Nesne Yönelimli Programlama

4.Hafta

Konu: Fonksiyonlar,Nesneye Yönelik Programlama Tarihçesi, Farklılıklar, Nesne ve Sınıf Kavramı

Dr. Öğr. Üyesi Güncel SARIMAN
E-posta: guncelsariman@mu.edu.tr

Fonksiyonlar

Metotlar belirli işlemleri yerine getiren kod bloklarıdır. Aynı kodların tekrar tekrar kullanılması gereken durumlarda büyük kolaylık sağlamaktadır. Oluşturulan **metot,** ismiyle çağırılarak içerisinde bulunan kod bloklarının çalıştırılması sağlanır.

```
<Erişim Biçimi>  <Geri Dönüş Tipi>  Metot_İsmi (Parametre)
{
    Metodun içinde çalışacak kodlar;
}
```

Fonksiyonlar

Metotların **erişim biçiminin** yazılması zorunlu değildir. Yazılmadığı takdirde **private** olarak kabul edilir. ama istenirse private , **public**, **static** vs.. erişim tipini belirleyebilirsiniz.

Geri dönüş tipinin metot tanımlanırken **mutlaka** belirtilmesi gerekir. Burada metottan geriye herhangi bir değer dönmeyecekse **void** anahtar sözcüğü kullanılmalıdır. Eğer geriye bir değer döndürülecek ise geri dönecek **değerin türü** (string, int, double, List<>,vs..) belirtilmelidir. Geriye değer döndüren metotlarda geriye döndürülecek değer **return** anahtar kelimesiye bildirilmelidir.

Fonksiyonlar

Fonksiyon **parametreli** veya **parametresiz** olabilmektedir. Eğer metot parametre almıyorsa yani dışarıdan değer verilmeyecekse "**( )**" parantez açılıp kapatılarak metot oluşturulur.

1. Geriye Değer Döndürmeyen ve Parametre Almayan metotlar.
2. Geriye Değer Döndürmeyen ve Parametre Alan metotlar.
3. Geriye Değer Döndüren ve Parametre Almayan metotlar.
4. Geriye Değer Döndüren ve Parametre Alan metotlar.

Fonksiyonlar

```
private static void Hesapla()
    {
      Console.Write("Kısa kenarı girin :");
      int kk=Convert.ToInt32(Console.ReadLine());
      Console.Write("Uzun kenarı girin :");
      int uk=Convert.ToInt32(Console.ReadLine());
      int alan=kk*uk;
      Console.WriteLine("Alan = {0}", alan);
    }
```

```
static void Main(string[] args)
    {
      Hesapla();
      Console.WriteLine("www.yazilimkodlama.com");
      Console.ReadKey();
    }
```

Fonksiyonlar

```
private static void Hesapla(int kisa,int uzun)
    {
      int alan=kisa*uzun;
      Console.WriteLine("Alan = {0}", alan);
    }
```

```
static void Main(string[] args)
    {
      Console.Write("Kısa kenarı girin :");
      int kk = Convert.ToInt32(Console.ReadLine());
      Console.Write("Uzun kenarı girin :");
      int uk = Convert.ToInt32(Console.ReadLine());
      Hesapla(kk,uk);
      Console.WriteLine("www.yazilimkodlama.com");
      Console.ReadKey();
    }
```

Fonksiyonlar

```
private static int Hesapla()
      {
         Console.Write("Kısa kenarı girin
:");
         int kk =
Convert.ToInt32(Console.ReadLine());
         Console.Write("Uzun kenarı girin
:");
         int uk =
Convert.ToInt32(Console.ReadLine());
         int alan=kk*uk;
         return alan;
      }
```

```
static void Main(string[] args)
    {
       int sonuc = Hesapla();
       Console.WriteLine("Alan = "+ sonuc);
       Console.WriteLine("www.Google.com");
       Console.ReadKey();
    }
 }
```

Fonksiyonlar

Örnek;

Kulllanıcıdan Vize ve Final yazılı notu istenecek ve bu değerler **“durum”** isimli metoda gönderilecektir. Metotta ise gerekli katsayılar olan vize ve final in değerlendirme kriterini arayüzden alacaktır.

geriye dönen «geçti,kaldı» string değerin ekranda gösterilmesini ise messagebox ile gerçekleştiriniz.

Listeler

Dinamik listeler için, .NET Framework, ArrayList ve List sınıflarını kullanıma sunmuştur.

ArrayList : Dizinin yaptığı tüm işleri yapar ama sınır getirilmediğinden istediğimiz kadar eleman ekleyebiliriz. Aldığı her elemanı boxing işlemi ile object olarak sakladığından Hertürlü değeri alabilir. Add,Remove,Sort gibi metodlar kullanışlı metodlardır.

List : List dizinin yaptığı tüm işlemleri yapar buna ek olarak en önemli özelliği belirlenen türde veri saklayabilir. Bu sebepten dolayı veri saklanırken herhangi bir boxing işlemine tabi tutulmayacağından performans olarak ArrayList'den daha iyidir.

Listeler

```
static void Main(string[] args)
{
 ArrayList objectList = new ArrayList();
List<int> intList = new List<int>();
List<Racer> racers = new List<Racer>();
 ArrayList objectList 2= new ArrayList(10);
 List<int> intList2 = new List<int>(25);
}
//İlk Değer Atama
List <int>intList = new List<int>() { 1, 2 };
List <string>stringList = new List<string>() { "bir", "iki", "üç" };
```

Listeler

```
List<int> intList = new List<int>();
intList.Add(1);
intList.Add(2);
List<string> stringList = new List<string>();
stringList.Add("Bir");
stringList.Add("İki");
```

Listeler

AddRange

```
List<Racer> racers = new List<Racer> (20);
racers.AddRange(new Racer[] {
                        new Racer("Niki","Lauda","Autria",25),
                        new Racer("Ayrton","Senna","Brazil",41)
                                    });
```

Insert() metodu ile belirli bir pozisyona eleman eklemek mümkündür.

```
racers.Insert(3, new Racer("Phil", "Hill", "USA", 3));
```

Arama için

IndexOf(), LastIndexOf(), FindIndex(), FindLastIndex(), Find(), FindLast() gibi metotlar kullanılabilir. List sınıfının Exists() metodu da kullanılabilir.

LINQ

LINQ kodlarımız içerisinde, farklı dil kullanmadan, çeşitli ortamlardaki verileri sorgulayıp, filtrelemek için kullanılır.

LINQ öncesinde daha çok kod yazarak yapılan işlemler artık daha az eforlai daha kısa zamanda yapılabiliyor. Bu durum da üretkenliği arttırıyor.

Çoğu SQL komutlarına benzer ya da aynı olan C# anahtar kelimelerinden oluşan ifadeler ile .NET koleksiyonlarına LINQ sorguları yazılır. Bu anahtar kelimeler, LINQ Standart Sorgu Operatörleri olarak anılır(LINQ Standart Query Operators).

LINQ

Farklı LINQ uygulamaları mevcuttur.

LINQ to Objects

Bellek üzerindeki nesnelerden oluşan koleksiyonları sorgulamak

LINQ to SQL

SQL Server veri tabanındaki tabloları sorgulamak

LINQ to DataSet

ADO.NET DataSet tiplerini sorgulamak.

LINQ to XML

XML verilerini sorgulamak.

LINQ to Entity

ADO.NET Entity Framework tarafından oluşturulan varlıkları sorgulamak.

LINQ

```
int[] scores = new int[] { 97, 92, 81, 60 };
IEnumerable<int> scoreQuery =from score in scores where score > 80 select score;
foreach (int i in scoreQuery)
{
   Console.Write(i + " ");
}
```

OOP Tarihçesi ve Farklılıklar

OOP mimarisi 1960 yıllarında Alan Kay tarafından ortaya atılmış ve bence yazılımın altın çağı diyebileceğimiz bir mimaridir.

Birçok yeni başlayan arkadaşın kafasını karıştıran bir yapı olsa da OOP mimarisi yazılımcıların sırtından çok büyük bir yük almış ve yazılım maliyetlerini ve süresini kısaltmıştır.

1960lar ve öncesinin benimsediği Prosedürel Programlama mimarisi ortak işlerin yapılacağı fonksiyonları yazarak onları her alanda kullanmak üzerine kuruluydu.

OOP Tarihçesi ve Farklılıklar

Uzun bir süre bu yönde ilerleyen yazılımcıların büyük ölçüde işlerini görüyordu ancak birçok zorluk ve sıkıntıyı da beraberinde getiriyordu. Örneğin;

- ✓ Uygulamayı bir modüler bir yapıda değil bir bütün olarak kodlanmak zorundaydı ve geliştiren ekibin her üyesi bütün uygulamaya hakim olmalıydı
- ✓ Yeni katılan geliştiricilerin uygulamada değişiklikler yapıp ekibe tam anlamıyla katılması için uzun bir adaptasyon süresinden geçmesi gerekiyordu
- ✓ Uygulama bütün halde kodlandığı için en ufak değişiklik de farklı bölümlerde büyük sıkıntılar çıkarabiliyordu ve debug olayları oldukça zorlaşıyordu.
- ✓ Yıllar geçtikçe donanım teknolojileri yükseldikçe yazılımdan istenilen özellikler artmaya başladı ve buda beraberinde çok fazla kod, çok fazla zaman getirdi. Projeler belirlenen zamanlarda bitirilemiyor, kontrolü zor olmaya başladı. Projelerin büyük bir kısmı zaman yetersizliğinden iptal edilmeye başlandı.

OOP Tarihçesi ve Farklılıklar

Nesne Mimarisi Sonucunda;

Uygulamalar nesnelerden oluşmalı, nesnelerde objelerden oluşmalıdır. Ve bu nesnele belirli bir iş yapmak için geliştirilebilmelidir.

Nesneler birbirleriyle haberleşebilmelidir.

Nesne Yönelimli Programlama 'da, programlama ortamındaki her şey bir nesne olarak kabul edilir ve nesnelerin özellikleri değiştirilerek onlara yeni biçimler verilebilir. Ayrıca her nesnenin duyarlı olduğu durumlar da mevcuttur. Her nesne üzerine uygulanabilecek farklı metotlar oluşturulmuştur.

Yapısal Programlama 'da ağırlık programlama komutlarındayken, Nesne Yönelimli Programlama 'da yazılımcının ortamdaki nesneler, bunların özellikleri, duyarlı oldukları olaylar ve nesnelere uygulanabilecek metotlar hakkında ayrıntılı bilgi sahibi olması gerekir.

Nesne Yönelimli Programlamanın yararlı yönleri

✓ **Yazılımın Bakım (Maintenance) Kolaylığı**

Nesne Yönelimli Programlama dillerinde, nesneler ait oldukları sınıfların sunduğu şablonlarla temsil edilirler. Birbirinin benzeri ya da aynı konu ile ilişkili sınıflarsa bir araya getirilerek *namespace* ya da *package (paket)* adı verilen sınıf kümeleri oluşturulur.

Bu durumda yazılımın bakımı demek; ya mevcut sınıflarda değişiklik ya da yeni sınıf eklenmesi anlamına gelir. Bu da yazılımın tümünü hiçbir şekilde etkilemeyecek olan bir işlemdir. Oysa klasik yapısal programlama dilleriyle geliştirilen programlarda yazılımın bakım maliyeti üretim maliyetinin çok önemli bir yüzdesi kadardır (%40'lar civarında).

Nesne Yönelimli Programlamanın yararlı yönleri

✓ **Genişletilebilirlik (Extensibility)**

Nesne Yönelimli Programlama 'da mevcut bir sınıfa yeni özellik ve metotlar ekleyerek artan işlevsellik sağlamak son derece kolaydır.

✓ **Üretilen Kodun Yeniden Kullanılabilirliği (Reusability)**

Nesnelerin üretildiği sınıflar, ortam içinde her uygulama geliştiricinin kullanımına açık olduğu için ortak bir kütüphane oluşturulmuştur.

Her kullanıcı bu ortak kütüphanede bulunan sınıfları kullanarak gerekli kodu yeniden yazmaktan kurtulur ve uygulamaya özgü kod parçaları, ortak kullanımdaki sınıfların kod uzunluklarına göre göz ardı edilebilir boyuttadır.

Sınıf ve Nesne

Ütü Örneği;

- ✓ Ütünün markası, modeli, rengi, çalıştığı elektrik voltajı, ne tür kumaşları ütüleyebildiği bu ütüye ait özelliklerdir (veri).
- ✓ Aynı zamanda ütümüzü ısıtabiliriz, ütüleme işinde kullanabiliriz ve soğumaya bırakabiliriz. Bunlar ise ütünün fonksiyonlarıdır(metot).

Eğer ütü ile ilgili bir program yapmış olsak ve nesne tabanlı programlama tekniğini kullansak hemen bir ütü sınıfı (class) oluşturmalıyız.

Bu sınıfta ütüye ait bilgiler (veriler) ve ütü ile yapabileceğimiz işler (metot) bulunur. O zaman nesne tabanlı programlamada bir sınıfta, sınıfa ait veriler ve bu verileri işleyip bir takım faydalı sonuçlar üreten fonksiyonlar/metotlar bulunur.

Sınıf ve Nesne

- ✓ Nesneye yönelik programlama yaklaşımı, gerçek hayattan alınmış problemi çözmek üzere oluşturulacak modelin, gene gerçek hayatta var olan nesneler ve bu nesneler arasındaki ilişkilerden faydalanılarak oluşturulmasını ilke edinmiştir.

- ✓ Problem aynen gerçek hayatta görüldüğü şekli ile sanal ortama aktarılabilirse, günlük yaşamında nesneler ve nesneler arası ilişkiler ile etkileşimde olan programcı, nesneye yönelik model sayesinde, üzerinde çalıştığı problemi aynen gerçek hayatta olduğu şekliyle görebilecektir.

Sınıf ve Nesne

- ✓ Bir otomobil düşünelim. Otomobilin marka ve modeli ne olursa olsun, gaza basınca hızlandığını, frene basınca yavaşladığını, direksiyonu herhangi bir tarafa çevirince otomobilin o tarafa döndüğünü hepimiz biliyoruz. Bunlar otomobillerin genel davranışlarıdır ve bütün otomobiller bu davranışları sergiler.

- ✓ Ayrıca, her otomobilin bir motoru, lastikleri, farları, direksiyonu, dikiz aynası vardır. Marka ve modeli ne olursa olsun, gene bütün otomobillerde bunlar ve daha birçok başka aksam bulunmaktadır. Başka bir deyişle, bütün otomobiller bu özellikleri taşımaktadır

Sınıf ve Nesne

- ✓ Bu örnekte otomobil bir sınıftır. "Otomobil" denilince aklımıza gelen temel davranışlar ve özellikler, bütün otomobillerde vardır.
- ✓ Her otomobil gaza basıldığında aynı sürede 100 km hıza ulaşamıyor olsa da, gaza basıldığında bütün otomobiller hızlanır. Ya da bütün otomobillerin lastikleri aynı büyüklükte olmasa da, bütün otomobillerin lastikleri vardır.
- ✓ Sınıf, aynı özellik ve davranışları sergileyen varlıkların ortak kümesini belirleyen tanımdır. Hiç otomobil görmemiş birisine otomobilin ne olduğunu anlatmaya kalksak, kapımızın önünde duran belirli bir otomobili değil, o kişinin bir otomobil gördüğünde tanımasını sağlayacak, bütün otomobiller için ortak olan bilgiyi aktarmaya çalışırız.

Sınıf ve Nesne

- ✓ Özetle sınıf, o sınıftan olan bütün varlıkların ortak özellik ve davranışlarını anlatan bir tanımdır.
- ✓ Nesne ise ait olduğu sınıftan gelen özelliklerin değerlerinin belli olduğu, sınıfı için tanımlı olan davranışları nasıl sergilediğinin bilindiği, somut olarak var olan, biricik bir kimliği olan varlıktır.
- ✓ Örneğin, otomobil sınıfına ait olan 06-XYZ-1234 plakalı bir otomobil nesnesinden bahsediyorsak, söz konusu nesnenin kimliği plaka numarasıdır ve genel olarak otomobillerden değil, markası, modeli, rengi belli olan, gaza basıldığında 100 km/saat hıza kaç saniyede ulaştığı bilinen, elle gösterilebilen tek bir varlıktan söz ediyoruz demektir.

Sınıf ve Nesne

- ✓ Aynı sınıfa ait birçok nesne olabilir, şu an Ankara'da binlerce otomobil bulunduğu gibi.
- ✓ Aynı sınıfa ait olan nesnelerin hepsinde, o sınıftan gelen özellik ve davranışlar bulunmaktadır.
- ✓ Ancak herbir nesne için bu özelliklerin değerleri farklı, davranışların gerçekleştirilişi de bu özelliklere bağlı olarak farklı olabilir.
- ✓ Örneğin, 06-ZBC-9876 plakalı kırmızı spor otomobil 100 km/saat hıza 6 saniyede ulaşırken, 06-XYZ-1234 plakalı mavi otomobil 100 km/saat hıza 12 saniyede ulaşıyor olabilir.
- ✓ Ya da spor otomobilimiz 100 km'lik yolda 10 lt benzin tüketirken, diğer otomobilimiz aynı mesafede 6,5 lt benzin tüketiyor olabilir. Burada önemli olan, her iki otomobilin de otomobil sınıfından gelen özellik ve davranışları kendilerine özgü bir biçimde sergiliyor olmalarıdır.