ana parçadan oluşur. Bunlar donanım(hardware) ve yazılım(software). Donanımın yazılım ile uyumlu çalişması sonucunda sistemlerimiz sorunsuz bir şekilde bizlere hizmet verirler. Ayrıca donanımın amacımimza uygun hizmet vermesi uygun yazılımın geliştirilip kullanılmasına baglıdır.   
  
Yazılım sektöründe program geliştirme konusunda günümüze kadar bir çok yaklaşim denenmiştir. Bunların ilki programın baştan aşağıya sırası ile yazılıp çalıştırılmasıdır. Bu yaklaşımla BASIC dili kullanılarak bir çok program yazıldığını biliyoruz. Burda sorun programın akışı sırasında değişik kısımlara **goto** deyimi ile atlanmasıdır. Program kodu bir kaç bin satır olunca, kodu okumak ve yönetmek gerçekten çok büyük sorun oluyordu.   
  
ıkinci yaklaşım ise prosedürel yaklaşımdır. Programlarda bir çok işin tekrar tekrar farklı değerleri kullanılarak yapıldığı farkedildi. Mesela herhangi bir programda iki tarih arasında ne kadar gün olduğunu bulmak birçok kez gerek olabilir. Bu durumda başlangıç ve bitiş tarihlerini alıp aradaki gün sayısını veren bir fonksiyon yazılabilir ve bu fonksiyon ihtiyaç duyulduğu yerde uygun parametrelerle çağrılıp istenen sonuç elde edilebilir. Prosedürel yaklaşım Pascal ve C dillerinde uzun yıllar başari ile kullanılmıştır.  
  
Ama her geçen gün programların daha karmaşık bir hal alması, program kodunun kurumsal uygulama projelerinde onbinlerce satırı bulması ve yazılım geliştirme maliyetinin çok arttiğını gören bilim adamları, programcılara yeni bir yaklaşımın kullanılabilineceğini öğrettiler. Bu yakaşımın ismi Nesne Yönelimli Programlama(Object Oriented Programlama)dır.   
  
Nesne yönelimli programlama tekniği, diger yaklaşımlara nazaran, yazılım geliştiren insanlara büyük avantajlar sağlamaktadır. Birincisi karmaşık yazılım projelerinin üretilmesini ve bakımını kolaylaştırıyor olmasıdır. Diğeri ise program kodunun tekrar kullanılabilmesine (code-reusability) olanak sağlamasıdır. Bu noktada program kodunun tekrar kullanılabilmesi profesyonel yazılım şirketlerinin maliyetlerini azaltmıştır. Dolayısi ile programların lisans ücretleri düşmüş ve sektörün sürekli olarak canlı kalmasına ve rekabet içinde gelişmesine yardımcı olmuştur.

**Nesne Yönelimli Programlama Nedir?**

Nesne yönelimli programlamada esas olan, gerçek hayatta varolan olguların programlamaya aktarılmasındaki yeni yaklaşımdır. Prosedürel programlamada verilerimiz ve fonksiyonlarımız vardı. Yani veri ve bu veriyi işleyen metodlar etrafinda dönüyordu herşey.   
  
Aslında nesne yönelimli programlamada da iki önemli birim veri ve veriyi işleyip mantıklı sonuçlar üreten metodlar bulunur. Ama burdaki fark gerçek hayattaki olguların da daha iyi gözlenip programlama dünyasına aktarılmasındadır.   
  
Mesela elimizde bir ütümüz olsun. Ütünün markası, modeli, rengi, çalıstığı elektrik voltajı, ne tür kumaşları ütüleyebildiği bu ütüye ait özelliklerdir (veri). Aynı zamanda ütümüzü ısıtabiliriz, ütüleme işinde kullanabiliriz ve soğumaya bırakabiliriz. Bunlar ise ütünün fonksiyonalarıdır(metod). Eğer ütü ile ilgili bir program yapmış olsak ve nesne yönelimli programlama tekniğini kullansak hemen bir ütü sınıfı(class) oluştururduk. Bu sınıfta ütüye ait bilgiler (veriler) ve ütü ile yapabileceğimiz işler(metod) bulunurdu. O zaman nesne yönelimli programlama da bir sınıfta, sınıfa ait veriler ve bu verileri işleyip bir takiı faydalı sonuçlar üreten fonksiyonlar/metodlar bulunur.  
  
Dahası, biz birtane ütü sinifi tasarlarsak bu sınıftan istediğimiz sayıda değişik ütüler(object veya instance) yapabiliriz. Ağagidaki şekilde ütü sınıfı ve bu sınıftan oluşturduğumuz neslerin görsel olarak anlatımı bulunmaktadır.



**Sınıf Nedir ve Nasıl Tasarlanır?**

Az önce de değindiğimiz gibi, sınıf bir yazılım kurgusudur ve gerçek hayattaki herhangi bir olguyu modelleyen ve bu olguya ait özellikleri(veri) ve davranışları(metdodlar) tarifleyen yapıdır.  
  
ısterseniz ütü ile ilgili bir örnek üzerinde çalışalım. Aşağıdaki örnek C# program kodunu lütfen dikkatlice inceleyeniz. 

|  |
| --- |
| using System;  class Ütü\_Örnegi {       static void Main(string[] args)      {           Ütü ütü1= new Ütü("Beyaz", "AyBakır");            Console.WriteLine(ütü1.Isın(70));           Console.ReadLine();      }  }  class Ütü {      public int sıcaklık;      public string renk;      public string marka;       public Ütü(string renk, string marka)      {            sıcaklık=15;           this.renk=renk;           this.marka= marka;            Console.WriteLine(sıcaklık+ "derece sıcaklığında,\n "                                    + renk + " renginde ve\n "                                    + marka +" markasıyla bir ütü nesnesi oluşturuldu\n\n");       }       public string Isın(int derece)      {           sıcaklık+=derece;           return "şu an sıcaklıgım: " + sıcaklık+ " derece";      }  } |

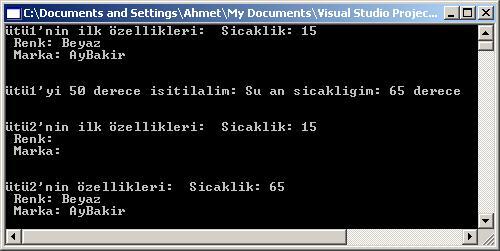
Yukarıdaki örnek programımızda önce altta bulunan **Ütü** sınıfımızı inceleyelim. Bu sınıfın sıcaklık, renk ve marka olmak üzere üç adet verisi vardır. Ayrıca Ütü sınıfımızın Ütü(string renk, string marka) şeklinde tanımlanmış yapılandırıcısı (constructor) vardır. Yapılandırıcılar bir sınıftan oluşturulan nesnelerin ilk değerlerini atama ve baslangıç işlemlerini yapmak için kullanılırlar. Ütü sınıfınımızın yapılandırcısı, oluşturulan her ütü nesnesinin sıcaklığını varsayılan değer olarak 15 dereceye ayarlıyor. Ayrıca paratmetre olarak alınan renk ve marka değerlerini de atayıp, ütüye ait özellikleri ekrana yazdırıyor. Ütü sınıfına ait olan diğer metod ise Isın(int derece) olarak tanımladığımız metoddur. Bu metod ütünün sıcaklığını derece parametresinde verilen değer kadar artırıp sonucu string tipinde geri dönderiyor.  
  
**Ütü\_Örnegi** sınıfına geri dönersek, burda sadece Main() metodunun bulunduğunu görürüz. Main() metodu, C# dilinde her programda bulunması gereken zorundu bir metoddur. Çünkü programın çaliştırılması buradan başlar. Ayrıca her C# programında sadece bir tane Main() metodu bulunur. Örneğimizde Main() metodundaki en önemli satir:   
  
        Ütü ütü1= new Ütü("Beyaz", "AyBakır");  
  
satırıdır. Bu satırda Ütü sınıfına ait ütü1 nesnemizi oluşturuyoruz. Yukarıdaki satırdan da görebileceğimiz gibi herhangi bir sınıfa ait yeni nesneyi oluştururken genel olarak şu yapş kullanılır:  
  
        SınıfAdı nesneAdı = new SınıfAdı(parametre1, parametre2, … parameter N)

**Referans Tipleri ve Nesneler**

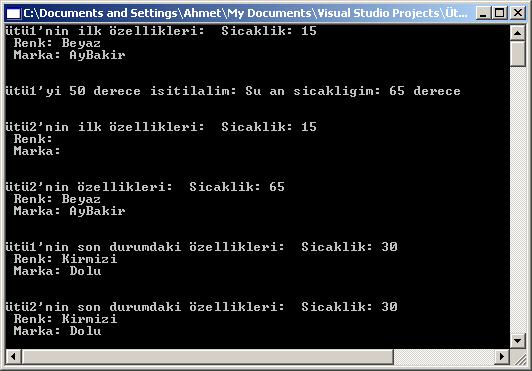
Nesnelerin en önemli özelliği referans tipinde olmalarıdır. Referans tiplerinde bir nesnenin değeri saklanmaz. Sadece nesnenin hafızadaki (heap memory) yeri saklanır. Bir nesnenin hafizadaki yerini new operatörü geri dönderir. Aşağıdaki program kodu ile referans tiplerine ait bir kaç temel özelliği inceleyelim:

|  |
| --- |
| using System;  class Ütü\_Örnegi {      static void Main(string[] args)     {         Ütü ütü1= new Ütü("Beyaz", "AyBakır");          Console.WriteLine("ütü1'nin ilk özellikleri: " +                                  ütü1.Özellikler());         Console.WriteLine("ütü1'i 50 derece ısıtalım: "+                                  ütü1.Isin(50)+ "\n\n");          // ütü2 nesnemizi renksiz ve markasız olarak olusturalım:         Ütü ütü2 = new Ütü("","");           //ütü2 nesnemizin özelliklerine bir göz atalım:         Console.WriteLine("ütü2'nin ilk özellikleri: " +                                  ütü2.Özellikler());          ütü2= ütü1; // ütü2 nesnemizi ütü1 nesnemize atıyoruz.          Console.WriteLine("ütü2'nin özellikleri: " +                                  ütü2.Özellikler());          Console.ReadLine();     }  } |
| class Ütü {     public int sıcaklık;     public string renk;     public string marka;      public Ütü(string renk, string marka)     {         sıcaklık=15;         this.renk=renk;         this.marka= marka;     }      public string Isın(int derece)     {         sıcaklık+=derece;         return "şu an sıcaklığım: " + sıcaklık+ " derece";     }       public string Özellikler()     {         string sonuc= " Sıcaklık: " + sıcaklık+                             "\n Renk: " + renk+                             "\n Marka: " + marka+ "\n\n";         return sonuc;      } } |

ısterseniz kodumuzu incelemeye yine Ütü sınıfından başlayalım. Bir önceki örneğe göre sınıfımızda degişikler yaptık. Ütü sınıfının özelliklerini gösteren ayrı bir metod yazdık. Bu metod Özellikler() isimli olsun. Doğal olarak Ütü sınıfının yapılandırıcısında, sınıf oluşturulduktan hemen sonra nesnenin özelliklerini gösteren kısım kaldırıldı. şimdi ise Ütü\_Örnegi sınıfındaki Main() metodu üzerinde yoğunlaşalım. Yine, birinci örnekte olduğu gibi, ütü1 nesnemizi   
  
        Ütü ütü1= new Ütü("Beyaz", "AyBakır");   
  
ile oluşturuyoruz. Sonra ütü1 nesnesinin özelliklerini ekrana yazdırıyoruz. Bir sonraki satırda ise ütü1 nesnemizi 50 derece ısıtıp sonucu ekrana yazıyoruz.  
  
Artık ikinci bir ütü nesnesinin oluşturmanin zamanı geldiğini düşünüp  
  
        Ütü ütü2 = new Ütü("","");   
  
satırında ütü2 nesnemizi oluşturuyoruz. Burada dikkatinizi çektiği gibi ütü2 nesnesinin renk ve marka özelliklerini boş olarak (null) oluşturuyoruz. ütü2 nesnesini oluşturunca özelliklerine gözatmak için   
  
        Console.WriteLine("ütü2'nin ilk özellikleri: " +   
                                 ütü2.Özellikler());  
  
satırlarını yazıyoruz. Bir sonaki satır programımızın en önemli satırıdır. Bu satırda ***ütü2= ütü1*** ifadesi ile ütü2 nesnesinin referansını ütü1 nesnesinin referansına atıyoruz. Değer tiplerinde "**=**" operatörü ile bir değişkenin değeri diğerine kopyalanır. Ama nesneler referans tipleri oldukları için, "**=**" operatörü bir nesnenin referansı diğerine atar. Bu durumda ütü2 nesnesi artik ütü1'in elemanlarının adreslerini içerir. Bu andan sonra ütü2'nin eskiden gösterdiği sicaklik, renk ve marka verilerine erişmemiz mümkün değildir. Artık onlar hafızada bir yerde ve biz bilmiyoruz. Bu verilerin hafızada kapladıkları alanı .NET Framework'nun çöp toplayıcısı (garbage collector) uygun gördüğü bir anda temizleyecektir.   
  
ütü2 nesnesinin son durumda özelliklerine bakacak olursak gerçektende ütü1'i referans ettiğini görürüz. Yukarıdaki programımızı çalıştırdığımızda aşağıdaki sonucu elde ederiz:



Son olarak şu ilginç soruya cevap vermeye çalışalım. "Bir önceki program kodunun sonuna aşağıdaki kısmı eklersek ne gibi bir değişiklik olur?"  
  
// ütü2 nesnemizin özelliklerini degiştiriyoruz:  
ütü2.sıcaklık=30;  
ütü2.renk="Kırmızı";  
ütü2.marka="Dolu";  
  
//ütü1 nesnemizin özellikerine bir göz atalım:  
Console.WriteLine("ütü1'nin son durumdaki özellikleri: " +   
                          ütü1.Özellikler());  
  
//ütü1 nesnemizin özellikerine bir göz atalım:  
Console.WriteLine("ütü2'nin son durumdaki özellikleri: " +   
                          ütü2.Özellikler());  
  
Yukarıdaki kodun eklenmiş halini derleyip çalıştirmayı deneyelim. Sonuç aşağıdaki gibi olacaktır:



Evet ütü1 ve ütü2'nin son durumdaki özellikleri aynıdır. Bu durumdan şu sonuca ulaşabiliriz: " ütü1 ve ütü2'nin herhangi biri ile özellikleri degiştirirsek diğeri de aynı özelliklere sahip olur. Çünkü her iki nesne de hafızada aynı yere referans ediyorlar."