C# – Koleksiyonlar

Nesne yönelimli programlama ile birlikte sınıf (class) kavramıyla tanıştık ve sıkça kullandığımız metotları tekrar tekrar yazmak yerine, bir sınıf içerisinde, bir defaya mahsus yazarak kod kalabalığının önüne geçmiş olduk. Bu programcı olarak bizlerin, kod karmaşıklığını ve kalabalığını azaltmaya yönelik olarak kendimizce almış olduğumuz bir önlem. Kullanmış olduğumuz platformda, programcılar tarafından sıkça kullanılan veri yapılarını hazır bir şekilde sunarak, benzeri bir kolaylık sağlamakta.

Örnek vermek gerekirse; Yığın ve Kuyruk programcılar tarafından sıkça kullanılan veri yapılarıdır. Bu veri yapılarına ait sınıfları her programcının yazmak zorunda kalması istenmeyen bir durumdur. Bu yüzden .Net Platformu programcılar tarafından sıkça kullanılan yapıları sisteme entegre ederek, hazır bir şekilde sunmaktadır. Platform tarafından bize sağlanan bu çözümleri ***Koleksiyon Sınıfları*** başlığı altında ele alacağız.

Koleksiyonların Sağladığı Avantajlar

Bir dizi nasıl birden fazla elemanı temsil ediyorsa, bir koleksiyon nesnesi de aynı şekilde birden fazla nesneyi temsil etmektedir. Diziler ile koleksiyonlar arasındaki farklılıklardan bahsedecek olursak;

1. Diziler sabit boyutludur ve eleman sayısının önceden belirtilmesi gerekir. Koleksiyonlar ise dinamik yapıdadır yani sabit boyutlu değildir. Eleman eklendikçe boyutu dinamik olarak artmaktadır.
2. Diziler aynı veri tipindeki elemanları içermektedir. Koleksiyonlarda ise böyle bir kısıtlama bulunmamaktadır. Farklı veri tipindeki elemanları genel amaçlı koleksiyonlar üzerinde tutabiliriz.

Koleksiyonların Dezavantajları

.Net Platformunda kullanmış olduğumuz veri tipleri, Değer tipleri ve Referans tipleri olarak ikiye ayrılmaktadır. Değer Tipleri stack bölgesinde tutulurken, Referans Tipleri heap bölgesinde tutulmaktadır.

1. **Değer Tipleri:** “int”, “long”, “float”, “double”, “decimal”, “char”, “bool”, “byte”, “short”, “struct”, “enum”
2. **Referans Tipleri:** “string”, “object”, “class”, “interface”, “array”, “delegate”, “pointer”

Bir Değer Tipinin, Referans Tipine dönüştürülmesi işlemine Boxing, tersi bir işlemede Unboxing denmektedir. Koleksiyonlar verileri object olarak tutmaktadır. Bu yüzden koleksiyonlara her değer tipli eleman eklediğimizde Boxing işlemi gerçekleşecektir. Yani verimiz object’e dönüştürülecektir. Koleksiyona eklenen verileri değer tipli bir değişene aktarmak istediğimizde de Unboxing işlemi gerçekleşecektir. Koleksiyonun eleman sayısındaki artışa bağlı olarak boxing ve unboxing işlemleri artacaktır ve buna bağlı olarak da uygulamamızın performansı düşecektir.

Boxing ve Unboxing işlemlerinin performans üzerindeki etkisini daha detaylı incelemek için bu bağlantıyı ziyaret edebilirsiniz.

Koleksiyon Tipleri

.Net Platformu 3 koleksiyon tipini desteklemektedir.

1. **Genel Amaçlı Koleksiyonlar:** Her tipten veriyi saklamak için kullanılabilirler.
   - • ***ArrayList***
   - • ***Dictionary***
   - • ***HashTable***
   - • ***Queue***
   - • ***SortedList***
   - • ***Stack***
2. **Özel Amaçlı Koleksiyonlar:** Belirli bir veri tipi veya çalışma şekli için optimize edilmişlerdir.
   - • ***CollectionsUtil***
   - • ***ListDictionary:*** Anahtar-Değer çiftlerini bir bağlı liste içinde saklar. Küçük veri kümeleri için tercih edilir.
   - • ***HybridDictionary:*** Belirli bir büyüklüğü geçene kadar Anahtar-Değer çiftlerini ListDictionary koleksiyonunda tutmaktadır, geçtikten sonra otomatik olarak bir Hashtable koleksiyonuna geçiş yapar.
   - • ***NameValueCollection:*** Anahtar-Değer çiftlerinin her ikisininde string tipinde olduğu durumlarda tercih edilebilir.
   - • ***StringCollection:*** Karakter katarlarını saklamak için optimize edilmiş bir koleksiyondur.
   - • ***StringDictionary:*** Anahtar-Değer çiftlerinin her ikisininde string tipinde olduğu bir hash tablodur. Anahtar-Değer çiftleri küçük harflere çevrilerek saklanır.
3. **Bit Tabanlı Koleksiyonlar:** İsminden de anlaşılacağı üzere bu koleksiyonlar bir grup biti içerisinde saklamaktadır. Bit tabanlı koleksiyonlara örnek olarak ***BitArray***'i verebiliriz. ***BitArray*** bitleri saklamak haricinde VE – VEYA gibi mantıksal işlemleri de gerçekleştirebilmektedir.

Koleksiyonları kullanım tekniği açısından Generic ve Generic olmayan olarak 2 tür üzerinde de inceleyebiliriz.

Generic koleksiyonlar System.Collections.Generic namespacei altında mevcutturlar.Non-Generic koleksiyonlarda tip güvenliği yoktur ancak Generic koleksiyonlar tip güvenliği vardır.Boxing – Unboxing işlemleri yapmamıza gerek kalmadığından performans açısından artış sağlayacaklardır.

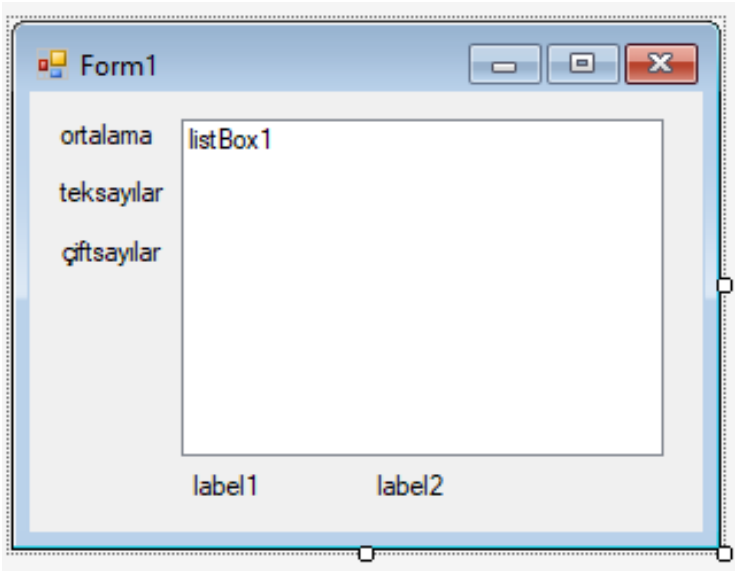
Öncelikle Generic koleksiyonların Non-Generic koleksiyonlardan artısı eksisi nedir bir bakalım.Non-Generic koleksiyonlar elemanları object tipten aldıkları için Boxing işlemine uğruyorlar.Haliyle elemanlara erişirkende Unboxing işlemi yapıyoruz.Bazen koleksiyonumuzda tek tipten elemanlar olduğu durumlar olur.Bu amaçla Non-Generic koleksiyonları kullanarak Boxing-Unboxing işlemleri performans açısından bizi olumsuz etkileyecektir.Halbuki Generic koleksiyonları kullanırsak, o tipten bir koleksiyon oluşturup , hem tip güvenliğini sağlarız hemde Boxing – Unboxing işlemine gerek duymamış oluruz. Buda performans açısından bize katkı sağlayacaktır.

Aşağıda Non-Generic kolleksiyonlara karşılık gelen Generic kolleksiyonlar listelenmiştir.

Generic Kolleksiyon	Non-Generic Kolleksiyon
Queue<T>	Queue
SortedDictionary<K,T>	SortedList
Stack<T>	Stack
List<T>	ArrayList
Dictionary<K,T>	Hashtable

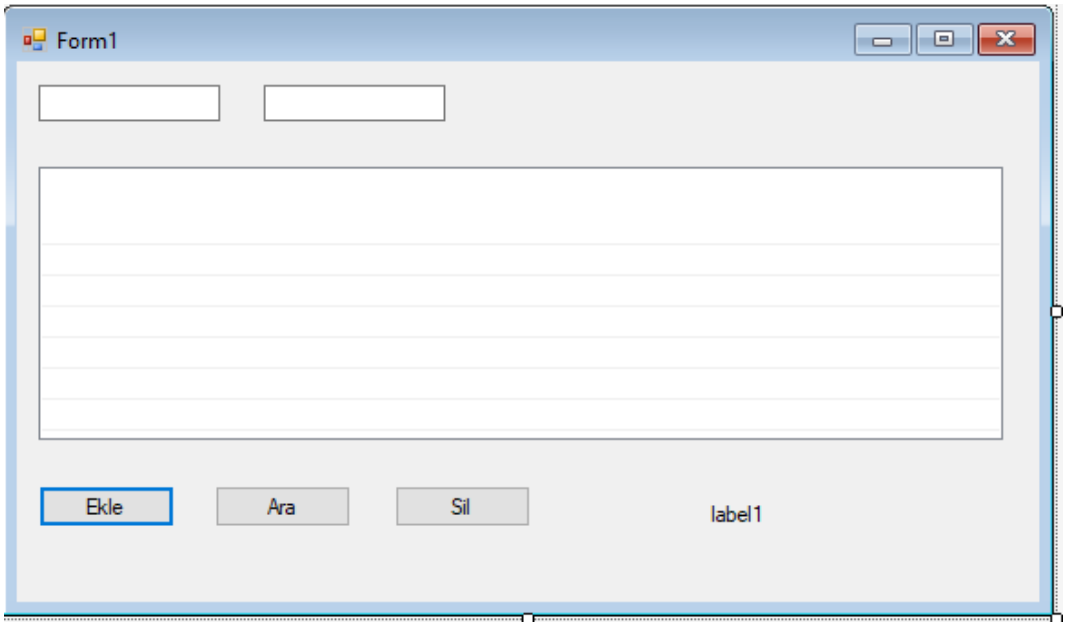
Bu koleksiyonlardan kullanılan en popülerlerini aşağıda örneklerle inceleyelim;

ArrayList



```
private void Form1_Load(object sender, EventArgs e)
{
    //ArrayList Non Jeneric kolleksiyondur. Tür ve sınır tanımlaması yok.
    ArrayList tekSayilar = new ArrayList();
    ArrayList ciftSayilar = new ArrayList();

    int ortalama = 0, toplam = 0;
    Random rnd = new Random();
    for (int i = 0; i < 1000; i++)
    {
        int sayi = rnd.Next(1, 100);
        listBox1.Items.Add(sayi);
        toplam += sayi;
        if (sayi % 2 == 0)
            ciftSayilar.Add(sayi);
        else
            tekSayilar.Add(sayi);
    }
    ortalama = toplam / (tekSayilar.Count + ciftSayilar.Count);

    lblOrt.Text = ortalama.ToString();
    lblTekSayilar.Text = tekSayilar.Count.ToString();
    lblCiftSayilar.Text = ciftSayilar.Count.ToString();

}

private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    label1.Text = listBox1.SelectedItem.ToString();
    label2.Text = listBox1.SelectedIndex.ToString();
}
```

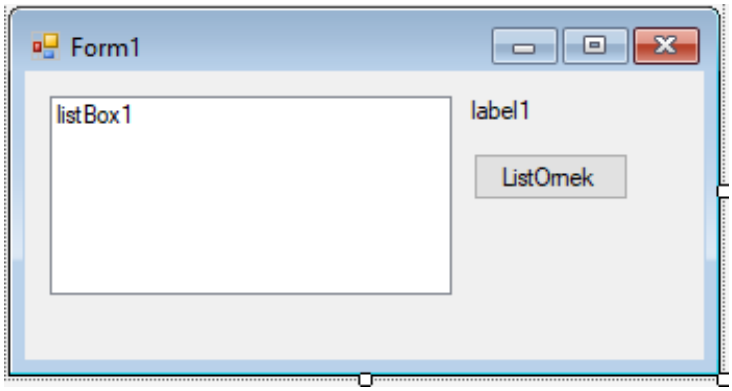
HashTable



```
Hashtable sehirler = new Hashtable();

private void Form1_Load(object sender, EventArgs e)
{
    sehirler.Add("322", "Adana");
    sehirler.Add("416", "Adıyaman");
    sehirler.Add("272", "Afyon");
    sehirler.Add("472", "Ağrı");
    sehirler.Add("382", "Aksaray");

    Listele();
}

public void Listele()
{
    ListViewItem item = new ListViewItem();
    ICollection kod = sehirler.Keys;
    listView1.Clear();
    //kolon başlıkları ve genişlikleri
    listView1.Columns.Add("Şehir Kodu", 60);
    listView1.Columns.Add("Şehir Adı", 120);

    //listview özellik ayarlama
    listView1.View = View.Details;
    listView1.GridLines = true;

    foreach (string eleman in kod)
    {
        item = listView1.Items.Add(eleman);
        item.SubItems.Add(sehirler[eleman].ToString());
    }

    listView1.Sorting = SortOrder.Ascending;
}

private void button1_Click(object sender, EventArgs e)
{
    sehirler.Add(textBox1.Text, textBox2.Text);
    textBox1.Text = "";
    textBox2.Text = "";
    Listele();
}
```

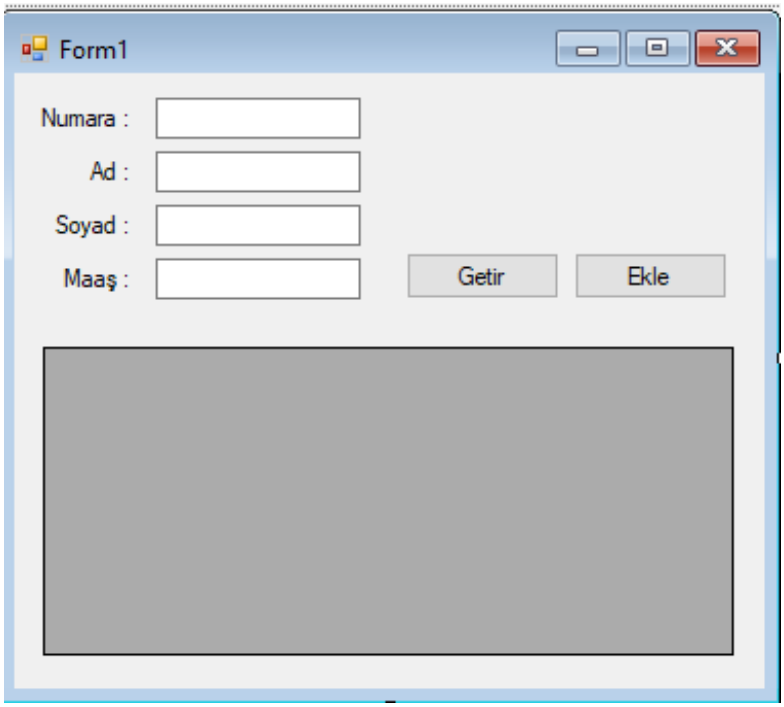
```
private void button2_Click(object sender, EventArgs e)
{
    string anahtar = textBox1.Text;
    label1.Text = sehirler[anahtar].ToString();
}

private void button3_Click(object sender, EventArgs e)
{
    sehirler.Remove(textBox1.Text);
    Listele();
}
```

List<T>



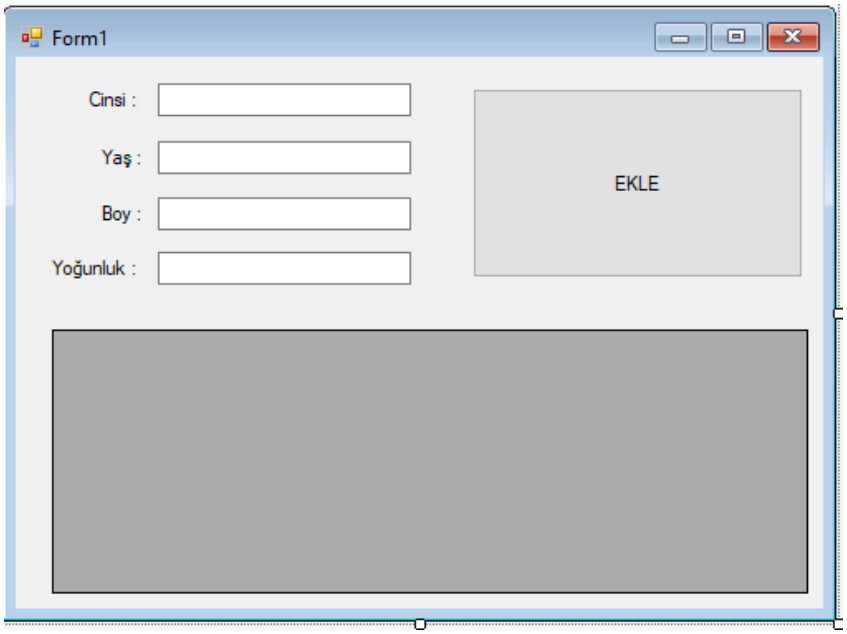

```
private void button1_Click(object sender, EventArgs e)
{
    List<string> gunler = new List<string>();
    //add ile eleman ekle
    gunler.Add("Pazartesi");
    gunler.Add("Salı");
    gunler.Add("Çarşamba");

    //eleman sırasını ters çevir
    gunler.Reverse();
    //Count eleman sayısını bul
    label1.Text = "Eleman Sayısı" + gunler.Count;

    //silme işlemi
    gunler.Remove("Salı");

    //Listeleme
    foreach(string eleman in gunler)
    {
        listBox1.Items.Add(eleman);
    }
    label1.Text = "Eleman Sayısı" + gunler.Count;
}
```

List ile 2. Örnek;



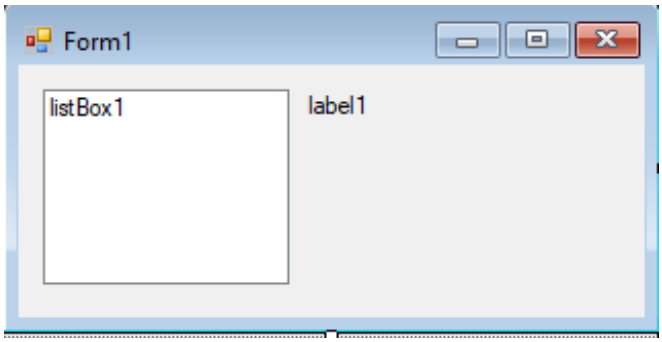

```
 //personel sınıfı oluşturuldu.
 //sınıf içerisinde kullanılacak propertyler belirlendi
 //prop + tab+tab
public class Personel
 {
     public int Numara { get; set; }
     public string Ad { get; set; }
     public string Soyad { get; set; }
     public decimal Maas { get; set; }
 }


List<Personel> pList = new List<Personel>();
private void button1_Click(object sender, EventArgs e)
{
    dataGridView1.DataSource = pList;
}
private void button2_Click(object sender, EventArgs e)
{
    Personel p = new Personel();
    p.Numara = int.Parse(txtNumara.Text);
    p.Ad = txtAd.Text;
    p.Soyad = txtSoyad.Text;
    p.Maas = decimal.Parse(txtMaas.Text);
    pList.Add(p);
    dataGridView1.DataSource = pList.ToList();
}
```

Dictionary<Key,Value>




```
public class Urun
{
    public int Cinsi { get; set; }
    public int Boy { get; set; }
    public int Yas { get; set; }
    public decimal Yogunluk { get; set; }
}

Dictionary<string, Urun> liste = new Dictionary<string, Urun>();
private void button1_Click(object sender, EventArgs e)
{
    string key = Guid.NewGuid().ToString();
    Urun u = new Urun();

    u.Cinsi = int.Parse(txtCinsi.Text);
    u.Boy = int.Parse(txtBoy.Text);
    u.Yas = int.Parse(txtYas.Text);
    u.Yogunluk = decimal.Parse(txtYogunluk.Text);

    liste.Add(key, u);
    MessageBox.Show("Key : " + key + "kaydedildi");
    dataGridView1.DataSource = liste.Values.ToList();
}
```

Stack<T>




```
Stack<string> isimler = new Stack<string>();
int i;

private void Form1_Load(object sender, EventArgs e)
{
    isimler.Push("Ahmet");
    isimler.Push("Mehmet");
    isimler.Push("Azra");
    isimler.Push("Murat");

    for(i=0;i<isimler.Count; i++)
    {
        listBox1.Items.Add(isimler.ElementAt(i));
    }

    label1.Text = isimler.Peek();
    isimler.Pop();

}
```

Queue<T>

```
int i;
Queue<string> isimler = new Queue<string>();

private void Form1_Load(object sender, EventArgs e)
{
    isimler.Enqueue("Ahmet");
    isimler.Enqueue("Mehmet");
    isimler.Enqueue("Azra");
    isimler.Enqueue("Murat");

    for(i=0;i<isimler.Count;i++)
    {
        listBox1.Items.Add(isimler.ElementAt(i));
    }

    label1.Text = isimler.Peek();

    isimler.Dequeue();

}
```

NESNEYE YÖNELİK PROGRAMLAMA(OOP)

Nesne Yönelimli Programlama, 1990'lı yıllarda başlayan ve günümüzde de yoğun olarak kullanılan bir programlama teknolojisidir. Yine aynı yıllarda özellikle İnternet uygulamalarına yönelik özellikleriyle öne çıkan Java dili saf (pure) bir nesneye yönelik programlama dilidir.

Nesne nedir?
Nesne, içinde veri ve bu veriler üzerinde işlem yapacak olan metotları (fonksiyon) bulunduran yazılım bileşenidir. Nesne bu tanıma uygun olarak, kendi işlevselliğini de içinde taşır. Nesneler her uygulamada tekrar tekrar kullanılabilir. Veri ve metotlar, birlikte nesnenin üyeleri (members) adını alır. Bir nesne yapısı, bir *sınıf (class)* içinde tanımlanır. Sınıf içinde nesneyi oluşturan üye değişkenler ve metotlar açıkça tanımlanır.

Nesne Yönelimli Programlama dillerinin teknik özelliklerini gözden geçirmeden önce, yukarıda belirtilen nesne yönelimli dillerin oluşturulma nedenlerini anlayabilmek için Nesne Yönelimli Programlama dillerinin yararlı yönlerini gözden geçirelim.

Yazılımın bakım (maintenance) kolaylığı
Nesne Yönelimli Programlama dillerinde, nesneler ait oldukları sınıfların sunduğu şablonlarla temsil edilirler. Birbirinin benzeri ya da aynı konu ile ilişkili sınıflar bir araya getirilerek *namespace* ya da *package (paket)*adı verilen sınıf kümeleri oluşturulur. Bu durumda yazılımın bakımı, 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).

Genişletilebilirlik (Extensibility)
Nesne Yönelimli Programlama'da mevcut bir sınıfa yeni özellik ve metotlar ekleyerek artan işlevsellik sağlamak çok 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.

Nesne Yönelimli Programlama Teorisi'nde 4 temel özelliğin gerçekleştirilmesi zorunlu sayılmıştır ve bu temel özelliklerden birini bile sağlamayan bir dil *saf (pure)* Nesne Yönelimli Programlama Dili sayılmaz.

Bu 4 temel özellik;

1. Soyutlama (Abstraction)
Soyutlama denilince, nesneyi bazı karakteristikleri olan ve bazı eylemleri gerçekleştirebilen bir veri tipi olarak genelleştirmek anlaşılmalıdır. Yeryüzü üzerinde bulunan her şey bir nesnedir. Bilgisayar yazılımı açısından nesne, bir varlığın temsil biçimidir. Örneğin; bina bir nesnedir. Binayı bilgisayar

yazılımı içinde temsil etmek istersek, öncelikle bu nesnenin yani binanın ayırt edici özelliklerini belirlemeliyiz.

Örneğin;

- Bina yüksekliği
- Dış yüzey rengi
- Kat sayısı
- Zemin alanı
- Zemin boyutları

gibi özellikler, binayı temsil etmek için ilk akla gelen birkaç özelliktir. O halde nesnenin bilgisayarda temsilinde, özellikleri kilit rol oynamaktadır.

Nesnenin karakteristikleri, özellikler ve gerçekleştirebileceği eylemler de metot adını alır. Bu anlamda, soyutlaştırmanın sonucunda bir nesne;

- Özellikleri (properties, member variables - Objective-C)
- Metotları (methods) ile temsil edilebilir.

Nesne Yönelimli Programlama Dilleri, soyutlamayı *sınıf (Class)* yapısı ile gerçekleştirirler. Sınıf yapısı içinde o nesneye ait özellikler ve metotlar tanımlanabilir.

Ancak sınıf soyut bir yapıdır ve doğrudan kullanılamaz. O sınıftan üretilen örnekler sınıfa ait tüm özelliklere ve metotlara sahip olurlar. Bunlar program içinde doğrudan kullanılabilirler. Sınıftan üretilen her örnek, aynı özellik ve metotlara sahip olacak ancak özelliklerin değerleri doğal olarak farklı olabilecektir.

Örneğin, insanı bir nesne olarak düşünürsek ve insan sınıfı olarak soyutlarsak, ilk akla gelen özellikleri; boy, kilo, IQ ve EQ değerleri, eğitim düzeyi vb. gibi özellikler olur. Bu özelliklerin değerleri doğal olarak her insan için farklıdır. İnsan için ilk akla gelen eylemler (metotlar) ise; yürüme, okuma-yazma, konuşma, araç kullanma vb. gibi metotlardır.

2. Sarmalama / Paketleme (Encapsulation)
Paketlemenin anlamı; sınıfı oluşturan metot ve özelliklerin gerçekleştirme biçiminin, bu sınıfı kullanacak olan kullanıcılardan gizlenmiş olmasıdır.

NESNE = VERİ + METODLAR

şeklinde ifade edilen bağıntı aslında Nesne Yönelimli Programlama'nın temelini açıklamaktadır. Veri (özellikler) ve veri üzerinde işlem yapan kod (metotlar) bir arada bulunur ve nesneyi oluşturur. Nesneyi tanımlayan sınıfın iç ayrıntıları, normal olarak programın artakalan kısmı için görünür değildir.

Bir nesne sınıfının gerçekleştirimini değiştirirsek, yani aynı sınıfı metot ve özellikleri aynı kalmak koşuluyla farklı bir programlama tekniğiyle oluşturursak, o sınıfın dış dünyaya olan arayüzü değişmediği sürece (metot ve özellikler aynı kaldığı sürece) bu sınıfı kullanan program kodlarınızda bir değişiklik yapmamıza gerek kalmayacaktır.

3. Miras Alma (Inheritance)
Nesneye yönelik programlamada, bir nesne, genellikle bir nesne sınıfına ait bir örnektir (instance).

Örneğin, Albert Einstein, insan sınıfının bir örneğidir. Bir nesne sınıfından alt sınıflar (subclasses) oluşturulabiliyorsa, türetme özelliği (derivation) var demektir. Örneğin insan sınıfı, canlı sınıfının bir alt sınıfıdır.

Kendisinden alt sınıf üretilen sınıfa, temel sınıf (base class) veya süper sınıf (super class**)** veya ana sınıf (parent class) adı verilir.

Subclass yerine child class terimi de kullanılmaktadır.

Alt sınıfın nesneleri, türetildikleri temel sınıfa ait özellikleri alıyorsa, burada miras alma (inheritance) özelliği vardır denir.

Bu anlamda, miras alma özellikli bir nesne yönelimli programlama dilinde, bir nesne sınıfından türetilen alt nesne sınıfına ait nesneler, üst sınıfın özelliklerini (properties) ve metodlarını (methods) aynen alırlar.

4. Çok Biçimlilik (Polymorphism)
Farklı nesnelerin, aynı mesaja (olaya ya da uyarıma) farklı şekillerde cevap verebilme yeteneğidir.

Her nesne sınıfı, kendi metotlarını paketlediği için ve bu metotlar programın kalan kısmı için gizli olduğundan, farklı sınıflar aynı isimde bazı metotlara sahip olabilirler.

Örneğin ŞEKİL adlı bir süper sınıfımız (super class) olsun. Bu sınıftan, DAİRE**,** KARE ve ÜÇGEN adlı 3 alt sınıf türettiğimizi varsayalım. Bu alt sınıfların her biri kendi örneklerini çizmek için ÇİZ adlı bir metoda sahip olabilir. Fakat, her bir alt sınıf için bu metot aynı olmasına karşın, bu metodu devreye sokmak için gönderilecek mesajdan sonra her alt sınıfa ait nesne farklı bir şekil (yani kendini, yani bir üçgen, daire veya kare) çizecektir.

Nesne Yönelimli Programlamanın sınıf türetebilme özelliği sayesinde, bir nesne hiyerarşisi oluşturma imkânı bulunnmaktadır.

Sınıf (class) yapısının bu sınıftan üretilecek nesneler için bir şablon görevi gördüğünü söyleyebiliriz.

Constructor (Kurucu Metod);

Oluşturulan bir class üzerinden nesne türetildiğinde ilk olarak çalışan metoddur. Burada metodun adı ile class' ın adı aynı olmak zorundadır. Parametreli veya parametresiz olarak kullanılabilir.

Örnek :

```
public  class Car
{

     public string model;
     public string color;
     public int year;

        public Car(string ModelName, string ColorName, int Year)
        {
        model = ModelName;
        color = ColorName;
        year = Year;
        }

}
```

```
private void Form1_Load(object sender, EventArgs e)
{
    Car cr = new Car("Volvo","red",2020);
    label1.Text = cr.model;
}

private void button1_Click(object sender, EventArgs e)
{
    Car cr = new Car(textBox1.Text, textBox2.Text, int.Parse(textBox3.Text));
    MessageBox.Show(cr.model + " " + cr.color + " " + cr.year);
}
```

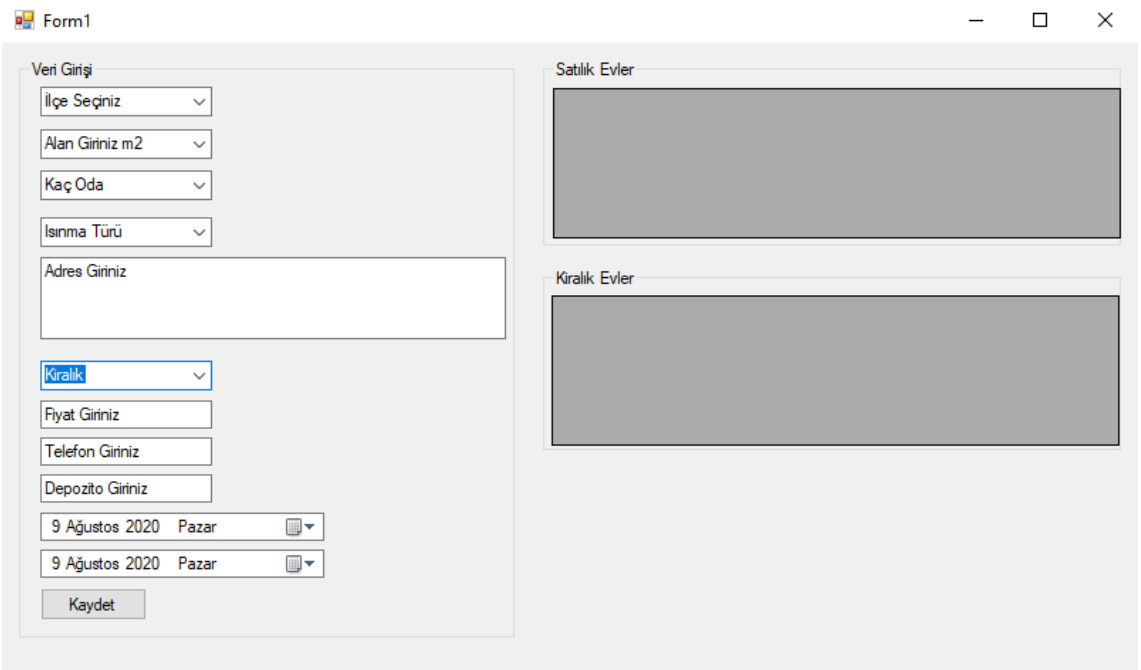
Miras Alma – Kalıtım Örneği;

```
public class INSAN
{
    public string Ad { get; set; }
    public string Soyad { get; set; }
    public int Yas { get; set; }
    public int Kilo { get; set; }
    public int Boy { get; set; }
}
```

```
public class PERSONEL : INSAN
{
    public string Departman { get; set; }
    public string Pozisyon { get; set; }
    public decimal Maas { get; set; }
}
```

```
List<PERSONEL> pList = new List<PERSONEL>();


private void button1_Click(object sender, EventArgs e)
{
    PERSONEL P = new PERSONEL();
    P.Ad = txtAd.Text;
    P.Soyad = txtSoyad.Text;
    P.Yas = int.Parse(txtYas.Text);
    P.Kilo = int.Parse(txtKilo.Text);
    P.Boy = int.Parse(txtBoy.Text);
    P.Departman = txtDepartman.Text;
    P.Pozisyon = txtPozisyon.Text;
    P.Maas = decimal.Parse(txtMaas.Text);

    pList.Add(P);

    dataGridView1.DataSource = pList.ToList();

}
```

Miras Alma – Kalıtım 2. Örneği;




```
public class Ev
{
    public string Ilce { get; set; }
    public int Alan { get; set; }
    public string Adres { get; set; }
    public int OdaS { get; set; }
    public string Isinma { get; set; }
}
```

```
public class SatilikEv : Ev
{
    public decimal Fiyat { get; set; }
    public string Tel { get; set; }
}


public class KiralikEv : Ev
{
    public decimal Fiyat { get; set; }
    public decimal Depozito { get; set; }
    public DateTime SozBas { get; set; }
    public DateTime SozBit { get; set; }
    public string Tel { get; set; }
}

class DevreMulk :KiralikEv
{
    //örnek olması için
}

private void Form1_Load(object sender, EventArgs e)
{
    txtFiyat.Visible = false;
    txtTel.Visible = false;
    txtDep.Visible = false;
    dateTimePicker1.Visible = false;
    dateTimePicker2.Visible=false;

    cmbAlan.Text = "Alan Giriniz m2";
    cmbIlce.Text = "İlçe Seçiniz";
    cmbIsinma.Text = "Isınma Türü";
    cmbOda.Text = "Kaç Oda";
    cmbTur.Text = "Konut Türü";

    for (int i = 40; i < 300; i++)
    {
        cmbAlan.Items.Add(i);
    }


    for (int i = 1; i <= 8; i++)
    {
        cmbOda.Items.Add(i);
    }

    cmbIlce.Items.Add("Eyüpsultan");
    cmbIlce.Items.Add("Avcılar");
    cmbIlce.Items.Add("Şişli");
```

```
    cmbIsinma.Items.Add("Kalorifer");
    cmbIsinma.Items.Add("Soba");
    cmbIsinma.Items.Add("Merkezi");

    cmbTur.Items.Add("Kiralık");
    cmbTur.Items.Add("Satılık");
}

private void cmbTur_SelectedIndexChanged(object sender, EventArgs e)
{
    if(cmbTur.SelectedIndex==1)
    {
        txtFiyat.Visible = true;
        txtTel.Visible = true;
        txtDep.Visible = false;
        dateTimePicker1.Visible = false;
        dateTimePicker2.Visible = false;

    }

   if (cmbTur.SelectedIndex == 0)
    {
        txtFiyat.Visible = true;
        txtTel.Visible = true;
        txtDep.Visible = true;
        dateTimePicker1.Visible = true;
        dateTimePicker2.Visible = true;

    }
```

```
List<SatilikEv> stlListe = new List<SatilikEv>();
List<KiralikEv> krlListe = new List<KiralikEv>();

private void button1_Click(object sender, EventArgs e)
{
    SatilikEv stl = new SatilikEv();
    KiralikEv krl = new KiralikEv();

    if(cmbTur.SelectedIndex==0)
    {
        krl.Ilce = cmbIlce.SelectedItem.ToString();
        krl.Alan = int.Parse(cmbAlan.SelectedItem.ToString());
        krl.OdaS = int.Parse(cmbOda.SelectedItem.ToString());
        krl.Isinma = cmbIsinma.SelectedItem.ToString();
        krl.Adres = textBox1.Text;
        krl.Fiyat = Convert.ToInt32(txtFiyat.Text);
        krl.Depozito = Convert.ToInt32(txtDep.Text);
        krl.Tel = txtTel.Text;
        krl.SozBas = dateTimePicker1.Value;
        krl.SozBit = dateTimePicker2.Value;
        krlListe.Add(krl);
        dataGridView2.DataSource = krlListe.ToList();
    }
```

```
if (cmbTur.SelectedIndex == 1)
{
    stl.Ilce = cmbIlce.SelectedItem.ToString();
    stl.Alan = int.Parse(cmbAlan.SelectedItem.ToString());
    stl.OdaS = int.Parse(cmbOda.SelectedItem.ToString());
    stl.Isinma = cmbIsinma.SelectedItem.ToString();
    stl.Adres = textBox1.Text;
    stl.Fiyat = Convert.ToInt32(txtFiyat.Text);
    stl.Tel = txtTel.Text;
    stlListe.Add(stl);
    dataGridView1.DataSource = stlListe.ToList();
}
```