BU DÖKÜMAN KAAN ASLAN TARAFINDAN C VE SİSTEM PROGRAMCILARI DERNEĞİNDE VERİLEN C# İLE .NET ORTAMINDA UYGULAMA GELİŞTİRME KURSUNDAKİ DERS İÇERİSİNDE TUTULAN NOTLARDAN OLUŞMAKTADIR. NOTLAR ÜZERİNDE HİÇBİR DÜZELTME YAPILMAMIŞTIR

C# UYGULAMALARI TEMEL

Dispose Kalıbı ve IDisposable Arayüzü: C# da bitiş fonksiyonu çöp toplayıcı tarafından nesne bellekten silinmeden hemen önce çağrılır. Bitiş fonksiyonlarının isimleri ***~<sınıf ismi>*** biçimdedir. Sınıfın tek bir bitiş fonksiyonu bulunabilir. Sıfır parametresi olmak zorundadır. Bitiş fonksiyonu kavramı işleyiş ve isim olarak C++ dan aktarılmıştır. Fakat C++ da bitiş fonksiyonları deterministiktir. Yani bitiş fonksiyonlarının hangi noktada çağrılacağı kesinlikle bilinmektedir. Fakat C# daki bitiş fonksiyonları deterministik değildir. Yani nesne seçilebilir duruma geldikten sonra tam olarak nesnenin ne zaman silineceği dolayısıyla bitiş fonksiyonunun çağrılacağı önceden belli değildir. Bu nedenle C# da bitiş fonksiyonları C++ daki kadar yoğun kullanılmamaktadır.

Bazen sınıfın başlangıç fonksiyonu içinde kritik bazı kaynaklar tahsis edilmiş olabilir. Referansın kullanımı bittikten sonra ya da belirli bir noktada bu kaynakların boşaltılması gerekebilir. Bu tür boşaltımların bitiş fonksiyonu içinde yapılması ve bitiş fonksiyonuna bırakılması iyi bir teknik değildir. İşte bu yüzden IDisposable arayüzü düşünülmüştür. IDisposable arayüzü tek bir fonksiyona sahiptir.

```
interface IDisposable
{
        void Dispose();
}
```

Eğer bir sınıf ya da yapı IDisposable arayüzünü destekliyorsa o sınıf ya da yapının bir Dispose fonksiyonu vardır ve bu fonksiyon gerekli olan boşaltım işlemlerini yapmaktadır. O halde biz böyle bir sınıf ya da yapıyı kullanıyorsak işimiz bitince Dispose fonksiyonunu çağırmalyız. Genellikle IDisposable arayüzünü destekleyen sınıflarda ve yapılarda tasarımcı bir bitiş fonksiyonu da yazmıştır ve bitiş fonksiyonunda da boşaltımı yapmıştır. Bu nedenle programcı Dispose işlemini bilinçli olarak yapabilir. Kaynaklar bitiş fonksiyonu sayesinde ve çöp toplayıcı sayesinde iade edilecektir.

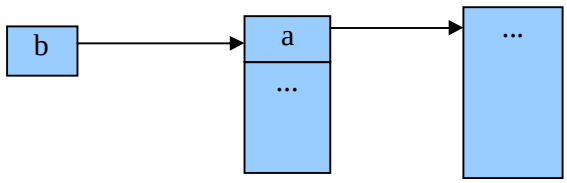
Fakat kaynak boşaltımının bazı önemli ayrıntıları vardır. Kaynaklar **“managed”, “unmanaged”** olmak üzere iki kısma ayrılmaktadır. Unmanaged kaynaklar işletim sistemi düzeyinde CLR nin kontrolü dışında tahsis edilmiş kaynaklardır. Bu tür kaynaklar sistem fonksiyonları ya da API fonksiyonları çağrılarak tahsis edilmektedir. Managed kaynaklar CLR nin kontrolü altındaki kaynaklardır. Bir sınıfın başkabir sınıfı kullanması durumunda kullanılan kaynaklar managed kaynaklardır.
Örneğin bir B sınıfı bir A sınıfı türünden bir nesneyi yaratarak kullanıyor olsun birde doğrudan API fonksiyonları çağırarak tahsis ettiği unmanaged kaynakları kullanıyor olsun. Şimdi biz B sınıfı türünden nesnenin boşaltılması işleminde iki şeyden sorumluyuz. Birincisi unmanaged kaynakların boşaltılması ikincisi A sınıfına ilişkin tahsis edilmiş nesnenin yani managed kaynakların boşaltılmasıdır.

.net ordamında bir sının başka bir sınıfı içermesi durumunda içeren sınıf nesnesi seçilebilir duruma geldiğinde çoğu kez içerilen nesne de secilebilir duruma gelmektedir.

```
class A
{
        //...
}
class B
{
        private A m_a;
```

```
        //...
        public B()
        {
                m_a = new A();
                //...
        }
        //...
}
```

B b = new B(); B nesnesi A nesnesi



Burada B nesnesi seçilebilir duruma geldiğinde A nesnesi de seçilebilir duruma gelmektedir. İşte çöp toplayıcının önce A nesnesinimi yoksa B nesnesinimi boşaltacağı bir kurala bağlanmamıştır.

Örneğin önce A nesnesi sonra B nesnesi heap ten silinebilir. Bu durumda önce A nesnesi için sonra B nesnesi için bitiş fonksiyonları çağrılacaktır. Bu nedenle elemana sahip sınıfın bitiş fonksiyonunda elemana ilişkin referansın kullanılması sorunlara yol açabilmektedir. Şüphesiz sınıfın değer türlerine ilişkin veri elemanlarının bitiş fonksiyonunda kullanılmasında bir sakınca yoktur.

Bir sınıfın hem managed(yani başka sınıf türünden bir veri elemanının olması durumu) ve unmaneged kaynaklara sahip olması durumunda onlar nasıl düzenli boşaltılabilir. İşte bu kalıba Disposa kalıbı(Dispose Pattern) denilmektedir. Sınıf unmanaged kaynak kullandığına göre IDisposable arayüzünü desteklemesi iyi olur. Boşaltma işlemi hem Dispose fonksiyonunda hem bitiş fonksiyonunda yapılmalıdr. Bu durumda biz Dispose fonksiyonunda hem managed hem unmaneged kaynakları ve bitiş fonksiyonunda ise unmanaged kaynakları boşaltmamız gerekir. Görüldüğü gibi burada her iki yerden de unmanaged kaynaklar boşaltılmaktadır. Ortak işlemlerin bir fonksiyona yaptırılması uygundur.

```
class A
{
        //...
}
class B
{
        private A m_a;
        //...
        public B()
        {
                m_a = new A();
                //...
        }
        //...
}
public void Dispose(bool disposing)
```

```
{
        if(disposing)
        {
                //managed kaynakları boşalt
        }
        // unmanaged kaynakları boşalt
}
public void Dispose()
{
        Dispose(true);
}
~B()
{
        Dispose(false);
}
//...
}
```

Buradaki bir problem kaynakların Dispose fonksiyonuyla boşaltılmasıyla ortaya çıkmaktadır.

```
B  b = new B();
//...
b.Dispose();
```

b nesnesi için Dispose edildiği halde yine bitiş fonksiyonu çağrılacaktır. Bitiş fonksiyonu ikinci kez unmanaged kaynakların boşaltılmasına yol açacaktır. Boşaltılmış kaynakların yeniden boşaltılmaya çalışması soruna yol açar. Bunu çözmenin bir yolu bir sınıf içinde bir bayrak(flag) tutarak bunu engellenmesidir. Fakat daha iyi bir yöntem bitiş fonksiyonunun çağrılmasını engellemektir. Bu işlem **GC sınıfının SuppressFinalize** fonksiyonuyla yapılır.

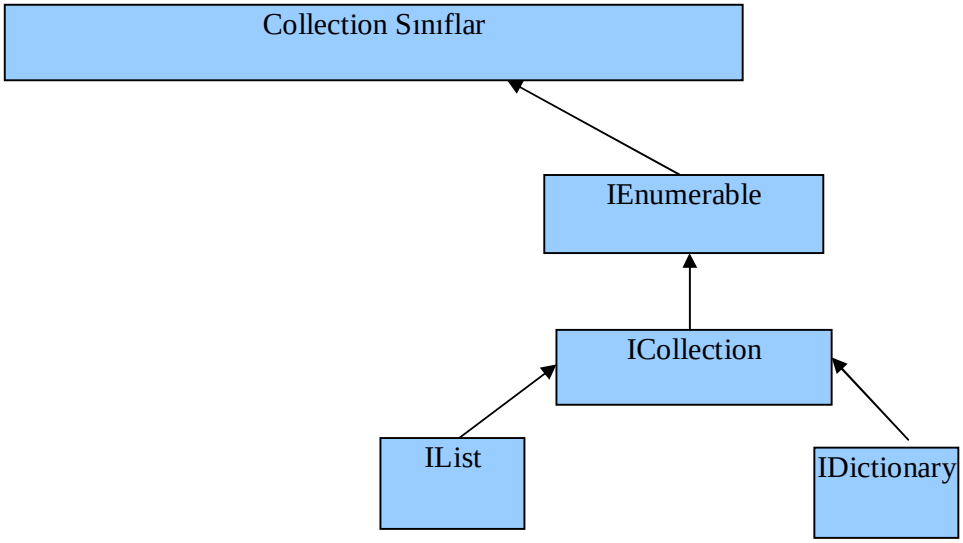
```
public static void SuppressFinalize(object obj)
```

O halde yukarıdaki kodun şu şekilde düzeltilmesi gerekir.

```
class A
{
        //...
}
class B
{
        private A m_a;
        //...
        public B()
        {
                m_a = new A();
                //...
        }
        //...
}
public void Dispose(bool disposng)
{
        if(disposing)
        {
                //managed kaynakları boşalt
```

```
            GC.SuppressFinalize(this);
        }
        // unmanaged kaynakları boşalt
}
public void Dispose()
{
        Dispose(true);
}
~B()
{
        Dispose(false);
}
//...
}
```
Artık Dispose fonksiyonu çağrıldığında boşuna bitiş fonksiyonu çağrılmayacaktır.

Collection Sınıflar: Birden fazla nesneyi belli bir veri yapısını kullanarak tutan özel sınıflara collection sınıflar denilmektedir. Collection sınıflar liste tarzı ve sözlük tarzı olmak üzere ikiye ayrılmaktadır. .netteki tüm collection sınıflar bazı arayüzleri desteklemektedir.



Collection sınıfların desteklediği en temel arayüz IEnumarable arayüzüdür. Bu arayüzden ICollection arayüzü türetilmiştir. ICollection arayüzündende IList ve IDictionary arayüzleri türetilmiştir. Liste tarzı sınıflar IList arayüzünü sözlük tarzı sınıflar IDictionary arayüzünü desteklemektedir. Örneğin ArrayList sınıfı IList arayüzünü desteklemektedir. Yani liste tarzı bir collection sınıftır.

IEnumerable Arayüzü: Bu arayüz çok biçimli olarak dolaşılabilirliği temsil etmektedir. Bu arayüzün tek bir elemanı vardır.
```
interface IEnumarable
{
        IEnumarator GetEnumerator();
}
```
Görüldüğü gibi bu arayüzün enumarator veren bir fonksiyonu vardır. IEnumarator arayüzünün elemanları şunlardır. Arayüzün MoveNext isimli fonksiyonu sonraki elemana geçiş sağlar.
```
bool MoveNext()
```

Fonksiyon son elemanı girilmişse false değerine girilmemişse true değerine geri döner. Enumaratör ilk alındığında ilk elemanın bir gerisinde olduğu varsayılmaktadır. Dolayısıyla yürüme işi şöyle yaplabilir.

```
while(r.MoveNext())
{
        //...
}
```

Arayüzün bir elemanıda Current isimli property dir.

```
Object Current { get; }
```

Bu property konumlanılmış olan pozisyondaki nesne ile geri dönmektedir ya da nesneyi geri vermektedir. Reset fonksiyonu Enumaratörü ilk konumuna getirir.

```
void Reset()
```

Eğer bir sınıf ya da yapı IEnumarable arayüzünü destekliyorsa biz o sınıf ya da yapının tuttuğu elemanları tek tek ele geçirebiliriz.

O halde Enumerator yoluyla dolaşım şöyle yapılabilir.

```
        int val;

        ien = c.GetEnumerator();

        while (ien.MoveNext())
        {
           val = (T)ien.Current;

        }
```

```
using System;
using System.Collections;

namespace CSD
{
   class App
   {
      public static void Main()
      {
         ArrayList al = new ArrayList();

         for (int i = 0; i<10; ++i)
            al.Add(i);

         IEnumerable ie = al;
         IEnumerator ien;
         int val;

         ien = ie.GetEnumerator();
         while (ien.MoveNext())
         {
            val = (int)ien.Current;
```

```
            Console.WriteLine(val);
        }
    }
  }
}
```

Anımsanacağı gibi tüm diziler System.Array sınıfından türetilmiştir. System.Array sınıfı da IEnumerable arayüzünü desteklemektedir. O halde tüm dizilerde Enumerator yoluyla dolaşılabilir.

```
using System;
using System.Collections;

namespace CSD
{
  class App
  {
    public static void Main()
    {
      int [ ] a = {1, 2, 3, 4, 5, 6, 7, 8, 9};
      IEnumerator ie = a.GetEnumerator();

      while (ie.MoveNext())
      {
        int val = (int)ie.Current;
        Console.WriteLine(val);
      }
    }
  }
}
```

foreach deyimi Ienumerable arayüzünü destekleyen her türlü sınıf ya da yapıyla kullanılabilir. Foreach döngüsünü gören derleyici aslında ilgili dizilime GetEnumerator fonksiyonunu uygulayarak yukarıdaki gibi bir döngü yoluyla elemanlara ulaşır.

Anahtar Notlar: Bir .exe ya da .dll ister .net sistemine ister doğal çalışma sistemine ilişkin olsun PE(Portable Exacutable) dosya formatına sahiptir. Doğal exe ve dll lerde CLR header kısmı yoktur. .net exe ve dll lerde bu başlık kısmı vardır. Dolayısıyla bir exe ya da dll in .net e ilişkin olup olmadığı hemen bu bölümün varlığı sorgulanarak belirlenebilir. Pratik olarak ilgili exe ya da dll in ILASM programı ile yüklenebilirliğine de bakılabilir(komut satırında debug içinde ki programları ildasm x.exe gibi).

C# standartlarına göre foreach deyiminin karşılığı şöyledir.
foreach (V v in x) embedded statement
açılımı

```
{
        E e = ((C) (x)).GetEnummerator();
        try {
                V v;
                while (e.MoveNext()) {
                        v = (V)(T)e.Current;
                        embedded-statement
                }
        }
```

```
        finally {
                //Dispose e
        }
}
```

IEnumareble arayüzünü destekleyen bir sınıf yazarsak onu bizde foreach deyimiyle birlikte kullanabiliriz. Ienumareble arayüzünü destekleyen bir sınıf şöyle yazılabilir.

1. Bir tane IEnumerator arayüzünü destekleyen bir sınıf oluşturulur. Bu sınıfın collection sınıfın içinde private olarak bildirilmesi iyi olur.
2. Ana collection sınıf için GetEnumerator fonksiyonu yazılır. Bu fonksiyon içinde Ienumerator arayüzünü destekleyen sınıf türünden nesne yaratılır ve o nesne ile geri dönülür. Nesne yaratılırken collection sınıfın kendi referansının sınıfa geçirilmesi gerekir.

```
using System;
using System.Collections;

namespace CSD
{
   class App
   {
      public static void Main()
      {
         MyCollection mc = new MyCollection();
         foreach (int x in mc)
            Console.WriteLine(x);

      }
      class MyCollection : IEnumerable
      {
         private int[] m_array;
         public MyCollection ()
         {
            m_array = new int[] {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
         }
         public IEnumerator GetEnumerator()
         {
            return new MyCollectionEnumerator(this);
         }
         //...
         private class MyCollectionEnumerator : IEnumerator
         {
            private MyCollection m_mc;
            private int m_index;

            public MyCollectionEnumerator(MyCollection mc)
            {
               m_mc = mc;
               m_index = -1;

            }
            public bool MoveNext()
            {
```

```
            ++m_index;
            return m_index < m_mc.m_array.Length;
        }
        public object Current
        {
            get
            {
                return  m_mc.m_array[m_index];

            }
        }
        public void Reset()
        {
            m_index = -1;

        }
        //...
    }
  }
 }
}
```

Anahtar Notlar: C# standartlarına göre bir collection sınıfın foreach deyimiyle kullanılabilmesi için IEnumerable arayüzünü desteklemek yerine doğrudan IEnumerator arayüzünün fonksiyonlarına sahip olması da yeterlidir.

ICollection Arayüzü: Bu arayüz ister liste ister sözlük tarzı olsun her türlü collection sınıfların ortak elemanlarını tanımlamaktadır. Bu arayüzün elemanları şunlardır.

int Count { get; }
Bu property eleman collection da tutulan eleman sayısını elde etmekte kullanılır.

void CopyTo(Array array, int index)
Görüldüğü gibi her collection sınıfının kendi tuttuğu elemanları aynı türden bir diziye kopyalayan bir CopyTo fonksiyonu olmak zorundadır. CopyTo fonksiyonunun 2.parametresi hedef dizideki kopyalamanın başlatılacağı index numarasını verir.

ICollection arayüzünün diğer elemanları MSDN dökümanlarında izlenebilir.

IList Arayüzü ve Liste Tarzı Collection Sınıflar: Liste tarzı sınıflar elemanların belli bir sıraya göre yerleştirildiği ve herhangi bir index teki elemanın elde edilebildiği Collection sınıfları belirtir. ArrayList buna tipik bir örnektir. ArrayList sınıfının pek çok fonksiyonu IList arayüzünden gelen fonksiyonlardır.

IList arayüzünün önemli elemanları şunlardır:

int Add(Object value)
Add fonksiyonu collection ın sonuna eleman eklemek için kullanılır. Geri dönüş değeri eklenen elemanın index numarasıdır.

void Clear()
Bu fonksiyon Collection içindeki tüm elemanları silmek için kullanılmaktadır.

void insert(int index, Object value)
Fonksiyonunu 1. parametresi insert pozisyonunu ikinci parametresi insert edilecek elemanı belirtmektedir.

void RemoveAt(int index)
Belirli indexteki elemanı silmek için kullanılır.

Ilist arayüzünün int parametreli indeksleyicisi belirli bir elemana erişmekte kullanılır. Ilist arayüzünün diğer elemanları MSDN dökümanlarından takipedilebilir.

ArrayList Sınıfının Bazı Ayrıntıları: ArrayList sınıfında kapasite büyütülmesi her zaman 2 kat yapılmaktadır. Başlangıç kapasitesi int parametreli başlangıç fonksiyonuyla belirlenebilir.

ICollection parametreli başlangıç fonksiyonu ilgili dizilimi dolaşarak onun elemanlarından ArrayList yapar. ArrayList sınıfındave pek çok liste tarzı Collection sınıfında AddRange gibi bir fonksiyon vardır. Bu fonksiyonlar birden fazla eleman eklemek için kullanılmaktadır.Benzer biçimde InsertRange gibi fonksiyonlarlada karşılaşılmaktadır.

Ilist arayüzünden gelen Remove fonksiyonu ArrayList içindeki belirli bir elemanı arayıp bulduktan sonra silmektedir. Arayıp bulma işlemi çok biçimli olarak Object sınıfının Equals sanal fonksiyonuyla yapılmaktadır. Object sınıfın Equals isimli sanal fonksiyonu şöyledir.

public virtual bool Equals(object obj)

int, long gibi temel yapılar için string sınıfı için zaten Equals fonksiyonları override edilmiştir. Eğer biz kendi sınıfımız ya da yapımız için Equals fonksiyonunu override etmezsek bu durumda object sınıfının ki çağrılır. Object sınıfının Equals fonksiyonu ise referans eşitliğine bakmaktadır.

```
using System;
using System.Collections;

namespace CSD
{
   class App
   {
      public static void Main()
      {
         ArrayList al = new ArrayList();

         al.AddRange(new Sample[] { new Sample(1), new Sample(2), new Sample(3), new
Sample(4), new Sample(5) });

         al.Remove(new Sample(3));

         foreach (Sample s in al)
            Console.WriteLine(s.A);

      }
   }
      class Sample
      {
         private int m_a;
```

```
        public Sample(int a)
        {
            m_a = a;
        }
        public int A
        {
            get { return m_a; }
        }
        public override bool Equals(object obj)
        {
            Sample s = (Sample)obj;
            return m_a == s.m_a;
        }
    }

}
```

ValueType sınıfının Equals fonksiyonu reflection işlemi uygulayarak ilgili türün byte uzunluğunu elde eder ve byte byte karşılaştırma yapar.

Windows Message System i: Windows sistemleri mesaj tabanlı sistemlerdir.

Anahtar Notlar: Mesaj sözcüğü daha çok aşağı seviyeli programlamada kullanılmaktadır. Mesaj yerine Event terimi de eşdeğer biçimde kullanılmaktadır.

Mesaj tabanlı sistemlerde klavye fare gibi girdi olayları daha ilk elden işletim sistemi tarafından ele alınır ve programcıya verilir. Halbuki klasik Console tabanlı çalışmada girdi olayları programcı tarafından elde edilmektedir. Mesaj tabanlı çalışma modeli grafik arayüze sahip işletim sistemlerinde zorunlu olarak uygulanmaktadır. Bugün Mac OS sistemleri Unix/Linux ortamlarındaki xwindow sistemleri mesaj tabanlı girdi modeli kullanmaktadır.

Windows işletim sistemi pencere yaratan her bir thread için bir mesaj kuyruğu oluşturmaktadır. Windows klavye, fare gibi kaynaklarda oluşan olayları izleyerek bu olayları bir mesaj biçiminde ilgili thread in mesaj kuyruğuna eklemektedir. Örneğin bir pencere üzerinde farenin sol tuşuna tıklamış olalım windows hangi pencereye tıklandığını tespit edecek bu pencerenin hangi programın threadi tarafından yaratıldığını bulacaktır. Bu tıklama olayını bir mesaj formuna dönüştürecek ve mesaj kuyruğuna ekleyecektir. Windows kuyruğa yerleştirdiği mesajlar için mesajın ne nedenle bırakıldığını anlatan bir mesaj numarası ve o olaya ilişkin diğer parametrik bilgileri de mesaja ekler.

Windows mesajı kuyruğa bırakarak mesaj konusundaki sorumluluğu programcıya devreder. Yani window kuyruğa mesajı bırakmakta ve artık başka bir şeyle ilgilenmemektedir.

Bir windows programının bir döngü içinde kuyruktaki mesajları alıp işlemesi gerekir. Buna programın ya da ilgili thread in mesaj döngüsü denilmektedir. Yani bir windows programı yaşamını mesaj döngüsünde geçirmektedir. Eğer thread in mesaj kuyruğunda mesaj yoksa thread bloke edilerek etkin bir biçimde bekletilmektedir. Yani bir programın (thread in) o anda mesaj kuyruğunda mesaj yoksa o program CPU zamanı harcamamaktadır. Birwindows programının etkin sonlanması şu biçimdedir: Kullanıcı X ikonuna tıklar. Windows mesaj kuyruğunna WM_CLOSE isimli mesajı bırakır. Bu mesajı alan program mesaj döngüsünden çıkar ve program böylece sonlanır.

Mesaj tabanlı sistemler de programlar tipik olarak bir mesaj döngüsü yoluyla kuyruktan mesajın

alınıp işlenmesi biçiminde oluşturulurlar. Programcı kuyrukta sırada bulunan mesajı alır ve hangi olay karşısında ne yapacağını belirler. Programlar “ne oldu ve ben ne yapmalıyım” biçiminde yazılmaktadır.

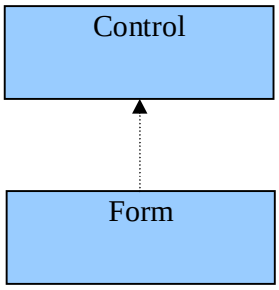
Pencere Kavramı ve Pencere Terminolojisi: İşletim sistemi tarafından bağımsız olarak kontrol edilebilen dikdörtgensel bölgelere pencere denir. İşletim sistemi yaratılan tüm pencereleri bir biçimde bir liste içinde tutmaktadır. Masa üstüne açılan pencerelere ana pencereler(top level windows) denilmektedir. Bir pencerenin içinde görüntülenen ve o pencerenin sınırları dışına çıkamayan pencerelere alt pencereler(chil windows) denilmektedir. Alt pencerelerin alt pencereleri olabilir. Birr alt pencerenin içinde bulunduğu pencereye o alt pencerenin üst penceresi (parent windows) denilmektedir. Üst pencereleri aynı olan pencere kardeş pencereler(sibliny windows) denilmektedir. Aslında en dıştaki masaüstü de bir penceredir. Bu durumda tüm anapencereler kardeş pencerelerdir. Ve masaüstü penceresinin alt pencereleri durumundadır. Hem alt pencere hem de ana pencere görünümünde olan özel bir pencere durumuna sahiplenilmiş pencerler (owned windows) denilmektedir. Sahiplenilmiş pencereler üst pencerelerin dışına çıkabilir ve ana pencere durumundadır. Bunlar hiçbir zaman üst pencerelerin altında görüntülenmezler Sahiplenilmiş pencereler ve üst pencerleri minimize edildiğinde onlarda minimize edilirler. Tipik olark dialog pencerleri böyledir. Tipik olarak bir anapencerede bir pencere başlığı ve sınır çizgileri bulunmaktadır. Fakat bir pencere klasik olarak gördüğümüz ana pencerelerin tüm bileşenlerine sahip olmak zorunda değildir. Genellikle alt pencereler başlık kısmına sahip olmazlar fakat bu bir zorunluluk cdeğildir. Aslında bir pencere sınır çizgilerine de sahip olmak zorunda değildir. Pencere başlığının altındaki çizim tapılabilen bölgeye çalışma alanı (client area) denilir. Çalışma alanının sol üst köşesi çizim alanı için orjin belirtmektedir. Bir pencere bütünsel olarak görünür ya da görünmez yapılabilir. Bir pencere klavye ve fare mesajlarına kapatılabilir.Bu duruma pencerenin pasif hale getirilmesi (disable) denilmektedir. Bu hale gelmiş bir pencere görünür bir durumdadır. Fakat windows klavye ve fare olayları için kuyruğa mesaj bırakmaz. Alt pencereler ile ana pencerler arasında ciddi bir işlecsel fazlalık ya da azlık yoktur. Bir thread in yarattığı tüm pencereleristerpencere olsun ister alt pencere oldun aynı mesaj kuyruğuna ilişkindir. Yani tüm bu pencerler için oluşan mesajlar aynı kuyruğa bırakılır. Mesaj kuyruğu mesaj başına bir tane değil thrread başına bir tanedir. Dolayısıyla bir mesaj döngüsü o threadin yrattığı tüm pencerelerin mesajlarını alıp işlemektedir. Windows da mesaj kavramı büyük ölçüde pencere kavramıyla ilişkilendirilmiştir. Yani biz bir pencere yaratmadıktan sonra bir mesaj kuyruğu ya da bir mesaj döngüsü söz konusu olmamaktadır.

Console ve GUI Uygulamaları: Windows uygulamaları kabaca Console ve GUI uygulamaları olmak üzere ikiye ayrılmaktadır. Console uygulamalarında program çalıştırıldığında siyah Console penceresi işletim sistemi tarafından otomatik yaratılmaktadır. Zaten bir exe dosyanın Console uygulaması olup olmadığı PE başlık bölümünde yazmaktadır. Şüphesiz birConsole uygulaması başka bir Console uygulaması tarafından çalıştırıldığında yeniden bir Console penceresi açılmayabilir. Fakat bir Console uygulaması isterse normal GUI pencereleri yaratıp normal mesaj döngüsüne girebilir.

GUI uygulaması tipik olarak pencereli programları anlatmaktadır. Fakat teknik anlamda bir GUI programının pencereye sahip olması da zorunlu değildir. Yani GUI uygulamaları teknik olarak Console uygulaması olmayan uygulamalardır. Yani bir GUI uygulaması hiçbir pencere yaratmazsa görsel bir uygulama sergilemez. Ayrıca bir GUI uygulaması ek bir Console penceresi de yaratabilir. Console uygulaması için csc derleyicisi /target:exe GUI uygulaması için /target:winexe dir. /target belirlemesi yapılmaz ise exe uygulaması yapıldığı kabul edilir.

İskelet GUI Programı: GUI uygulamaları için kullanılan türlerin büyük çoğunluğu Windows.Forms.dll içinde bulunmaktadır. Bu türlerin büyük çoğunluğu yine System. Windows.Forms dolayısıyla bu dll e referans edip bu isim alanını using direktifi ile kullanmak

gerekir. Fakat microsoft çizim ile ilgili system.drawing dll içine yerleştirmiştir. Bu isim alanı da System.Drawing dir.
Düğmeler, edit alanları, listeleme kutuları gibi görsel öğelerde hep birer penceredir ve tüm pencerelerin bazı ortak öğeleri vardır. Pencerelerin ortak öğelerinin hepsi kontrol sınıfında toplanmıştır. Programlama penceresi kontrol sınıfından dolaylı olarak türetilmiş form sınıfıyla temsil edilmektedir.



İskelet GUI programı şöyledir. Şu dll lere referans yapılacaktır. System.dll, System.Windows.dll, System.Drawing.dll

Application sınıfının static Run fonksiyonları mesaj döngüsünü oluşturmaktadır. Yani bu fonksiyon döngüsünde mesaj kuyruğundan mesaj alınarak sıradaki mesaj işlenir. Windows programının sonlanması için bu fonksiyonun geri dönmesi ve Main fonksiyonunun bitmesi gerekir. Anapnecerenin yaratılması Form sınıfından bir nesne yaratmakla otomatik yapılmaktadır. Fakat bir takım eklemeler yapabilmek için form sınıfını doğrudan değilde türeterek kullanmak daha doğrudur.

```
using System;
using System.Windows.Forms;
using System.Drawing;


namespace CSD
{
   class App
   {
      public static void Main()
      {
         MyForm mf = new MyForm();

         Application.Run(mf);

      }
      class MyForm : Form
      {
         public MyForm()
         {
            //...
         }
         //...
      }

   }
}
```

Application sınıfının parametresiz ve form türünden parametre alan Run fonksiyonları vardır. İskelet programda biz form parametreli Run fonksiyonunu kullandık. Bu fonksiyon aynı zamanda pencereyi visible (görünür) duruma da getirmektedir. Prametresiz Run fonksiyonu kullanıldığında Form nesnesi yaratıldığında pencere yaratıldığında ama görünür değildir. Anapencereyi görünür hale getirmek için Control sınıfından gelen Show fonksiyonu çağrılabilir ya da Control sınıfından gelen bool türden Visible propertisi true yapılabilir.

```
using System;
using System.Windows.Forms;
using System.Drawing;


namespace CSD
{
   class App
   {
      public static void Main()
      {
         MyForm mf = new MyForm();

         mf.Visible = true;

         Application.Run();

      }
      class MyForm : Form
      {
         public MyForm()
         {
            //...
         }
         //...
      }

   }
}
```

Bir programda birden fazla anapencere olabilir. Windows un görev çubuğu çalışan programları değil anapencereleri göstermektedir.

Parametresiz Application.Run() fonksiyonundan anapencereleri kapatmakla çıkamayız. Fakat parametreli Run fonksiyonundan parametre olarak verdiğimiz anapencereyi kapatarak çkabiliriz. Form nesnesinin yaratılması tek hamledede yapılabilir. Bu durumda Msin fonksiyonu tek bir satırdan oluşur.

```
using System;
using System.Windows.Forms;
using System.Drawing;


namespace CSD
{
   class App
   {
```

```
        public static void Main()
        {
            MyForm mf1 = new MyForm();

            mf1.Visible = true;

            MyForm mf2 = new MyForm();

            mf2.Visible = true;

            Application.Run(mf2);

        }
        class MyForm : Form
        {
            public MyForm()
            {
                //...
            }
            //...
        }

    }
}
```

Bundan sonraki iskelet program kopyalanacak

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
    class App
    {
        public static void Main()
        {
            Application.Run(new MyForm());

        }
        class MyForm : Form
        {
            public MyForm()
            {
                //...
            }
            //...
        }

    }
}
```

Control Sınıfının Önemli Property Elemanları: Control sınıfı tüm pencere sınıflarının ortak taban sınıfı olduğuna göre buradaki elemanlar yalnızca anapencereler için değil her türlü pencereler için anlamlıdır.

Control sınıfının çeşitli property elemanları set edildiğinde pencere üzerinde çeşitli görsel değişiklikler oluşturulabilmektedir. Aslında biz bir property e değer atıyor olsakta arka planda property nin set bölümü çalıştırılıp karmaşık API fonksiyonlarıyla işlemler yürütülmektedir. Yani işlemler bir property e değer atamak biçiminde basitleştirilmiştir.

Visible Property si: Bu property bool türden read/write property dir. Bu property ye true atarsak pencere görünür hale gelir false atarsak görünmez hale gelir.
Anapencerenin propertylerini set etme işlemi Main içinde yapılabileceği gibi pencere sınıfının başlangıç fonksiyonunda da yapılabilir. Başlangıç fonksiyonunda set işlemi iyi bir tekniktir.

Text Property si : Bu property string türünden read/write property dir. Bu property pencere yazısını oluşturur. Pencere yazısı anapencere için pencere başlığındaki yazıdır. Düğmeler için düğme üzerindeki yazı, edit alanları için adit alanı üzerindeki yazıyı çalıştırır.

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
   class App
   {
      public static void Main()
      {
         Application.Run(new MyForm());

      }
      class MyForm : Form
      {
         public MyForm()
         {
            this.Text = "Örnek Pencere";
         }
         //...
      }

   }
}
```

Color Yapısı ve BackColor Property Yapısı: BackColor property si Color yapısı türündendir. Bu property ye atama yapıldığında pencerenin zemin rengi değişir. Color yapısı renk tutmak için kullanılmaktadır. Rengin 0-255 arası red, green, blue tonel değerleri vardır. 4. bileşenine alfa faktörü denilmektedir. Alfa faktörü transparanlık belirtir. Color yapısının read only RGB ve A propertyleri byte türündendir. A propertyleri rengin ilgili Tonel bileşenlerini elde etmek için kullanılır. Color yapısının çeşitli statik propertyleri çeşitli renkleri Color yapısı olarak vermektedir. Böyle 100 ün üzerinde renk vardır. Örneğin Color.Redkırmızı rengi veren Color.Blue mavi rengi veren propertylerdir. Eğer programcı istediği tonel bileşimlerden bir renk elde etmek istiyorsa Color yapısının FromArgb statik fonksiyonlarını kullanmalıdır. 3 parametreli FromArgb fonksiyonu sırasıyla Red, Green, Blue değerlerini alır. Fonksiyonun geri dönüş değeri ilgili tonel değerlere

ilişkin renktir. Bu fonksiyon Alfa bileşenini 255 de tutar. Yani saydamsız bir renk elde edilir. Örneğin:

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
   class App
   {
      public static void Main()
      {
         Application.Run(new MyForm());

      }
      class MyForm : Form
      {
         public MyForm()
         {
            this.Text = "Örnek Pencere";
            this.BackColor = Color.Aqua;

                     }
         //...
      }

   }
}
```

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
   class App
   {
      public static void Main()
      {
         Application.Run(new MyForm());

      }
      class MyForm : Form
      {
         public MyForm()
         {
            this.Text = "Örnek Pencere";
            this.BackColor = Color.FromArgb(128, 128, 128);


         }
         //...
```

```
    }

  }
}
```

4 parametreli FromArgb fonksiyonu Alfa faktörünü de almaktadır.

```
this.Text = "Örnek Pencere";
        this.BackColor = Color.FromArgb(128, 255, 0, 0);
```

Color yapısının yalnızca default başlangıç fonksiyonu vardır.

Pencerenin Konumlandırılmasına ilişkin Yapılar ve Property ler: Konumlandırma da anapencereler için orjin masaüstünün solt üst köşesi alt pencerelerde ise üst pencerenin çalışma alanının sol üst köşesidir. Point yapısı System.Drawing isim alanı içindedir. Point yapısının amacı bir x ,y noktasını tutmaktadır. Point yapısının 2 parametreli başlangıç fonksiyonu x ve y değerlerini set ederek nesneyi oluşturur.

```
Point pt = new Point(100, 200);
```

Point yapısının X ve Y isimli int türden read/write property elemanları noktanın bileşenlerini get ve set etmek için kullanılmaktadır.

Örneğin:

```
Point pt;
        pt.X = 100;
        pt.Y = 200;
```

Point yapısının Point parametreli ve int parametreli offset fonksiyonları noktayı X Y değeri kadar öteler.

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
  class App
  {
    public static void Main()
    {
      Application.Run(new MyForm());

    }
    class MyForm : Form
    {
      public MyForm()
      {
        this.Text = "Örnek Pencere";
        Color c = Color.Yellow;

        Point pt = new Point(100, 200);
        pt.Offset(1, 1);

      }
      //...
    }

  }
```

```
}
```
Point yapısının diğer elemanları MSDN kütüphanesinden izlenmelidir.

Control sınıfının Point türünden read/write location isimli property elemanı pnecereyi konumlandırmak için kullanılmaktadır. Örneğin:

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
   class App
   {
      public static void Main()
      {
         Application.Run(new MyForm());

      }
      class MyForm : Form
      {
         public MyForm()
         {
            this.Text = "Örnek Pencere";
            this.BackColor = System.Drawing.Color.Red;
            this.Location = new Point(300, 100);

         }
         //...
      }

   }
}
```

Anapencerelerin konumlandırmada ve boyutlandırmada istisnai bir durumu vardır. Form sınıfının (Control sınıfının değil) StarPosition isimli property elemanı Form.StartPosition isimli bir enum türündendir. İşletim sistemi anapencerenin ilk konumlandırmasını bu property e göre yapmaktadır. Bu enum türünün elemanları ve anlamları şöyledir.

CenterSecreen: Bu durumda anapencere programcının belirttiği boyutta ekranın ortasında konumlandırılır.

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
   class App
   {
      public static void Main()
      {
         Application.Run(new MyForm());

      }
```

```
    class MyForm : Form
    {
        public MyForm()
        {
            this.Text = "Örnek Pencere";
            this.BackColor = System.Drawing.Color.Red;
            this.Location = new Point(300, 100);
            this.StartPosition = FormStartPosition.CenterScreen;

        }
        //...
    }

  }
}
```

WindowsDefaultLocation: Burada konumlandırma işletim sisteminin isteğine bırakılır fakat boyutlandırma yapılabilir. Bu default durumdur.

Manual: Bu durumda konumlandırma ve boyutlandırma tamamen programcıya bırakılır. Yani pencereyi location property sine değer atayarak istediğimiz yerde görüntülemek için startposition manuele çekilmelidir.

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
  class App
  {
    public static void Main()
    {
      Application.Run(new MyForm());

    }
    class MyForm : Form
    {
      public MyForm()
      {
        this.Text = "Örnek Pencere";
        this.BackColor = System.Drawing.Color.Red;
        this.Location = new Point(300, 100);
        this.StartPosition = FormStartPosition.Manual;

      }
      //...
    }

  }
}
```

Control sınıfının Top, Left read/write Right, Bottom read only int türden property elemanları Yukarı, sol, sağ ve aşağı koordinatlara ilişkindir. Yani Location propertysini set etmek yerine Left ve Top propertlerini set etmek aynı etkiyi yaratır.

```
public MyForm()
        {
            this.Text = "Örnek Pencere";
            this.BackColor = System.Drawing.Color.Red;
            this.Location = new Point(300, 300);
            this.Left = 0;
            this.StartPosition = FormStartPosition.Manual;

        }
```

Control sınıfının int türden width ve Height türden property elemanları genişlik ve yükseklik ayarlarını ayarlamaktadır.

```
this.Width = 100;
this.Height = 200;
```

Size isimli yapı tamemen point yapısı gibi iki değer tutmaktadır. Size yapısının kullanımı Point yapısının kullanımına banzemektedir. Bu yapı genişlik ve yükseklik değerlerini tutmaktadır. Yapının Width ve Heigth propertlyleri read/Write propertylerdir ve genişlik yükseklik değerlerinin set etmekte kullanılır.Set işlemi yapının başlangıç fonksiyonundan da yapılabilir.
Örneğin:

```
Size s = new Size(100, 200);
            s.Width = 50;
            s.Height = 75;
```

Controle sınıfının Size isimili property elemanı size türündendir buna atanan değer pencereyi boyutlandırır.

```
this.Size= new Size(100, 200)
```

Yani Size property sine değer atamakla width ve Heigth propertylerine değer atamak aynı şeydir.

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
   class App
   {
      public static void Main()
      {
         Application.Run(new MyForm());

      }
      class MyForm : Form
      {
         public MyForm()
         {
            this.Text = "Örnek Pencere";
            this.BackColor = System.Drawing.Color.Red;
            this.Location = new Point(300, 300);
            this.Size = new Size(100, 200);
```

```
            this.StartPosition = FormStartPosition.Manual;

        }
        //...
    }

  }
}
```

Baund Rectangle Yapısı ve Baund Property Elemanı: Rectangel yapısı dikdörtgensel bir bölgeyi tutmak için kullanılır. Rectangle yapısının 4 parametreli başlangıç fonksiyonu sol üst köşe ve genişlik, yükseklik biçiminde alınan dikdörtgensel bölgeyi tutar.

public Rectangle(int x , int y, int width, int height)

Point ve Size parametreli başlangıç fonksiyonu sol üst köşe ve genişlik yükseklik değerlerinin point ve size yapısı biçiminde alır.

public Rectangle(Point location, Size size)

```
Rectangle rect = new Rectangle(new Point(100, 200), new Size(10, 20));
```

Rectangle yapısının Read/Write Location property si sol üst köşeyi Read/Write Size property si genişlik, yükseklik bilgisinin alıp set etmekte kullanılır. Rectangle yapısının Read Only Top, Bottom, Left ve Write Property elemanları dikdörtgenin sol üst ve sağ alt köşegenlerine ilişkin koordinatları verir. X ve Y Property elemanları yine solt üst köşeye ilişkin koordinatları belirtir. Left ve Top dan farklı olarak bunlar Read/Write property lerdir. Yine Yapının Width ve Heigth propertyleri vardır.

Rectangle yapısının Contains fonksiyonları bir noktanın ya da Rectangle yapısının diğerinin içinde olup olmadığını belirlemektedir.

Rectangle yapısının Offset isimli fonksiyonları dikdörtgeni delta X, delta Y miktarı kadar kaydırır.

Yapının inflate fonksiyonları dikdörtgensel bölgeyi iki taraftan açıp büzmek için kullanılabilir.

Rectangle yapısının Intersect fonksiyonları iki dikdörtgenin kesişimine ilişkin, dikdörtgenin Union fonksiyonun birleşimine ilişkin fonksiyonudur.

Control sınıfının Read/Write Bounds isimli property elemanı Rectangle yapısı türündendir. Bu propertye değer atadığımızda pencerenin konumu bu Rectangle da belirtilen değerlere gelir. Örneğin:

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
  class App
  {
    public static void Main()
```

```
        {
            Application.Run(new MyForm());

        }
        class MyForm : Form
        {
            public MyForm()
            {
                this.Text = "Örnek Pencere";
                this.BackColor = System.Drawing.Color.Red;
                this.Bounds = new Rectangle(100, 100, 200, 200);
             }
            //...
        }

    }
}

using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
    class App
    {
        public static void Main()
        {
            Application.Run(new MyForm());

        }
        class MyForm : Form
        {
            public MyForm()
            {
                this.Text = "Örnek Pencere";
                this.BackColor = System.Drawing.Color.Red;
                this.StartPosition = FormStartPosition.Manual;
                this.Bounds = new Rectangle(100, 100, 200, 200);

            }
            //...
        }

    }
}
```

ClientRectangle ve ClientSize Property Elemanları: Control sınıfının ClientRectangle Read Only property elemanı çalışma alanının koordinatlarını Rectangle olarak verir. Fakat bu property nin verdiği Rectangle yine çalışma alanı orjinlidir. Yani verilen dikdötgenin sol üst köşesi 0, 0 durumundadır. ClientSize property si Read/Write bir property dir ve Size türündendir. Biz bu property e atama yaparsak pencere bulunduğu konumda çalışma alanının genişliği ve yüksekliği

belirlenen değer dolacak biçimde genişletilir.

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
   class App
   {
      public static void Main()
      {
         Application.Run(new MyForm());

      }
      class MyForm : Form
      {
         public MyForm()
         {
            this.Text = "Örnek Pencere";
            this.BackColor = System.Drawing.Color.Red;
            this.StartPosition = FormStartPosition.Manual;
            this.Bounds = new Rectangle(100, 100, 0, 0);

            this.ClientSize = new Size(100, 100);
            MessageBox.Show(this.ClientRectangle.ToString());

         }
         //...
      }

   }
}

using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
   class App
   {
      public static void Main()
      {
         Application.Run(new MyForm());

      }
      class MyForm : Form
      {
         public MyForm()
         {
            this.Text = "Örnek Pencere";
            this.BackColor = System.Drawing.Color.Red;
```

```
            this.StartPosition = FormStartPosition.Manual;
            this.Bounds = new Rectangle(100, 100, 0, 0);

            this.ClientSize = new Size(100, 100);
            MessageBox.Show(this.Bounds.ToString());

        }
        //...
    }

  }
}
```

Name Property si: Control sınıfının Name property si string türündendir ve read/write property dir. Name property si çalışma sırasında kütüphane tarafından kullanılmamaktadır. Programcı buraya bir değer atayıp sonra onu alabilir. Visual Studio IDE si forma yerleştirdiği her kontrole bir isim verir ve bu ismi Name property sine atar.

```
this.Name = "Test";
```

Mesaj Diyalog Penceresinin Çıkartılması ve MessageBox Sınıfı: Mesaj diyalog penceresi çıkartmak için MessageBox sınıfının statik Show fonksiyonları kullanılır. En genel bir Show fonksiyonu aşağıdaki 4 parametreli fonksiyondur. Aslında bir parametreli, 2 parametreli ve 3 parametreli show fonksiyonları bu 4 parameteli fonksiyonu default değerlerle çağırmaktadır. Örneğin programcı pratiklik için 1 parametreli Show fonksiyonunu kullanabilir. Bu durumda 4 parametreli fonksiyonun diğer parametreleri belirlenmemiş gibi etki söz konusudur. 4 parametreli show fonksiyonu şöyledir.

```
public static DialogResult Show(
        string text,
        string caption,
        MessageBoxButtons buttons,
        MessageBoxIcon icon
)
```

Fonkjsiyonun 1. parametresi message penceresi içinde görüntülenecek yazıyuı belirtir. 2. Parametre pencere başlık yazısında görüntülenecek yazıyı belirtir. 3. parametre MessageBoxButton isminde bir enum türündendir. Mesaj penceresi ile yalnızca bazı tuş takımları görüntülenebilir. Tuş takımları şunlardır.

Ok, OkCancel, AbortRetryIgnore, YesNoCancel, YesNo, RetryCancel...

Fonkjsiyonun son parametresi MessageBoxIcon isimli bir enum türündendir. Bu parametre diyalog penceresinden icon resmini belirtmektedir.

Fonksiyonun geri dönüş değeri diyalog result türündendir.

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
   class App
   {
      public static void Main()
```

```
        {
            Application.Run(new MyForm());

        }
        class MyForm : Form
        {
            public MyForm()
            {
                this.Text = "Örnek Pencere";
                this.BackColor = System.Drawing.Color.Red;
                this.StartPosition = FormStartPosition.Manual;
                this.Bounds = new Rectangle(100, 100, 0, 0);

                this.ClientSize = new Size(100, 100);
                MessageBox.Show("this is a test", "Error", MessageBoxButtons.OKCancel,
MessageBoxIcon.Question);

            }
            //...
        }

    }
}
```

Geri dönüş değeri hangi tuşla çıktığımızı belirtir.

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
    class App
    {
        public static void Main()
        {
            Application.Run(new MyForm());

        }
        class MyForm : Form
        {
            public MyForm()
            {
                this.Text = "Örnek Pencere";
                this.BackColor = System.Drawing.Color.Red;
                this.StartPosition = FormStartPosition.Manual;
                this.Bounds = new Rectangle(100, 100, 0, 0);

                this.ClientSize = new Size(100, 100);
                DialogResult result = MessageBox.Show("this is a test", "Error",
MessageBoxButtons.OKCancel, MessageBoxIcon.Question);

                if (result == DialogResult.OK)
                    MessageBox.Show("Ok");
```

```
            else if (result == DialogResult.Cancel)
               MessageBox.Show("Cancel");
         }
         //...
      }

   }
}
```

Alt Pencerelerin Yaratılması: Anımsanacağı gibi child windows üst pencerenin çalışma alanı dışına çıkamayan pencerelerdir.Alt pencereler için orjin noktası onun üst penceresinin çalışma alanının sol üst köşesidir. Windows da düğme gibi edit alanları gibi, listeleme kutuları gibi pek çok alt pencere zaten oluşturulmuş biçimdedir. Programcı bu hazır alt pencereleri programında kullanabilir. Fakat programcı doğrudan kontrol sınıfını kullanarak ya da genişletilebilir olması için kontrol sınıfından sınıf türeterek kendi alt pencerelerini oluşturabilir.

Alt pencere yaratmak için kontrol sınıfı türünden ya da kontrol sınıfından türetilmiş bir sınıf türünden bir nesne new operatörüyle oluşturmak yeterlidir.

Her pencerenin bir alt pencere listesi vardır. Bu alt pencere listesi kontrol sınıfının Controls isimli property elemanıyla temsil edilmektedir. Controls property elemanı Ilist arayayüzünü destekleyen bir sınıf türündendir.
public Control.ControlCollection Controls{get;}

Görüldüğü gibi ControlCollection sınıfı Control sınıfının içinde bildirilmiş bir sınıftır. Fakat bizim için önemli olan Controls property sinin Liste tarzı bir Collection sınıf türünden olduğudur. Alt pencerelere temsil edilen referansların üst pencereyi temsil edne sınıfın veri elemanı olarak bulundurulması nesne yönelimli programlama tekniği bakımından anlamlıdır. O halde örneğin bir anapencereye şu adımlardan geçilerek bir alt pencere yerleştirilebilir.

1. Altpencere sınıfı türünden (yani kontrol ya da kontrolden türetilmiş bir sınıf türünden) bir sınıf referansı üst pencereyi temsil eden pencere sınıfının private veri elemanı olarak bildirilir.
2. Alt pencere nesnesi üst pencereyi temsil eden sınıfın başlangıç fonksiyonunda yaratılır.
3. Altpencere referansı üst pencerenin control isimli collection elemanına eklenir.

En basit bir alt pencere kontrol sınıfı kullanılarak yaratılabilir. Bir alt pencere yaratıldığında zemin rengi üst pencerenin zemin rengi ile aynı olur. Eğer biz pencerenin sınır çizgilerini çizmezsek yaratılmış alt pencereyi fark edemeyiz. Ayrıca alt pencereleri visible duruma getirmeye gerek yoktur. Alt pencereler otomatik olarak zaten visible durumdadır. Bu durumda örneğin kontrol sınıfı kullanılarak bir alt pencere şöyle oluşturulabilir.

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
   class App
   {
      public static void Main()
      {
         Application.Run(new MyForm());
```

```
    }
    class MyForm : Form
    {
        private Control m_child;

        public MyForm()
        {
            this.Text = "Test Child";

            m_child = new Control();
            m_child.BackColor = Color.Red;
            m_child.Bounds = new Rectangle(100, 100, 100, 100);

            this.Controls.Add(m_child);

            //...
        }
        //...
    }

  }
}
```

Control sınıfını doğrudan kullanmak yerine control sınıfından bir sınıf türeterek kullanmak daha iyi bir tekniktir.

Anahtar Notlar: Alt pencereler işletim sistemi düzeyinde aslında alt pnecere nesnesi new ile yaratılırken değil üst pencerenin alt pencere listesine eklewnirken yaratılmaktadır.

Alt pencereyi üst pencerenin pencere listesine eklemenin diğer bir yolu control sınıfının parents property sini kullanmaktır. Bu property aşağıdaki gibi bildirilmiştir.
public Control Parent{get; set;}

Yanialt pencereyi üst pencerenin pencere listesine eklemek yerine alt pencerenin parents propertysinne üst pencereyi atamak aynı anlamdadır.
m_child = new MyControl(); ile *m_child.Parent = this;* aynı anlamdadır. Muhtemelen Parent propertysinin set bölümü şöyle yazılmıştır.

```
public Control Parent
        {
            set
            {
                value.Controls.Add(this);
            }
        }
```

Controls Collection elemanının ilişkin olduğu Collection sınıfının AddRange isimli bir fonksiyonu da vardır. AddRange fonksiyonu oparametre olarak Controls sınıfı türünden bir dizi alır ve dizinin tüm elemanlarını pencerenin altpencere listesine ekler. Bu durumda tek hamlede biz birden fazla alt pencereyi üst pencereye ekleyebiliriz.
Örneğin:

```
m_c1 = new MyControl();
m_c2 = new MyControl();
m_c3 = new MyControl();

this.Controls.AddRange(new Control [ ] {m_c1, m_c2, m_c3});
```

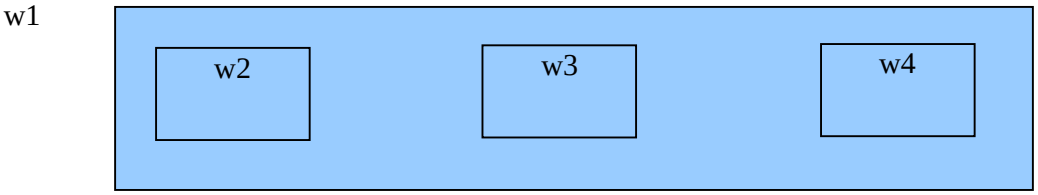
Bir alt pencere nesnesi yaratıldığında pencerenin Location ve Size property leri 0 durumundadır. Şüphesiz böyle bir pencere ekranda görülemez.

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
   class App
   {
      public static void Main()
      {
         Application.Run(new MyForm());

      }
      class MyForm : Form
      {
         private MyControl m_c1, m_c2, m_c3;

         public MyForm()
         {
            this.Text = "Test Child";


            m_c1 = new MyControl();
            m_c1.Bounds = new Rectangle(0, 0, 100, 100);
            m_c2 = new MyControl();
            m_c2.Bounds = new Rectangle(150, 0, 100, 100);
            m_c3 = new MyControl();
            m_c3.Bounds = new Rectangle(300, 0, 100, 100);

            this.Controls.AddRange(new Control [] {m_c1, m_c2, m_c3});
            //...
         }
         //...

      }
      class MyControl : Control
      {


         public MyControl()
         {
            this.BackColor = Color.Red;

         }
```

```
    }

  }
}
```

Pencere Mesajlarının İşlenmesi: Anımsanacağı gibi işletim sistemi üst ya da alt pencere üzerinde bir işlem gerçekleştiğinde gerçekleşen işleme ilişkin bir mesajı pencerenin yaratıldığı thread in mesaj kuyruğuna eklemektedir. Mesaj kuyruğundan mesajı alıp işleyen fonksiyon Application.Run fonksiyonlarıdır. Bir mesaj karşısında programcının belirlediği bir kod u çalıştırması gerekir. Bu işlem biraz karmaşık bir süreçle gerçekleşmektedir.

.net ortamı yaratılan her pencerenin pencere ister alt ister ana pencere olsun control referansını kendi içinde bir collection da saklar. Örneğin bir anapencere üçte alt pencere söz konusu olsun. Bu pencerelerinin hepsi dolaylı olarak control sınıfından türetilmiştir.



Burada .net ortamı bu dört pencerenin control referansınıda collection içinde tutuyor durumdadır. Windows un kuyruğa bıraktığı her mesajın en azından ortak bazı bileşenleri vardır. Örneğin windows her mesaj için o mesaj hangi pencereye ilişkin olduğunu belirten bir handle değeri içerir. Aynı zamanda her mesajın bir numarası vardır. Mesajın numarası mesajın hangi olaydan dolayı kuyruğa bırakıldığını belirtmektedir. Ayrıca her mesaj ilave bazı parametrik bilgilere sahiptir.

Application.Run fonksiyonları kuyruktan mesajı alır. Mesajın handle değerinden haraketle Collection içinde arama yaparak mesajın ilişkin olduğu pencerenin control referansını elde eder.

Application.Run fonksiyonları mesajın ilişkin olduğu control referansını bulduktan sonra mesaj numarasını inceler. Her mesaj için control sınıfının bir tane OnXXX isimli protected bir sanal fonksiyonu vardır. İşte Run fonksiyonları mesajın numarasına bakarak ilgili control referansını kullanıp sanal OnXXX fonksiyonlarını çağırmaktadır. Yukarıdaki açıklamalara göre programcı bir olay gerçekleştiğinde kendi kodunun çalışmasını şöyle sağlayabilir.

1. Programcının ilgili mesaj olayı için çağrılacak OnXXX fonksiyonunu bilmesi gerekir.
2. Programcı belirlediği OnXXX fonksiyonunu kendi sınıfında Override eder. Böylece Application.Run fonksiyonları bu sanal OnXXX fonksiyonunu çağırdığında programcının fonksiyonu çağrılmış olur.

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
  class App
  {
    public static void Main()
    {
      Application.Run(new MyForm());

    }
```

```
    class MyForm : Form
    {
        private MyControl m_c1, m_c2, m_c3;

        public MyForm()
        {
            this.Text = "Test Child";


            m_c1 = new MyControl();
            m_c1.Bounds = new Rectangle(0, 0, 100, 100);
            m_c2 = new MyControl();
            m_c2.Bounds = new Rectangle(150, 0, 100, 100);
            m_c3 = new MyControl();
            m_c3.Bounds = new Rectangle(300, 0, 100, 100);

            this.Controls.AddRange(new Control [] {m_c1, m_c2, m_c3});
            //...
        }
        //...
        protected override void OnClick(EventArgs e)
        {
            MessageBox.Show("test");
        }

    }
    class MyControl : Control
    {


        public MyControl()
        {
            this.BackColor = Color.Red;

        }
    }

  }
}
```

Mesaj Parametre Sınıfları: Anımsanacağı gibi windows her mesaj için Handle değeri ve mesaj numarasının yanısıra o mesaja özgü ek birtakım bilgiler de yerleştirebiliyordu. İşte Application.Run fonksiyonları kuyruktan mesajı aldığında mesajın parametrik bilgilerini elde edip bu parametrik bilgileri YYYEventArgs isimli bir sınıf nesnesi biçimine dönüştürüp OnXXX fonksiyonuna parametre olarak geçirmektedir. Yani programcı hangi mesajı işliyorsa o mesajın parametrik bilgilerini OnXXX fonksiyonunun parametresinden elde edebilir. Tüm YYYEventArgs isimli sınıflar EventArgs isimli bir taban sınıftan türetilmiştir. EventArgs isimli sınıf parametresiz sınıfı bir sınıfı temsil etmektedir. Yani EventArgs parametre sınıfı bir mesaj parametresine sahip olmayan bir sınıftır.

Event Yoluyla Mesaj İşlenmesi: Mesaj işlemenin diğer bir yolu control sınıfının Event elemanlarını kullanmaktır. Mesajların bu biçimde işlenmesi daha pratik olduğu için tercih

edilmektedir.

Kontrol sınıfının her mesaj için bir OnXXX sanal fonksiyonunun yanısıra buna karşı gelen bir XXX Event elemanı vardır. Örneğin OnClick fonksiyonuna karşılık gelene Event elemanının ismi Click OnKeyDown fonksiyonuna karşılık gelen Event elemanının ismi KeyDown dur. İşte control sınıfının OnXXX fonksiyonları XXX isimli event elemanının fonksiyonlarını çağırmaktadır. Böylece programcı bir mesaj için OnXXX isimili sanal fonksiyonunu Override etmek yerine XXX elemanına fonksiyonda girebilir. Sık yapılan bir hata bir mesajın her iki yöntemle birden işlenmesi sırasında oluşmaktadır. Programcının hem OnXXX isimli sanal fonksiyonunu Override ettiğini hemde XXX Event elemanına += operatörü ile fonksiyon girdiğini düşünelim. Bu durumda ilgili mesaj oluştuğunda OnXXX fonksiyonu çağrılacak fakat XXX Event fonksiyonu çağrılmayacaktır. Çünkü XXX Event fonksiyonlarını çağıran Control sınıfının OnXXX fonksiyonudur. Halbuki artık Override edilmiş fonksiyon çağrılmaktadır. Eğer programcı Control sınıfının OnXXX nin fonksiyonunu da çağırmak istiyorsa bunu aşağıdaki gibi base.OnXXX yaparak sağlamalıdır.

Kontrol sınıfının OnXXX sanal fonksiyonuna karşılık bir XXX event elamanı vardır. Bu Event eleleman YYY isimli bir delege türündendir. Programcının bu event elemana fonksiyon ekleyebilmesi için ilgili delege türünden nesne yaratması gerekir. Ve çağrılacak event fonksiyonu o delegenin parametrik yapısına uygun olmak zorundadır. Örneğin MouseDown isimli Event elemanının ilişkin olduğu delege sınıfı MouseEvent Handler isimli sınıftır. Bu delege sınıfı geri dönüş değeri void olan 1. parametresi object 2. parametresi MouseEventArgs türünden olan fonksiyonları tutabilir. Dolayısıyla programcının bu Event elemana fonksiyon girerken MouseEventHandler türünden bir nesne yaratması ve gireceği fonksiyonu burada belirtmesi gerekir. Örneğin: (kırmızı bölüm)

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
   class App
   {
      public static void Main()
      {
         Application.Run(new MyForm());

      }
      class MyForm : Form
      {
         public MyForm()
         {
            this.MouseDown += new MouseEventHandler(MouseDownHandler);
         }
         public void MouseDownHandler(object sender, MouseEventArgs mea)
         {
            MessageBox.Show("Test");
         }
         //...
      }

   }
}
```

Çağrılacak fonksiyon sarı ile işaretlenmiştir.

YYYEventHandler isimli delegenin ikinci parametresi oluşan mesaja ilişkin mesaj parametre sınıfı türündendir. Bu tür OnXXX sanal fonksiyonuna geçirilen parametre sınıf türüyle aynıdır. Mesaj işleme konusunun anahtar noktaları şöyledir.

- Mesajlar her zaman Control sınıfının OnXXX sanal fonksiyonunun Override edilmesiyle işlenebilir.
- OnXXX sanal fonksiyonu Override etmek yerine Control sınıfının XXX isimli YYYEventHandler isimli delege türünden Event elemanına bir delege nesnesi eklenebilir.
- XXX Event elemanının ilişkin olduğu YYYEventHandler isimli delege sınıfının geri dönüş değeri void, 1. parametresi object ve 2. parametresi mesaj parametre sınıfı türündendir. Bu ikinci parametrenin türü OnXXX fonksiyonunun türüyle aynıdır.

Mesaj oluştuğunda çağrılacak mesaj fonksiyonunun birinci object parametresi ne anlama gelmektedir? Bu parametre mesaj hangi nesne üzerinde oluşmuşsa o nesnenin referansını belirtmektedir. Örneğin Forum a ilişkin bir mesaj oluşmuşsa object parametresi yaratılmış form sınıfının referansını bir düğme içinde olay gerçekleşmişse object parametresi bu düğme nesnesinin referansını belirtmektedir. Olayları daha iyi anlamak için kontrol sınıfının OnMouseDown sanal fonksiyonunun nasıl yazıldığını gösterelim:

```
class Control
    {
      //...
      public event MouseEventHandler MouseDown;
      protected virtual void OnMouseDown(MouseEventArgs e);
      {
        //...
        MouseDown(this, e);
        //...

    }
```

Bir mesajın işlenmesi için nelerin bilinmesi gerekir?

1. Programcının hangi olay gerçekleştiğinde ilgili mesajın kuyruğa bırakıldığını bilmesi gerekir.
2. Programcı mesaj parametre sınıfını incelemeli böylece ilgili mesaj oluştuğunda hangi ek bilgilerinde verildiğini belirleyebilmelidir.
3. Programcı bu mesaj karşısında çağrılacak kontrol sınıfının OnXXX fonksiyonunun hangisi olduğunu bilmelidir.
4. Programcının ilgili mesaja ilişkin event elemanının ne olduğunu ve bunun hangi delege sınıfından olduğunu bilmesi gerekir.

Temel Pencere Mesajları

Click Mesajı: Click mesajı fareyle pencerenin çalışma alanı üzerine tıklanıp el çekildiğinde gönderilir. Bu mesaj karşısında Control sınıfının çağrılan sanal fonksiyonu OnClick isimli fonksiyondur. Mesajın mesaj parametres sınıfı EventArgs sınıfıdır. Yanibu mesaj bize ekstra bir bilgi vermektedir. Tetiklenen Event eleman Click ismindedir. Bu event eleman EventHandler isimli delege türündendir. EventHandler delegesinin geri dönüş değeri void, 1. parametresi object, 2. parametresi EventArgs türündendir.

```
using System;
using System.Windows.Forms;
using System.Drawing;
```

```
namespace CSD
{
   class App
   {
      public static void Main()
      {
         Application.Run(new MyForm());

      }
      class MyForm : Form
      {
         public MyForm()
         {
            this.Text = "Sample Event";

            this.Click += new EventHandler(clickEventHandler);

         }
         private void clickEventHandler(object sender, EventArgs e)
         {
            MessageBox.Show("Click");
         }
      }
   }
}
```

MouseDown ve MouseUp Mesajları: Fareyle tuşa basıldığında MouseDown mesajı el tuştan çekildiğinde MouseUp mesajı oluşur. Çağrılan sanal fonksiyonlar Control sınıfının OnMouseDown ve OnMouseUp fonksiyonlarıdır. İlgili Event elemanı MouseDown ve MouseUp isimli elemanlardır.

Bu mesajlar da çalışma alanı mesajlarıdır. Nu mesajlar oluştuğunda hangi tuşa basılmış ya da çekilmiş olduğu ve hangi koordinatta bu işin gerçekleştiği bilgileri de verilmektedir. Bu mesajların mesaj parametre sınıfı MouseEventArgs isimli sınıftır. Bu sınıfın X ve Y isimli int türden Read Only property elemanları basım ve çekim koordinatlarını vermektedir. Sınıfın Location isimli property elemanı ise aynı koordinatı point olarak verir. Hangi tuşa basıldığı Button property(public MouseButtons Button { get; }) si ile elde edilebilir. Bu property MouseButtom isimli bir enum türündendir. Sınıfın diğer elemanları MSDN dökumanlarından izlenmelidir. MouseDown ve MouseUp event elemanları MouseDownHandler delege türündendir.

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
   class App
   {
      public static void Main()
      {
         Application.Run(new MyForm());

      }
```

```
        class MyForm : Form
        {
            public MyForm()
            {
                this.Text = "Sample Event";

                this.MouseDown += new MouseEventHandler(mouseDownHandler);

            }
            private void mouseDownHandler(object sender, MouseEventArgs mea)
            {
                MessageBox.Show(mea.Location.ToString());
            }
        }
    }
}

using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
    class App
    {
        public static void Main()
        {
            Application.Run(new MyForm());

        }
        class MyForm : Form
        {
            public MyForm()
            {
                this.Text = "Sample Event";

                this.MouseDown += new MouseEventHandler(mouseDownHandler);

            }
            private void mouseDownHandler(object sender, MouseEventArgs mea)
            {

                if (mea.Button == MouseButtons.Left)
                {
                    MessageBox.Show(mea.Location.ToString());
                }
            }
        }
    }
}
```

Resize Mesajı: Bir ana pencerenin boyutlarının değiştirilebilmesi için Form sınıfının

FormBorderStyle elemanın Sizable olması gereekir. Zaten Form sınıfı için bu property default olarak bu değerdedir. Eğer programcı anapencerenin boyutlandırılabilir olmasını istemiyorsa FormBorderStyle property sine Sizable yerine Fixed içeren başka bir değer atamalıdır.
Örneğin:

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
   class App
   {
      public static void Main()
      {
         Application.Run(new MyForm());

      }
      class MyForm : Form
      {
         public MyForm()
         {
            this.Text = "Sample Event";

            this.FormBorderStyle = FormBorderStyle.Fixed3D;//boyutlandırılamaz.

            this.MouseDown += new MouseEventHandler(mouseDownHandler);

         }
         private void mouseDownHandler(object sender, MouseEventArgs mea)
         {

            if (mea.Button == MouseButtons.Left)
            {
               MessageBox.Show(mea.Location.ToString());
            }
         }
      }
   }
}
```

Görüldüğü gibi FormBorderStyle propertysi hem anapencerenin kenar çizgilerini belirlemekte hem de anapencerenin boyutlandırılabilir olup olmadığını belirlemektedir.

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
   class App
   {
      public static void Main()
      {
         Application.Run(new MyForm());
```

```
    }
    class MyForm : Form
    {
        public MyForm()
        {
            this.Text = "Sample Event";

            this.FormBorderStyle = FormBorderStyle.Sizable;//kenardan çekerek boyutlandırılabilir
form

            this.MouseDown += new MouseEventHandler(mouseDownHandler);

        }
        private void mouseDownHandler(object sender, MouseEventArgs mea)
        {

            if (mea.Button == MouseButtons.Left)
            {
                MessageBox.Show(mea.Location.ToString());
            }
        }
    }
  }
}
```

Pencerenin boyutu fareyle ya da programlama yoluyla değiştirildiğinde windows Resize mesajını kuyruğa bırakmaktadır. Bu mesaj oluştuğunda çağrılan sanal fonksiyon OnRize isimli fonksiyondur. Mesajın parametre sınıfı EventArgs sınıfıdır. Yani bu mesaj bize ek bir bilgi vermemektedir. Mesajın ilişkin olduğu Event elemanı Resize isimli elemandır. Resize event elemanı EventHandler isimli delege türündendir.

Anahtar Notlar: Windows 95, 98 ME sistemlerinde pencereyi genişletirken ya da sürüklerken default olarak pencere içeriği tazelenmemektedir. Halbuki Windows XP ve Vista da default durum pencere içeriğinin de tazelenmesidir. İşte Windows XP ve Vista daki bu default surumda Resize mesajı genişletme ya da daraltma sırasında sürekli gönderilmektedir. Halbuki diğer modda genişletme ya da daraltma işleminden sonra elimizi fareden çektiğimizde bu mesaj bir kez gönderilir.

Programcı Risize mesajı oluştuğunda artık Control sınıfının Size, Width, Heigth gibi propery elemanlarına bakarak boyutlandırma sonucundaki yeni pencere genişlik ve yüksekliğini elde edebilir. Programcı ClientSize ve ClientRectangle propertyleri ile çalışma alanının yeni boyutlarını elde edebilir.

Bazı uygulamalarda programcı pencerenin belirli bir boyuttan daha fazla küçültülmesini ya da büyütülmesini istemeyebilir. Bunu sağlamak için Form sınıfının Size türünden MaximumSize ve MinumumSize propertylerini kullanmaktadır. **Örneğin:**

```
this.MinimumSize = new Size(200, 200);
```

```
using System;
using System.Windows.Forms;
using System.Drawing;
```

```
namespace CSD
{
   class App
   {
      public static void Main()
      {
         Application.Run(new MyForm());

      }
      class MyForm : Form
      {
         public MyForm()
         {
            this.Text = "Sample Event";

            this.FormBorderStyle = FormBorderStyle.Sizable;

            this.MouseDown += new MouseEventHandler(mouseDownHandler);
            this.Resize += new EventHandler(resizeEventHandler);
            this.MinimumSize = new Size(200, 200);

         }

         private void resizeEventHandler(object sender, EventArgs mea)
         {
            MessageBox.Show(this.Size.ToString());
         }

         private void mouseDownHandler(object sender, MouseEventArgs mea)
         {

            if (mea.Button == MouseButtons.Left)
            {
               MessageBox.Show(mea.Location.ToString());
            }
         }
      }
   }
}
```

Resize mesajı maximize ve minimize işlemleri sırasında da oluşturulur. Fakat .net te (API de böyle değil) pencere ilk kez açıldığında Resize mesajı oluşturulmamaktadır. Programcı bazen pencerenin maximize ve minimize gibi özel durumlarda olup olmadığını anlamak isteyebilir. Ya da pencereyi programlama yoluyla maximize ve minimize yapmak isteyebilir.

Diğer Fare Mesajları: Fare mesajlarının hepsinin mesaj parametre sınıfı MouseEventArgs sınıfıdır. Bu sınıf MouseDown mesajında ele alınmıştır. Yine mesaj oluştuğunda tetiklenecek event elemanlar MouseEventHandler delege türündendir.

MouseUp mesajı el fareden çekildiğinde oluşur. MouseUp mesajı için çağrılacak sanal fonksiyon OnMouseUp fonksiyonudur. Bu fonksiyonda MouseUp event elemanının fonksiyonlarını çağırır.

Farenin tuşuna basılı olsun ya da olmasın fare çalışma alanı üzerinde hareket ettrildiğinde windows kuyruğa MouseMove mesajları bırakır. Bu mesajlar fare hareket ettririlirken her pixel için oluşturulmak zorunda değildir. Hangi yoğunlukta oluşturulacağı sistemin içinde bulunduğu duruma bağlıdır. Bu mesaj için çağrılacak sanal fonksiyon OnMouseMove fonksiyonudur. Bu fonksiyonda MouseMove event elemanını tetikler.

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
   class App
   {
      public static void Main()
      {
         Application.Run(new MyForm());

      }
      class MyForm : Form
      {

         private Point m_prevPoint;
         private Pen m_redPen;


         public MyForm()
         {
            this.Text = "ScratchPad Appilication";
            this.BackColor = Color.White;
            this.Size = new Size(800, 400);

            this.MouseDown += new MouseEventHandler(mouseDownHandler);
            this.MouseMove += new MouseEventHandler(mouseMoveHandler);
         }
         //...

         private void mouseDownHandler(object sender, MouseEventArgs mea)
         {
            if (mea.Button == MouseButtons.Left)
               m_prevPoint = mea.Location;
         }

         private void mouseMoveHandler(object sender, MouseEventArgs mea)
         {
            if (mea.Button == MouseButtons.Left)
            {
               Graphics g = this.CreateGraphics();
               g.DrawLine(Pens.Red, m_prevPoint, mea.Location);

               m_prevPoint = mea.Location;
            }
         }
```

```
        }

    }
}

using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
    class App
    {
        public static void Main()
        {
            Application.Run(new MyForm());

        }
        class MyForm : Form
        {

            private Point m_prevPoint;
            private Pen m_redPen;


            public MyForm()
            {
                this.Text = "ScratchPad Appilication";
                this.BackColor = Color.White;
                this.Size = new Size(800, 400);

                m_redPen = new Pen(Brushes.Red, 5);

                this.MouseDown += new MouseEventHandler(mouseDownHandler);
                this.MouseMove += new MouseEventHandler(mouseMoveHandler);
            }
            //...

            private void mouseDownHandler(object sender, MouseEventArgs mea)
            {
                if (mea.Button == MouseButtons.Left)
                    m_prevPoint = mea.Location;
            }

            private void mouseMoveHandler(object sender, MouseEventArgs mea)
            {
                if (mea.Button == MouseButtons.Left)
                {
                    Graphics g = this.CreateGraphics();
                    g.DrawLine(m_redPen, m_prevPoint, mea.Location);

                    m_prevPoint = mea.Location;
```

```
            }
        }
    }

  }
}
```

System Renkleri: Anımsanacağı gibi denetim masasında çeşitli pencere desenleri global düzeyde ayarlanabilmektedir. Örneğin biz denetim masasında pencere zemin rengini değiştirirsek pek çok programın pencerelerinin zemin rengi değişecektir. Denetim masasından çeşitli ögelerin renkleri değiştirildiğinde windows arka planda tüm pencerelere rengin değiştiğini belirten mesajlar göndermektedir. Fakat .net te programın denetim masasındaki değişikliklere duyarlı hale getirilmesi oldukça kolaylaştırılmıştır. SystemColors isimli sınıfın Color türünden pek çok static property si vardır. Bu sınıfın denetim masasındaki çeşitli ögeleri temsil eden elemanları vardır. Örneğin SystemColor.Window denetim masasındaki pencere zemin rengini temsil eder. Yani biz bizden bir renk istendiği zaman o rengi SystemColors.Window biçiminde verirsek denetim masasında pencerenin zemin rengi değiştirildiğinde bu renk artık değişmiş rengi temsil eder.

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
   class App
   {
      public static void Main()
      {
         Application.Run(new MyForm());

      }
      class MyForm : Form
      {

         private Point m_prevPoint;
         private Pen m_redPen;


         public MyForm()
         {
            this.Text = "ScratchPad Appilication";
            this.BackColor = SystemColors.Window;
            this.Size = new Size(800, 400);

            m_redPen = new Pen(Brushes.Red, 5);

            this.MouseDown += new MouseEventHandler(mouseDownHandler);
            this.MouseMove += new MouseEventHandler(mouseMoveHandler);
         }
         //...

         private void mouseDownHandler(object sender, MouseEventArgs mea)
```

```
        {
            if (mea.Button == MouseButtons.Left)
                m_prevPoint = mea.Location;
        }

        private void mouseMoveHandler(object sender, MouseEventArgs mea)
        {
            if (mea.Button == MouseButtons.Left)
            {
                Graphics g = this.CreateGraphics();
                g.DrawLine(m_redPen, m_prevPoint, mea.Location);

                m_prevPoint = mea.Location;
            }
        }
    }

  }
}
```

Pencerelerin Yok Edilmesi: Bir pencerenin nesne olarak yaratılmasıyla işletim sistemi genelinde yaratılması farklı zamanlarda gerçekleştirilmektedir. Control sınıfı ya da o sınıftan türetilmiş bir sınıf türünden nesneyi new operatörü ile yarattığımızda henüz windows düzeyinde alt pencere yaratılmamış durumdadır. Alt pencerenin windows düzeyinde yaratılması üst pencerenin pencere listesine eklenmesi sırasında yapılmaktadır. Fakat ana pencereler için durum böyle değildir. Ana pencereler gerçekten de nesne new operatörüyle yaratıldığında yaratılmaktadır. Peki bir pencere nasıl yok edilir. Alt pencerelerin fiziksel olarak yok edilmesi tamamen ters bir işlem olarak alt pencerenin üst pencerenin pencere listesinden silinmesiyle gerçekleştirilebilir. Fakat ana pencerenin yok edilmesi form sınıfının Close fonksiyonuyla yapılmaktadır. Şüphesiz üst pencere yok edildiğinde onun tüm alt pencereleri de yok edilmektedir.

Görüldüğü gibi programlama yoluyla ana pencereyi kapatıp mesaj döngüsünden çıkmak istiyorsak tek yapılacak şey Form sınıfının Close fonksiyonun çağırmaktır. Ana pencereyi kapatmanın bir yolu da pencerenin sağ üst köşesindeki X simgesine tıklamaktır. Şüphesiz kullanıcı X simgesine tıkladığında bu işlemin iptal edilebilmesine olanak verilmelidir. İşte . net te X tuşuna tıklandığında Form sınıfının bazı event elemanları tetiklenmektedir.

Form sınıfının FormClosing isimli event elemanı FormClosingEventHandler isimli delege türündendir. Windows tarafından oluşturulan bu mesajın mesaj parametre sınıfı FormClosingEventArgs isimli sınıftır.

FormClosingEventArgs sınıfının bool türden Cancel isimli property elemanına event fonksiyonu çağrıldıktan sonra .net tarafından bakılmaktadır. Eğer bu elemanda true varsa pencerenin kapatılması iptal edilir. Bu event çağrıldığında bu eleman içinde default olarak false vardır.

Form sınıfının Close fonksiyonu yine FormClosing mesajını oluşturmaktadır. O halde çıkış kontrolü (örneğin save edilmesine yönelik) FormClosing mesaında gerçekleştirilebilir.

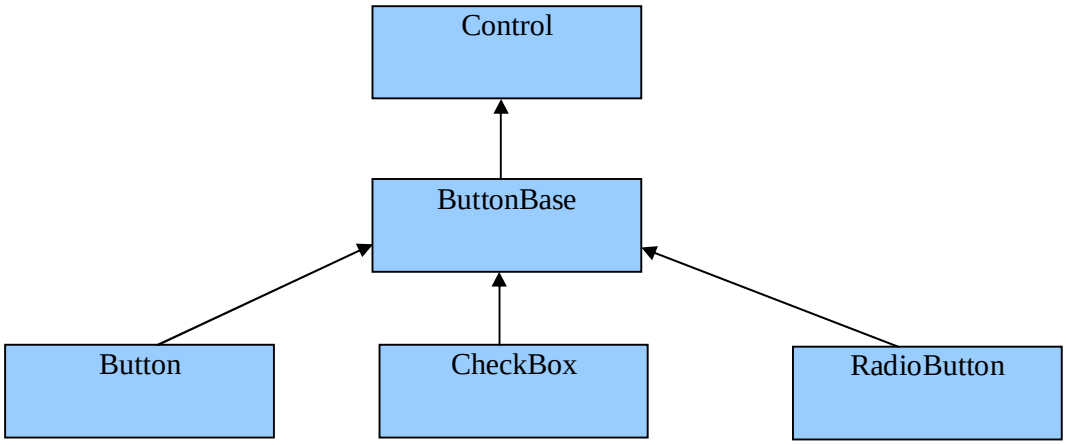
FormClosing event elemanı anapencere henüz yok edilmeden önce tetiklenmektedir. Fakat bazen ana pnecere yok edildikten sonra da bir takım işlemlerin yapılması gerekebilir. Bunun için FormClosed event elemanı kullanılır. Şüphesiz artık bu event oluştuğunda pencerenin kapatılmasının iptal edilmesi söz konusu değildir. O halde FormClosing mesajı işlemin iptal

edilmesi için FormClosed mesajı kaynakların boşaltılması için kullanılmalıdır.

Bir uygulamayı sonlandırmanın diğer bir yolu da Appilicationsınıfının exit static fonksiyonunun çağrılmasıdır. Bu fonksiyon doğrudan mesaj döngüsünden çıkmaya yarar.

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
   class App
   {
      public static void Main()
      {
         Application.Run(new MyForm());

      }
      class MyForm : Form
      {
         public MyForm()
         {
            this.Text = "Form Closing Example";

            this.MouseDown +=new MouseEventHandler(MyForm_MouseDown);
            this.FormClosing += new FormClosingEventHandler(closingEventHandler);
         }
         //...
         void MyForm_MouseDown(object sender, MouseEventArgs e)
         {
            this.Close();
         }

         private void closingEventHandler(object sender, FormClosingEventArgs fcea)
         {
            DialogResult dr;
           dr = MessageBox.Show("Save Changes?", "Message",
MessageBoxButtons.YesNoCancel);
            if (dr == DialogResult.Cancel)
               fcea.Cancel = true;
         }
      }

   }
}
```

Düğme Kontrolü: Düğme kontolleri genellikle bir olayı başlatmak için kullanılır. Düğmeler(push buttons) seçenek kutuları(check boxes) ve radyo düğmeleri(radio buttons) ile benzerlikler göstermektedir. Bu nedenle bu üç sınıfın ortak elemanları buttonbase isimli bir taban sınıfta toplanmıştır ve bu üç kontrolü temsil eden sınıflar ButtonBase isimli sınıftan türetilmişlerdir.



Görüldüğü gibi düğmeler Button sınıfından elde edilmektedir.

Bir düğme şöyle kullanılır.

1. Button sınıfı türünden bir nesne yaratılır. Nesne üst pencerenin pencere listesine eklenir.
2. Nesnenin Text property si düğmenin üzerinde çıkacak yazıyı belirtir. Programcı isterse diğer property lerle görüntüyü ayarlayabilir.
3. Button sınıfının Control sınıfından gelen Click Event elemanı düğmeye tıklandığında işlemin başlatılması için kullanılabilir.

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
   class App
   {
      public static void Main()
      {
         Application.Run(new MyForm());

      }
      class MyForm : Form
      {
         private Button m_buttonOk;
         private Button m_buttonCancel;

         public MyForm()
         {
            m_buttonOk = new Button();
            m_buttonOk.Text = "Ok";
            m_buttonOk.Location = new Point(30, 100);
            m_buttonOk.Click += new EventHandler(ClickOkEventHandler);
            m_buttonOk.ForeColor = Color.Red;

            m_buttonCancel = new Button();
            m_buttonCancel.Text = "Cancel";
            m_buttonCancel.Location = new Point(120, 100);
```

```
            m_buttonCancel.Click += new EventHandler(ClickCancelEventHandler);
            m_buttonCancel.ForeColor = Color.Red;

            this.Controls.AddRange(new Control[] { m_buttonOk, m_buttonCancel });

        }
        private void ClickOkEventHandler(object sender, EventArgs ea)
        {
            MessageBox.Show("Ok");
        }

         private void ClickCancelEventHandler(object sender, EventArgs ea)
        {
            MessageBox.Show("Cancel");
        }
        //...
    }

  }
}
```

Control sınıfının bool türden read/write Enabled isimli property elemanı pencereyi kontrolü klavye ve fare mesajlarına kapatmaktadır. Şüphesiz kontrol nesnesi yaratıldığında bu property default true durumdadır. Pasif duruma getirilen pek çok standart kontrol solgun renkle gösterilmektedir. Aslında bu solgun görüntüleme işlemi windows tarafından otomatik yapılmamaktadır. Windows pencere pasif duruma geçerken kontrole EnabledChanged mesajı gönderir. Kontrolü yazan kişi de bu mesajı işleyerek solgun renkli çizimi yapmaktadır.

Kontrollerin Fareyle Sürüklenmesi: Şekil sürükleme ve kontrol sürükleme gibi işlemlerin mantığı aynıdır. İki mouseMove mesajı arasında farenin önceki ve sonraki konumları dikkate alınarak **Δx** ve **Δy** yer değiştirmeleri hesaplanır. Şüphesiz programcı ilgili kontrolün MouseDown ve MouseMove mesajlarını işlemelidir.

Programcı konrolü Location property sini kullanarak Δx - Δy miktarı kadar ötelemelidir. Bu ötelemeyi yaptıktan sonra farenin hareket ettirildiği nokta ilk nokta haline gelir. Yani farenin ilk tıklandığı nokta konrol ötelendiğinde yine aynı nokta olarak kalmaktadır. Başka bir deyişle tıklanan nokta kontrolün sol üst köşesi orjinli noktası elde edilir.

Anahtar Notlar: Bir propert eleman tamamen bir fonksiyon gibi davranmaktadır. Örneğin: m_buttonOk.Location işleminde Location property sinin get bölümü çalıştırılacaktır. Bu propert muhtemelen aşağıdaki gibi yazılmıştır.

```
class Control
{
        private Point m_Location;
        //...
        public Point Location
        {
                get
                {
                        return m_Location;
                }
                set
                {
```

```
                    m_Location = value;
                    //...
                }
            }
            //...
}
```

Property bize bir yapı veriyorsa biz o yapının bir elemanına eriştiğimizde sınıf içerisindeki yapının elemanına erişmiş olmayız. Property nin verdiği return ifadesiyle yaratılan geçici yapı değişkeninin elemanına erişmiş oluruz. Yani valx ve valy değerlerini biz örneğin düğmenin location property isinin x ve y kısımlarına şöylr yerleştiremeyiz.

```
m_buttonOK.Location x = valx;
m_buttonOK.Location y = valy;
```

bu işlemi aşağıdaki gibi yapmalıyız.

```
m_button.Location = new Point(valx, valy);
```

Kontrol ün sürüklenmesi için kontrol ün (örneğimizde biz düğme diyoruz)mouseDown, mouseUp ve mouseMove event fonksiyonları şöyle yazılabilir.

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
   class App
   {
      public static void Main()
      {
        Application.Run(new MyForm());
      }
   }


   class MyForm : Form
   {
      private Button m_buttonOK;
      private Point m_clickPoint;
      private bool m_flag;

      public MyForm()
      {
         this.Text = "Dragging Example";

         m_buttonOK = new Button();
         m_buttonOK.Text = "&Ok";
         m_buttonOK.MouseDown += new MouseEventHandler(m_buttonOK_MouseDown);
         m_buttonOK.MouseUp += new MouseEventHandler(m_buttonOK_MouseUp);
         m_buttonOK.MouseMove += new MouseEventHandler(m_buttonOK_MouseMove);

         m_buttonOK.Parent = this;
      }

      void m_buttonOK_MouseDown(object sender, MouseEventArgs e)
```

```
    {
        if (e.Button == MouseButtons.Left)
        {
            m_clickPoint = e.Location;
            m_flag = true;

        }
    }

    void m_buttonOK_MouseUp(object sender, MouseEventArgs e)
    {
        if (e.Button == MouseButtons.Left)
            m_flag = false;

    }

    void m_buttonOK_MouseMove(object sender, MouseEventArgs e)
    {
        if (m_flag)
        {
            int deltaX = e.X - m_clickPoint.X;
            int deltaY = e.Y - m_clickPoint.Y;

            m_buttonOK.Left += deltaX;
            m_buttonOK.Top += deltaY;
        }
    }
  }
}
```

Mevcut Bir Control ün Türeterek Genişletilmesi: Elimizde başkaları tarafından yazılmış olan bir kontrol sınıfı bulunuyor olsun. (Örneğin button sınıfı başkaları tarafında yazılmış bir control sınıfıdır) Biz bu mevcut kontrol ü genişletmek istersek bu control sınıfından türetme yapabiliriz. Ve artık pencereri bu türetilmiş sınıfı kullanarak oluşturabiliriz. Örneğin button sınıfını genişletmek istersek Button sınıfından MyButton isimli bir sınıf türetebiliriz. Ve button nesneleri yerine MyButton nesneleri yaratabiliriz. Örneğin biz fareyle sürüklenebilen bir Button sınıfı yazabiliriz. Bunun için Button sınıfından DargButton gibi bir sınıf türetiriz. Button sınıfının MouseDown, MouseUp, MouseMove event elemanlarını bu DragButton sınıfında işleyebiliriz. Böylece Button yerine DragButton sınıfını kullanarak sürüklenebilen düğmeler oluşturabiliriz.

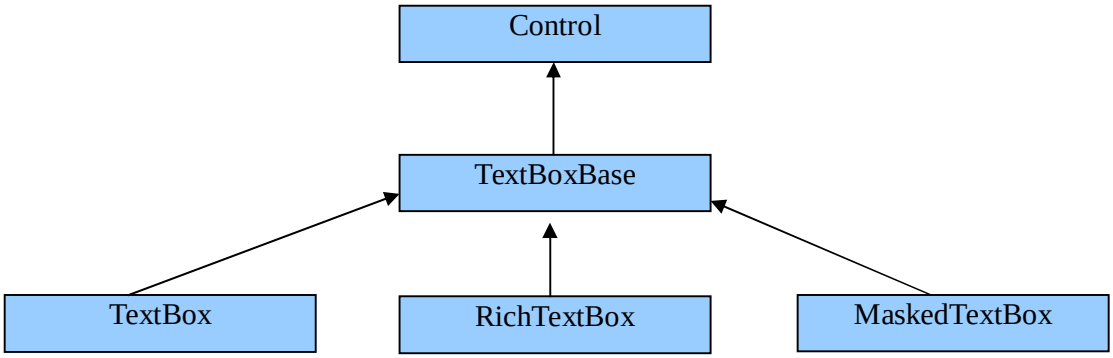
```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
  class App
  {
    public static void Main()
    {
      Application.Run(new MyForm());
    }
  }
```

```
class MyForm : Form
{
   private DragButton m_buttonOK;
   private DragButton m_buttonCancel;

   public MyForm()
   {
      this.Text = "Dragging Example";

      m_buttonOK = new DragButton();
      m_buttonOK.Text = "&Ok";

      m_buttonCancel = new DragButton();
      m_buttonCancel.Text = "&Cancel";
      m_buttonCancel.Location = new Point(100, 100);

      this.Controls.Add(m_buttonOK);
      this.Controls.Add(m_buttonCancel);
   }

}


class DragButton : Button
{
   private Point m_clickPoint;
   private bool m_flag;

   public DragButton() : base()
   {
      this.MouseDown += new MouseEventHandler(mouseDownHandler);
      this.MouseUp += new MouseEventHandler(mouseUpHandler);
      this.MouseMove += new MouseEventHandler(mouseMoveHandler);
   }

   void mouseDownHandler(object sender, MouseEventArgs e)
   {
      if (e.Button == MouseButtons.Left)
      {
         m_clickPoint = e.Location;
         m_flag = true;

      }
   }

   void mouseUpHandler(object sender, MouseEventArgs e)
   {
      if (e.Button == MouseButtons.Left)
         m_flag = false;
   }
```

```
    void mouseMoveHandler(object sender, MouseEventArgs e)
    {
        if (m_flag)
        {
            int deltaX = e.X - m_clickPoint.X;
            int deltaY = e.Y - m_clickPoint.Y;

            this.Left += deltaX;
            this.Top += deltaY;
        }
    }

  }
}
```

TextBox Control ü: Bu control (API programlama da EditBox olarak bilinir) kullanıcıdan bir edit alanı içinden bilgi almak için kullanılır. Bu control kendi içinde undo gibi karmaşık mekanizmaları içermektedir. Adeta bir editörün özelliklerine sahiptir. Control ün en yalın kullanımı kullanıcın girdiği yazının alınmazası biçimidir. Programcı control ü textbox ın default başlangıç fonksiyonu ile yaratır textbox kontrolü kontrol sınıfından gelen text property si içindeki yazıyı temsil eder. Yani biz bu property ye yazı girersek yazı textbox içinde görüntülenir. Bu property den yazı alırsak control ün içine girmiş yazıyı almış oluruz.Tıpkı Button sınıfında olduğu gibi TextBox sınıfıyla benzerlik gösteren RichTextBox ve MaskedTextBox gibi sınıflar da vardır.

Bu sınıfların ortak özellikleri TextBoxBase içinde toplanmıştır. TextBox sınıfıda TextBoxBase sınıfından türetilmiştir.




```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
  class App
  {
    public static void Main()
    {
      Application.Run(new MyForm());
    }
  }
```

```
    class MyForm : Form
    {
        private TextBox m_textBoxName;
        private Button m_buttonOK;

        public MyForm()
        {
            m_textBoxName = new TextBox();
            m_textBoxName.Location = new Point(100, 100);

            m_buttonOK = new Button();
            m_buttonOK.Text = "&Ok";
            m_buttonOK.Location = new Point(160, 140);
            m_buttonOK.Click += new EventHandler(buttonOKClickHandler);

            this.Controls.AddRange(new Control[] { m_buttonOK, m_textBoxName });



        }

        private void buttonOKClickHandler(object sender, EventArgs ea)
        {
            MessageBox.Show(m_textBoxName.Text);
        }

    }
}
```

TextBox Control ünün bool türden multiLine property elemanı control ü çok satırlı mod a geçirmek için kullanılabilir. Bu property nin default durumu false dur. Yani control tek satırlıdır.

Control ün int türden maxLength property si TextBoxBase sınıfından gelmektedir. Control e girecek karakter sayısını sınırlamakta kullanılır. Control ün bool türden readOnly property si programı Read Only mod a geçirmekte kullanılır. Bu property nin default değeri false biçimdedir. Yani property read only değildir.

ReadOnly Text Box a klavye ile yazı girilemez fakat text property si yoluyla yazı girilebilir.

Control ün ScrollBars isimli property elemanı çok satırlı durumda etkili olur. Bu property ScrollBars isimli enum türündendir. Default durumu scrollBars.none biçimindedir.

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
    class App
    {
        public static void Main()
        {
            Application.Run(new MyForm());
```

```
    }
  }


  class MyForm : Form
  {
    private TextBox m_textBoxName;
    private Button m_buttonOK;

    public MyForm()
    {
      m_textBoxName = new TextBox();
      m_textBoxName.Bounds = new Rectangle(0, 0, 100, 100);
      m_textBoxName.BackColor = Color.White;
      m_textBoxName.Multiline = true;
      m_textBoxName.ReadOnly = true;

      m_buttonOK = new Button();
      m_buttonOK.Text = "&Ok";
      m_buttonOK.Location = new Point(160, 140);
      m_buttonOK.Click += new EventHandler(buttonOKClickHandler);

      this.Controls.AddRange(new Control[] { m_buttonOK, m_textBoxName });

    }

    private void buttonOKClickHandler(object sender, EventArgs ea)
    {
      m_textBoxName.Text = "Süleyman";
    }

  }
}
```

Control ün bool türden WordWrap property si sarma yapılıp yapılmayacağını belirtir. Default durum true yani sarma yapılmasıdır. Sarma kaldırılırsa enter a basılmadıktan sonra aşağı satıra geçilmez.

Şüphesiz yatay kaydırma çubuğu ancak sarma yoksa söz konusu olabilir. Control ün Lines isimli Read/Write property si özellikle çok satırlı mod için anlamlıdır. Bu propert string dizisi türündendir. Control deki yazıyı satır satır tek hamlede elde etmek için kullanılır.

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
  class App
  {
    public static void Main()
    {
      Application.Run(new MyForm());
    }
  }
```

```
    class MyForm : Form
    {
        private TextBox m_textBoxName;
        private Button m_buttonOK;

        public MyForm()
        {
            m_textBoxName = new TextBox();
            m_textBoxName.Bounds = new Rectangle(0, 0, 100, 100);
            m_textBoxName.BackColor = Color.White;
            m_textBoxName.Multiline = true;
            m_textBoxName.ScrollBars = ScrollBars.Both;
            m_textBoxName.WordWrap = false;
            m_textBoxName.Lines = new string[] { "Ankara", "İzmir", "Adana" };

            m_buttonOK = new Button();
            m_buttonOK.Text = "&Ok";
            m_buttonOK.Location = new Point(160, 140);
            m_buttonOK.Click += new EventHandler(buttonOKClickHandler);

            this.Controls.AddRange(new Control[] { m_buttonOK, m_textBoxName });

        }

        private void buttonOKClickHandler(object sender, EventArgs ea)
        {
            foreach (string line in m_textBoxName.Lines)
                MessageBox.Show(line);
        }

    }
}
```

Control ün bool türden Modified property si Control ilk yaratıldığında false durumdadır. Her hangi bir karakter girildiğinde control tarafından true yapılır. Tipik olarak editör programlarında save işleminden sonra bu property programcı tarafından false yapılır çıkışta da kontrol edilir.

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
    class App
    {
        public static void Main()
        {
          Application.Run(new MyForm());
        }
    }
```

```
class MyForm : Form
{
    private TextBox m_textBoxName;
    private Button m_buttonOK;

    public MyForm()
    {
        this.Text = "TextBox Sample";
        this.FormClosing += new FormClosingEventHandler(MyForm_FormClosing);


        m_textBoxName = new TextBox();
        m_textBoxName.Bounds = new Rectangle(0, 0, 300, 200);
        m_textBoxName.BackColor = Color.White;
        m_textBoxName.Multiline = true;
        m_textBoxName.ScrollBars = ScrollBars.Both;
        m_textBoxName.WordWrap = false;


        m_buttonOK = new Button();
        m_buttonOK.Text = "&Ok";
        m_buttonOK.Location = new Point(160, 240);
        m_buttonOK.Click += new EventHandler(buttonOKClickHandler);

        this.Controls.AddRange(new Control[] { m_buttonOK, m_textBoxName });



    }

    void MyForm_FormClosing(object sender, FormClosingEventArgs e)
    {
        if (m_textBoxName.Modified)
        {
            DialogResult dr = MessageBox.Show("Save Changes? ", "Message",
MessageBoxButtons.YesNoCancel);
            if (dr == DialogResult.Cancel)
            {
                e.Cancel = true;
                return;
            }
            if (dr == DialogResult.Yes)
            {
                MessageBox.Show("Saving");
            }
            else
            {

            }

        }
```

```
    }

    private void buttonOKClickHandler(object sender, EventArgs ea)
    {
      foreach (string line in m_textBoxName.Lines)
        MessageBox.Show(line);
    }

  }
}
```

Control ün int türden read only TextLength property si girilen yazının uzunluğunu vermektedir. Bu tamamen Text property sinin Length ini elde etmekle Length değerini elde etmek aynı anlamdadır.

Control ün TextAlign isimli propert elemanı HorizontalAligntment isimli enum türündendir. Yatay hizalama da kullanılır. Default durum Left biçimindedir. Control ün char türden PasswordChar property si set edildiğinde control parola girme modu na geçer. Property ye girilen karakter parola girme karakteri olur.

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
  class App
  {
    public static void Main()
    {
      Application.Run(new MyForm());
    }
  }


  class MyForm : Form
  {
    private TextBox m_textBoxName;
    private Button m_buttonOK;

    public MyForm()
    {
      this.Text = "TextBox Sample";
      this.FormClosing += new FormClosingEventHandler(MyForm_FormClosing);


      m_textBoxName = new TextBox();
      m_textBoxName.Bounds = new Rectangle(0, 0, 300, 200);
      m_textBoxName.BackColor = Color.White;
      m_textBoxName.Multiline = true;
      m_textBoxName.ScrollBars = ScrollBars.Both;
      m_textBoxName.WordWrap = false;
      m_textBoxName.TextAlign = HorizontalAlignment.Center;
      m_textBoxName.PasswordChar = '*';
```

```
            m_buttonOK = new Button();
            m_buttonOK.Text = "&Ok";
            m_buttonOK.Location = new Point(160, 240);
            m_buttonOK.Click += new EventHandler(buttonOKClickHandler);

            this.Controls.AddRange(new Control[] { m_buttonOK, m_textBoxName });



        }

        void MyForm_FormClosing(object sender, FormClosingEventArgs e)
        {
            if (m_textBoxName.Modified)
            {
                DialogResult dr = MessageBox.Show("Save Changes? ", "Message",
MessageBoxButtons.YesNoCancel);
                if (dr == DialogResult.Cancel)
                {
                    e.Cancel = true;
                    return;
                }
                if (dr == DialogResult.Yes)
                {
                    MessageBox.Show("Saving");
                }
                else
                {

                }

            }
        }

        private void buttonOKClickHandler(object sender, EventArgs ea)
        {
            foreach (string line in m_textBoxName.Lines)
                MessageBox.Show(line);
        }

    }
}
```

Control deki herhangi bir bölgeyi programlama yoluyla seçmek için int türden SelectionStart ve SelectionLength property leri kullanılmaktadır. Text Box içindeki her karakterin bir indeksi vardır. (Enter tuşu için iki karakter kontrole basılmaktadır) Bu propert seçilmiş olan bölgeyi belirlemek için de kullanılabilir. Seçilmiş alandaki yazı Control ün SelectedText propert si ile elde edilebilir. Eğer hiçbir alan seçilmemişse SelectedText property si boş bir string içerir. SelectedText Read/Write property dir. Eğer bu property ye bir yazı atanırsa seçilmiş olan yazı silinir. Yerine sanki paste işlemi yapılıyormuş gibi atanan yazı yerleştirilir. Eğer hiçbir yazı seçilmemişse imlecin

bulunduğu yere insert işlemi yapılır.

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
   class App
   {
      public static void Main()
      {
        Application.Run(new MyForm());
      }
   }


   class MyForm : Form
   {
      private TextBox m_textBoxName;
      private Button m_buttonOK;

      public MyForm()
      {
         this.Text = "TextBox Sample";
         this.FormClosing += new FormClosingEventHandler(MyForm_FormClosing);


         m_textBoxName = new TextBox();
         m_textBoxName.Bounds = new Rectangle(0, 0, 300, 200);
         m_textBoxName.BackColor = Color.White;
         m_textBoxName.Multiline = true;
         m_textBoxName.ScrollBars = ScrollBars.Both;
         m_textBoxName.WordWrap = false;


         m_buttonOK = new Button();
         m_buttonOK.Text = "&Ok";
         m_buttonOK.Location = new Point(160, 240);
         m_buttonOK.Click += new EventHandler(buttonOKClickHandler);

         this.Controls.AddRange(new Control[] { m_buttonOK, m_textBoxName });



      }

      void MyForm_FormClosing(object sender, FormClosingEventArgs e)
      {
         if (m_textBoxName.Modified)
         {
            DialogResult dr = MessageBox.Show("Save Changes? ", "Message",
```

```
MessageBoxButtons.YesNoCancel);
            if (dr == DialogResult.Cancel)
            {
               e.Cancel = true;
               return;
            }
            if (dr == DialogResult.Yes)
            {
               MessageBox.Show("Saving");
            }
            else
            {

            }

         }
      }

      private void buttonOKClickHandler(object sender, EventArgs ea)
      {
         m_textBoxName.SelectedText = "kaan";
      }

   }
}
```

Tipik bir uygulama (özellikle chat programlarında görülür. Text Box ın sonuna programlama yoluyla yazı eklemektedir. Bu işlem çoğu kez yanlış yapılmaktadır. Pek çok programcı yanlışlıkla yazıyı control ün Text property sine eklemektedir. Böyle yapıldığında Text property sine yeni bir yazı girildiği için kaydırma çubuğu başta görünür. Kaydırma çubuğunun sonda kalması için Text property sine atama yerine insert uygulanmalıdır. Yani imleç sona çekilip paste işlemi yapılmalıdır. Paste işlemi daha önce açıklandığı gibi SelectedText property si ile yapılabilir. İmlecin sona çekilmesi SelctionStart property sine uygun değerin atanmasıyla sağlanabilir. SelectionStart property si aynı zamanda seçme işleminin yanısıra imleci de taşımaktadır.

Yanlış uygulama:

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
   class App
   {
      public static void Main()
      {
        Application.Run(new MyForm());
      }
   }


   class MyForm : Form
   {
```

```
    private TextBox m_textBoxName;
    private Button m_buttonOK;
    private int m_count;

    public MyForm()
    {
        this.Text = "TextBox Sample";
        this.FormClosing += new FormClosingEventHandler(MyForm_FormClosing);


        m_textBoxName = new TextBox();
        m_textBoxName.Bounds = new Rectangle(0, 0, 300, 200);
        m_textBoxName.BackColor = Color.White;
        m_textBoxName.Multiline = true;
        m_textBoxName.ScrollBars = ScrollBars.Both;
        m_textBoxName.WordWrap = false;


        m_buttonOK = new Button();
        m_buttonOK.Text = "&Ok";
        m_buttonOK.Location = new Point(160, 240);
        m_buttonOK.Click += new EventHandler(buttonOKClickHandler);

        this.Controls.AddRange(new Control[] { m_buttonOK, m_textBoxName });



    }

    void MyForm_FormClosing(object sender, FormClosingEventArgs e)
    {
        if (m_textBoxName.Modified)
        {
            DialogResult dr = MessageBox.Show("Save Changes? ", "Message",
MessageBoxButtons.YesNoCancel);
            if (dr == DialogResult.Cancel)
            {
                e.Cancel = true;
                return;
            }
            if (dr == DialogResult.Yes)
            {
                MessageBox.Show("Saving");
            }
            else
            {

            }

        }
    }
```

```
    private void buttonOKClickHandler(object sender, EventArgs ea)
    {
      m_textBoxName.Text += m_count.ToString() + "\r\n";
      ++m_count;
    }

  }
}
```

Doğru Uygulama:

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
  class App
  {
    public static void Main()
    {
      Application.Run(new MyForm());
    }
  }


  class MyForm : Form
  {
    private TextBox m_textBoxName;
    private Button m_buttonOK;
    private int m_count;

    public MyForm()
    {
      this.Text = "TextBox Sample";
      this.FormClosing += new FormClosingEventHandler(MyForm_FormClosing);


      m_textBoxName = new TextBox();
      m_textBoxName.Bounds = new Rectangle(0, 0, 300, 200);
      m_textBoxName.BackColor = Color.White;
      m_textBoxName.Multiline = true;
      m_textBoxName.ScrollBars = ScrollBars.Both;
      m_textBoxName.WordWrap = false;


      m_buttonOK = new Button();
      m_buttonOK.Text = "&Ok";
      m_buttonOK.Location = new Point(160, 240);
      m_buttonOK.Click += new EventHandler(buttonOKClickHandler);

      this.Controls.AddRange(new Control[] { m_buttonOK, m_textBoxName });
```

```
        }

        void MyForm_FormClosing(object sender, FormClosingEventArgs e)
        {
            if (m_textBoxName.Modified)
            {
                DialogResult dr = MessageBox.Show("Save Changes? ", "Message",
MessageBoxButtons.YesNoCancel);
                if (dr == DialogResult.Cancel)
                {
                    e.Cancel = true;
                    return;
                }
                if (dr == DialogResult.Yes)
                {
                    MessageBox.Show("Saving");
                }
                else
                {

                }

            }
        }

        private void buttonOKClickHandler(object sender, EventArgs ea)
        {
            m_textBoxName.SelectionStart = m_textBoxName.TextLength;

            m_textBoxName.SelectedText =  "\r\n" + m_count.ToString() ;
            ++m_count;
        }

    }
}
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
    class App
    {
        public static void Main()
        {
            Application.Run(new MyForm());
        }
    }


    class MyForm : Form
```

```
{
    private TextBox m_textBoxChat;
    private TextBox m_textBoxUser;
    private Button m_buttonOK;


    public MyForm()
    {
        this.Text = "TextBox Sample";
        this.FormClosing += new FormClosingEventHandler(MyForm_FormClosing);
        this.FormBorderStyle = FormBorderStyle.Fixed3D;
        this.Size = new Size(600, 450);
        this.MaximizeBox = false;

        m_textBoxChat = new TextBox();
        m_textBoxChat.Bounds = new Rectangle(10, 10, this.ClientSize.Width - 20, 300);
        m_textBoxChat.BackColor = Color.White;
        m_textBoxChat.Multiline = true;
        m_textBoxChat.ScrollBars = ScrollBars.Vertical;
        m_textBoxChat.WordWrap = false;
        m_textBoxChat.ReadOnly = true;
        m_textBoxChat.TabStop = false;

        m_textBoxUser = new TextBox();
        m_textBoxUser.Bounds = new Rectangle(10, 330, this.ClientSize.Width - 20, 50);
        m_textBoxUser.BackColor = Color.White;

        m_buttonOK = new Button();
        m_buttonOK.Text = "&Ok";
        m_buttonOK.Location = new Point(250, 380);
        m_buttonOK.Click += new EventHandler(buttonOKClickHandler);
        m_textBoxUser.KeyDown += new KeyEventHandler(MyForm_KeyDown);


        this.Controls.AddRange(new Control[] { m_textBoxUser, m_textBoxChat, m_buttonOK });

    }

    void MyForm_KeyDown(object sender, KeyEventArgs e)
    {
        if (e.KeyCode == Keys.Enter)
            buttonOKClickHandler(null, null);

    }

    void MyForm_FormClosing(object sender, FormClosingEventArgs e)
    {
        if (m_textBoxChat.Modified)
        {
            DialogResult dr = MessageBox.Show("Save Changes? ", "Message",
MessageBoxButtons.YesNoCancel);
            if (dr == DialogResult.Cancel)
```

```
            {
                e.Cancel = true;
                return;
            }
            if (dr == DialogResult.Yes)
            {
                MessageBox.Show("Saving");
            }
            else
            {

            }

        }
    }

    private void buttonOKClickHandler(object sender, EventArgs ea)
    {
        m_textBoxChat.SelectionStart = m_textBoxChat.TextLength;
        m_textBoxChat.SelectedText = "<Kaan>: " + m_textBoxUser.Text + "\r\n";
        m_textBoxUser.Text = string.Empty;

        m_textBoxUser.Focus();
    }
  }
}
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
  class App
  {
    public static void Main()
    {
      Application.Run(new MyForm());
    }
  }


  class MyForm : Form
  {
    private TextBox m_textBoxChat;
    private TextBox m_textBoxUser;
    private Button m_buttonOK;


    public MyForm()
    {
      this.Text = "TextBox Sample";
      this.FormClosing += new FormClosingEventHandler(MyForm_FormClosing);
```

```
        this.FormBorderStyle = FormBorderStyle.Fixed3D;
        this.Size = new Size(600, 450);
        this.MaximizeBox = false;

        m_textBoxChat = new TextBox();
        m_textBoxChat.Bounds = new Rectangle(10, 10, this.ClientSize.Width - 20, 300);
        m_textBoxChat.BackColor = Color.White;
        m_textBoxChat.Multiline = true;
        m_textBoxChat.ScrollBars = ScrollBars.Vertical;
        m_textBoxChat.WordWrap = false;
        m_textBoxChat.ReadOnly = true;
        m_textBoxChat.TabStop = false;

        m_textBoxUser = new TextBox();
        m_textBoxUser.Bounds = new Rectangle(10, 330, this.ClientSize.Width - 20, 50);
        m_textBoxUser.BackColor = Color.White;
        m_textBoxUser.KeyDown += new KeyEventHandler(MyForm_KeyDown);

        m_buttonOK = new Button();
        m_buttonOK.Text = "&Ok";
        m_buttonOK.Location = new Point(250, 380);
        m_buttonOK.Click += new EventHandler(buttonOKClickHandler);
        m_buttonOK.Enabled = false;




        this.Controls.AddRange(new Control[] { m_textBoxUser, m_textBoxChat, m_buttonOK });

    }

    void MyForm_KeyDown(object sender, KeyEventArgs e)
    {

        if (e.KeyCode == Keys.Enter)
            buttonOKClickHandler(null, null);
        else
            m_buttonOK.Enabled = true;
    }

    void MyForm_FormClosing(object sender, FormClosingEventArgs e)
    {
        if (m_textBoxChat.Modified)
        {
            DialogResult dr = MessageBox.Show("Save Changes? ", "Message",
MessageBoxButtons.YesNoCancel);
            if (dr == DialogResult.Cancel)
            {
                e.Cancel = true;
                return;
            }
```

```
            if (dr == DialogResult.Yes)
            {
               MessageBox.Show("Saving");
            }
            else
            {

            }

         }
      }

      private void buttonOKClickHandler(object sender, EventArgs ea)
      {
         if (m_textBoxUser.Text == string.Empty)
         {
            return;
         }

         m_textBoxChat.SelectionStart = m_textBoxChat.TextLength;
         m_textBoxChat.SelectedText = "<Kaan>: " + m_textBoxUser.Text + "\r\n";
         m_textBoxUser.Text = string.Empty;
         m_buttonOK.Enabled = false;

         m_textBoxUser.Focus();
      }
   }
}
```

TextBox control ünün tam olarak clipboard ve undo desteği vardır. Control ün TextBoxBase sınıfından gelen Copy, Cut, Paste fonksiyonları seçili alan üzerinde clipboard işlemi yapar. Şüphesiz kullanıcı bu işleri klavyeyi kullanarak da yapabilir. Fakat bu fonksiyonlar bu işlerin programlama yoluyla yapılmasını sağlamaktadıt.

TextBox sınıfının kendine özgü(yani Control sınıfından gelmeyen) event elemanları da vardır. Bu event elemanlarının bazıları TextBoxBase sınıfından gelmiştir. Örneğin sınıfın TextBoxBase sınıfından gelen ModifiedChanged event elemanı TextBox a ilkez karakter girildiğinde tetiklenir. Yani Control ün Modified property si false yapıldıktan sonra Control a karakter girildiğinde modified property si değiştiği için bu event tetiklenir. TextBox sınıfını diğer event elemanları dökümanlardan incelenmelidir.

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
   class App
   {
      public static void Main()
      {
        Application.Run(new MyForm());
```

```
    }
}


class MyForm : Form
{
    private TextBox m_textBoxChat;
    private TextBox m_textBoxUser;
    private Button m_buttonOK;


    public MyForm()
    {
        this.Text = "TextBox Sample";
        this.FormClosing += new FormClosingEventHandler(MyForm_FormClosing);
        this.FormBorderStyle = FormBorderStyle.Fixed3D;
        this.Size = new Size(600, 450);
        this.MaximizeBox = false;

        m_textBoxChat = new TextBox();
        m_textBoxChat.Bounds = new Rectangle(10, 10, this.ClientSize.Width - 20, 300);
        m_textBoxChat.BackColor = Color.White;
        m_textBoxChat.Multiline = true;
        m_textBoxChat.ScrollBars = ScrollBars.Vertical;
        m_textBoxChat.WordWrap = false;
        m_textBoxChat.ReadOnly = true;
        m_textBoxChat.TabStop = false;

        m_textBoxUser = new TextBox();
        m_textBoxUser.Bounds = new Rectangle(10, 330, this.ClientSize.Width-20, 50);
        m_textBoxUser.BackColor = Color.White;
        m_textBoxUser.KeyDown += new KeyEventHandler(MyForm_KeyDown);
        m_textBoxUser.ModifiedChanged += new
EventHandler(m_textBoxUser_ModifiedChanged);

        m_buttonOK = new Button();
        m_buttonOK.Text = "&Ok";
        m_buttonOK.Location = new Point(250, 380);
        m_buttonOK.Click += new EventHandler(buttonOKClickHandler);
        m_buttonOK.Enabled = false;

        this.Controls.AddRange(new Control[] { m_textBoxUser, m_textBoxChat, m_buttonOK});

    }

    void m_textBoxUser_ModifiedChanged(object sender, EventArgs e)
    {
        m_buttonOK.Enabled = true;


    }
```

```
        void MyForm_KeyDown(object sender, KeyEventArgs e)
        {

            if (e.KeyCode == Keys.Enter)
            {
                buttonOKClickHandler(null, null);
            }
            else
                m_buttonOK.Enabled = true;
        }

        void MyForm_FormClosing(object sender, FormClosingEventArgs e)
        {
            if (m_textBoxChat.Modified)
            {
                DialogResult dr = MessageBox.Show("Save Changes? ", "Message",
MessageBoxButtons.YesNoCancel);
                if (dr == DialogResult.Cancel)
                {
                    e.Cancel = true;
                    return;
                }
                if (dr == DialogResult.Yes)
                {
                    MessageBox.Show("Saving");
                }
                else
                {

                }

            }
        }

        private void buttonOKClickHandler(object sender, EventArgs ea)
        {
            if (m_textBoxUser.Text == string.Empty)
            {
                return;
            }

            m_textBoxChat.SelectionStart = m_textBoxChat.TextLength;
            m_textBoxChat.SelectedText = "<Kaan>: " + m_textBoxUser.Text + "\r\n";
            m_textBoxUser.Text = string.Empty;
            m_buttonOK.Enabled = false;

            m_textBoxUser.Focus();
        }
    }
}
```

Anahtar Notlar: Internetten "windows forms faq" listesi incelenmelidir. Bu dökumanlarla paralel olacak biçimde Windows 2005 C# Recipes, C# Cookcook ve C# programmers Cookbook benzer temalı kitaplardır. Ayrıca Microsoft un .net le ilgili bazıları yoğun bazıları az yoğun olan haber gurupları vardır.
Bu haber grupları microsoft.public.dotnet ön ekiyle başlamaktadır ve Google tüm USNET guruplarında olduğu gibi bu guruplarda arama ve post işlemlerine izin vermektedir. Örneğin "microsoft.public.dotnet.languages.csharp" genel konular için "microsoft.public.dotnet.faq" çok sorulan sorular için, form programlama için"microsoft.public.dotnet.framework.windowsforms" ve bunların altındaki guruplara bakılabilir.

Label Control: Amacı sadece bir yazıyı göstermek olan basit alt pencerelere API dünyasında "static" kontrol .net te ise "Label" denilmektedir. Label bir alt penceredir. Tek görevi yazı göstermektir. Pencere pasif durumdadır. Bu nedenle Fare mesajları aşağıya iletilir. Aynı zamanda Control ün zemin rengi transparan olduğu için kullanıcı bir pencere görüntüsüyle karşılaşmaz. Tabi şüphesiz diğer kontrollerde olduğu gibi kenar çizgileri ve zemin rengi istenildiği gibi ayarlanabilir.

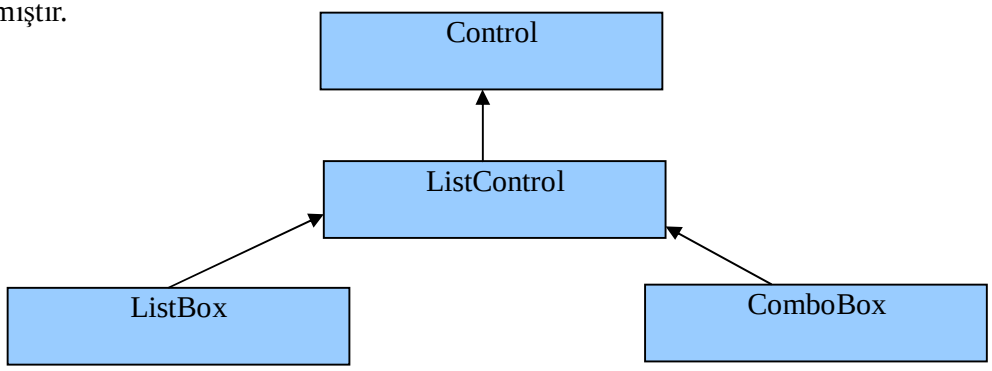
LabelControl ü Label sınıfıyla temsil edilmiştir. Label sınıfı doğrudan control sınıfından türetilmiştir. Programcı label türünden nesneyi default başlangıç fonksiyonu ile yaratır. Sonra Control ün text property sine görüntgülenecek yazıyı girer.

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
   class App
   {
      public static void Main()
      {
        Application.Run(new MyForm());
      }
   }


   class MyForm : Form
   {
      private TextBox m_textBoxChat;
      private TextBox m_textBoxUser;
      private Button m_buttonOK;
      private Label m_label;


      public MyForm()
      {
         this.Text = "TextBox Sample";
         this.FormClosing += new FormClosingEventHandler(MyForm_FormClosing);
         this.FormBorderStyle = FormBorderStyle.Fixed3D;
         this.Size = new Size(600, 450);
         this.MaximizeBox = false;

         m_textBoxChat = new TextBox();
         m_textBoxChat.Bounds = new Rectangle(10, 10, this.ClientSize.Width - 20, 300);
```

```
        m_textBoxChat.BackColor = Color.White;
        m_textBoxChat.Multiline = true;
        m_textBoxChat.ScrollBars = ScrollBars.Vertical;
        m_textBoxChat.WordWrap = false;
        m_textBoxChat.ReadOnly = true;
        m_textBoxChat.TabStop = false;

        m_textBoxUser = new TextBox();
        m_textBoxUser.Bounds = new Rectangle(10, 330, this.ClientSize.Width-20, 50);
        m_textBoxUser.BackColor = Color.White;
        m_textBoxUser.KeyDown += new KeyEventHandler(MyForm_KeyDown);
        m_textBoxUser.ModifiedChanged += new
EventHandler(m_textBoxUser_ModifiedChanged);

        m_buttonOK = new Button();
        m_buttonOK.Text = "&Ok";
        m_buttonOK.Location = new Point(250, 380);
        m_buttonOK.Click += new EventHandler(buttonOKClickHandler);
        m_buttonOK.Enabled = false;

        m_label = new Label();
        m_label.Text = "Enter Text: ";
        m_label.Location = new Point(10, 315);

        this.Controls.AddRange(new Control[] { m_textBoxUser, m_textBoxChat, m_buttonOK,
m_label});

    }

    void m_textBoxUser_ModifiedChanged(object sender, EventArgs e)
    {
        m_buttonOK.Enabled = true;


    }

    void MyForm_KeyDown(object sender, KeyEventArgs e)
    {

        if (e.KeyCode == Keys.Enter)
        {
            buttonOKClickHandler(null, null);
        }
        else
            m_buttonOK.Enabled = true;
    }

    void MyForm_FormClosing(object sender, FormClosingEventArgs e)
    {
        if (m_textBoxChat.Modified)
        {
            DialogResult dr = MessageBox.Show("Save Changes? ", "Message", MessageBoxButtons.YesNoCancel);
```

```
            if (dr == DialogResult.Cancel)
            {
                e.Cancel = true;
                return;
            }
            if (dr == DialogResult.Yes)
            {
                MessageBox.Show("Saving");
            }
            else
            {

            }

        }
    }

    private void buttonOKClickHandler(object sender, EventArgs ea)
    {
        if (m_textBoxUser.Text == string.Empty)
        {
            return;
        }

        m_textBoxChat.SelectionStart = m_textBoxChat.TextLength;
        m_textBoxChat.SelectedText = "<Kaan>: " + m_textBoxUser.Text + "\r\n";
        m_textBoxUser.Text = string.Empty;
        m_buttonOK.Enabled = false;


        m_textBoxUser.Focus();
    }
  }
}
```

ListBox Control ü: Listeleme kutuları bir gurup elemanı listelemek bir ya da daha fazlasının seçilmesine olanak vermektedir. Listeleme kutuları ile seçimli listeleme kutuları(combo boxes)bir birlerine benzeyen kontrollerdir. Bu nedenle bunların ortak elemanları ListControl isimli bir sınıfta toplanmıştır.



ListBox Control tipik olarak şöyle kullanılır:

1. Control ListBox sınıfının default başlangıç fonksiyonuyla yaratılır. (Text Property sinin bu control de bir önemi yoktur)
2. Sınıfın Items isimli property elemanı ListBox.ObjectCollection isimli bir collection sınıf türündendir. Bu collection sınıf IList arayüzünü desteklemektedir. Bu sınıf adeta ArrayList gibi object tutan bir sınıftır. ListBox Control ü bu collection elemana eklenen her nesne için sanal ToString fonksiyonunu çağırır ve oradan elde ettiği yazıyı control e yerleştirir.
3. ListBox sınıfının çeşitli property ve event elemanları aşağıda ele alınacak olan faydalı işlere yol açmaktadır.

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
   class App
   {
      public static void Main()
      {
        Application.Run(new MyForm());
      }
   }


   class MyForm : Form
   {
      private ListBox m_listBox;

      public MyForm()
      {
         this.Text = "Sample ListBox";
         this.Size = new Size(500, 500);

         m_listBox = new ListBox();
         m_listBox.Bounds = new Rectangle(100, 50, 200, 300);
         for (int i = 0; i < 100; ++i)
            m_listBox.Items.Add(new Sample(i));

         this.Controls.AddRange(new Control[] { m_listBox });

      }
      //...
   }

   class Sample
   {
      private int m_a;

      public Sample(int a)
      {
         m_a = a;
```

```
        }

        public override string ToString()
        {
            return "Number: " + m_a.ToString();
        }

        public int A
        {
            get { return m_a; }
        }
    }

}
```

Görüldüğü gibi ListBox control ü aynı zamandabir collection gibidir. Control ün Bool türden Sorted isimli property elemanı eklenen elemanları sıralı bir biçimde göstermekte kullanılır.

Anahtar Notlar: C# da bir dizi refansının başka bir dizi referansına atanabilmesi için her iki dizi referansınında referans türlerine ilişkin olması ve kaynak dizi türünün hedef dizi türünden türetilmiş olması gerekmektedir.

```
Object [] o;
string []s;
int [] x;
//...
o=s; //geçerli
o= x; //geçersiz;
s=x;// geçersiz.
```

Int türü, Double türüne doğrudan atanabilir. Int dizi referansı double dizi referansına doğrudan atanabilemez.

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
    class App
    {
        public static void Main()
        {
            Application.Run(new MyForm());
        }
    }


    class MyForm : Form
    {
        private ListBox m_listBox;

        public MyForm()
        {
```

```
        this.Text = "Sample ListBox";
        this.Size = new Size(500, 500);

        m_listBox = new ListBox();
        m_listBox.Bounds = new Rectangle(100, 50, 200, 300);
        m_listBox.Sorted = true;

        string[] cities = {"Ankara", "Eskişehir", "İzmir", "Bursa", "Adana", "İstanbul", "Mersin",
                           "Bartın", "Yalova", "Sivas", "Bilecik"};


        m_listBox.Items.AddRange(cities);

        this.Controls.AddRange(new Control[] { m_listBox });

    }
    //...
}

class Sample
{
    private int m_a;

    public Sample(int a)
    {
        m_a = a;
    }

    public override string ToString()
    {
        return "Number: " + m_a.ToString();
    }

    public int A
    {
        get { return m_a; }
    }
}

}
```

Bu property yalnızca Items.Add yapıldığında etkili olmaktadır. Insert yapıldığında geçerli olmaz. Yani insert işleminde bu property ye bakılmamaktadır. Yalnızca Add' e bakılmaktadır.

ListBox sınıfının Bool türden HorizontalScrollBar property si yatay kaydırma çubuğunu görüntülemkte kullanılır. Bu property nin default durumu false dur. (Düşey Scroll Bar her zaman görüntülenmektedir.

Bir elemana çft tıklandığında o elemana ilişkin bir işlemin başlatılması istenebilir. Bunun için ListBox sınıfının control sınıfından gelen DoubleClick elemanı kullanılır. DoubleClick event elemanı EventHandler isimli delege türündendir. Bu event oluştuğunda seçilmiş olan eleman verilmemektedir. Seçilmiş olan eleman herhangi bir zaman object türünden SelectedItem property si

ile elde edilebilir. Eğer hiçbir eleman seçili değilse SelectedItem elemanı Null referans verir.

ListBox Control ü tek seçimli ya da çok seçimli olabilir. Default durum tek seçimli olmasıdır. Sınıfın SelectionMode isimli property si SelectionMode isimli enum türündendir. Bu enum türünün elemanları şunlardır.
None: Bu durumda her hangi bir eleman seçilemez.
One: Bu durumda bir tane eleman seçilebilir. Default durumdur.
MultiSimple: Birden fazla eleman seçilebilir.
MultiExtended: Hem ayrık hemde ardışıl seçim yapılabilir.

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
   class App
   {
      public static void Main()
      {
        Application.Run(new MyForm());
      }
   }


   class MyForm : Form
   {
      private ListBox m_listBox;
      private Button m_buttonOK;

      public MyForm()
      {
         this.Text = "Sample ListBox";
         this.Size = new Size(500, 500);

         m_listBox = new ListBox();
         m_listBox.Bounds = new Rectangle(100, 50, 200, 300);
         m_listBox.Sorted = true;
         m_listBox.DoubleClick += new EventHandler(m_listBox_DoubleClick);
         m_listBox.SelectionMode = SelectionMode.MultiSimple;

         string[] cities = {"Ankara", "Eskişehir", "İzmir", "Bursa", "Adana", "İstanbul", "Mersin",
                            "Bartın", "Yalova", "Sivas", "Bilecik"};


         m_listBox.Items.AddRange(cities);

         m_buttonOK = new Button();
         m_buttonOK.Text = "&Ok";
         m_buttonOK.Location = new Point(350, 200);
         m_buttonOK.Click += new EventHandler(m_buttonOK_Click);
```

```
            this.Controls.AddRange(new Control[] { m_listBox, m_buttonOK });

        }

        void m_buttonOK_Click(object sender, EventArgs e)
        {
            MessageBox.Show((string)m_listBox.SelectedItem);
        }

        void m_listBox_DoubleClick(object sender, EventArgs e)
        {
            MessageBox.Show((string)m_listBox.SelectedItem);
        }


        //...
    }


}
```

ListBox sınıfının selectedIndices isimli property si ListBox.SelectedIndexCollection isimli collection sınıf türündendir. Eğer çoklu seçime izin verilmişse seçilmiş elemanın indexleri SelctedIndices property si ile alınabilir. Eğer tek eleman seçilebiliyorsa seçilmiş elemanın index i int türden selctedIndex property si ile alınabilir. Benzer biçimde SlectedItems isimli property si de ListBox.SelectedObjectCollection isimli collection türündendir. Tüm seçilmiş elemanları verir.

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
    class App
    {
        public static void Main()
        {
          Application.Run(new MyForm());
        }
    }


    class MyForm : Form
    {
        private ListBox m_listBox;
        private Button m_buttonOK;

        public MyForm()
        {
            this.Text = "Sample ListBox";
            this.Size = new Size(500, 500);
```

```
            m_listBox = new ListBox();
            m_listBox.Bounds = new Rectangle(100, 50, 200, 300);
            m_listBox.Sorted = true;
            m_listBox.DoubleClick += new EventHandler(m_listBox_DoubleClick);
            m_listBox.SelectionMode = SelectionMode.MultiExtended;

            string[] cities = {"Ankara", "Eskişehir", "İzmir", "Bursa", "Adana", "İstanbul", "Mersin",
                               "Bartın", "Yalova", "Sivas", "Bilecik"};


            m_listBox.Items.AddRange(cities);

            m_buttonOK = new Button();
            m_buttonOK.Text = "&Ok";
            m_buttonOK.Location = new Point(350, 200);
            m_buttonOK.Click += new EventHandler(m_buttonOK_Click);


            this.Controls.AddRange(new Control[] { m_listBox, m_buttonOK });

        }

        void m_buttonOK_Click(object sender, EventArgs e)
        {
            string text = null;

            foreach (string s in m_listBox.SelectedItems)
                text += s + "  ";

            MessageBox.Show(text);
        }

        void m_listBox_DoubleClick(object sender, EventArgs e)
        {
            MessageBox.Show((string)m_listBox.SelectedItem);
        }


        //...
    }

}
```

Görüldüğü gibi SelectedIndex ve SelectedIndices propertyleri ile SelectedItem property si Read/Write propertylerdir. Yani biz bu sayede programlama yoluyla istenilen bir elemanı seçili bir duruma getirebiliriz.

ListBox sınıfının Bool türden MultiColumn property si listeleme kutusunu çok sütunlu yapmakta kullanılır. Default durum tek sütünlu olmasıdır.

```
using System;
using System.Windows.Forms;
using System.Drawing;
```

```
namespace CSD
{
   class App
   {
      public static void Main()
      {
        Application.Run(new MyForm());
      }
   }


   class MyForm : Form
   {
      private ListBox m_listBox;
      private Button m_buttonOK;

      public MyForm()
      {
         this.Text = "Sample ListBox";
         this.Size = new Size(500, 500);

         m_listBox = new ListBox();
         m_listBox.Bounds = new Rectangle(100, 50, 200, 100);
         m_listBox.Sorted = true;
         m_listBox.DoubleClick += new EventHandler(m_listBox_DoubleClick);
         m_listBox.SelectionMode = SelectionMode.MultiExtended;

         string[] cities = {"Ankara", "Eskişehir", "İzmir", "Bursa", "Adana", "İstanbul", "Mersin",
                            "Bartın", "Yalova", "Sivas", "Bilecik"};


         m_listBox.Items.AddRange(cities);
         m_listBox.SelectedItem = "Eskişehir";
         m_listBox.MultiColumn = true;

         m_buttonOK = new Button();
         m_buttonOK.Text = "&Ok";
         m_buttonOK.Location = new Point(350, 200);
         m_buttonOK.Click += new EventHandler(m_buttonOK_Click);


         this.Controls.AddRange(new Control[] { m_listBox, m_buttonOK });

      }

      void m_buttonOK_Click(object sender, EventArgs e)
      {
         string text = null;

         foreach (string s in m_listBox.SelectedItems)
            text += s + "  ";
```

```
            MessageBox.Show(text);
        }

        void m_listBox_DoubleClick(object sender, EventArgs e)
        {
            MessageBox.Show((string)m_listBox.SelectedItem);
        }


        //...
    }


}
```

ListBox sınıfının yukarıdan gelmeyen yani sınıfın kendisine özgü olan SelectedIndexChanged isimli event elemanı seçili olan eleman değiştiğinde tetiklenir. Bu event eleman EventHandler isimli delege türündendir.

```
using System;
using System.Windows.Forms;
using System.Drawing;
using System.Collections;

namespace CSD
{
    class App
    {
        public static void Main()
        {
            Application.Run(new MyForm());
        }
    }


    class MyForm : Form
    {
        private ListBox m_listBox;
        private ListBox m_listBoxOther;

        public MyForm()
        {
            this.Text = "Sample ListBox";
            this.Size = new Size(800, 500);

            m_listBox = new ListBox();
            m_listBox.Bounds = new Rectangle(30, 50, 200, 300);
            m_listBox.Sorted = true;
            m_listBox.DoubleClick += new EventHandler(m_listBox_DoubleClick);
            m_listBox.SelectionMode = SelectionMode.MultiExtended;

            m_listBox.Items.Add(new City("İstanbul", "Beşiktaş", "Şişli", "Üsküdar", "Kadiköy",
```

```
            "Kartal", "Pendik", "Tuzla", "Adalar"));

        m_listBox.Items.Add(new City("Adana", "Ceyhan", "Seyhan", "Kozan"));
        m_listBox.Items.Add(new City("Eskişehir", "Mihalıççık", "Seyitgazi", "Alpu"));
        m_listBox.Items.Add(new City("Antalya", "Finike", "Seril", "Serik", "Alanya",
"Manavgat"));


        m_listBox.SelectedItem = "Eskişehir";
        m_listBox.MultiColumn = true;
        m_listBox.SelectedIndexChanged += new
EventHandler(m_listBox_SelectedIndexChanged);




        m_listBoxOther = new ListBox();
        m_listBoxOther.Bounds= new Rectangle(350, 50, 200, 300);


        this.Controls.AddRange(new Control[] { m_listBox, m_listBoxOther});

    }

    void m_listBox_SelectedIndexChanged(object sender, EventArgs e)
    {
        City city = (City)m_listBox.SelectedItem;

        m_listBoxOther.Items.Clear();
        foreach (string s in city.Towns)
            m_listBoxOther.Items.Add(s);
    }

    void m_listBox_DoubleClick(object sender, EventArgs e)
    {
        MessageBox.Show((string)m_listBox.SelectedItem);
    }


    //...
}


class City
{
    private string m_cityName;
    private ArrayList m_towns;

    public City(string cityName, params string[] tokens)
    {
        m_cityName = cityName;
        m_towns = new ArrayList(tokens);
```

```
        }

        public string CityName
        {
            get { return m_cityName; }
        }

        public ArrayList Towns
        {
            get { return m_towns; }
        }

        public override string ToString()
        {
            return m_cityName;
        }


        //...
    }

}
```

Seçimli Listeleme Kutuları: Seçimli listeleme kutuları aslında listeleme kutusuna bir textBox ya da Label kontrolünün iliştirilmesi ile elde edilmiştir. Seçilen eleman listeleme kutusunun başlık kısmında görüntülenir. Şüphesiz bu kontrol de çoklu seçim özelliği yoktur. Control kapalıyken az yer kapladığı için ListBox kontrolüne göre oldukça sık tercih edilmektedir.

ComboBox kullanımı ListBox kullanımına çok benzer.Control ComboBox sınıfıyla temsil edilmiştir. Control ListControl sınıfınıdan türetilmiştir.

ComboBox kullanımı büyük ölçüde ListBox kullanımına benzemektedir. Burada farklılıkları üzerinde durulacaktır.

```
using System;
using System.Windows.Forms;
using System.Drawing;
using System.Collections;

namespace CSD
{
    class App
    {
        public static void Main()
        {
            Application.Run(new MyForm());
        }
    }


    class MyForm : Form
```

```
{
    private ComboBox m_comboBox;
    private ListBox m_listBoxOther;

    public MyForm()
    {
        this.Text = "Sample ListBox";
        this.Size = new Size(800, 500);

        m_comboBox = new ComboBox();
        m_comboBox.Bounds = new Rectangle(30, 50, 200, 300);
        m_comboBox.Sorted = true;
        m_comboBox.DoubleClick += new EventHandler(m_listBox_DoubleClick);


        m_comboBox.Items.Add(new City("İstanbul", "Beşiktaş", "Şişli", "Üsküdar", "Kadiköy",
              "Kartal", "Pendik", "Tuzla", "Adalar"));

        m_comboBox.Items.Add(new City("Adana", "Ceyhan", "Seyhan", "Kozan"));
        m_comboBox.Items.Add(new City("Eskişehir", "Mihalıççık", "Seyitgazi", "Alpu"));
        m_comboBox.Items.Add(new City("Antalya", "Finike", "Seril", "Serik", "Alanya",
"Manavgat"));


        m_comboBox.SelectedItem = "Eskişehir";
        m_comboBox.SelectedIndexChanged += new
EventHandler(m_listBox_SelectedIndexChanged);

        m_listBoxOther = new ListBox();
        m_listBoxOther.Bounds= new Rectangle(350, 50, 200, 300);


        this.Controls.AddRange(new Control[] { m_comboBox, m_listBoxOther});

    }

    void m_listBox_SelectedIndexChanged(object sender, EventArgs e)
    {
        City city = (City)m_comboBox.SelectedItem;

        m_listBoxOther.Items.Clear();
        foreach (string s in city.Towns)
            m_listBoxOther.Items.Add(s);
    }

    void m_listBox_DoubleClick(object sender, EventArgs e)
    {
        MessageBox.Show((string)m_comboBox.SelectedItem);
    }


    //...
```

```
    }


    class City
    {
        private string m_cityName;
        private ArrayList m_towns;

        public City(string cityName, params string[] tokens)
        {
            m_cityName = cityName;
            m_towns = new ArrayList(tokens);
        }

        public string CityName
        {
            get { return m_cityName; }
        }

        public ArrayList Towns
        {
            get { return m_towns; }
        }

        public override string ToString()
        {
            return m_cityName;
        }


        //...
    }

}
```

ComboBox sınıfının DropDownStyle isimli property elemanı ComboBoxStyle isimli enum türündendir. Bu eneum türünün elemanları ve anlamları şöyledir.
Simple: Bu modda combobox sürekli açıktır ve kapatılamaz.

```
using System;
using System.Windows.Forms;
using System.Drawing;
using System.Collections;

namespace CSD
{
    class App
    {
        public static void Main()
        {
          Application.Run(new MyForm());
        }
```

```
}


class MyForm : Form
{
    private ComboBox m_comboBox;
    private ListBox m_listBoxOther;

    public MyForm()
    {
        this.Text = "Sample ListBox";
        this.Size = new Size(800, 500);

        m_comboBox = new ComboBox();
        m_comboBox.Bounds = new Rectangle(30, 50, 200, 300);
        m_comboBox.Sorted = true;
        m_comboBox.DoubleClick += new EventHandler(m_listBox_DoubleClick);


        m_comboBox.Items.Add(new City("İstanbul", "Beşiktaş", "Şişli", "Üsküdar", "Kadiköy",
              "Kartal", "Pendik", "Tuzla", "Adalar"));

        m_comboBox.Items.Add(new City("Adana", "Ceyhan", "Seyhan", "Kozan"));
        m_comboBox.Items.Add(new City("Eskişehir", "Mihalıççık", "Seyitgazi", "Alpu"));
        m_comboBox.Items.Add(new City("Antalya", "Finike", "Seril", "Serik", "Alanya",
"Manavgat"));
        m_comboBox.SelectedItem = "Eskişehir";
        m_comboBox.SelectedIndexChanged += new
EventHandler(m_listBox_SelectedIndexChanged);
        m_comboBox.DropDownStyle = ComboBoxStyle.Simple;

        m_listBoxOther = new ListBox();
        m_listBoxOther.Bounds= new Rectangle(350, 50, 200, 300);


        this.Controls.AddRange(new Control[] { m_comboBox, m_listBoxOther});

    }

    void m_listBox_SelectedIndexChanged(object sender, EventArgs e)
    {
        City city = (City)m_comboBox.SelectedItem;

        m_listBoxOther.Items.Clear();
        foreach (string s in city.Towns)
            m_listBoxOther.Items.Add(s);
    }

    void m_listBox_DoubleClick(object sender, EventArgs e)
    {
        MessageBox.Show((string)m_comboBox.SelectedItem);
    }
```

```
        //...
    }


    class City
    {
        private string m_cityName;
        private ArrayList m_towns;

        public City(string cityName, params string[] tokens)
        {
            m_cityName = cityName;
            m_towns = new ArrayList(tokens);
        }

        public string CityName
        {
            get { return m_cityName; }
        }

        public ArrayList Towns
        {
            get { return m_towns; }
        }

        public override string ToString()
        {
            return m_cityName;
        }


        //...
    }

}
```

DropDown: Bu default durumdur. Programcı edit alanına yazı girebilir. Yani girilecek yazı Combobox içindeki yazılardan bir olmak zorunda değildir.

DropDownList: Bu modda ComboBox control ünün edit alanı read only dir. Dolayısıyla kesinlikle listede olan elemanlardan biri seçilir.

```
using System;
using System.Windows.Forms;
using System.Drawing;
using System.Collections;

namespace CSD
{
```

```
class App
{
    public static void Main()
    {
      Application.Run(new MyForm());
    }
}


class MyForm : Form
{
    private ComboBox m_comboBox;
    private ListBox m_listBoxOther;

    public MyForm()
    {
        this.Text = "Sample ListBox";
        this.Size = new Size(800, 500);

        m_comboBox = new ComboBox();
        m_comboBox.Bounds = new Rectangle(30, 50, 200, 300);
        m_comboBox.Sorted = true;
        m_comboBox.DoubleClick += new EventHandler(m_listBox_DoubleClick);


        m_comboBox.Items.Add(new City("İstanbul", "Beşiktaş", "Şişli", "Üsküdar", "Kadiköy",
              "Kartal", "Pendik", "Tuzla", "Adalar"));

        m_comboBox.Items.Add(new City("Adana", "Ceyhan", "Seyhan", "Kozan"));
        m_comboBox.Items.Add(new City("Eskişehir", "Mihalıççık", "Seyitgazi", "Alpu"));
        m_comboBox.Items.Add(new City("Antalya", "Finike", "Seril", "Serik", "Alanya",
"Manavgat"));
        m_comboBox.SelectedItem = "Eskişehir";
        m_comboBox.SelectedIndexChanged += new
EventHandler(m_listBox_SelectedIndexChanged);
        m_comboBox.DropDownStyle = ComboBoxStyle.DropDownList;


        m_listBoxOther = new ListBox();
        m_listBoxOther.Bounds= new Rectangle(350, 50, 200, 300);


        this.Controls.AddRange(new Control[] { m_comboBox, m_listBoxOther});

    }

    void m_listBox_SelectedIndexChanged(object sender, EventArgs e)
    {
        City city = (City)m_comboBox.SelectedItem;

        m_listBoxOther.Items.Clear();
        foreach (string s in city.Towns)
```

```
            m_listBoxOther.Items.Add(s);
        }

        void m_listBox_DoubleClick(object sender, EventArgs e)
        {
            MessageBox.Show((string)m_comboBox.SelectedItem);
        }


        //...
    }


    class City
    {
        private string m_cityName;
        private ArrayList m_towns;

        public City(string cityName, params string[] tokens)
        {
            m_cityName = cityName;
            m_towns = new ArrayList(tokens);
        }

        public string CityName
        {
            get { return m_cityName; }
        }

        public ArrayList Towns
        {
            get { return m_towns; }
        }

        public override string ToString()
        {
            return m_cityName;
        }


        //...
    }

}
```

DropDown modda edit alanı içine girilen yazı seçilmiş olan anlamına gelmez. Yani edit alanındaki yazı seçilmiş elemanın yazısı olmak zorunda değildir. Bu nedenle programcı bu modda seçilmiş elemanı almak yerine kontrol ün edit alanındaki yazının alınması daha anlamlı olabilir. Control ün edit alanındaki yazı ComboBox kontrolünün Text property si ile elde edilebilir.

```
using System;
using System.Windows.Forms;
```

```
using System.Drawing;
using System.Collections;

namespace CSD
{
   class App
   {
      public static void Main()
      {
        Application.Run(new MyForm());
      }
   }


   class MyForm : Form
   {
      private ComboBox m_comboBox;
      private ListBox m_listBoxOther;

      public MyForm()
      {
         this.Text = "Sample ListBox";
         this.Size = new Size(800, 500);
         this.Click += new EventHandler(MyForm_Click);

         m_comboBox = new ComboBox();
         m_comboBox.Bounds = new Rectangle(30, 50, 200, 300);
         m_comboBox.Sorted = true;
         m_comboBox.DoubleClick += new EventHandler(m_listBox_DoubleClick);


         m_comboBox.Items.Add(new City("İstanbul", "Beşiktaş", "Şişli", "Üsküdar", "Kadiköy",
               "Kartal", "Pendik", "Tuzla", "Adalar"));

         m_comboBox.Items.Add(new City("Adana", "Ceyhan", "Seyhan", "Kozan"));
         m_comboBox.Items.Add(new City("Eskişehir", "Mihalıççık", "Seyitgazi", "Alpu"));
         m_comboBox.Items.Add(new City("Antalya", "Finike", "Seril", "Serik", "Alanya",
"Manavgat"));
         m_comboBox.SelectedItem = "Eskişehir";
         m_comboBox.SelectedIndexChanged += new
EventHandler(m_listBox_SelectedIndexChanged);
         m_comboBox.DropDownStyle = ComboBoxStyle.DropDown;


         m_listBoxOther = new ListBox();
         m_listBoxOther.Bounds= new Rectangle(350, 50, 200, 300);


         this.Controls.AddRange(new Control[] { m_comboBox, m_listBoxOther});

      }
```

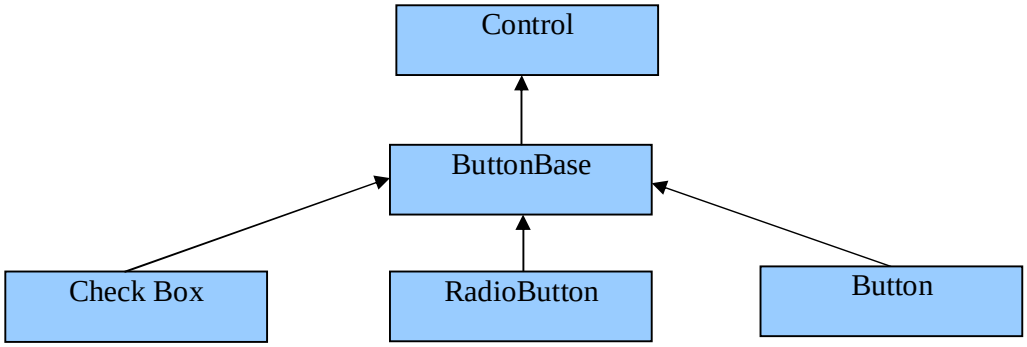
```
    void MyForm_Click(object sender, EventArgs e)
    {
        MessageBox.Show(m_comboBox.Text);
    }

    void m_listBox_SelectedIndexChanged(object sender, EventArgs e)
    {
        City city = (City)m_comboBox.SelectedItem;

        m_listBoxOther.Items.Clear();
        foreach (string s in city.Towns)
            m_listBoxOther.Items.Add(s);
    }

    void m_listBox_DoubleClick(object sender, EventArgs e)
    {
        MessageBox.Show((string)m_comboBox.SelectedItem);
    }


    //...
}


class City
{
    private string m_cityName;
    private ArrayList m_towns;

    public City(string cityName, params string[] tokens)
    {
        m_cityName = cityName;
        m_towns = new ArrayList(tokens);
    }

    public string CityName
    {
        get { return m_cityName; }
    }

    public ArrayList Towns
    {
        get { return m_towns; }
    }

    public override string ToString()
    {
        return m_cityName;
    }


    //...
```

```
    }

}
```

ComboBox sınıfının diğer elemanları MSDN dökumanlarından okunabilir.

Seçenek Kutuları: Check Boxes bir küçük kutucuk ve yanında bir yazıdan oluşmaktadır. Bir özelliğin bir bütüne dahil edilip edilmeyeceğini belirlemektedir. Kontrol den elde edilecek tek bilgi control ün çarpılanıp çarpılanmadığıdır.

Seçenek kutuları CheckBox sınıfıyla temsil edilmiştir. Daha önce de bahsedildiği gibi bu sınıf ButtonBase sınıfından türetilmiştir.



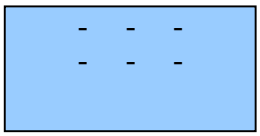
Seçenek Kutuları şöyle kullanılır:

1. CheckBox türünden bir sınıf nesnesi sınıfın default başlangıç fonksiyonu ile yaratılır.
2. Control ün Bool türden AutoCheck property si seçenek kutusunu otomatik ve ya manuel yapmak için kullanılır. Otomatik modda çarpılanıp çarpının kaldırılması(uncheck) tıklandığında kontrol tarafından yapılır. Manuel biçim modda ise control e tıklandığında event oluşur. Çarpılama programcı tarafından yapılır. Default durum otomatik moddur.
3. Contol ün Bool türden Read/Write Checked property si Control ün çarpılanıp çarpılanmadığını ve programlama yoluyla çarpılamak için kullanılır.

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
   class App
   {
      public static void Main()
      {
        Application.Run(new MyForm());
      }
   }


   class MyForm : Form
   {
      private CheckBox m_checkBox;
      private Button m_buttonOK;
```

```
    public MyForm()
    {
        m_checkBox = new CheckBox();
        m_checkBox.Text = "Backup alınsın mı?";
        m_checkBox.Bounds = new Rectangle(100, 100, 200, 100);

        m_buttonOK = new Button();
        m_buttonOK.Text = "&Ok";
        m_buttonOK.Location= new Point(150, 200);
        m_buttonOK.Click += new EventHandler(m_buttonOK_Click);


        m_checkBox.Parent = this;
        m_buttonOK.Parent = this;
    }

    void m_buttonOK_Click(object sender, EventArgs e)
    {
        MessageBox.Show(m_checkBox.Checked ? "Checked" : "Unchecked");
    }
    //...
  }
}
```

Yazının kutuya göre hizalanması sınıfın CheckAlign property si ile sağlanabilir. Bu property ContentAlingment isimli enum türündendir. Bu enum türünün aşağıdaki biçimde hizalama sağlar.



TestAlingment property si ise hizalamayı yazıya göre yapar.

Radyo Düğmeleri (Radio Button): Radio düğmeleri genellikle bir gurup oluşturacak biçimde birden fazla kullanılır. Bir laç seçenekten yalnızca birinin seçileceği durumlarda tercih edilmektedir. Radio düğmesi bir küçük yuvarlakçık ve bir de yazıdan oluşur. Örneğin 5 seçenekli bir radio düğmesi gurubu oluşturacaksak 5 ayrı radio düğmesi nesnesi yaratmalıyız.

Üst pencereleri aynı olan tüm radio düğmeleri aynı gurup içindedir. Bu nedenle doğrudan bir form üzerine birden fazla gurup radio düğmesi yerleştirilemez. Bunu sağlamak için GroupBox denilen özel alt pencereler kullanılır. Bu durumda programcı tipik olarak form üzerine birden fazla groupBox penceresi oluşturur. Radio düğmelerini de bu GroupBox penceresinin içinde yaratır.

Bir grup radio düğmesi şöyle yaratılır:

1. Radio düğmeleri RadioButton sınıfıyla temsil edilmiştir. Programcı sınıfın default başlangıç fonksiyonunu kullanarak birden fazla RadioButton nesnesi yaratır.
2. Nesnelerin Text propertyleri uygun yazılarla set edilir.
3. Radio düğmeleri de otomatik ve manuel olabilmektedir. Default durum yine otomatik moddur.
4. Hangi radio düğmesinin çarpılandığını anlamak için tek tek radio düğmelerinin Checked property sine bakılmalıdır.

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
   class App
   {
      public static void Main()
      {
        Application.Run(new MyForm());
      }
   }


   class MyForm : Form
   {
      private RadioButton m_radio1, m_radio2, m_radio3, m_radio4;
      private Button m_buttonOK;

      public MyForm()
      {
         m_radio1 = new RadioButton();
         m_radio1.Text = "A";
         m_radio1.Location = new Point(50, 50);

         m_radio2 = new RadioButton();
         m_radio2.Text = "B";
         m_radio2.Location = new Point(50, 80);

         m_radio3 = new RadioButton();
         m_radio3.Text = "C";
         m_radio3.Location = new Point(50, 110);

         m_radio4 = new RadioButton();
         m_radio4.Text = "D";
         m_radio4.Location = new Point(50, 140);

         m_buttonOK = new Button();
         m_buttonOK.Text = "&Ok";
         m_buttonOK.Location = new Point(160, 95);
         m_buttonOK.Click += new EventHandler(m_buttonOK_Click);

         this.Controls.AddRange(new Control[] { m_radio1, m_radio2, m_radio3, m_radio4,
m_buttonOK });
      }

      void m_buttonOK_Click(object sender, EventArgs e)
      {
         foreach (Control c in this.Controls)
         {
            RadioButton rb;
```

```
            rb = c as RadioButton;
            if (rb != null)
            {
               if (rb.Checked)
               {
                  MessageBox.Show(rb.Text + " Checked");
                  break;
               }
            }

         }
      }
      //...
   }
}
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
   class App
   {
      public static void Main()
      {
        Application.Run(new MyForm());
      }
   }


   class MyForm : Form
   {
      private GroupBox m_group1, m_group2;
      private RadioButton m_radio1, m_radio2, m_radio3, m_radio4;
      private RadioButton m_radio5, m_radio6, m_radio7, m_radio8;
      private Button m_buttonOK;

      public MyForm()
      {
         m_group1 = new GroupBox();
         m_group1.Text = "Group1";
         m_group1.Bounds = new Rectangle(30, 30, 150, 160);


         m_group2 = new GroupBox();
         m_group2.Text = "Group2";
         m_group2.Bounds = new Rectangle(30, 200, 220, 220);


         m_radio1 = new RadioButton();
         m_radio1.Text = "A";
```

```
            m_radio1.Location = new Point(50, 50);

            m_radio2 = new RadioButton();
            m_radio2.Text = "B";
            m_radio2.Location = new Point(50, 80);

            m_radio3 = new RadioButton();
            m_radio3.Text = "C";
            m_radio3.Location = new Point(50, 110);

            m_radio4 = new RadioButton();
            m_radio4.Text = "D";
            m_radio4.Location = new Point(50, 140);

            m_group1.Controls.AddRange(new Control[] { m_radio1, m_radio2, m_radio3, m_radio4,
});

            m_radio5 = new RadioButton();
            m_radio5.Text = "X";
            m_radio5.Bounds = new Rectangle(0, 100, 50, 50);

            m_radio6 = new RadioButton();
            m_radio6.Text = "Y";
            m_radio6.Bounds = new Rectangle(50, 100, 50, 50);

            m_radio7 = new RadioButton();
            m_radio7.Text = "Z";
            m_radio7.Bounds = new Rectangle(100, 100, 50, 50);

            m_radio8 = new RadioButton();
            m_radio8.Text = "K";
            m_radio8.Bounds = new Rectangle(150, 100, 50, 50);

            m_group2.Controls.AddRange(new Control[] { m_radio5, m_radio6, m_radio7, m_radio8,
});

            m_buttonOK = new Button();
            m_buttonOK.Text = "&Ok";
            m_buttonOK.Location = new Point(160, 95);
            m_buttonOK.Click += new EventHandler(m_buttonOK_Click);

            this.Controls.AddRange(new Control[] {m_group1, m_group2, m_buttonOK });
        }

        void m_buttonOK_Click(object sender, EventArgs e)
        {
            foreach (Control c in this.Controls)
            {
                RadioButton rb;

                rb = c as RadioButton;
                if (rb != null)
```

```
            {
               if (rb.Checked)
               {
                  MessageBox.Show(rb.Text + " Checked");
                  break;
               }
            }


         }
      }
      //...
   }
}
```

Kontrollerin Demirlenmesi: Pencereleri genişletip daralttığımızda kontrollerin belirli kenarlara aynı uzaklıkta kalmasını isteyebiliriz. Bunu gerçekleştirmek için ilk akla gelen yöntem resize mesajında location property sini kullanarak control ü tekrar konumlandırmaktır.

Control sınıfının Anchor isimli property elemanı AnchorStyles isimli enum türündendir.

Anahtar Notlar: Bir enum türüne FlagsAttribute özniteliği atanmışsa bu enum türünün elemanları bir düzeyinde or işlemine sokulabilir. Bu durumdan birden fazla özelliğin dahil edilmesi gibi bir anlam çıkmaktadır.

Anchor property sine atanan değer hangi kenarlara demirleme yapılacağını belirtir.

Örneğin: *m_buttonOK.Anchor = AnchorSteyles.Right | AnchorSteyles.Bottom;*

Bu property nin default değeri *AnchorSteyles.Top | AnchorSteyles.Left;*

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
   class App
   {
      public static void Main()
      {
        Application.Run(new MyForm());
      }
   }
   class MyForm : Form
   {
      private Button m_buttonOK;

      public MyForm()
      {
         this.Text = "Anchor Sample";
         this.Size = new Size(400, 400);

         m_buttonOK = new Button();
```

```
            m_buttonOK.Text = "&Ok";
            m_buttonOK.Location = new Point((this.ClientSize.Width - m_buttonOK.Width) / 2,
(this.ClientSize.Height - m_buttonOK.Height) / 2);
            m_buttonOK.Anchor = AnchorStyles.Top | AnchorStyles.Left | AnchorStyles.Bottom |
AnchorStyles.Right;

            this.Controls.Add(m_buttonOK);
        }
        //...
    }
}
```

Bu property control sınıfının OnResize sanal fonksiyonunda ele alınmaktadır. Yani biz üst pencerenin OnResize sanal fonksiyonunu override edersek Control sınıfının OnResize fonksiyonunun çağrılmasını sağlamalıyız.

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
    class App
    {
        public static void Main()
        {
          Application.Run(new MyForm());
        }
    }


    class MyForm : Form
    {
        private Button m_buttonOK;

        public MyForm()
        {
            this.Text = "Anchor Sample";
            this.Size = new Size(200, 200);

            m_buttonOK = new Button();
            m_buttonOK.Text = "&Ok";
            m_buttonOK.Location = new Point((this.ClientSize.Width - m_buttonOK.Width) / 2,
(this.ClientSize.Height - m_buttonOK.Height) / 2);
            m_buttonOK.Anchor = AnchorStyles.Top | AnchorStyles.Left | AnchorStyles.Bottom |
AnchorStyles.Right;

            this.Controls.Add(m_buttonOK);
        }

        protected override void OnResize(EventArgs e)
        {
            base.OnResize(e);
        }
```

```
        //...
    }
}
```

Controllerin Yuvalanması: Bir alt pencere üst pencerenin herhangi bir kenarına yuvalanırsa bu kenarı tamamen kaplayacak duruma gelir. Örneğin menüler, araç çubukları üst kenara yuvalanmış kontrollerdir. Benzer biçimde durum pencereleri(States Bar) alt kenara yuvalanmış pencerelerdir.

Control sınıfının Dock isimli property elemanı DockStyle isimli enum türündendir. DockStyle enum türünün, None, Top, Bottom, Left, Rigth ve Fill isimli elemanları vardır.

Bir kontrol ün üst pencerenini çalışma alanını kaplar hale getirilmesi en pratik olarak DosckStyle.Fill propertsiile yapılabilir.

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
    class App
    {
        public static void Main()
        {
          Application.Run(new MyForm());
        }
    }


    class MyForm : Form
    {
        private Button m_buttonOK;

        public MyForm()
        {
            this.Text = "Anchor Sample";
            this.Size = new Size(200, 200);

            m_buttonOK = new Button();
            m_buttonOK.Text = "&Ok";
            m_buttonOK.Location = new Point((this.ClientSize.Width - m_buttonOK.Width) / 2,
(this.ClientSize.Height - m_buttonOK.Height) / 2);
            m_buttonOK.Dock = DockStyle.Fill;

            this.Controls.Add(m_buttonOK);
        }

        protected override void OnResize(EventArgs e)
        {
            base.OnResize(e);
        }
        //...
    }
```

```
}
```

Dock özelliği de control sınıfının OnResize fonksiyonunda sağlanmaktadır. Dolaysıyla Anchor property si için söylenenler bu property içinde söylenebilir.

Diyalog Pencereleri: Diyalog pencereleri her zaman üst pencerelerin üzerinde görüntülenen Owned penceredir. Diyalog pencereleri Modal ve Modeless olmak üzere iki bölüme ayrılır.
Modal diyalog penceresi en çok kullanılan tipik diyalog penceresidir.
Modal diyalog penceresi açıldığında kullanıcı arka plandaki üst pencere ile etkileşebilemez.Başka bir deyişle arka plandaki pencereye gönderilen mesajlar doğrudan atılmaktadır.

Halbuki Modeless pencerelerde diyalog penceresi açıkken kullanıcı arka plandaki üst pencereyle de etkileşebilir.

MessageBox tipik bir modal pencereyken Find and Replace tipik bir Modeless penceredir.

GUI tabanlı uygulamalarda diyalog pencereleri çok sık kullanılmaktadır.

Aslında bir ana pencerenin diyalog penceresinden tek farkı Owned pencere olmamasıdır. Biz Form sınıfından yarattığımız ana pencereyi owned pencere durumuna getirirsek diyalog penceresi yaratmış oluruz. Bu işlem .net tw oldukça pratik bir biçimde yapılmaktadır. Bir form penceresi Show fonksiyonuyla değil de ShowDialog fonksiyonuyla görüntülenirse Modal diyalog penceresi haline gelir. Anımsanacağı gibi Form sınıfının Show fonksiyonunu çağırmakla Visible property sine true atamak aynı etkiyi yaratmaktadır. O halde bir form nesnesi show fonksiyonu ile görüntülenirse normal bir ana pencere, ShowDialaog fonksiyonuyla görüntülenirse Modal diyalog penceresi olur.

O halde Modal diyalog penceresi oluşturmak için şunlar yapılmaldır.

1. Form sınıfından bir sınıf türetilir ve new operatörüyle bu sınıf türünden bir nesne yaratılır.
2. Form nesnesi ShowDialaog fonksiyonuyla görüntülenir.

Bir diyalog penceresinde en azından Okey ve Cancel biçiminde iki tuş bulunur.

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
   class App
   {
      public static void Main()
      {
        Application.Run(new MyForm());
      }
   }


   class MyForm : Form
   {
      private Button m_buttonOK;

      public MyForm()
```

```
        {
            m_buttonOK = new Button();
            m_buttonOK.Text = "&Ok";
            m_buttonOK.Location = new Point(100, 100);
            m_buttonOK.Click += new EventHandler(m_buttonOK_Click);


            m_buttonOK.Parent = this;
        }

        void m_buttonOK_Click(object sender, EventArgs e)
        {
            TestForm tf = new TestForm();
            tf.ShowDialog();


        }
        //...
    }

    class TestForm : Form
    {
        //...
    }
}
using System;
using System.Windows.Forms;
using System.Drawing;

namespace CSD
{
    class App
    {
        public static void Main()
        {
          Application.Run(new MyForm());
        }
    }


    class MyForm : Form
    {
        private Button m_buttonOK;

        public MyForm()
        {
            this.Text = "Sample Dialog";
            this.Size = new Size(600, 400);

            m_buttonOK = new Button();
            m_buttonOK.Text = "&Ok";
            m_buttonOK.Location = new Point(100, 100);
```

```
            m_buttonOK.Click += new EventHandler(m_buttonOK_Click);


            m_buttonOK.Parent = this;
        }

        void m_buttonOK_Click(object sender, EventArgs e)
        {
            TestForm tf = new TestForm();

            tf.Show();
        }
        //...
    }

    class TestForm : Form
    {
        private Button m_buttonOK;
        private Button m_buttonCancel;

        public TestForm()
        {
            this.Text = "Dialog Test";
            this.Size = new Size(400, 200);

            m_buttonOK = new Button();
            m_buttonOK.Text = "&Ok";
            m_buttonOK.Location = new Point(180, 130);

            m_buttonCancel = new Button();
            m_buttonCancel.Text = "&Cancel";
            m_buttonCancel.Location = new Point(280, 130);


            this.Controls.AddRange(new Control[] { m_buttonOK, m_buttonCancel });

        }
    }
}
```

Diyalog penceresinin ilk görünüş yerini ayarlamak için yine form sınıfının yine startPosition property si kullanılır.

Modal diyalog penceresini kapatmak için en pratik yöntem Form sınıfının DialogResult türünden DialogResult isimli property sine değer atamaktır. Bu property ye değer atandığında dialog penceresi otomatik kapatılır. ShowDialog fonksiyonu atanan bu değerle geri döner. Örneğin Dialog penceresindeki Cancel tuşuna click yapıldığında dialog penceresini şöyle kapatabiliriz.

```
this.DialogResult=DialogResult.Cancel;
using System;
using System.Windows.Forms;
using System.Drawing;
```

```
namespace CSD
{
   class App
   {
      public static void Main()
      {
        Application.Run(new MyForm());
      }
   }


   class MyForm : Form
   {
      private Button m_buttonOK;

      public MyForm()
      {
         this.Text = "Sample Dialog";
         this.Size = new Size(600, 400);

         m_buttonOK = new Button();
         m_buttonOK.Text = "&Ok";
         m_buttonOK.Location = new Point(100, 100);
         m_buttonOK.Click += new EventHandler(m_buttonOK_Click);


         m_buttonOK.Parent = this;
      }

      void m_buttonOK_Click(object sender, EventArgs e)
      {
         TestForm tf = new TestForm();

         DialogResult dr = tf.ShowDialog();

         if (dr == DialogResult.Cancel)
            MessageBox.Show("Cancel ile çıkıldı");
      }
      //...
   }

   class TestForm : Form
   {
      private Button m_buttonOK;
      private Button m_buttonCancel;

      public TestForm()
      {
         this.Text = "Dialog Test";
         this.Size = new Size(400, 200);
         this.FormBorderStyle = FormBorderStyle.FixedDialog;
```

```
            this.MaximizeBox = false;
            this.StartPosition = FormStartPosition.CenterParent;

            m_buttonOK = new Button();
            m_buttonOK.Text = "&Ok";
            m_buttonOK.Location = new Point(180, 130);

            m_buttonCancel = new Button();
            m_buttonCancel.Text = "&Cancel";
            m_buttonCancel.Location = new Point(280, 130);
            m_buttonCancel.Click += new EventHandler(m_buttonCancel_Click);


            this.Controls.AddRange(new Control[] { m_buttonOK, m_buttonCancel });

        }

        void m_buttonCancel_Click(object sender, EventArgs e)
        {
            this.DialogResult = DialogResult.Cancel;
        }
    }
}
```

ShowDialog fonksiyonu çağrıldığında artık akış ShowDialaog fonksiyonunun içindedir. Ancak diyalog penceresi kapandığında fonksiyon geri döner.

DialogResult property sine değer atandığında bir flag(bayrak) set edilir. Diyalog penceresinin kapatılması akış mesaj döngüsüne geri döndüğünde yapılır. Aslında dialogResult property sine değer atandıktan sonra hemen Close fonksiyonuyla da pencere kapatılabilir.

Örneğin:

```
this.DialogResult = DialogResult.OK;
this.Close();
```

Fakat bu gereksizdir.

Modal Diyalog Pencerelerinde Oluşturulan Bilgilerin Elde Edilmesi: Bir diayalog penceresi kapatıldığında programcı kullanıcının diyalog penceresinde oluşturuduğu bilgileri elde etmek ister.

Bir pencerenin kapatılmış olması o pencere nesnesinin yok edilmiş olduğu anlamına gelmez. Pencere nesnesinin yok edilmesi çöp toplayıcı tarafından yapılmaktadır. Örneğin bir diyalog penceresi üzerine bir textBox yerleştirmiş olalım.

```
class MyDialog:Form
{
        private TextBox m_textBoxName;
        //...
}
```

Şimdi biz bu diyalog penceresini aşağıdaki gibi açmış olalım.

```
MyDialog md = new MyDialog();
md.ShowDialog();
```

Diyalog penceresi kapatıldığında diyalog penceresi fiziksel olarak yok olur. Dolayısıyla pencere üzerindeki kontrollerde yok edilmiştir. Ancak örnekteki md nesnesi ve onun içindeki

m_textBoxName nesnesi yaşamaya devam etmektedir. Yani örneğin yapabilsek *md.m_textBoxName .Text* ifadesiyle yok edilmiş olan TextBox kontrolünün içineki yazıyı elde edebiliriz. Fakat tipik olarak bu kontroller MyDialog sınıfının private bölümünde olduğu için biz bu erişimi yapamayacağız. **İşte önerilen tipik yöntem şudur:**

1. Diyalog penceresinden elde edilecek her bilgi için get ve set bölümlerine sahip public property yazılır.

```
class MyDialog:Form
{
        private TextBox m_textBoxName;
        //...
        public string Name
        {
                get{return m_textBoxName;}
                set{m_textBoxName = value;}
        }
}
```

2. Diyalog penceresi ShowDialaog ile açılmadan önce bu kontrollere ilk değerleri atanabilir. Pencere kapatıldıktan sonra kontrollerde oluşan değerler alınabilir. **Örneğin:**

```
MyDialog md = new MyDialog();
md.Name = "Kaan Aslan";
//...
md.ShowDialog();

MessageBox(md.Name);
```

Visual Shirbazının Kullanılması: Yeni bir proje yaratılırken template olarak bazı seçenekler bizim için hazır bir başlangıç kodu hazırlamaktadır. Aynı zamanda bu sihirbaz seçenekleri form editörün kullanılmasına da olanak sağlamaktadır. En temel sihirbaz şablonu File/New/project menüsünden template olarak Windows Form Appilication seçilerek elde edilir. Bu şablon seçeneği ile proje yaratıldığında sihirbaz şu dosyaları oluşturmaktadır.

Program.cs: Bu dosya da program isimli br sınıf bildirilmiş ve içine main fonksiyonu yerleştirilmiştir. Main fonksiyonu içerisinde klasik olarak Application.Run çalıştırılmıştır.

Form1.cs: Bu dosyada form sınıfından türetilmiş olan Form1 isimli sınıf tanımlanmıştır. Form1 sınıfı partial olarak bildirilmiştir. Bu sınıfın diğer parçası Form1.Designer.cs içerisindendir.

Anahtar Notlar: Framework 2005 ile birlikte sınıflar için partial bildirimi de eklenmiştir. Aynı isim alanında aynı sınıf birden fazla partial belirleyicisi ile bildirilirse bu sınıf derleyici tarafından birleştirilmektedir. Partial anahtar sözcüğü sınıfın her parçası için bulundurulmak zorundadır.

Sihirbazın(Wizard) yazdığı Form1 sınıfın başlangıç fonksiyonunda InitializeComponent isiml ibir fonksiyon çağrılmıştır. InitializeComponent fonksiyonu Form1.Designer.cs içinde tanımlanmıştır. Form editör kullanılarak fare hareketleriyle yapılan işlemler kod a dönüştürülerek InitializeComponent içine yazılmaktadır. Programcı manuel ekleme yapacaksa eklemeleri InitializeComponent içine değil başlangıç fonksiyonunun içinde yapmalıdır.

Form1.Designer.cs: Bu dosya da Form1 sınıfının diğer parçası bulundurulmaktadır. InitializeComponent fonksiyonu buradadır ve sihirbaz doğrudan bu dosya üzerine kod eklemektedir. Ayrıca bu dosyada Form1 Sınıfına bir Dispose kalıbı da eklenmiştir.

AssemlyInfo.cs: Bu dosyada Assemly düzeyinde öz nitelik bilgileri bulunmaktadır.

Form1.resx: Bu dosya Form1 isimli Form a yönelik xml tarzı kaynak bildirimlerini bulundurmaktadır. Kaynak konusu ileride ele alınacaktır.

Resources.resx: Her Form için ayrı bir kaynak dosyası bulundurulabilir. Fakata Resources.resx dosyası global kaynak dosyasıdır. Bu dosyada xml tabanlı kaynakları içerir.

Resources.Designer.cs: Bu dosya global kaynak dosyasındaki türlü(typed) kaynakları bildirmektedir.

Sihirbazın oluşturduğu projede temel bazı dll lere referans edilmiştir. Aslında bu referans edilen dll lerin bir kısmına gerek yoktur. Sihirbazın ürettiği kod üzerinde hemen programcı tarafından hemen bazı küçük değişiklikleri yapmalıdır. Öncelikle anapencereyi temsil eden sınıfın ismi değiştirilmelidir. Solution Explorer de Form1.cs dosyası ismi değiştirilmek istendiğinde sihirbaz kodlarda bu sınıfın ismini de değiştirmektedir.

Ayrıca programcı isim alanı isimini de değiştirmek isteyebilir. Visual studio default olarak herşeyi proje ismiyle belirtilen isim alanına yerleştirmektedir. Maalesef isim alanı değiştirmenin henüz pratik bir yolu yoktur. Fakat tipik olarak Find and Replace da Entire Solution seçeneği seçilerek tüm isimler değiştirilir.

Proje seçeneklerinden default namespace ayrıca değiştirilmelidir. Buradaki değişiklik bildirilmiş olana isimalanlarının ismini değiştirmez. Fakat daha sonra oluşturulacak isim alanlarını değiştirebilir.

Form Editörün Kullanılması: Sihirbazla çalışmanın en önemli faydası Form Editördür. Form Editör yalnızca GUI işlemleri için değil diğer işlemle için de kullanılabilen işlevsel bir araçtır. Programcı Form Editörde herhangi bir ögenin üzerine gelip bağlan menüsünden properties elemanını seçerse o ögeninin görsel olarak değiştirlebilecek elemanları görüntülenebilir. Properties içinde sınıfın property elemanları ve event elemanları görüntülenmekte ve bunlar üzerinde görsel değişiklikler yapılabilmektedir.

Control sınıfılarının ve diğer sınıfların bütün public propertyleri menüde yer almaktadır. Ve bunlar hemen menüde değiştirilebilir. Form Editör yapılan değişikliği kod a yansıtır.

Properties menüsünde event kısmına geçildiğinde ilgili sınıfın tüm event elemanları listelenmektedir. Listedeki bir event elemanına double click yapılırsa Form Editör ilgili event elemanına += ile event elemanına fonksiyon ekleyen kodu InitializeComponent içine ekler ve ilgili fonksiyonu da içi boş olarak tanımlar. Ayrıca aynı delege türünden olan event elemanlarında yazılmış olan tüm fonksiiyonlarda listelenmektedir. Böylece farklı event elemanları için aynı fonksiyonların verilmesi mümkün olabilir.

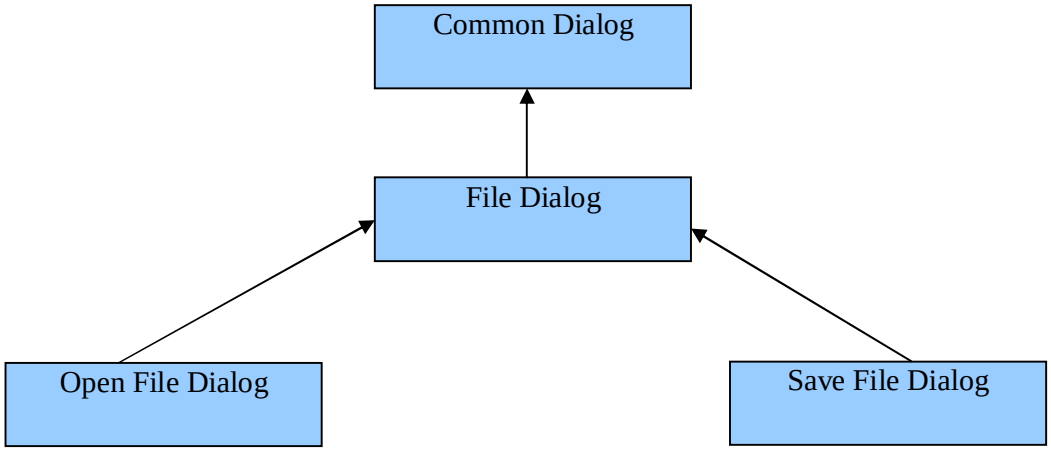
Form Editörün en önemli özelliği "ToolBox" dan seçerek kontrol yerleştirilebilmesidir. Form Editör bunlar için gerekli kodları oluşturmaktadır. Controller kolaylıkla hizalanmakta propertyleri ve event elemanları kolaylıkla set edilebilmektedir. Form editörde bazı elemanların üzerine double click yapıldığında en çok kullanılan event set edilir. Örneğin bir düğmenin üzerine double click yaptığımızda düğmenin en çok kullanılan event i click olduğu için click ecent i set edilir. Bazı controller forma yerleştirildiğinde control ün sağ üst köşesinde küçük bir ok çıkar bu ok a tıklandığında en çok kullanılan ögelerle karşılaştırılır.

Form Editöre yerleştirilen her bir kontrolere Form Editör otomotik olarak Control ismi ve bir sayıdan oluşan bir isim vermektedir. Programcının da bunu düzeltmesi yerinde olacaktır. İsimlerin otomotik değiştirilmesi için control ün main name propertysinin değiştirilmesi yeterlidir.

Anahtar Notlar: Visual Studio IDE sinde private bir veri elemanın üzerine gelinip farenin sağ tuşuyla bağlan menüsünden refactor Encapsulet Field seçilirse otomatik olarak o veri elemanı için temel property yazılır.

File Seçme Diyalog Penceresi: Windows da Dosya açma, saklama, renk seçme, bul ve değiştir, printer gibi diyalog pencereleri standart bir biçimde bulundurulmaktadır. . Nette bu diyalog pencereleri CommonDialog sınıfından türetilmiş sınıflar la temsil edilmektedir.

Bir dosyayı açmak üzere dosyayı seçmek için OpenFileDialog saklamak için SaveFileDialog pencereleri kullanılır. Bu pencereler aynı isimli sınıflarla temsil edilmiştir.



OpenFileDialog ve SaveFileDialog pencerelerinden istediğimiz yegane şey kullanıcının seçtiği dosyanın isimin bize verilmesidir. Bu diyalog pencereleri isminin aksine seçilen dosyayı açmazlar yalnızca bize kullanıcının seçtiği dosya ismini veriler. Dosyanın açılması bize kalmıştır.

OpenFileDialog sınıfının CommonFileDialog sınıfından gelen ShowDialaog fonksiyonu dosya seçme diyalog penceresini açar. O halde Dosya seçme diyalog penceresi tipik olarak şöyle açılır.

```
OpenFileDiyalog  ofs = new OpenFileDiyalod();

if(ofd.ShowDialog() == DialogResult.OK)
{
        //...
}
```

OpenFileDialog sınıfının CommonFileDialog sınıfından gelen String türünden FileName isimli property elemanı seçilen dosya ismini yol ifadesiyle vermektedir. Bu property Read/Write property dir. Bu property ye değer atanırsaa bu yazı Dialog penceresinin FileName kısmında çıkar. OpenFileDialog penceresi default olarak seçilen dosyanın da varlığını araştırmaktadır. Eğer dosya yoksa çıkmaya izin vermemektedir.

OpenFileDialog sınıfının FileDialog sınıfından gelen string türünden Filter isimli property elemanı filtreleme bilgilerini oluşturur.
Filtreleme yazısı type karakterleriyle ayrılmış çiftlerden oluşur. Çiftin solundaki yazı

görüntülenecek yazıdır. Bunun gerçek filtre ile bir ilgisi yoktur. Sağındaki yazı filtreleme bilgisini içerir.
Örneğin:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace OpenFileDialogSample
{
   public partial class OpenFileDialogForm : Form
   {
      public OpenFileDialogForm()
      {
         InitializeComponent();
      }

      private void m_buttonOk_Click(object sender, EventArgs e)
      {
         OpenFileDialog ofd = new OpenFileDialog();
         ofd.Filter = "C Files (*.c)|*.c";

         if (ofd.ShowDialog() == DialogResult.OK)
         {
            MessageBox.Show(ofd.FileName);
         }
      }
   }
}
```

Eğer gerçek filtreleme birden fazla seçenekten oluşuyorsa ikinci kısım ; lerle çoğaltılabilir. Örneğin:

```
OpenFileDialog ofd = new OpenFileDialog();
        ofd.Filter = "Image Files |*.BMP;*.JPEG";
```

Sınıfın FileDialog sınıfından gelen int türden FilterIndex isimli property elemanı Dialog penceresi açıldığında hangi elemanın aktif olacağını belirtir. İlk eleman 1 numaralı indextedir. Property nin default değeri birdir yani ilk filtre aktiftir.

```
 ofd.Filter = "Image Files |*.BMP;*.JPEG|All Files|*.*";
        ofd.FilterIndex = 2;
```

Sınıfın String türden Read/Write InitialDirectory isimli property elemanı Dialog Penceresi açıldığında görüntülenecek dizini belirtir. Bazen programcılar son açılan dizinin görüntülenmesini isteyebilir. Bu durumda çıkışta bu dizinin sınıfın bir veri elemanında saklanıp diyalog penceresi açılmadan önce bu property ye atanması gerekir. Bu işlem şöyle yapılabilir.

Anahtar Notlar: System.IO isim aslanındaki Path isimli sınıfın çeşitli statik fonksiyonları bir yol ifadesini ayrıştırmakta kullanılabilir. GetFileName yol ifadesinin sonundaki dosyanın isimini

uzantısıyla verir. GetExtention yalnızca dosyanın uzantısını verir. GetDirectoryName dosyanın içinde bulunduğu dizinin yol ifadesini verir. Sınıfın başka pek çok faydalı fonksiyonu da vardır.

OpenFileDialog sınıfının bool türden MultiSelect isimli property elemanı çoklu seçime izin verir. Bu durumda seçilen tüm dosyalar FileNames property elemanından elde edilmektedir.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace OpenFileDialogSample
{
   public partial class OpenFileDialogForm : Form
   {
      public OpenFileDialogForm()
      {
         InitializeComponent();
      }

      private void m_buttonOk_Click(object sender, EventArgs e)
      {
         OpenFileDialog ofd = new OpenFileDialog();
         ofd.Filter = "Image Files |*.BMP;*.JPEG|All Files|*.*";
         ofd.FilterIndex = 2;
         ofd.Multiselect = true;

         if (ofd.ShowDialog() == DialogResult.OK)
         {
            string text = null;

            foreach (string file in ofd.FileNames)
               text += file + "\n";
            MessageBox.Show(text);
         }
      }
   }
}
```

OpenFileDialog sınıfının OpenFile isimli fonksiyonu seçilmiş dosyayı Read/Write olarak açar.

FileDialog sınıfının bool türden CheckFileExists isimli property si seçilen dosyanın varlığını test etmekte kullanılır. Bu property nin default değeri true dır. OpenFileDialog sınıfını diğer elemanları MSDN dökumanlarından incelenmelidir.

```
 private void m_buttonOk_Click(object sender, EventArgs e)
      {
         OpenFileDialog ofd = new OpenFileDialog();
         ofd.Filter = "Image Files |*.BMP;*.JPEG|All Files|*.*";
         ofd.FilterIndex = 2;
```

```
        if (ofd.ShowDialog() == DialogResult.OK)
        {
            textBox1.Text = File.ReadAllText(ofd.FileName);

        }
    }
```

SaveFileDialog sınıfı kullanım bakımından OpenFileDialog sınıfına çok benzemektedir. Zaten pek çok elemanı ortaktır.

Renk Seçme Diyalog Penceresinin Kullanımı: Renk seçme diyalog penceresi ColorDialog sınıfı ile temsil edilmiştir. Bu sınıfta CommonDialog sınıfından türetilmiştir. Programcı önce ColorDialog sınıfı türünden bir nesne yaratır ve sınıfın CommonDialog sınıfından gelen ShowDialaog fonksiyonunu çağırır.Sınıfın Color türünden Color isimli property elemanı seçilen rengi vermektedir.

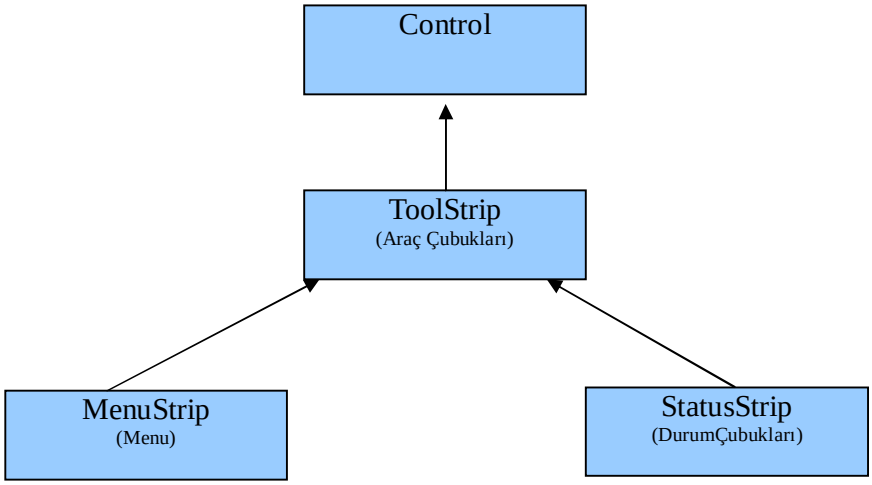
```
private void m_buttonOk_Click(object sender, EventArgs e)
    {
        ColorDialog cd = new ColorDialog();

        if (cd.ShowDialog() == DialogResult.OK)
            this.BackColor = cd.Color;
    }
```

ColorDialog sınıfının bool türden FullOpen isimli property elemanı diyalog penceresinin bütünsel açılmasını sağlar.

Sınıfın int [ ] türünden CustomColors isimli property elemanı kullanıcının ayarladığı renkleri RGB biçiminde kodlanmış bir dizi olarak elde eder. Bu değerler saklanıp yeniden yüklenerek kullanıcının seçtiği renklerin kalıcılığı sağlanır. Diyalog penceresinin default Custom renklerle başlatılması için int bir diziye renk değerlerinin int bir sayı olarak atanması gerekir. Renk değerlerini int türle ifade etmenin çok pratik bir yolu yoktur. Programcı istediği renklerin RGB bileşenleriyle int türüne dönüştürebilir.

Menü İşlemleri: Menü sistemleri Framework 2.0 ile birlikte tamamen değiştirilmiştir. Her nekadar Framwork2.0 ve sonrasında hala Framework 1.1 in menüleri desteklense de obsulate yapılmıştır. Zaten sihirbaz desteği kaldırılmıştır. Framework 2.0 ın menüleri birer kontrol biçimindedir. Dolaysıyla Menü yukarıya yuvalandığında çalışma alanının sol üst köşesi menünün altında kalmaktadır.

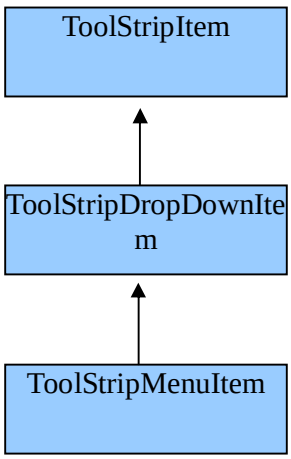
Framework 2.0 da menüler, araç çubukları ve durum çubukları ortak özelliğe sahip birer şerit(strip) kontrolleridir. Bunların hepsinin ortak özellikleri vardır ve ortak özellikler ToolStrip isimli sınıfta toplanmıştır. Araç çubukları ToolStrip sınıfının kendisiyle oluşturulur. Menüler ToolStrip sınıfından türetilmiş olan MenuStrip sınıfıyla oluşturulmaktadır ve Durum Çubukları da ToolStrip sınıfından türetilmiş StatusStrip sınıfıyla oluşturulmaktadır.



Bir menü sistemi bir menü çubuğundan ve POPup menülerden oluşmaktadır. Popup pencereler menü elemalanrına sahiptir. Menü elemanlarıda ayrı popup pencereler biçiminde olabilir. Bu durumda bir menü sistemi için mantıksal olarak şunlar yapılmalıdır.
1.Menü çubuğu, popup pencerler ve menü elemanları birer sınıf nesnesi biçiminde yaratılır.
2. Menü elemanları popup pencerelere popup pencereleri de menü çubuğuna bağlanır.

Menü çubuğu MenuStrip sınıfı türünden bir nesne yaratılıp formun control kümesine eklendiğinde otomatik olarak yaratılır.

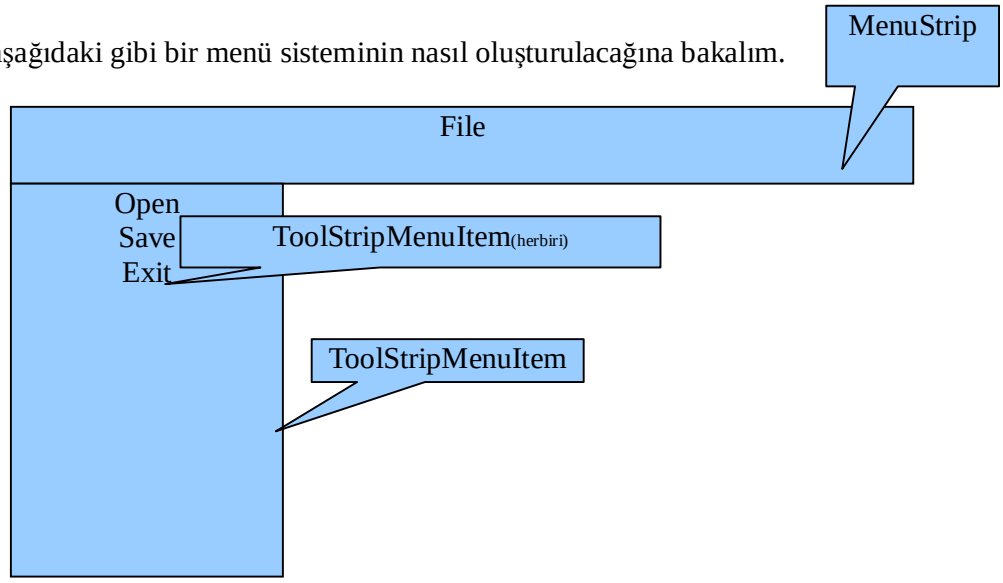
MenuStrip sınıfın ToolStrip sınıfından gelen Item isimli property elemanı ToolStripItemCollection isimli Collection sınıf türündendir. Bu collection sınıf ToolStripItem sınıf türünden nesneleri tutmaktadır. Araç çubukları, menüler, durum çubukları üzerindeki elemanlar ToolStripItem sınıfından türetilmiş sınıflarla temsil edilmektedir. Bir menü elemanı ya da popup menü dolaylı olarak ToolStripItem sınıfından türetilmiştir.



ToolStripMenuItem sınıfı hem popup pencereleri hem de sıradan menü elemanlarını temsil eder. Bir ToolStripMenuItem nesnesinin sıradan bir menü elemanımı yoksa bir popup mı olduğu sınıfın DropDownItemsCollection property siyle belirlenmektedir. DropDownItems propertysi ToolStripItem isimli Collection sınıf türündendir. Anaımsanacağı gibi bu sınıf ToolStripItem

nesneleri tutmaktadır. .net Framwork bu collection elemanın boş olup olmadığına bakar eğer bu elemana hiçbir şey eklenmediyse bu sıradan bir menü elemanıdır. Eklenmişse bu bir popup elemandır ve ekleneler de bunun içindeki elemaları belirtir.

Şimdi aşağıdaki gibi bir menü sisteminin nasıl oluşturulacağına bakalım.



Bu sistemde toplam 5 farklı nesne yaratılmalıdır. 1 menü çubuğu için MenuStrip sınıfı türünden 2 File popup için ToolStripMenuItem sınıfı türünden diğer üç tanesi open, save, exit menü elemaları için ToolStripMenuItem snıfı türündendir. Open, Save ve Exit elemanları için yarattığımız ToolStripMenuItem nesnelerini File popup ın DropDownItem collectionına File popup ıda MenuStrip sınıfının Items collectionına ekleriz.
MenuStrip nesnesinide formun control listesine ekleriz.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace SampleMenu
{
   public partial class Form1 : Form
   {
      private MenuStrip m_mainMenu;
      private ToolStripMenuItem m_filePopup;
      private ToolStripMenuItem m_openItem, m_saveItem, m_exitItem;

      public Form1()
      {
         InitializeComponent();

         m_mainMenu = new MenuStrip();
```

```
            m_filePopup = new ToolStripMenuItem();
            m_filePopup.Text = "&File";

            m_openItem = new ToolStripMenuItem();
            m_openItem.Text = "&Open";

            m_saveItem = new ToolStripMenuItem();
            m_saveItem.Text = "&Save";

            m_exitItem = new ToolStripMenuItem();
            m_exitItem.Text = "&Exit";

            m_filePopup.DropDownItems.AddRange(new ToolStripItem[] { m_openItem, m_saveItem,
m_exitItem });
            m_mainMenu.Items.Add(m_filePopup);


            this.Controls.Add(m_mainMenu);
        }
    }
}
```

ToolStripMenuItem sınıfının check isimli bool türden property elemanı menu elemanını Checked ya da Unchecked yapmakta kullanılır. ToolStripMenuItem sınıfının Enabled isimli bool türden property elemanı ilgili menü elemanını aktif ya da pasif duruma getirir.

ToolStripMenuItem sınıfının Font isimli property elemanı eleman yazısının fontunu ayarlamakta kullanılır.

Yine sınıfın klasik BackColor ve ForeColor propertyleri menü elemanın zemin ve şekil renklerini oluşturur.

ToolStripMenuItem sınıfının Size isimli property elemanı ile menü elemanının genişlik ve yüksekliği ayarlanabilir. Örneğin biz menü elemanını diğerlerinden daha yüksek yapabiliriz. Bu durumda eleman yazısı sınıfın TestAlingn propertysi ile ayarlanabilir.

Menü elemanı için kısayol tuşu ShortcutKeys propertysi ile atanır. Bu property Keys isimli enum türündendir.

Bir menü elemanı seçildiğinde sınıfın Click event elemanı tetiklenir. Kısayol tuşunun görüntülenip görüntülenmeyeceği sınıfın bool türden ShowShortcutKeys propertysi ile belirlenmektedir. Eğer istenirse kısayol için istenilen bir yazıda ShowShortcutDsiplayString propertysi ile görüntülenebilir.

ToolStripMenuItem sınıfının ToolStripItem sınıfından gelen ToolTipText propertysi ipucu yazısını çıkartmakta kullanılır. Fakat istenirse ipucu yazısı ShowItemToolTips propertysi ile gösterilebilir ya da gösterilmeyebilir. Ayrıca ToolStripMenuItem sınıfının ToolStripItem sınıfından gelen Bool türden AutoToolTip properyty si default olarak menü yazısının ipucu yazısı olarak görüntülenmesini sağlar.

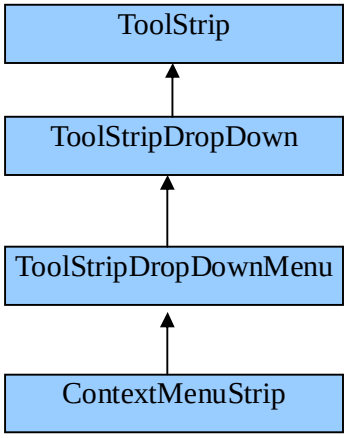
Ayrıca bir popup için menü olaylarının takip etmeye yönelik çeşitli event elemanlarda vardır.

Örneğin popup açıldığında ve kapandığında çeşitli eventler tetiklenmektedir. Ya da örneğin menü elemanları üzerinde gezinirken yine eventler tetklenmektedir. Tüm bu eventler MSDN dökumanlarından incelenmelidir.

Pek çok programda birden fazla menü bulunmaktadır ve menüler arasında geçiş yapılmaktadır. Bunun için çeşitli yöntemler kullanılabilir.
Örneğin bütün menü çubukları Forma eklenebilir. Fakat yalnızca birinin Visible property si true yapılır. Duruma göre diğerinin Visible propertysi tru yapılarak geçiş sağlanabilir. Ya da bu işlem ilgili menü çubuğu control ünün formun control listesine eklenip çıkarılarakta yapılabilir. Bazen bir menü elemanını seçtiğimizde başka popuplar ya da başka menü elemanlarıda eklenebilmektedir. Bu işlem menüye dinamik olarak Add ve Remove fonksiyonlarıyla ekleme çıkarma yapılarak sağlanabilir. Ya da benzer teknik kullanılabilir. Programcı tüm popupları ekler gerektiğinde Visible propertysini true ya da False çeker.

Bazı programlarda az kullanılan menü elemanlarını popup açıldığında görüntülenmemektedir. Bu şimdilik otomatik yapılan bir özellik değildir.

Bağlam Menüleri: Çalışma alanı üzeride farenin sağ tuşuna basılarak görüntülenen menüye bağlam menü(Context Menu) denilir. Bağlam menüsü de Framework2.0 ile birlikte değiştirilmiştir. Bunun için eskiden Form sınıfının ContextMenu propertysi kullanılıyordu. Artık ContextMenuStrip propertysi kullanılmaktadır. Form sınıfının ContextMenuStrip propertysi ContextMenuStrip isimli bir sınıf türündendir. Bu sınıfını türetme şeması şöyledir.



Programcı bağlam menüsünü manuel olarak şöyle oluşturur.

1. ContextMenuStrip türünden bir nesne yaratılır.
2. ContextMenuStrip sınıfının ToolStripItemCollection isimli Items propertysine ekleme yapar. ContextMenuStrip sınıfının kullanımı MenuStrip sınıfının kullanımına çok benzemektedir.

Pek çok programda birden fazla bağlam menüsü vardır ve fare belirli bölgelerdeyken farklı bağlam menüleri görüntülenmektedir. Bu işlem bir kaç biçimde yapılabilir.
Birincisi Form sınıfının ContextMenuStrip propertysini kullanmak yerine tamamen MouseDown Eventini kullanarak koordinatlara bakmak ve duruma göre ContextMenuStrip sınıfının show fonksiyonu ile açım yapılmalı.
Diğer bir yöntem farenin durumuna göre sınıfın ContextMenuStrip propertysini ayarlamaktır. Yani programcı fare hareketlerini izler bir property e uygun elemanı atar. Bu atama işlemi MouseMove mesajında yapılabileceği gibi MouseDown mesajında da yapılabilir. Çünkü farenin sağ tuşuna basıldığından önce MouseDown eventi tetiklenmekte sonra bağlam menüsü görüntülenmektedir.

```
using System.Text;
using System.Windows.Forms;

namespace SampleMenu
{
   public partial class Form1 : Form
   {
      private MenuStrip m_mainMenu;
      private ToolStripMenuItem m_filePopup;
      private ToolStripMenuItem m_openItem, m_saveItem, m_exitItem;

      private ContextMenuStrip m_contextMenu;
      private ToolStripMenuItem m_appleItem, m_fruitItem;

      public Form1()
      {
         InitializeComponent();

         m_mainMenu = new MenuStrip();

         m_filePopup = new ToolStripMenuItem();
         m_filePopup.Text = "&File";

         m_openItem = new ToolStripMenuItem();
         m_openItem.Text = "&Open";
         m_openItem.ShortcutKeyDisplayString = "Kontrol O";
         m_openItem.ShortcutKeys = Keys.Control | Keys.O;
         m_openItem.ToolTipText = "This is a sample";
         m_openItem.Click += new EventHandler(m_openItem_Click);

         m_saveItem = new ToolStripMenuItem();
         m_saveItem.Text = "&Save";

         m_exitItem = new ToolStripMenuItem();
         m_exitItem.Text = "&Exit";

         m_filePopup.DropDownItems.AddRange(new ToolStripItem[] { m_openItem, m_saveItem,
m_exitItem });
         m_mainMenu.Items.Add(m_filePopup);

         m_contextMenu = new ContextMenuStrip();

         m_appleItem = new ToolStripMenuItem();
         m_appleItem.Text = "&Apple";
         m_appleItem.Click += new EventHandler(m_appleItem_Click);

         m_fruitItem = new ToolStripMenuItem();
         m_fruitItem.Text = "Fruit";

         m_contextMenu.Items.AddRange(new ToolStripItem[] { m_appleItem, m_fruitItem });

         //this.ContextMenuStrip = m_contextMenu;
```

```
        this.Controls.Add(m_mainMenu);
    }

    void m_appleItem_Click(object sender, EventArgs e)
    {
        MessageBox.Show("Apple selected");
    }

    void m_openItem_Click(object sender, EventArgs e)
    {
        OpenFileDialog ofd = new OpenFileDialog();

        if (ofd.ShowDialog() == DialogResult.OK)
        {
            MessageBox.Show(ofd.FileName);
        }
    }

    private void Form1_MouseDown(object sender, MouseEventArgs e)
    {
        if (e.Button == MouseButtons.Right)
        {
            Rectangle rect = new Rectangle(100, 100, 100, 100);

            if (rect.Contains(e.Location))
                this.ContextMenuStrip = m_contextMenu;
        }
    }
  }
}
```

Aslında Form sınıfının ContextMenuStrip propertysi Control sınıfından gelmektedir. Yani her kontrol ün ContextMenuStrip elemanı zaten vardır. Böylece biz örneğin bir düğmenin üzerinde sağ tuşa bastığımızda bağlam menüsünün görüntülenmesini sağlayabiliriz.

Menülerin Form Editör Yoluyla Kullanılması: Menlüler form editörde tamamen görsel biçimde oluşturulabilir. Form editör menü çubuğu için, her popup için ve her menü elemanı için ayrı birer nesne oluşturmaktadır.

Nesnelerin Seri Hale Getirilmesi: Seri hale getirme(Serialization) sınıf nesnelerinin bir kaynağa (muhtemelen disk) yazılması ve oradan geri alınmasıanlamına gelmektedir. Bir sınıf nesnesinin saklanması aslında onun datalarının saklanması anlamına gelir. Geri alma sırasında da o veri elemanları yeniden doldurulur.

Seri hale getirme otomatik ve manuel biçimde yapılabilir. Manuel(custom) seri hale getirme kursumuzun kapsamı dışındadır.

Seri hale getirme işlemi şu adımlardan geçiler yapılmaktadır.

1. Seri hale getirilecek sınıf yada yapı serializable attribute öznitelik sınıfı kullanılarak özniteliklendirilmelidir. Bunun için ilgili sınıf ya da yapının önüne [Serializable]

belirlemesinin yapılması gerekir.

```
[Serializable]
class Sample
{
    //...
    }
```

2. Seri hale getirilecek hedef yaratılır. Hedefin Stream sınıfından türetilmiş bir sınıf olması gerekir. Tipik hedef dosyadır. Bunun için FileStream sınıfı türünden bir nesne yaratılır.
3. Seri hale getirme işlemi sırasında yazmanın formatı değişebilir. Yazma düz binary yapılabileceği gibi html/xml biçiminde yapılabilir. İşte yazım formatına uygun bir nesnesinin yapılması gerekir. Bu tür sınıflar genellikle XXFormatter biçiminde isimlendirilmiştir. Örneğin binary yazım için BinaryFormatter sınıfı kullanılabilir.
   Örneğin: *BinaryFormatter bf= new BinaryFormatter();*
4. Tüm XXXFormatter sınıfları IFormatter arayüzünü desteklemektedir. Bu arayüzün de Serialize ve Deserialize fonksiyonları vardır. Seri hale getirme işlemi ilgili formatter sınıfının Serialize fonksiyonu çağrılarak yapılır.
   void Serialize(
   Stream serializationStream,
   Object graph
) Fonksiyonun birinci parametresi hedefi ikinci parametresi seri hale getirilecek nesneyi belirtmektedir. Geri alma işlemi Deseralize fonksiyonuyla yapılır.
   Object Deserialize(
   Stream serializationStream,
) Fonksiyon parametre olarak kaynağı almaktadır.Geri dönüş değeri yaratılmış olan nesnedir. Fonksiyon nesneyi yaratı içine bilgileri doldurur ve bize nesnenin referansını verir.

Seri hale getirme işlemi basit bir işlem değildir. Seri hale getireceğimiz sınıf ya da yapı başka sınıflar türünden referans veri elemanlarına sahipse yalnızca ana nesnesinin hedefe yazılması yetmez. Aynı zamanda referans veri elemalarının gösterdikleri nesnelerin de hedefe yazılması gerekir. Yani bir ağaç sözkonusudur. Tüm bir ağacın belli bir sırada yazılması ve okunması gerekir. Aynı durum taban sınıf için de geçerlidir. Biz bir türemiş sınıf nesnesini hedefe yazacaksak onun tüm taban sınıfılarını da yazmak zorundayız.

Bir sınıf ya da yapıya biz Serializable özniteliğini atamışsak bu sınıfın tüm taban sınıflarının ve tüm veri elemanlarının da bu özniteliğe sahip olması gerekir. Zaten .net in int32, int64, String gibi tüm temel yapı ve sınıfları bu özniteliğe sahiptir.

.net içindeki bazı collection sınıflarda Serializable durumdadır. Örneğin ArrayList sınıfı böyledir. Biz tewk hamlede bir ArrayList nesnesini diske yazıp yine tek hamlede geri alabiliriz. Arka planda yapılan işlemler basit değildir. Mekanzima ArrayList nesnesinin elemanlarını onların tuttuğu nesneleri yani bütün bir ağacı özyinelemeli olarak hedefe yazabilmektedir. Geri alım sırasında tüm bu nesneler mekanizma taraından yeniden yaratılır.

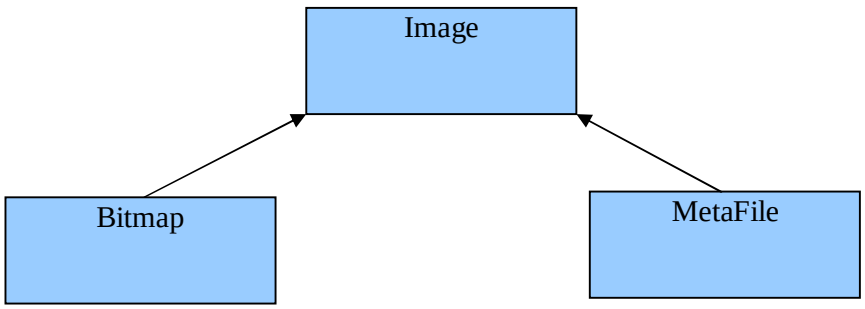
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Collections;
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;

namespace TestSerialization
```

```
{
    class Program
    {
        static void OldMain(string[] args)
        {
            ArrayList al = new ArrayList();

            al.Add(new Person("Kaan Aslan", 123));
            al.Add(new Person("Ali Serçe", 456));
            al.Add(new Person("Necati Ergin", 43));
            al.Add(new Person("Guray Sonmez", 876));
            al.Add(new Person("Ertan Geyik", 1239));
            al.Add(new Person("Gurbuz Aslan", 568));


            FileStream fs = null;

            try
            {
                fs = new FileStream("test.dat", FileMode.CreateNew, FileAccess.Write);
                BinaryFormatter bf = new BinaryFormatter();
                bf.Serialize(fs, al);

            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
            finally
            {
                if (fs != null)
                    fs.Close();
            }

            Console.WriteLine("Ok");

        }

        public static void Main()
        {
            FileStream fs = null;
            ArrayList al;

            try
            {
                fs = new FileStream("test.dat", FileMode.Open, FileAccess.Read);
                BinaryFormatter bf = new BinaryFormatter();
                al = (ArrayList) bf.Deserialize(fs);

                foreach (Person per in al)
                    per.Disp();
```

```
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
        }
        finally
        {
            if (fs != null)
                fs.Close();
        }

    }
}

[Serializable]
class Person
{
    private string m_name;
    private int m_no;

    public Person()
    { }

    public Person(string name, int no)
    {
        m_name = name;
        m_no = no;
    }

    public void Disp()
    {
        Console.WriteLine("{0}   {1}", m_name, m_no);
    }
}
}
```

Resimlerle İşlemYapmak:

.Net'de her türlü resimsel görüntü image sınıfı ile temsil edilmektedir. İmage sınıfından çeşitli sınıflar türetilmiştir. Image sınıfı abstract sınıftır.



Bitmap sınıfı resimsel görüntüyü oluşturduğumuz somut bir sınıftır. Bitmap sınıfı (ismi bmp dosyalarını çağrıştırsa da) aslında JPEG,GIF,PNG,TIFF,BMP,EXIF gibi pek çok dosya formatını desteklemektedir. Bir resim sınıfı Bitmap başlangıç fonksiyonu ile yaratılabilir.

public Bitmap(string filename);

fonksiyon parametre olarak resim dosyasının yol ifadesini almaktadır.

PictureBox isimli kontrol oldukça basittir. Tek yaptığı şey bir resmi alıp onu göstermektir. PicturteBox sınıfının Image türden proıperty elemanı görüntülenecek resmi alır.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace ImageSample
{
    public partial class ImageSampleForm : Form
    {
        private Bitmap m_bitmap;

        public ImageSampleForm()
        {
            InitializeComponent();
        }

        private void openToolStripMenuItem_Click(object sender, EventArgs e)
        {
            OpenFileDialog ofd = new OpenFileDialog();
            ofd.Filter = "Jpeg Files|*.jpg;*.jpeg|Bitmap Files|*.bmp|All Files|*.*";

            if (ofd.ShowDialog() == DialogResult.OK)
            {
                m_bitmap = new Bitmap(ofd.FileName);
                m_pictureBox.Image = m_bitmap;
            }
        }
    }
}
```

PictureBox kontrol Debug olarak resmi gerçek boyutu ile gösterir. resim buyuse biz onun yalnızsca sol ust kosesini goruruz. gösterim biçimi sınıfın pictureBox.SizeMode isimli enum türünden sizeMode isimli property elemanı ile ayarlanabilir. Bu enum türünün elemanları şunlardır:

Normal: Bu default durumdur. Resim orijinal büyüklükte ve yalnızca sol üst köşesi görüntülenir.

StrechImage: Bu seçenek resmi kontrol boyutuna getirir.

AutoSize: Burada pictureBox kontrolü resim boyutuna çekilir.

CenterImage: burada resim Kontrolden küçükse resim kontrolün ortasında görüntülenir. fakat resim kontrolden buyukse resmin ortası goruntulenir. burada resim kontrol boyutuna getirilir fakat aynı

zamanda yatay-düşey orantı dikkate alınır. Duruma göre kontrolün kenarlarında boşluk bırakılır.

```
namespace ImageSample
{
    public partial class ImageSampleForm : Form
    {
        private Bitmap m_bitmap;

        public ImageSampleForm()
        {
            InitializeComponent();
        }

        private void openToolStripMenuItem_Click(object sender, EventArgs e)
        {
            OpenFileDialog ofd = new OpenFileDialog();
            ofd.Filter = "Jpeg Files|*.jpg;*.jpeg|Bitmap Files|*.bmp|All Files|*.*";
            ofd.InitialDirectory = @"e:\dotnetappbasic\ImageSample\ImageSample";

            if (ofd.ShowDialog() == DialogResult.OK)
            {
                m_bitmap = new Bitmap(ofd.FileName);
                m_pictureBox.SizeMode = PictureBoxSizeMode.Zoom;
                m_pictureBox.Image = m_bitmap;
            }
        }
    }
}
```

Bitmap sınıfının Image sınıfından gelen size,width ve height propertyleri resmin boyutunu belirler.

```
namespace ImageSample
{
    public partial class ImageSampleForm : Form
    {
        private Bitmap m_bitmap;

        public ImageSampleForm()
        {
            InitializeComponent();
        }

        private void openToolStripMenuItem_Click(object sender, EventArgs e)
        {
            OpenFileDialog ofd = new OpenFileDialog();
            ofd.Filter = "Jpeg Files|*.jpg;*.jpeg|Bitmap Files|*.bmp|All Files|*.*";
            ofd.InitialDirectory = @"e:\dotnetappbasic\ImageSample\ImageSample";

            if (ofd.ShowDialog() == DialogResult.OK)
            {
                m_bitmap = new Bitmap(ofd.FileName);
                m_pictureBox.SizeMode = PictureBoxSizeMode.Zoom;
                m_pictureBox.Image = m_bitmap;
                this.Text = ofd.FileName + "  " + m_bitmap.Size.ToString();//boyutu
yazar…
            }
        }
    }
}
```

Bitmap sınıfının RawFormat isimli property elemanı nesneye tutulan resmin formatını verir.Bu property Image foırmat isimli sınıf türündendir. Bu sınıfın çeşitli static propertyleri çeşitli formatları belirtir. Bitmap sınıfının pixcel format propertysi renk pixel yapısını verir.

Bitmap sınıfının getpixel ve setpixel fonksiyonları resimndeki bir pixelin rengini alıp set etmekte kullanılır.

```
OpenFileDialog ofd = new OpenFileDialog();
            ofd.Filter = "Jpeg Files|*.jpg;*.jpeg|Bitmap Files|*.bmp|All Files|*.*";
            ofd.InitialDirectory = @"e:\dotnetappbasic\ImageSample\ImageSample";

            if (ofd.ShowDialog() == DialogResult.OK)
            {
                m_bitmap = new Bitmap(ofd.FileName);
                for (int i = 0; i < 100; ++i)
                    for (int k = 0; k < 100; ++k)
                        m_bitmap.SetPixel(100 + i, 100 + k, Color.Red);
                m_bitmap.RotateFlip//…
                m_pictureBox.SizeMode = PictureBoxSizeMode.Zoom;
                m_pictureBox.Image = m_bitmap;
                this.Text = ofd.FileName + "  " + m_bitmap.Size.ToString();
```

Bitmap sınıfının RotateFlip isimli fonksiyonu resmi tersine çevirir.

```
m_bitmap.RotateFlip(RotateFlipType.Rotate180FlipX);
```

Bitmap sınıfının Save fonksiyonu ilgili Bitmap nesnesini belirli boyutta diske saklamak için kullanılır.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Drawing.Imaging;

namespace ImageSample
{
    public partial class ImageSampleForm : Form
    {
        private Bitmap m_bitmap;

        public ImageSampleForm()
        {
            InitializeComponent();
        }

        private void openToolStripMenuItem_Click(object sender, EventArgs e)
        {
            OpenFileDialog ofd = new OpenFileDialog();
            ofd.Filter = "Jpeg Files|*.jpg;*.jpeg|Bitmap Files|*.bmp|All Files|*.*";
            ofd.InitialDirectory = @"e:\dotnetappbasic\ImageSample\ImageSample";

            if (ofd.ShowDialog() == DialogResult.OK)
            {
```

```
                m_bitmap = new Bitmap(ofd.FileName);
                for (int i = 0; i < 100; ++i)
                    for (int k = 0; k < 100; ++k)
                        m_bitmap.SetPixel(100 + i, 100 + k, Color.Red);
                m_bitmap.RotateFlip(RotateFlipType.Rotate180FlipX);
                m_pictureBox.SizeMode = PictureBoxSizeMode.Zoom;
                m_pictureBox.Image = m_bitmap;
                this.Text = ofd.FileName + "  " + m_bitmap.Size.ToString();
            }
        }

        private void saveToolStripMenuItem_Click(object sender, EventArgs e)
        {

        }

        private void saveAsToolStripMenuItem_Click(object sender, EventArgs e)
        {
            SaveFileDialog sfd = new SaveFileDialog();

            if (sfd.ShowDialog() == DialogResult.OK)
            {
                try
                {
                    m_bitmap.Save(sfd.FileName, ImageFormat.Jpeg);
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message);
                }

            }
        }
    }
}
```

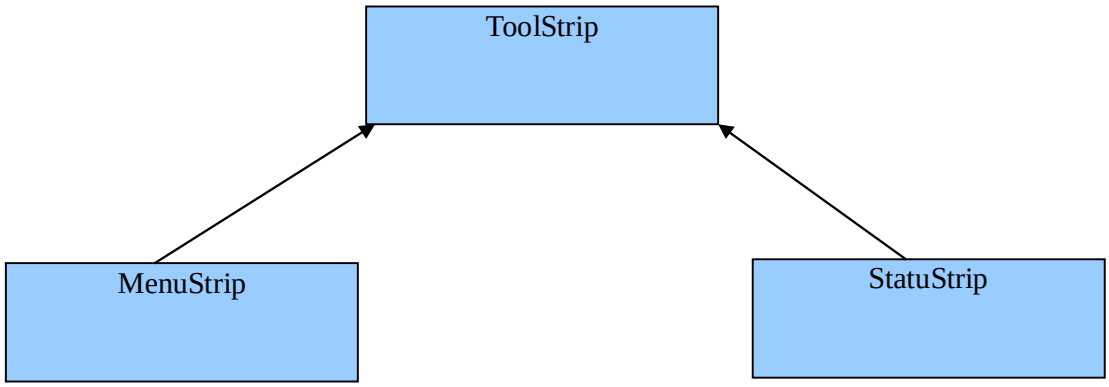
Pek çok kontrol kullanımında Image özelliği vardır. Örneğin Menu elemanlarının solunda istersek küçük bir resim gösterebiliriz. ToolStripMenuItem sınıfının Image propertysi Menu elemanının solunda görüntülenecek resmi belirler.

Araç çubukları gibi menuler gibi kontroller de pek çok küçük resme dolaylı olarak gereksinim duyulmaktadır. Bu resimlerinin hepsinin RunTime sırasında dosyadan hareketle oluşturulması hataya çok açık bir durum olusturur. Çünkü o resimlerin bir tanesi bile silinse proje bozulacaktır. İşte resimler gibi pek çok data Resorce kavramı altında Assambly dosyasına gömülebilmektedir. Resource konusu detaylı ve ayrı bir konudur. Fakat FormEditor bu kontrollerin Image propertysine tıklandıgında secilen resmi Resource olarak gomup kullanmaktadır.

Visiul Studio IDEsi Install edildiginde Comman7 dizini içerisinde VS2008ImageLibrary dizininde çok kullanılan pek çok küçük icon goruntusu bulunmaktadır.

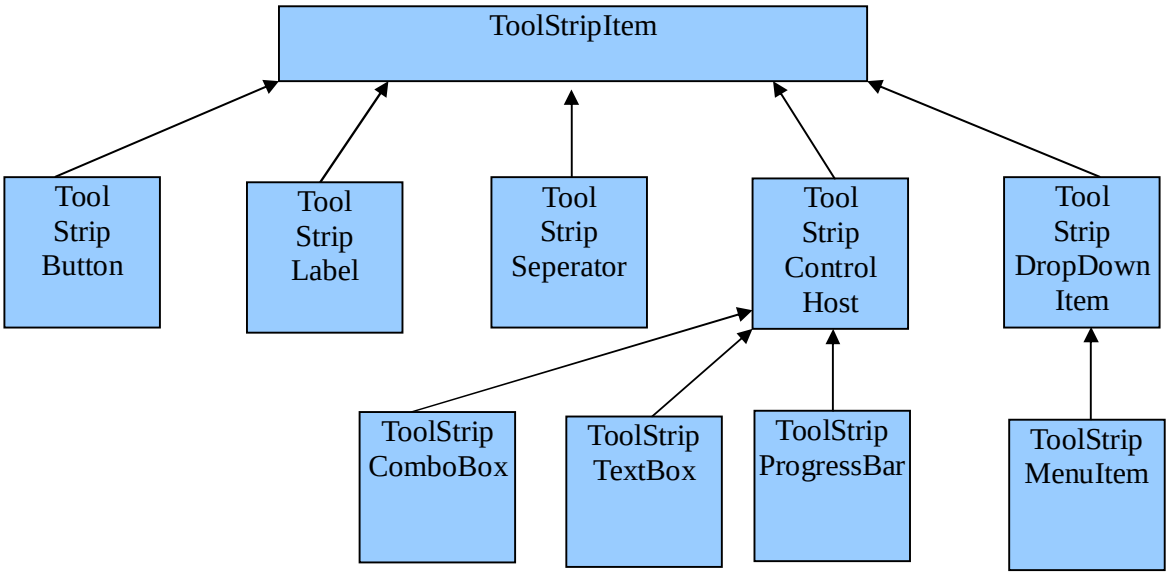
Düğme kontrollerinde oldugu gibi bazı kontrollerde Image propertyleri vardır.Eğer kontrolumuzun Image propertysi varsa biz bu property ye resim girdiğimizde kontrolun zemininde resim goruntulenir.

Araç Çubukları(Tool Bar): Araç çubukları ToolStrip sınıfı ile temsil edilmektedir yani ToolStrip sınıfı hem ToolStripMenuStrip,StatuStrip sınıfının taban sınınfıdır.Fakat aynı zamanda araç çubugu olarak da kullanılır.



Buradaki mantıga görte aslında araç çubugu bir nevi menu cubugudur.Onun özel bir bbiçimidir. Anımsanacağı gibi Araç Çubukları ToolStripItem sınıfından türetilmiş sınıflarla temsil edilmektedir.

ToolStripItem sınıfı Abstract bir sınıfıtır. Menu elemanları aslında dolaylı olarak bu sınıftan türetilmiştir. ToolStripItem sınıfının türetme şeması şöyledir.



Araç Çubuğu eleman oluşturmak için ToolStripButton, ToolStripLabel, ToolStripSeperator, ToolStripConrolHost sınıfları kullanılır. ToolStripDropDownItem ve ondan türemiş sınıflar menu oluşturmak için kullanılır.

Tipik olarak araç çubuğu elemanı bir düğmedir. Yani programcı ToolStripButton türünden bir nesne yaratır ve bu nesneyi ToolStrip sınıfının Items Collection ına ekler. ToolStripButton nesnesi görüntü olarak yalnızca bir yazı, yalnızca bir resim ya da hem yazı hem resim gösterebilmektedir. Default durum yazı ve resmin birlikte görüntülenmesidir. Genellikle bu durum istenen bir durum değildir. Nesnenin ToolStripItem sınıfından gelen DisplayStyle isimli property elemanı bu belirlemeyi yapmakta kullanılır. ToolStripButton nesnesinin istediğimiz nesneyi görüntüleyebilmesi için Image türünden Image property si kullanılır.

Anahtar Notlar: Çok tipik araç çubuğu ve icon boyutları 16x16 32x32 türündendir.

Araç çubuğu elemanına tıklandığında ToolStripButton sınıfının ToolStripItem sınıfından gelen click event i tetiklenir. Öncelikle Araç çubuğu elemanlarını büyütmek için araç çubuğunun kendisini

AutoSize mod dan çıkarmak gerekir. ToolStrip sınıfının Size property elemanı ancak Auto Size false ise etkili olmaktadır. Halbuki AutoSize true durumdadır. Araç çubuğu elemanlarını büyütmek için sırasıyla şunlar yapılmalıdır.

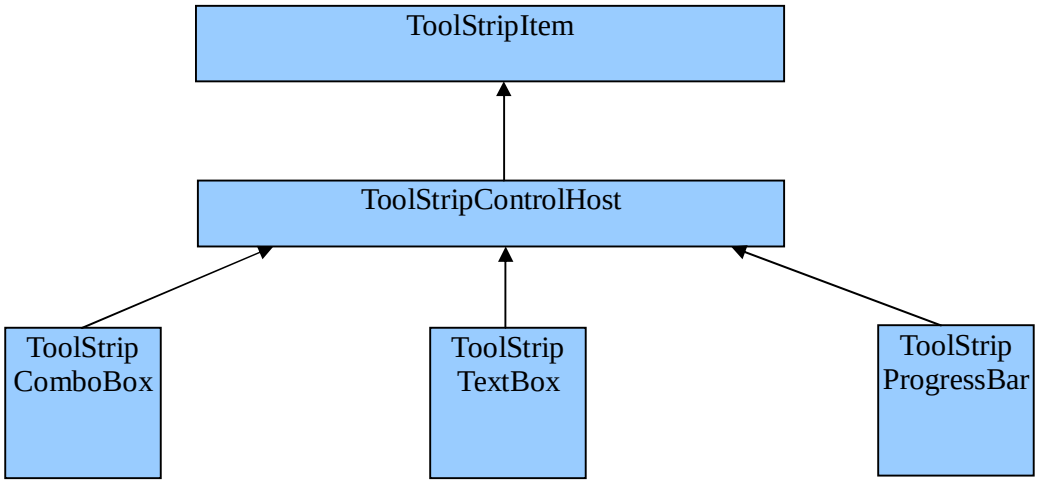
1. ToolStrip nesnesi AutoSize moddan çıkartılıp Size propertysi düzenlenir. Şüphesizaraç çubuğunun genişliği yukarıya yuvalandığı için değişemez. Ama yüksekliği değiştirilebilir.
2. ToolStrip nesnesinin ImageScalingSize isimli property elemanı araç çubuğu elemanları kaça kaç olacaksa o biçimde ayarlanır.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Toolbar
{
   public partial class Form1 : Form
   {
      private ToolStripButton m_toolbutton;

      public Form1()
      {
         InitializeComponent();

         try
         {

            m_toolbutton = new ToolStripButton();
            m_toolbutton.Text = "Deneme";
            m_toolbutton.DisplayStyle = ToolStripItemDisplayStyle.Image;
            m_toolbutton.Image = new Bitmap(@"E:\DotNetAppBasic\Toolbar\Toolbar\audio.bmp");
            m_toolBar.Items.Add(m_toolbutton);
            m_toolbutton.Click += new EventHandler(m_toolbutton_Click);

         }
         catch (Exception ex)
         {
            MessageBox.Show(ex.Message);
         }


      }

      void m_toolbutton_Click(object sender, EventArgs e)
      {
         MessageBox.Show("Click");
      }
   }
}
```

3. Araç çubuğuna yerleştirilen ToolStripItem nesnelerinin ImageScaling propertyleri ayrıca SizeToFit durumunda olmalıdır.

Normal olarak tüm arça çubuğu elemanlarının aynı boyutta olması arzu edilir. Fakat bu zorunlu değildir. İstenirse her araç çubuğu elemanı farklı boyutlarda olabilir. Bunun için araç çubuğu elemanlarını autosize moddan çıkarmak ve onlara yeni Size vermek gerekir. ToolStrip sınıfının ImageScalingSize elemanı ToolStrip nesnelerinin AutoSize true ise etkili olmaktadır.

Bazen bir araç çubuğu elemanının ComboBox olması ve TextBox olması durumu ile karşılaşılmaktadır. ComboBox ToolStripComboBox sınıfıyla temsil edilmiştir. TextBox ise ToolStripTextBox sınıfıyla temsil edilmiştir. Araç çubuğu elemanı ProgressBar da olabilir. Bu da ToolStripProgressBar sınıfıyla temsil edilmiştir. Bu sınıfların hepsi ToolStripControlHost sınıfından türetilmiştir.



Nihayet ToolStripControlHost sınıfı aslında herhangi bir controle evsahipliği yapan bir sınıftır. Biz bu sınıfın Control Property sine herhangi bir control yerleştirisek aslında araç çubuğu elemanı olarak o control görüntülenir. Bunun için en iyi yöntem ToolStripControlHost sınıfından türetme yapmaktır. Zaten aslında ToolStripComboBox isimli sınıflarda bu yöntemle elde edilmiştir.

Tab Control: TabControl API terminolojisinde Property Sheet ismiyle bilinmektedir. TabControl sayfalı diyalog penceresidir. Bir diyalog penceresi açılır. Sayfaları vardır. Aslında her sayfa bağımsız bir diyalog penceresi gibidir. Böylece kullanıcı pek çok belirlemeyi bir arada yapabilir.

TabConrol Control ünün her bir sayfası TabPAge isimli sınıfla temsil edilmiştir. Dolayısıyla programcı önce sayfalar için TabPage nesnelerini oluşturur. Sonra bu nesneleri TabControl sınıfının TabPageCollection sınıfının TabPages elemanına ekler. TabPAge sınıfı Panel sınıfından türetilmiştir. Panel sınıfı ise Form sınıfı gibidir.

Anahtar Notlar: Panel isimli sınıf Form sınıfına benzemekle birlikte bir alt pencere oluşturmaktadır. Panel adeta alt pencere biçiminde bir form gibidir.

Bir TabControl Form Editörde çok pratik oluşturulabilmektedir. FormEditör TabPage nesnelerini kendisi oluşturup TabControl sınıfına eklemektedir. Fakat bu işlemlerin en az br defa manuel yöntemle yapılması tavsiye edilir.

Pek çok yaygın programda gördüğümüz Options menü elemanı içinde TabControl ün bulunduğu bir diyalog penceresini açmaktadır. Tipik olarak TabControl diyalog penceresinin içine yerleştirilir.

Yine tipik olarak diyalog penceresinin Ok, Cancel , Apply gibi tuşları olur. Apply tuşu bu tür diyalog pencerelerinde sık karşılaşılan bir tuştur. Bu tuş yapılan değişikliklerin o anda etki yapmasını sağlar ve diyalog penceresinin kapatılmasını sağlamaz. TabControl içeren bir diyalog penceresi kapatıldığında tüm sayfalardak bilgilerin ele geçirilmesi gerekir. Bunun birkaç yolu olabilir.

1. Diyalog penceresini temsil eden form sınıfının TabControl elemanı ya public yapılır ya da bir property ile elde edilir. Diyalog penceresi kapatıldıktan sonra bu elemandan faydalınılarak TabPAge nesnelerine erişilip tüm bilgiler elde edilir.
2. Klasik yöntem uygulanır. Yani Tüm sayfalarda elde edilecek bilgiler diyalog penceresini temsil eden Form sınıfına property olarak girilir. Diyalog penceresi ok tuşu ile kapatılmadan önce bu propertylere yerleştirme yapılır.

Anahatar Notlar: Bir tab Kontrolün sayafalarına kontroller yerleştirildiğinde form editör bu kontrolleri anaformun veri elemanı olarak tanımlar. Fakat tab kontrolün panellerine yerleştirir.

Çizim İşlemeleri: Bir pencerenin çalışma alanına çizim yapıldığında windows bu çizimi bizim için saklamaz. Eğer penceremizin üstüne başka bir pencere gelirse ve bu pencere çekilirse görüntü saklanmadığı için yaptığımız çizimler kaybolacaktır. Bir pencerenin görünmeyen kısmı görünür hale geldiğinde windows pencereye paint mesajı göndermektedir. Bizimde paint mesajı geldiğinde bozulmuş olan çizimi yeniden yaparak bozulmayı telefi etmemiz gerekir. Görüldüğü gibi windows yapılan çizimleri bizim için tutup geri basmaz fakat görüntü bozulduğunda pencereye mesaj göndererek bizi uyarır.

Paint mesajı pencerenin görünmeyen bir kısmı görünür hale geldiğinde gönderilmektedir. Örneğin tiik olarak pencereinin üzerinde bulunan başka bir pencereyi çektiğimizde daha önce görünmeyen kısım görünür hale gelmiştir ve paint mesajı gönderilir. Ya da örneğin pencerenin bir kısmını masaüstünün dışına çıkarmış olalım ve yeniden o kısmı masaüstüne çekelim yine paint mesajı gönderilir. Pencere büyütüldüğünde ya da küçültüldüğünde paint mesajının gelip gelmemesi .nette control sınıfının bool türden ResizeRedraw property si ile tespit edilmektedir. Bu propertynin default durumu false biçimindedir. Yani pencere resize edildiğinde paint mesajı oluşmaz.

Fakat paint mesajı bazı durumlarda oluşturulmamaktadır. Pencere taşınırken windows pencere içindeki görüntüyü pencere ile birlikte taşır. Menüler açılır kapanırken pencerenin görünmeyen bir kısmı görünür bir hale geldiği halde yine paint mesajı gelmememektedir. Çünkü bozulan görüntüyü menü sınıfı kendisi tutup basmaktadır.

ResizeRedraw şöyle etki göstermektedir.

1. Property false ise(default durum) pencere küçültüldüğünde paint mesajı oluşmaz. Fakat büyürtüldüğünde paint mesajı oluşur. Ancak güncelleme alanı (update region) yalnızca genişletilmiş olan bölge olur.
2. Eğer bu property true ise bu durumda pencere daraltılsa da genişletilse de paint mesajı oluşur ve güncelleme alanı tüm çalışma alanı olur.

Denetim masası ayarlarında “sürüklerken pencere içeriğini görüntüle” modundaysak (xp ve vista da bu durum defaulttur.) Görünmeyen kısım görünür hale getirilirken bir tane değil bir dizi mesaj gönderilir. Yani sürüklerken sürükleme boyunca mesaj gönderilir. Biz herhangi bir yerde ve herhangi bir zamanda çizim yaparsak tüm bu çizim kayıtlarını tutup paint mesajı geldiğinde yeniden yapmalıyız. Ya da daha pratik olarak tüm çizimleri paint te yapabiliriz. Genellikle uygulanan teknik tüm çizimlerin paint mesajında yapılmasıdır.

Çizim işlemleri için Graphics sınıfı kullanılmaktadır. Graphics sınıfının Drawxxx ve Fillxxx

biçiminde bir gurup fonksiyonu vardır. Bu fonksiyonlar çizimleri yapar. Graphics nesnesi new operatörüyle yaratılamaz. Çünkü tasarımcı bunu istememiştir. Sınıfın başlangıç fonksiyonunu sınıfın private bölümüne yerleştirmiştir. Graphics nesnesi elde etmenin 2 yolu vardır.

1. Control sınıfının CreateGraphics fonksiyonunu çağırmak.
2. Paint event i oluştuğunda framework event fonksiyonuna mesaj parametre sınıfı olarak PaintEventArgs nesnesi geçirir. Bu nesnenin Graphics isimli property elemanı yaratılmış Graphics nesnesini verir.

Özetle Paint event i oluştuğunda graphics nesnesini biz yaratmayız. Graphics nesnesi zaten yaratılmış olarak bize verilir.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace PaintMessage
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }

      private void Form1_Paint(object sender, PaintEventArgs e)
      {
         MessageBox.Show("Paint");
      }
   }
}
```

Graphics nesnesi nispeten kıt bir kaynaktır. Dolayısıyle bunun çöp toplayıcıya bırakılmadan sisteme iade edilmesi uygun olur. Eğer Graphics nesnesini biz CreateGraphics fonksiyonu ile yaratmışsak işimiz bitince Dispose fonksiyonunu çağırmalıyız. Yok eğer Graphics nesnesi Framework tarafından yaratılıp PaintEvent fonksiyonuna geçirilmişse bizim bir şey yapmamıza gerek yoktur. Programdan çıkıldığında framework zaten Dispose yapmaktadır.

Tüm çizimlerin Paint mesajında yapılması ve küçük bir böllge yüzünden tüm çizimlerin yeniden yapılması normal bir durummudur? Örneğin biz Paint mesajında yüzlerce çizim yapmış olalım ve pencerenin çok küçük bir alanı görünür hale gelsin. Paint mesajı geldiğinde aslında bizim yalnızca bozulan bölgeyi çizmemiz yeterli olacaktır. Her nekadar bozulan bölgeyi elde edebilirsekte yalnızca orayı çizmek mümkün olmayabilir. Çünkü fonksiyonlar tüm çizim yapmak için oraya yerleştirilmiştir.

Windows ister alt pencere olsun ister ana pencere olsun her pencere için güncelleme alanı(update region) denile dikdörtgensel bir bölge tutar. Pencerenin görünmeyen bir kısmı görünür hale geldiğinde güncelleme alanı güncellenir ve windows belli periyotlarda güncelleme alanlarını inceleyerek güncelleme alanı boş küme olmayan pencerelere paint mesajı gönderir.

Güncelleme alanının boş küme olması demek pencerenin hiçbir bölümünün görüntü olarak

bozulmamış olması demektir.

Graphics sınıfının bir çizim fonksiyonunu ele alalım. Çizim fonksiyonları aslında iki aşamada işlemini yapmaktadır. Önce ekrana basılacak noktalar hesapla elde edilir.(Rendering) Sonra pixeller basılır. Pixel basma işlemi noktaların hesaplama işlemine göre çok yavaştır. İşte biz paint mesajında yüzlerce çizim fonksiyonu çağırsak bile burada ciddi bir zaman kaybı oluşmayacaktır. Çünkü çizim fonksiyonları eğer basılmak istenen pixel güncelleme alanı içinde kalıyorsa onları gerçekten basmak demektir.

Anahtar Notlar: Çeşitli söylentilere göre microsoft pencere içindeki görüntüyü saklamayıp bunun yerine paint mesajı gönderme stratejisini uzun dönemde değiştirmek istemektedir. Gerçektende Vista da görüntüyü saklayıp geri basma mekanizması daha güçlendirilmiştir. Vista default ayarlarda başka bir pencerenin pencereyi kaplayıp açılması durumunda görüntüyü otomatik tutup basmaktadır.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace PaintMessage
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         this.ResizeRedraw = true;


      }

      private void Form1_Paint(object sender, PaintEventArgs e)
      {
         Graphics g = e.Graphics;

         g.DrawLine(Pens.Red, 0, 0, this.ClientSize.Width, this.ClientSize.Height);
      }

      private void Form1_MouseDown(object sender, MouseEventArgs e)
      {

      }

      private void Form1_MouseMove(object sender, MouseEventArgs e)
      {

      }
   }
```

}

Çizim Nesneleri: En önemli iki çizim nesnesi kalem ve fırçadır. Çizgiler kalemle çizilir. Boyamalar fırçayla yapılır.

Kalem(Pen) nesnesinin en önemli üç property si rengi, çizim biçimi ve kalınlığıdır. Fırçaların ise en önemli property leri rengi ve desenidir. Örneğin biz bir resimden fırça yapabiliriz. Bu durumda boyama yaptığımızda aslında o resim çizilir.

Kalem nesnesi Pen sınıfıyla temsil edilmiştir. Bir kalem nesnesini yaratabilmek için Pen sınıfının başlangıç fonksiyonları kullanılabilir.

Pen sınıfının başlangıç fonksiyonu 1 birimlik istenilen renkte kalem yaratır. Color ve Float parametreli başlangıç fonksiyonu belirlenen renkte istenilen kalınlıkta kalem oluşturur. Pen sınıfının Color ve Witdh property elemanları da bu özelliklerin değiştirilmesinde kullanılabilir.

Pen sınıfının PenType isimli property elemanı PenType isimli enum türündendir ve kalemin desen sitilini belilemekte kullanılır. DashStyle propert elemanı ise kalemin biçimini belirlemek için kullanılır.

Anahtar Notlar: Bir sınıfın pek çok özelliği söz konusu olabilir. Tüm bu özelliklerin sınıfın başlangıç fonksiyonunda belirlenmesi mümkün olmadığı için tasarımcı en öenmli bazı özelliklerin başlangıç fonksiyonunda belirlenmesini diğerlerinin propert yoluyla set edilmesini sağlayabilir. Hatta bazen sınıfın o kadar çok özelliği söz konusu olabilirki programcı bunlardan hiçbirini başlangıç fonksiyonunu kullanıcıdan istemez. Tüm bu değerler default değerlerle set edilir. (Örneğin Button sınıfında ve TextBox sınıfında olduğu gibi)

Kalemin bitiş şekli EndCap property si ile belirlenebilir. Başlangıç şekil biçimi ise StartCap ile belirlenebilir. Eğer istenirse başlangıç ve bitiş şekilleri programcının istediği desne şeklinde ayarlanabilir. Bu işlemler CustomStartCap ve CustomEndCap propertyleri ile belirlenebilir.

Anahtar Notlar: Menüler ve araç çubukları Framework 2.0 da birer pencere oldukları için ana pencerenin çalışma alanının üzerinde görüntülenmektedir. Bu durumda çalışma alanının sol üst köşesi menü alanının altında kalmaktadır. Bu durum istenmeyen bir durumdur. Bunun için öenrilen tipik bir yöntem bir panel penceresi yaratıp bunu çalışma alanını kaplar hale getirmek ve çizimide panel üzerinde yapmaktır.

İki çizgi uc uca getirildiğinde eğer kalem kalın ise bunlar birbirini kapatmayabilir. Sınıfın LineJoin property elemanı bu amaçla kullanılmaktadır. Ayrıca çizgili kalem biçiminde çizgilerin uzunluklarıda DashPattern property si ile ayarlananbilmektedir. Pen sınıfının diğer elemanları MSDN dökümanlarından izlenebilir.

Ayrıca Pens isimli bir sınıfın bir kalınlıkta kalem veren pek çok static propertyleri vardır.(Kalınlık=Pixel) Yani bizw bir kalem gerektiği zaman acil olarak onu Pens sınıfıyla sağlayabiliriz.

Fırça nesneleri boyama amaçlı kullanılır ve bu nesneler Abstract Brush sınıfından türetilmiş nesnelerle temsil edilmektedir. Brush sınıfından SolidBrush, TextureBrush ve HatchBrush gibi sınıflar türetilmiştir.

SolidBrush düz renkli fırçayı temsil etmektedir.
using System;

```
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Drawing.Drawing2D;

namespace DrawingSample
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }

      private void Form1_Paint(object sender, PaintEventArgs e)
      {

      }

      private void panel1_Paint(object sender, PaintEventArgs e)
      {
         Graphics g = e.Graphics;

         Pen pen = new Pen(Color.Red, 10);
         pen.EndCap = System.Drawing.Drawing2D.LineCap.Triangle;

         pen.DashStyle = System.Drawing.Drawing2D.DashStyle.Dash;
         g.DrawLine(Pens.MediumSeaGreen, new Point(0, 0), new Point(100, 100));

         LinearGradientBrush sb = new LinearGradientBrush(new Rectangle(100, 100, 100, 100),
Color.Yellow, Color.Red, 0.5F);

         g.FillEllipse(sb, new Rectangle(100, 100, 100, 100));
      }
   }
}
```

Temel Çizim Fonksiyonları: Graphics sınıfının Drawxxx ve Fillxxx biçiminde 2 gurup çizim fonksiyonları vardır. Drawxxx fonksiyonları bizden bir kalem ister ve şeklin ylanızca sınır çizgilerini çizer. Fillxxx ise bizden bir fırça ister ve şeklin içini boyar.

DrawLine fomksiyonları bir kalem iki nokta alarak doğru çizmektedir.

DrawEllipse ve FillEllippse fonksiyonları bir dikdörtgenin teğet ve iç teğet elipslerini çizmektedir.

DrawRectangel ve FillREctangel fonksiyonları bir dikdörtgeni çizip boyamak için kullanılırlar.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
```

```
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Drawing.Drawing2D;

namespace DrawingSample
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }

      private void Form1_Paint(object sender, PaintEventArgs e)
      {

      }

      private void panel1_Paint(object sender, PaintEventArgs e)
      {
         Graphics g = e.Graphics;

         g.DrawRectangle(Pens.Red, new Rectangle(100, 100, 100, 100));
         g.FillRectangle(Brushes.Yellow, new Rectangle(101, 101, 99, 99));
      }
   }
}
```

Ayrıca Brushes isimli sınıf temel renklere ilişkin düz renkli fırça veren pek çok static property elemanlarına sahiptir. O halde bizden bir fırça istendiğinde biz Brushes sınıfını kullanarak bunu hemen sağlayabiliriz.

DrawLines isimli fonksiyon bir point dizisi içindeki noktaları birleştirerek kırıklı doğrular çizer.

DrawPolygon fonksiyonu DrawLinesfonksiyonu gibidir. Fakat ilk nokta ile son noktayı birleştirir. FillPolygon fonksiyonu da kapalı bir şeklin içini boyamakta kullanılır.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Drawing.Drawing2D;

namespace DrawingSample
{
   public partial class Form1 : Form
```

```
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e)
        {

        }

        private void panel1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;

            g.DrawRectangle(Pens.Red, new Rectangle(100, 100, 100, 100));
            g.FillRectangle(Brushes.Yellow, new Rectangle(101, 101, 99, 99));


            Point[] pt = { new Point(100, 150), new Point(150, 100), new Point(500, 500) };

            g.FillPolygon(Brushes.Green, pt);
        }
    }
}
```

Bir resmi çizmek için bir grup DrawImage fonksiyonu kullanılmaktadır. Örneğin PictureBox controlleri de çizimi DrawImage fonksiyonlarıyla yapmaktadır.

Uint parametreli DrawImage fonksiyonu resmi Orijinal boyutuyla bir noktadan başlayarak bütünsel olarak çizer.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Drawing.Drawing2D;

namespace DrawingSample
{
    public partial class Form1 : Form
    {
        private Image m_image;

        public Form1()
        {
            InitializeComponent();

            Environment.CurrentDirectory = @"E:\DotNetAppBasic\DrawingSample\DrawingSample";
```

```
            try
            {
                m_image = new Bitmap("forest.jpg");

            }
            catch (Exception e)
            {
                MessageBox.Show(e.Message);
            }

        }

        private void panel1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;

            g.DrawImage(m_image, new Point(100, 100));

        }
    }
}
```

Bu fonksiyonun sol üst köşe noktasını ayrı int değerlerle alan biçimi de vardır.

Rectangel parametreli DrawImage fonksiyonu resmi belirtilen dikdörtgen boyutuna getirtir.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Drawing.Drawing2D;

namespace DrawingSample
{
    public partial class Form1 : Form
    {
        private Image m_image;

        public Form1()
        {
            InitializeComponent();

            Environment.CurrentDirectory = @"E:\DotNetAppBasic\DrawingSample\DrawingSample";

            try
            {
                m_image = new Bitmap("forest.jpg");
```

```
            }
            catch (Exception e)
            {
                MessageBox.Show(e.Message);
            }

        }

        private void panel1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;

            g.DrawImage(m_image, new Rectangle(100, 100, 100, 100));

        }
    }
}
```

Point dizi parametreli DrawImage fonksiyonları 3 elemanlı bir point dizisi alır ve şekli döndürerek çizer. Dizinin birinci elemanı sol üst köşe, 2. elemanı sağ üst köşe ve 3. elemanı sol alt köşeyi belirtir. Dördüncü köşe zaten bu değerlerden elde edilebilmektedir.

Anahtar Notlar: Kontrol sınıfının ResizeRedraw property elemanı protected bölümdedir. Bu nednele Maalesef form üzerine panel açtığımızda panelin ResizeRedraw property sini set edemeyiz. Bu işlemi yapabilmek için mecburen panel sınıfından bir sınıf türetmemiz gerekir.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Drawing.Drawing2D;

namespace DrawingSample
{
    public partial class Form1 : Form
    {
        private Image m_image;

        public Form1()
        {
            InitializeComponent();


            Environment.CurrentDirectory = @"E:\DotNetAppBasic\DrawingSample\DrawingSample";

            try
            {
```

```
            m_image = new Bitmap("forest.jpg");

        }
        catch (Exception e)
        {
            MessageBox.Show(e.Message);
        }

    }

    private void panel1_Paint(object sender, PaintEventArgs e)
    {
        Graphics g = e.Graphics;

        g.DrawImage(m_image, new Point[] {new Point(panel1.ClientSize.Width / 2, 0),
                new Point(panel1.ClientSize.Width, panel1.ClientSize.Height / 2),
                new Point(0, panel1.ClientSize.Height / 2)});

    }
  }

  class MainPanel : Panel
  {
    public MainPanel()
    {
        this.ResizeRedraw = true;
    }
  }
}
```

Bu DrawImage fonksiyonu şüphesiz aynı zamanda şekli belirlenen boyuta da getirmektedir.

Diğer DrawImage fonksiyonları MSDN dökümanlarından incelenmelidir.

Pencerenin çalışma alanına bir yazının yazılması aslında bir çizim faaliyetidir ve bu işlemler DrawString fonksiyonlarıyla yapılmaktadır. Başka da yazı yazan bir fonksiyon yoktur.

Yazı yazmak doğrudan font konusuyla da ilgilidir. DrawString fonksiyonları bizden bir font nesnesi ister. Yazıyı o font nesnesindeki belirlemelere göre yazar. Bir font un temel özellikleri şunlardır.

- Yazının temel karakteristiği(Typeface)
- Yazının ikincil karakteristik özellikleri(Bold, italik)
- Yazının yazı karaketrelerinin boyutu
- Diğer özellikler.

Font nesnesi Font sınıfıyla temsil edilmiştir. Bir Font nesnesi Font sınıfının başlangıç fonksiyonlarıyla yaratılır. Font sınıfının en çok kullanılan başlangıç fonksiyonlarından biri şöyledir.

```
public Font(
        string familyName,
        float emSize
        )
```

Burada 1. parametre font ailesini belirten yazıdır. 2. parametre font un büyüklüğünü belirtir. Bir punto 1/71 dpi(dot per inch) dir. Fonksiyonun birinci parametresi yanlış girilirse fonksiyon başarısız

olmaz fakat default font yüklenir.

Font sınıfının diğer bir başlangıç fonksiyonu da 3 parametreli aşağıdaki fonksiyondur.

```
public Font(
        string familyName,
        float emSize,
        FontStyle style
)
```

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace FontSample
{
   public partial class Form1 : Form
   {
      private Font m_font;

      public Form1()
      {
         InitializeComponent();

         m_font = new Font("Times new Raman", 20);
      }

      private void Form1_Paint(object sender, PaintEventArgs e)
      {
         Graphics g = e.Graphics;

         g.DrawString("This is a test...", m_font, Brushes.Red, new Point(100, 100));

      }
   }
}
```

FontStyle paramtresi FontStyle isimli bir enum türündendir. Font un Bold, İtalik, UnderLine gibi özelliklerini belirtir. Birden fazla özellik çubuk operatörü ile birleştirilebilir. Elimizde mevcut bir font varsa biz o fontun diğer özelliklerine dokunmadan italik, bold gibi bir versiyonunu elde edebiliriz. Bu işlem aşağıdaki başlangıç fonksiyonu ile sağlanabilir.

```
public Font(
        Font prototype,
        FontStyle newStyle
)
```

Font ailesi FontFamily isimli bir sınıfla temsil edilmiştir. Font sınıfının FontFamily property si bunu verir.

Örneğin:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace FontSample
{
   public partial class Form1 : Form
   {
      private Font m_font;

      public Form1()
      {
         InitializeComponent();

         m_font = new Font("Times new Roman", 50, FontStyle.Strikeout| FontStyle.Italic);
      }

      private void Form1_Paint(object sender, PaintEventArgs e)
      {
         Graphics g = e.Graphics;

         g.DrawString("This is a test...", m_font, Brushes.Red, new Point(100, 100));

      }
   }
}
```

Burada Times new Romans a ilişkin Font nesnesi elde edilmiştir.

Font sınıfı da IDisposable arayüzünü deteklemektedir. Yani kullanımından sonra dispose fonksiyonunu çağırmak iyi bir tekniktir.

Bir font sabit geişlikli ya da değişken genişlikli olabilir. Örneğin Times New Roman değişken genişlikte Courier new sabit genişlikte fontlardır. Programla editörleri hemen her zaman sabit genişlikli fontlar kullanır.

Satır satır yazı yaqzabilmek için font yüksekliğini bilmek gerekir. Font sınıfının Heigth isimli property elemanı pixel cinsinden font yüksekliğini vermektedir.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
```

```
using System.Text;
using System.Windows.Forms;

namespace FontSample
{
   public partial class Form1 : Form
   {
      private Font m_font;

      public Form1()
      {
         InitializeComponent();

         m_font = new Font("Times new Roman", 12);


         foreach (FontFamily ff in FontFamily.Families)
            comboBox1.Items.Add(ff.Name);

      }

      private void Form1_Paint(object sender, PaintEventArgs e)
      {
         Graphics g = e.Graphics;

         int y = 0;

         g.DrawString("Bu bir denemedir", m_font, Brushes.Red, new Point(0, y));
         y += m_font.Height;
         g.DrawString("Bu bir denemedir", m_font, Brushes.Red, new Point(0, y));

      }

      private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
      {
         m_font = new Font((string) comboBox1.SelectedItem, 50);
         this.Invalidate();
         m_font.Dispose();
      }
   }
}
```

Yazı yazmak için kullanılan tek fonksiyon DrawString fonksiyonudur. En temel DrawString fonksiyonu aşağıdaki gibidir.

```
public void DrawString (
         string s,
         Font font,
         Brush brush,
         float x,
         float y
```

)

Fonksiyonun x, y parametresi yazının yazılacağı noktanın sol üst köşesidir. Yani yazılacak yazı bir dikdörtgen içine alınsa bu dikdörtgenin sol üst köşesi bu koordinat olacaktır. Diğer bir DrawString fonksiyonu yazıyı belirli bir dikdörtgensel alanın içerisine yazar.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace FontSample
{
   public partial class Form1 : Form
   {
      private Font m_font;

      public Form1()
      {
         InitializeComponent();

         m_font = new Font("Times new Roman", 12);


         foreach (FontFamily ff in FontFamily.Families)
            comboBox1.Items.Add(ff.Name);

      }

      private void Form1_Paint(object sender, PaintEventArgs e)
      {
         Graphics g = e.Graphics;

         g.DrawString("Bu bir denemedir evet evet denemedir", m_font, Brushes.Black,
                  new RectangleF(100, 100, 100, 100));

      }

      private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
      {
         m_font = new Font((string) comboBox1.SelectedItem, 50);
         this.Invalidate();
         m_font.Dispose();
      }
   }
}
```

public void DrawString (

```
        string s,
        Font font,
        Brush brush,
        RectangleF layoutRectangle
)
```

Fonksiyonun bu versiyonu sözcüklerden sarmalama yapmaktadır.

Yukarıdaki iki fonksiyonun StringFormat parametreli versiyonu da vardır.

```
public void DrawString (
        string s,
        Font font,
        Brush brush,
        PointF point,
        StringFormat format
)
```

Bu fonksiyon belirtilen noktaya göre hizalama yapmaktadır. StringFormat sınıfının Alignment ve LineAlignment property leri StringAlignment isimli Enum türündendir. Bu durumda örneğin çalışma alanının tam ortasına şöyle bir yazı yazabiliriz.

Yukarıdaki fonksiyonun ayrıca Rectangel içeren versiyonu da vardır.

```
public void DrawString (
        string s,
        Font font,
        Brush brush,
        RectangleF layoutRectangle,
        StringFormat format
)
```

Ekranın ortasına yazma işlemi şöyle de yapılabilir.

```
g.DrawString("Deneme", m_font, Brushes.Red, thisClientRectangle, sf);
```

Diğer DrawString fonksiyonları Point-PointF farklılığı içermektedir.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace FontSample
{
   public partial class Form1 : Form
   {
      private Font m_font;
```

```
        public Form1()
        {
            InitializeComponent();

            this.ResizeRedraw = true;

            m_font = new Font("Times new Roman", 20);


            foreach (FontFamily ff in FontFamily.Families)
                comboBox1.Items.Add(ff.Name);

        }

        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;

            StringFormat sf = new StringFormat();

            sf.Alignment = StringAlignment.Center;
            sf.LineAlignment = StringAlignment.Center;

            Rectangle rect = new Rectangle(100, 100, 100, 100);
            g.DrawRectangle(Pens.Red, rect);

            g.DrawString("Deneme", m_font, Brushes.Red, rect, sf);

        }

        private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
        {
            m_font = new Font((string) comboBox1.SelectedItem, 50);
            this.Invalidate();

        }
    }
}
```

Graphics Nesnesi ve Çizim Alanı: Graphics nesnesini elde etmenin temel olarak iki yolu vardır. Birinci yol Paint mesajıdır. Paint mesajı ile bize verilen graphics nesnesini kullanarak pencerenin her yerine çizim yapamayız. Yalnızca bozulmuş olan bozulma alanına çizim yapabiliriz.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
```

```
namespace FontSample
{
   public partial class Form1 : Form
   {
      private Font m_font;
      private Random m_rand;

      public Form1()
      {
         InitializeComponent();

         m_rand = new Random();

         this.ResizeRedraw = true;

         m_font = new Font("Times new Roman", 20);


         foreach (FontFamily ff in FontFamily.Families)
            comboBox1.Items.Add(ff.Name);

      }

      private void Form1_Paint(object sender, PaintEventArgs e)
      {
         Graphics g = e.Graphics;

         SolidBrush sb = new SolidBrush(Color.FromArgb(m_rand.Next(256), m_rand.Next(256),
m_rand.Next(256)));
         g.FillRectangle(sb, this.ClientRectangle);

      }

      private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
      {
         m_font = new Font((string) comboBox1.SelectedItem, 50);
         this.Invalidate();

      }
   }
}
```

Fakat Graphics nesnesi CreateGraphics fonksiyonu ile elde edilirse biz o Graphics nesnesi ile pencerenin her yerine çizim yapabiliriz. Zaten Paint mesajı işlendikten sonra güncelleme alanı boş küme haline getirilmektedir. Yani Biz örneğin fare mesajlarında çizim yaptıpımızda güncelleme alanı boş küme durumunda olacaktır.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
```

```
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace FontSample
{
   public partial class Form1 : Form
   {
      private Font m_font;
      private Random m_rand;

      public Form1()
      {
         InitializeComponent();

         m_rand = new Random();

         this.ResizeRedraw = true;

         m_font = new Font("Times new Roman", 20);


         foreach (FontFamily ff in FontFamily.Families)
            comboBox1.Items.Add(ff.Name);

      }

      private void Form1_Paint(object sender, PaintEventArgs e)
      {
         Graphics g = this.CreateGraphics();

         SolidBrush sb = new SolidBrush(Color.FromArgb(m_rand.Next(256), m_rand.Next(256),
m_rand.Next(256)));
         g.FillRectangle(sb, this.ClientRectangle);

         g.Dispose();

      }

      private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
      {
         m_font = new Font((string) comboBox1.SelectedItem, 50);
         this.Invalidate();

      }
   }
}
```

Bitmap'a Çizim Yapmak: Paint mesajından elde ettiğimiz Graphics nesnesi ya da CreateGraphics fonksiyonu ile elde ettiğimiz Graphics nesnesi ekrana çizim yapmak için kullanılır. Eğer Graphics sınıfının FromImage static fonksiyonu ile birGraphics nesnesi elde edilirse biz bu Graphics nesnesi

ile çizim yaptığımızda ekrana değil ilgili resmin üzerine çizim yapılır.

```
public static Graphics FromImage (
        Image image
)
```

Örneğin:

```
Bitmap bmp = new Bitmap(1000, 1000);
Graphics g = Graphics.FromImage(bmp);

g.FillRectangle(Brushes.Red, new Rectangle(0, 0, 1000, 1000));
//...
g.Dispose();
```

Şekil Taşıma İşlemleri: Daha önceden kontrolleri taşımıştık. Kontroller kendi çizimlerini yaptıkları için yani birer pencere oldukları için taşıma işlemi problemsizdir. Control ün Location property sinin güncellenmesi yeterlidir. Halbuki kendi çizdiğimiz bir dikdörtgeni ve Elipsi taşırken eski şeklin silinmesi şeklin taşınan bölgede yeniden çizilmesini gerektirir. Bir şeklin silinmesi demek onun zemin rengiyle yeniden çizilmesi demektir.

Bir pencerenin görünmeyen bir kısmının görünür hale geldiğini düşünelim .net PaintEvent fonksiyonunu çağırmadan önce pencerenin bozulmuş olan güncelleme alanını belirlenen zemin rengiyle boyamaktadır. Yani akış Paint fonksiyonuna geldiğinde bozulmuş alanın zemini boyanmış durumdadır. Yani bozulmuş alanın üzerindeki çizimler de kaybolmuş durumdadır. Biz Paint mesajında çizim yaparken zeminle hiç uğraşmadan doğrudan şekilleri çizebiliriz. Çünkü zaten zemin o noktada yenidem boyanmış durumdadır.

Anahtar Notlar: Pens sınıfının yanısıra SystemPens isimli bir sınıf da vardır. Bu sınıf da static elemanlardan oluşmuştur. Bu sınıfı bize denetim masası ayarlarına göre bir kalınlıkta renkli kalem vermektedir. Benzer biçimde SystemBrushes sınıfı da denetim masası renklerine göre bize fırça verir.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace DraggingShape
{
   public partial class Form1 : Form
   {
      private Rectangle m_rect;
      private Point m_prevPoint;
      private bool m_leftFlag;

      public Form1()
      {
         InitializeComponent();
```

```
        m_rect = new Rectangle(100, 100, 100, 100);

    }

    private void Form1_MouseMove(object sender, MouseEventArgs e)
    {
        if (m_leftFlag)
        {
            Graphics g = this.CreateGraphics();

            int deltaX = e.X - m_prevPoint.X;
            int deltaY = e.Y - m_prevPoint.Y;

            g.FillEllipse(SystemBrushes.Window, m_rect);
            m_rect.Offset(deltaX, deltaY);
            g.FillEllipse(Brushes.Red, m_rect);

            m_prevPoint = e.Location;

            g.Dispose();
        }

    }

    private void Form1_MouseDown(object sender, MouseEventArgs e)
    {
        if (e.Button == MouseButtons.Left)
        {
            if (m_rect.Contains(e.Location))
            {
                m_leftFlag = true;
                m_prevPoint = e.Location;
            }
        }
    }

    private void Form1_MouseUp(object sender, MouseEventArgs e)
    {
        if (e.Button == MouseButtons.Left)
        {
            m_leftFlag = false;
        }
    }

    private void Form1_Paint(object sender, PaintEventArgs e)
    {
        Graphics g = e.Graphics;

        g.FillEllipse(Brushes.Red, m_rect);
    }
}
```

```
}
```

Invalidate Fonksiyonlar: Control sınıfının invalidate fonksiyonları paint mesajını programlama yoluyla oluşturmak için kullanılır. Parametresiz invalidate fonksiyonu tüm çalışma alanını geçersiz hale getirir. Yani sanki pencerenin tamamı kapalıymış dasonra görünür hale gelmiş gibi bir etki yaratır. Tabi bu durumda paint mesajı geldiğinde paint mesajı içinde yapılan çizimler pencerenin her yerine yapılır. Bu durumda yavaşlığa yol açabilir. Rectangle parametreli invalidate fonksiyonu verilen rectangle ı geçersiz hale getirerek paint mesajı oluşturur. Ayrıca bool parametreli invalidate fonksiyonları da vardır. Bu fonksiyonlar alt pencerelerinde geçersiz hale getirilp getirlimeyeceğini belirtir.

Tüm Çizimlerin Paint Mesajında Yapılması: Uygulanan klasik teknik tüm çizimlerin paint mesajında yapılmasıdır. Çizim dışarıda yapılacak olsa bile yapılacak çizimin dataları saklanır sonra invalidate fonksiyonu çağrılır. Çizim yine paint mesajında yapılmış olur.

Invalidate fonksiyonu çağrıldığında zaten geçersiz hale getirilmiş zemin paint mesajı oluşturulup event fonksiyonu çağrılmadan önce zemin rengiyle boyanmış durumdadır. Yani bizim geçersiz bölgeyi silmemize gerek yoktur. O bölge zaten paint mesajı dolayısıyla silinmektedir.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace DraggingShape
{
   public partial class Form1 : Form
   {
      private Rectangle m_rect;
      private Point m_prevPoint;
      private bool m_leftFlag;

      public Form1()
      {
         InitializeComponent();

         m_rect = new Rectangle(100, 100, 100, 100);

      }

      private void Form1_MouseMove(object sender, MouseEventArgs e)
      {
         if (m_leftFlag)
         {
            Graphics g = this.CreateGraphics();

            int deltaX = e.X - m_prevPoint.X;
            int deltaY = e.Y - m_prevPoint.Y;
```

```
            m_rect.Offset(deltaX, deltaY);

            this.Invalidate();

            m_prevPoint = e.Location;

            g.Dispose();
        }

    }

    private void Form1_MouseDown(object sender, MouseEventArgs e)
    {
        if (e.Button == MouseButtons.Left)
        {
            if (m_rect.Contains(e.Location))
            {
                m_leftFlag = true;
                m_prevPoint = e.Location;
            }
        }
    }

    private void Form1_MouseUp(object sender, MouseEventArgs e)
    {
        if (e.Button == MouseButtons.Left)
        {
            m_leftFlag = false;
        }
    }

    private void Form1_Paint(object sender, PaintEventArgs e)
    {
        Graphics g = e.Graphics;

        g.FillEllipse(Brushes.Red, m_rect);
    }
  }
}
```

Tüm çizimlerin paint mesajında yapıldığı durumda geçersiz hale getirilmek istenen alanın minimal düzeyde tutulması en etkin yöntemdir. Bu hesabın yapılması bazen zor olabilmektedir. Kesin bir hesap yerine kaba bir hesap yapılabilir.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
```

```
using System.Text;
using System.Windows.Forms;

namespace DraggingShape
{
   public partial class Form1 : Form
   {
      private Rectangle m_rect;
      private Point m_prevPoint;
      private bool m_leftFlag;
      private Rectangle m_testRect;

      public Form1()
      {
         InitializeComponent();

         m_rect = new Rectangle(100, 100, 100, 100);

         m_testRect = new Rectangle(0, 0, 100, 100);

      }

      private void Form1_MouseMove(object sender, MouseEventArgs e)
      {
         if (m_leftFlag)
         {
            Graphics g = this.CreateGraphics();

            int deltaX = e.X - m_prevPoint.X;
            int deltaY = e.Y - m_prevPoint.Y;

            Rectangle temp = m_rect;

            m_rect.Offset(deltaX, deltaY);

            Rectangle invalidateRect = Rectangle.Union(temp, m_rect);

            this.Invalidate(invalidateRect);

            m_prevPoint = e.Location;

            g.Dispose();
         }

      }

      private void Form1_MouseDown(object sender, MouseEventArgs e)
      {
         if (e.Button == MouseButtons.Left)
         {
            if (m_rect.Contains(e.Location))
            {
```

```
                m_leftFlag = true;
                m_prevPoint = e.Location;
            }
        }
    }

    private void Form1_MouseUp(object sender, MouseEventArgs e)
    {
        if (e.Button == MouseButtons.Left)
        {
            m_leftFlag = false;
        }
    }

    private void Form1_Paint(object sender, PaintEventArgs e)
    {
        Graphics g = e.Graphics;

        g.FillEllipse(Brushes.Red, m_rect);

        g.FillRectangle(Brushes.Yellow, m_testRect);
    }
  }
}
```

Titreme Problemi: Tüm çizimlerin paint mesajında yapıldığını varsayalım. Bir şekli fareyle sürüklediğimizde invalidate fonksiyonunu çağırdığımız zaman önce zemin, zemin rengiyle boyanacak sonra paint fonksiyonu çalışacaktır. Bu durumda şeklin eski konumuyla yeni konumunun kesişim alanı önce zemin rengiyle boyanıp sonra yeniden şekil rengiyle boyanacaktır. Bu durum titremeye(flickering) yol açar.

Titreme probleminin klasik çözümü çizimlerin önce bir bitmap e yapılması(Offscreen) sonrapaint mesajı geldiğinde o bitmap in DrawImage fonksiyonuyla ekrana çizilmesidir. Bitmap yöntemini mauel uygulamak için şu adımlar uygulanır.

1. Maksimum ekran boyutu kadar bir bitmap yaratılır.
2. Fareyle çizim o anda bitmap üstüne yapılır. Ya da paint mesajı geldiğinde çizim bitmap üzerine yapılır.
3. Nihayet paint mesajında bitmap in ilgili bölgesi DrawImage fonksiyonuyla ekrana çizilir.

Framework 2.0 ile birlikte yukarıda açıklanan yöntem otomatik uygulanmaya başlanmıştır. Control sınıfının bool türden protected DoubleBuffered isimli property elemanı bu işlemi otomotize etmektedir. Bu property nin default değeri flase biçimindedir. Bu property true yapılırsa şunlar gerçekleşir: Kütüphane sistemi Control sınıfının OnPaint fonksiyonunda kendi içerisinde bir bitmap nesnesi yaratır ve bu bitmap e ilişkin bir graphics nesnesi elde eder. Event fonksiyonunu bu graphics nesnesi ile çağırır. Böylece biz paint mesajında bu graphics nesnesini kullanarak yaptığımız çizimleri aslında ekrana değil birmap e yapmış oluruz. Nihayet paint fonksiyonu bittiğinde kütüphane kendisi bu bitmap deki görüntüyü geçersiz olan güncelleme alanına çizer.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
```

```
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace DraggingShape
{
   public partial class Form1 : Form
   {
      private Rectangle m_rect;
      private Point m_prevPoint;
      private bool m_leftFlag;
      private Rectangle m_testRect;

      public Form1()
      {
         InitializeComponent();

         this.DoubleBuffered = true;

         m_rect = new Rectangle(100, 100, 100, 100);

         m_testRect = new Rectangle(0, 0, 100, 100);

      }

      private void Form1_MouseMove(object sender, MouseEventArgs e)
      {
         if (m_leftFlag)
         {
            Graphics g = this.CreateGraphics();

            int deltaX = e.X - m_prevPoint.X;
            int deltaY = e.Y - m_prevPoint.Y;

            Rectangle temp = m_rect;

            m_rect.Offset(deltaX, deltaY);

            Rectangle invalidateRect = Rectangle.Union(temp, m_rect);

            this.Invalidate(invalidateRect);

            m_prevPoint = e.Location;

            g.Dispose();
         }

      }

      private void Form1_MouseDown(object sender, MouseEventArgs e)
```

```
        {
            if (e.Button == MouseButtons.Left)
            {
                if (m_rect.Contains(e.Location))
                {
                    m_leftFlag = true;
                    m_prevPoint = e.Location;
                }
            }
        }

        private void Form1_MouseUp(object sender, MouseEventArgs e)
        {
            if (e.Button == MouseButtons.Left)
            {
                m_leftFlag = false;
            }
        }

        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;

            g.FillEllipse(Brushes.Red, m_rect);

            g.FillRectangle(Brushes.Yellow, m_testRect);
        }
    }
}
```

Timer İşlemleri: Pek çok uygulamada bazı fonksiyonların periyodik olarak çağrılması gerekmektedir. Örneğin bir oyun programında bir şeklin belirli bir hızda hareketini sağlamak için böyle bir mekanizma gerekmektedir.

Bu tür işlemler Timer sınıflarıyla yapılmaktadır.

Kullanımı birbirine çok benzeyen farklı isim alanlarında bulunan üç timer sınıfı vardır. System.Threading isim alanındaki Timer sınıfı kendi içinde thread oluşturarak Timer mekanizmasını sağlar. Bazı uygulamalarda bu yöntem özellikle tercih edilmektedir. System.Windows.Forms isim alanı içindeki Timer sınıfı windows un WM_TIMER mesaj sistemini kullanır. Dolaysıyla hiç Thread açmaz. Bu en klasik yöntemdir. Nihayet System.Timers isim alanı içindeki Timer sınıfı da yine thread kullanmaktadır. Bu sınıf Component sınıfından türetildiği için tamamen form editörde kullanılabilmektedir. (System.Windows.Forms içinteki Timers sınıfıda Component sınıfından türetilmiştir. Dolaysıyla onun için de ToolBox desteği vardır)

Timer sınıflarının hepsinin kullanımı birbirine çok benzer. Burada System.Windows.Forms isim alanı içindeki Timers sınıfı ele alınacaktır.

Timer sınıfının Interval isimli property elemanı periyodu belirlemekte kullanılır. Default periyod 100 milisaniyedir.

Sınıfın bool türden Enabled property elemanı Timer ı aktif ya da pasif duruma getirir. Default değer False biçimdedir. Yani Timer nesnesi yaratıldığında henüz işlev görmemektedir. İşlemin başlatılması için ya Enabled property si true yapılır ya da start fonksiyonu çağrılır. Benzer biçimde Enabled property sini False yapmakla stop fonksiyonunu çağırmak aynı anlamdadır. Timer sınıfının Tick isimli event elemanı EventHandler isimli delege türündendir ve çağrılacak fonksiyonu belirler.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace DigitalClock
{
   public partial class Form1 : Form
   {
      private Timer m_timer;
      private string m_strClock;
      private Font m_font;

      public Form1()
      {
         InitializeComponent();
         this.ResizeRedraw = true;

         this.DoubleBuffered = true;

         m_timer = new Timer();
         m_timer.Interval = 1000;
         m_timer.Tick += new EventHandler(m_timer_Tick);
         m_timer.Start();
         m_strClock = "";

         m_timer_Tick(null, null);


         m_font = new Font("Times New Roman", 100);
      }

      void m_timer_Tick(object sender, EventArgs e)
      {
         DateTime dt = DateTime.Now;
         m_strClock = string.Format("{0:D2}:{1:D2}:{2:D2}", dt.Hour, dt.Minute, dt.Second);

         this.Invalidate();
      }

      private void Form1_Paint(object sender, PaintEventArgs e)
      {
```

```
            Graphics g = e.Graphics;

            StringFormat sf = new StringFormat();

            sf.Alignment = StringAlignment.Center;
            sf.LineAlignment = StringAlignment.Center;

            g.DrawString(m_strClock, m_font, Brushes.Red, this.ClientRectangle, sf);

        }
    }
}

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace DigitalClock
{
    public partial class Form1 : Form
    {
        private Timer m_timer;
        private string m_strClock;
        private Font m_font;

        public Form1()
        {
            InitializeComponent();
            this.ResizeRedraw = true;

            this.DoubleBuffered = true;

            m_timer = new Timer();
            m_timer.Interval = 1000;
            m_timer.Tick += new EventHandler(m_timer_Tick);
            m_timer.Start();

            m_font = new Font("Times New Roman", 100);

            m_strClock = "99:99:99";

            Graphics temp = this.CreateGraphics();

            SizeF sf = temp.MeasureString(m_strClock, m_font);

            this.MinimumSize = new Size((int) sf.Width + 10, (int) sf.Height + 10);
```

```
            temp.Dispose();



            m_timer_Tick(null, null);



        }

        void m_timer_Tick(object sender, EventArgs e)
        {
            DateTime dt = DateTime.Now;
            m_strClock = string.Format("{0:D2}:{1:D2}:{2:D2}", dt.Hour, dt.Minute, dt.Second);

            this.Invalidate();
        }

        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;

            StringFormat sf = new StringFormat();

            sf.Alignment = StringAlignment.Center;
            sf.LineAlignment = StringAlignment.Center;

            g.DrawString(m_strClock, m_font, Brushes.Red, this.ClientRectangle, sf);

        }
    }
}
```

Region ve GraphicsPath İşlemleri: Region çalışma alanı içindeki çizim için kullanılabilecek bir bölgeyi belirtir. GraphicsPath kavramı Region kavramından daha sonra onun işlevini genişletecek biçimde tasarlanmıştır. Programcı önce Region Ya da GraphicsPath tanımlamasını yapar sonra bu tanımlanmış alan üzerinde çeşitli işlemleri yapar.

Bir Region Region sınıfıyla temsil edilmektedir. Region nesnesi default başlangıç fonksiyonu ile yaratılabilir. Bir GraphicsPath Regiona dönüştürülebilir. Ya da bir dikdörtgensel bölgeden Region oluşturulabilir.

Region sınıfının bazı faydalı fonksiyonları vardır. Örneğin intersect fonksiyonu Region nın belirttiği alanla verilen dikdörtgenin kesişim alanını belirler Regionu bu kesim alanı haline dönüştürür. Ya da iki Region nın kesişim kümesinden bir Region elde edilebilir. Benzer biçimde sınıfın Union fonksiyonları kesişim yerine birleşim işlemi yapar. XOR fonksiyonları da kesişim dışındaki alanı Region yapar. Region sınıfı System.Drawing isim alanı içindedir.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
```

```
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace RegionSample
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();

      }

      private void Form1_Paint(object sender, PaintEventArgs e)
      {
         Graphics g = e.Graphics;

         Region reg1 = new Region(new Rectangle(100, 100, 100, 100));
         Region reg2 = new Region(new Rectangle(150, 150, 100, 100));

         reg1.Union(reg2);


         g.FillRegion(Brushes.Red, reg1);

      }
   }
}
```

GraphicsPath işlemleri de GraphicsPath sınıfıyla temsil edilmiştir. Bu sınıfı System.Drawing.Drawing2D isim alanı içindedir. GraphicsPath sınıfı daha gelişkin bir Region kavamı sunmaktadır. Bu sınıf sayesinde düzgün olmayanbölgeler tanımlayabiliriz. Sınıfın bir gurup Addxxx fonksiyonu vardır. Her Add fonksiyonu çağrıldıkça bölgeye yeni eleman eklenir. CloseFigure fonksiyonu son eklenen elemanla ilk noktayı birleştirerek şekli kapatır. Tabi bu fonksiyon kapalı olmayan çizimlerde önemlidir.

Bir GraphicsPath Region biçimine dönüştürülebilir. Graphics sınıfının nasıl FillRegion DrawRegion gibi fonksiyonları varsa DrawPath, FillPath gibi fonksiyonları da vardır.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Linq;
using System.Text;
using System.Windows.Forms;
```

```
namespace RegionSample
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();

      }

      private void Form1_Paint(object sender, PaintEventArgs e)
      {
         Graphics g = e.Graphics;

         GraphicsPath gp = new GraphicsPath();

         gp.AddLine(50, 300, 100, 200);
         gp.AddArc(new Rectangle(100, 100, 100, 100), 135, 270);
         gp.AddLine(200, 200, 250, 300);
         gp.CloseFigure();


         g.FillPath(Brushes.Red, gp);
         g.DrawPath(Pens.Black, gp);
       }
   }
}
```

Bir noktanın bir Region nın veya bir GraphicsPath içinde olup olmadığını anlamak için Region sınıfının ve GraphicsPath sınıfının isVisible fonksiyonları kullanılır(Hit Testing).

Region ve GraphicsPath sınıflarının Translate fonksiyonları bölgeyi belirli bir miktar ötelemek için kullanılır. Böylelikle belirli bir Region ya da GraphicsPath nesnesini taşıyabiliriz.

Anımsanacağı gibi Control sınıfının invalidate fonksiyonları tüm çalışma alanını veya bir bölgesini sanki görüntü bozulmuş gibi gübcelleme alanı haline getirebilir. Böylece paint mesajı oluşmakta ve önce günceleme alanının resmi silinmektedir. İşte Region parametreli invalidate fonksiyonları da vardır.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace RegionSample
```

```
{
    public partial class Form1 : Form
    {
        private bool m_flag;
        private int m_prevX, m_prevY;
        private Region m_reg;

        public Form1()
        {
            InitializeComponent();

            this.DoubleBuffered = true;

            GraphicsPath gp = new GraphicsPath();

            gp.AddLine(50, 300, 100, 200);
            gp.AddArc(new Rectangle(100, 100, 100, 100), 135, 270);
            gp.AddLine(200, 200, 250, 300);
            gp.CloseFigure();
            m_reg = new Region(gp);

        }

        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;

            g.FillRegion(Brushes.Red, m_reg);

        }

        private void Form1_MouseDown(object sender, MouseEventArgs e)
        {
            if (e.Button == MouseButtons.Left)
            {
                if (m_reg.IsVisible(e.Location))
                {
                    m_flag = true;
                    m_prevX = e.X;
                    m_prevY = e.Y;

                }
            }
        }

        private void Form1_MouseMove(object sender, MouseEventArgs e)
        {
            if (m_flag)
            {

                this.Invalidate(m_reg);
                m_reg.Translate(e.X - m_prevX, e.Y - m_prevY);
```

```
            this.Invalidate(m_reg);

            m_prevX = e.X;
            m_prevY = e.Y;

            this.Text = (e.X - m_prevX).ToString() + " " + e.X.ToString();

            this.Invalidate();
        }
    }

    private void Form1_MouseUp(object sender, MouseEventArgs e)
    {
        m_flag = false;
    }
  }
}
```

Windows aslında güncelleme alanına değil Clip alanına pikselleri yerleştirir. Clip alanı normal olarak güncelleme alanıyla aynı alandır. Fakat biz clip alanını ayrı bir biçimde tanımlayabiliriz. Bu durumda çizim paint mesajı içinde güncelleme alanı içinde kalan clip alanına yapılır. Yani çizimler paint mesajı içinde güncelleme alanıyla clip alanının kesişim bölgesine yapılmaktadır.

Graphics sınıfının Region türünden Clip isimli property elemanı Clip bölgesini belirlemede kullanılır.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace CircleImage
{
  public partial class Form1 : Form
  {
    private GraphicsPath m_gp;

    public Form1()
    {
      InitializeComponent();

      this.ResizeRedraw = true;

      m_gp = new GraphicsPath();

      m_gp.AddEllipse(this.ClientRectangle);
```

```
        }

        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;

            g.Clip = new Region(m_gp);

            try
            {
                g.DrawImage(Image.FromFile(@"E:\DotNetAppBasic\CircleImage\forest.jpg"),
this.ClientRectangle);
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        private void Form1_Resize(object sender, EventArgs e)
        {
            m_gp = new GraphicsPath();

            m_gp.AddEllipse(this.ClientRectangle);
        }
    }
}
```

Graphics sınıfının Clip isimli property elemanını set etmek yerine SetClip fonksiyonu da aynı işi görür. Graphics sınıfının IntersectClip isimli fonksiyonları mevcut clip alanına başka bir dikdörtgen ya da regionla kesiştirerek kesişimi clip alanı yapar. Benzer biçimde ExcludeClip isimli fonksiyon belirli bir region ya da dikdörtgeni clip alanından atar. ResetClip fonksiyonu Clip alanını boş küme yapar. Yani onu boşaltır. Translate Clip fonksiyonları klip alanını deltax-deltay kadar öteler.

Ekrandan görüntünün belli bir bölgeye kopyalanması: Framework 2.0 ile birlikte çok gereksinim duyulan ekran görüntüsünün elde edilmesi çok kolaylaştırılmıştır. Graphics sınıfının CopyFromScreen fonksiyonları ekranın belli bir dikdörtgensel alanını çizebilmektedir. CopyFromScreen fonksiyonlarının en çok kullanılan biçimi şöyledir. Fonksiyonun ilk iki parametresi ekran bölgesinde kopyalanmak istenen yerin sol üst köşesini belirtir. Fonksiyonun 3. ve 4. parametreleri hedef çizim alanında bir sol üst köşe belirtir. Son parametre belirlenen noktadan itibaren kopyalanacak dikdörtgenin genişlik ve uzunluğunu belirtir.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace CopyFromScreen
```

```
{
   public partial class Form1 : Form
   {


      public Form1()
      {
         InitializeComponent();
         this.ResizeRedraw = true;
      }

      private void captureToolStripMenuItem_Click(object sender, EventArgs e)
      {
         panel1.Invalidate();
      }

      private void panel1_Paint(object sender, PaintEventArgs e)
      {
         Graphics g = e.Graphics;

         g.CopyFromScreen(0, 0, 0, 0, new Size(200, 200));

      }
   }
}
```

Ekran kopyalama uygulamalarında programcı ekranın (yani masaüstünün) yatay ve düşey pixel çözünürlüğünü bilmek isteyebilir. Gelişmiş donanımlarda artık bilgisayara bağlı birden fazla ekran sözkonusu olabilmektedir. Gerçektende .net sınıf sisteminde eğer bilgisayarımıza birden fazla monitör bağlıysa bunların hepsini elde edebilmekteyiz. Fakat ekranlardan biri işletim sistemi tarafından birincil ekran olarak değerlendirilmektedir.

Screen isimli sınıf ekran kavramını temsil etmektedir. Bu sınıfın Bounds isimli property elemanı sol üst köşesi sıfır sıfır olacak biçimde ekranın pixel şözünürlüğünü rectangle olarak verir. Screen sınıfının static PimaryScreen isimli property elemanı birincil ekrana ilişkin screen nesnesini vermektedir. Yine Screen sınıfının static AllScreens isimli property elemanı tüm ekranların bilgilerini bir dizi halinde alır.

Screen sınıfının BitsPerPixel propertysi aynı zamanda ekranın renk çözünürlüğünü vermektedir.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace CopyFromScreen
{
   public partial class Form1 : Form
```

```
    {


        public Form1()
        {
            InitializeComponent();
            this.ResizeRedraw = true;

        }

        private void captureToolStripMenuItem_Click(object sender, EventArgs e)
        {
            panel1.Invalidate();
        }

        private void panel1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;

            g.CopyFromScreen(0, 0, 0, 0, new Size(Screen.PrimaryScreen.Bounds.Width,
Screen.PrimaryScreen.Bounds.Height));


        }
    }
}
```

Screen sınıfının WorkingArea isimli property elemanı masaüstünün kullanılabilir bölgesini rectangle olarak vermektedir. Yani bu verilen alanda görev çubuğu ve diğer çerçevesel farklar çıkartılmaktadır.

Ekran görüntüsünü bir bitmap olarak save etmek de oldukça kolaydır. Yapılacak işlem yeni bir bitmap yaratmak bu bitmap için bir graphics nesnesi elde etmektir.

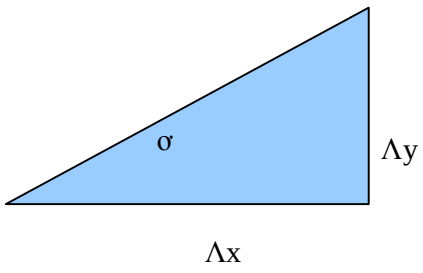
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace CopyFromScreen
{
    public partial class Form1 : Form
    {


        public Form1()
        {
```

```
            InitializeComponent();
            this.ResizeRedraw = true;

        }

        private void captureToolStripMenuItem_Click(object sender, EventArgs e)
        {
            Bitmap bmp = new Bitmap(Screen.PrimaryScreen.Bounds.Width,
                                    Screen.PrimaryScreen.Bounds.Height);
            Graphics g = Graphics.FromImage(bmp);
        }

        private void panel1_Paint(object sender, PaintEventArgs e)
        {


        }
    }
}

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace CopyFromScreen
{
    public partial class Form1 : Form
    {
        private Bitmap m_bmp;

        public Form1()
        {
            InitializeComponent();
            this.ResizeRedraw = true;

        }

        private void captureToolStripMenuItem_Click(object sender, EventArgs e)
        {
            m_bmp = new Bitmap(Screen.PrimaryScreen.Bounds.Width,
                               Screen.PrimaryScreen.Bounds.Height);
            Graphics g = Graphics.FromImage(m_bmp);

            g.CopyFromScreen(0, 0, 0, 0, new Size(Screen.PrimaryScreen.Bounds.Width,
                               Screen.PrimaryScreen.Bounds.Height));

            panel1.Invalidate();
```

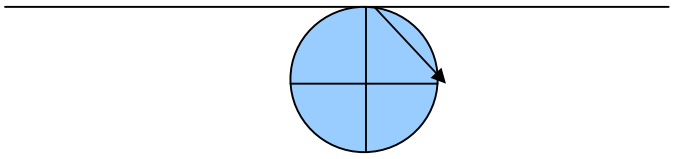
```
        }

        private void panel1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;

            if (m_bmp != null)
                g.DrawImage(m_bmp, panel1.ClientRectangle);

        }
    }
}
```

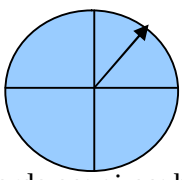
Yansıyan Top Örneği: Tula kırmaca gibi, tenis gibi pek çok oyunda bir top hareket etmekte bir engelle karşılaştığında çarparak uzaklaşmaktadır. Top engele çarpıca düzgün yansıması işlemi basit trigonometrik bir hesapla yapılabilir. Topun başlangıçta bir gidiş açısı vardır. Bu açı yansımadan sonra değiştirilir.

Topun yatayla alfa açısıyla hareket ettiğini düşünelim bu açıda topun a birim hareket edebilmesi için deltax ve deltay uzunlukları şöyle olmalıdır.

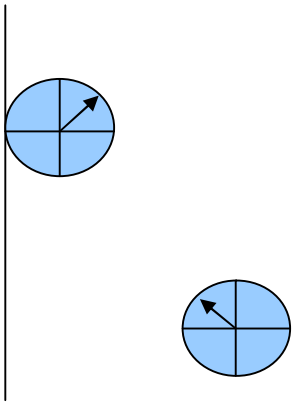


O halde topun hareketi için başlangıçta bir alfa açısı ve a değeri tespit edililir. Sonra timer belirli bir periyoda kurulur. Her timer mesajı geldiğinde top deltax, deltay miktarı kadar hareket ettirilir. Şekil düşey ya da yatay engelle karşılaştığında açı değiştirilir. Yatay çarpmalarda açı negatif değere çekilmelidir.





Düşey çarpışmalarda açı pi-açı biçimine gelmektedir(radyan cinsinden(2piRadyan 360 dır))



```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace ReflectingBall
{
    public partial class Form1 : Form
    {
        private Rectangle m_ball;
        private const int DefMoveUnit = 10;
        private const int Radius = 20;
        private double m_angle;

        public Form1()
        {
            InitializeComponent();
            this.DoubleBuffered = true;

            m_angle = Math.PI / 6;
            m_ball = new Rectangle(this.ClientSize.Width / 2 - Radius, this.ClientSize.Height / 2 -
Radius,
                        2 * Radius, 2 * Radius);


            timer1.Start();
        }

        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;

            g.FillEllipse(Brushes.Purple, m_ball);
```

```
        }

        private void timer1_Tick(object sender, EventArgs e)
        {
            this.Invalidate(m_ball);

            double deltaX = Math.Cos(m_angle) * DefMoveUnit;
            double deltaY = Math.Sin(m_angle) * DefMoveUnit;
            m_ball.Offset((int) deltaX, (int) deltaY);

            this.Invalidate(m_ball);
        }
    }
}
Form1
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace CSD
{
    public partial class Form1 : Form
    {
        private ReflectingBall m_rb1;
        private ReflectingBall m_rb2;
        private ReflectingBall m_rb3;

        public Form1()
        {
            InitializeComponent();
            this.DoubleBuffered = true;

            m_rb1 = new ReflectingBall(100, 100, this.ClientRectangle);
            m_rb1.Radius = 20;
            m_rb1.Angle = Math.PI / 6;
            m_rb1.Color = Color.Blue;
            m_rb1.Speed = 1;
            m_rb1.Move += new BallEventHandler(m_rb_Move);
            m_rb1.Start();

            m_rb2 = new ReflectingBall(200, 100, this.ClientRectangle);
            m_rb2.Radius = 10;
            m_rb2.Angle = -Math.PI / 6;
            m_rb2.Color = Color.Red;
            m_rb2.Speed = 80;
            m_rb2.Move += new BallEventHandler(m_rb_Move);
            m_rb2.Start();
```

```
            m_rb3 = new ReflectingBall(200, 200, this.ClientRectangle);
            m_rb3.Radius = 10;
            m_rb3.Angle = -Math.PI / 6;
            m_rb3.Color = Color.Purple;
            m_rb3.Speed = 50;
            m_rb3.Move += new BallEventHandler(m_rb_Move);
            m_rb3.Start();
        }

        void m_rb_Move(object o, BallEventArgs bea)
        {
            this.Invalidate();
        }

        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;

            m_rb1.Draw(g);
            m_rb2.Draw(g);
            m_rb3.Draw(g);
        }

        private void Form1_Resize(object sender, EventArgs e)
        {
            m_rb1.Frame = this.ClientRectangle;
            m_rb2.Frame = this.ClientRectangle;
            m_rb3.Frame = this.ClientRectangle;
        }

        private void Form1_MouseDown(object sender, MouseEventArgs e)
        {
            if (e.Button == MouseButtons.Right)
                StopBalls();
            else if (e.Button == MouseButtons.Left)
                StartBalls();
        }

        private void StartBalls()
        {
            foreach (ReflectingBall rb in ReflectingBall.Balls)
                rb.Start();
        }

        private void StopBalls()
        {
            foreach (ReflectingBall rb in ReflectingBall.Balls)
                rb.Stop();
        }
    }
}
```

Ball.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Drawing;
using System.Windows.Forms;

namespace CSD
{
    class BallEventArgs : EventArgs
    {
        private int m_x;
        private int m_y;

        public BallEventArgs(int x, int y)
        {
            m_x = x;
            m_y = y;
        }

        public int X
        {
            get { return m_x; }
        }

        public int Y
        {
            get { return m_y; }
        }
    }

    delegate void BallEventHandler(object o, BallEventArgs bea);

    class ReflectingBall
    {
        private Rectangle m_frame;
        private int m_centerX, m_centerY;
        private Color m_color;
        private int m_speed;
        private int m_radius;
        private double m_angle;
        private Timer m_timer;
        private Brush m_brush;
        private const int DefMoveUnit = 5;

        private static List<ReflectingBall> ms_balls;

        public event BallEventHandler Move;

        static ReflectingBall()
        {
```

```
    ms_balls = new List<ReflectingBall>();
}

public ReflectingBall(int centerX, int centerY, Rectangle frame)
{
    m_centerX = centerX;
    m_centerY = centerY;
    m_frame = frame;
    m_color = Color.Red;
    m_brush = new SolidBrush(m_color);
    m_speed = 50;
    m_radius = 10;
    m_angle = Math.PI / 6;

    m_timer = new Timer();
    m_timer.Interval = 1000 / m_speed;
    m_timer.Tick += new EventHandler(defaultTimerProc);

    ms_balls.Add(this);

}

public static List<ReflectingBall> Balls
{
    get { return ms_balls; }
}

public Rectangle Frame
{
    get { return m_frame; }
    set { m_frame = value; }
}

public Point Center
{
    get { return new Point(m_centerX, m_centerY); }
    set { m_centerX = value.X; m_centerY = value.Y; }
}

private void defaultTimerProc(object o, EventArgs e)
{
    this.Move(this, new BallEventArgs(m_centerX, m_centerY));

}

public void Draw(Graphics g)
{
    if (m_centerX + m_radius > m_frame.Right ||
                m_centerX - m_radius < m_frame.Left)
        m_angle = Math.PI - m_angle;

    if (m_centerY + m_radius > m_frame.Bottom ||
```

```
                    m_centerY - m_radius < m_frame.Top)
            m_angle = -m_angle;

        double deltaX = Math.Cos(m_angle) * DefMoveUnit;
        double deltaY = Math.Sin(m_angle) * DefMoveUnit;
        m_centerX += (int)deltaX;
        m_centerY += (int)deltaY;

        g.FillEllipse(m_brush, m_centerX - m_radius, m_centerY - m_radius, m_radius * 2,
m_radius * 2);

    }

    public void Start()
    {
        m_timer.Start();
    }

    public void Stop()
    {
        m_timer.Stop();
    }

    public Color Color
    {
        get { return m_color; }
        set
        {
            m_color = value;
            m_brush = new SolidBrush(m_color);
        }
    }

    public int Speed
    {
        get { return m_speed; }
        set
        {
            m_speed = value;
            m_timer.Interval = 1000 / m_speed;
        }
    }

    public int Radius
    {
        get { return m_radius; }
        set { m_radius = value; }
    }

    public double Angle
    {
        get { return m_angle; }
```

```
        set { m_angle = value; }
      }

   }
}
```

Komut Satırı Argümanları: Komut satırından program çalıştırırken programın isminin yanına yazılan yazılara komut satırı argümanları denilmektedir. Örneğin.
Test.exe ali veli selami:

Burada programın ismi test.exe dir. Ali veli selami komur satırı argümanlarıdır.

C# standartlarına göre main fonksiyonunun parametrik yapısı aşağıdakilerden biri biçiminde olabilir.

```
static void Main() {...}
static void Main (string[] args ){...}
static int Main (){...}
static int Main (string[] args ){...}
```

Görüldüğü gibi Main fonksiyonun geri dönüş değeri void ya da int olabilir. Parametresi olmayabilir ya da String dizisi olabilir. String dizisinin ilk elemanı program isminden sonraki ilk komut satırı argümanını belirtir.

Programcı isterse tüm komut satırı argümanlarını tek bir yzı biçiminde de alabilir. Bunun için Environment sınıfının String türünden static CommentLine property si kullanılabilir. Bu property bize program ismi ile birlikte bize tüm komut satırını tek bir string olarak vermektedir. Ayrıca Environment sınıfının GetCommentLneArgs isimli static fonksiyonu bize program ismi de dahil olmak üzere komut satırı argümanlarını bize bir string dizisi olarak vermektedir.

Komut satırı argümanları ayrıca proje propertylerinde debug sekmesinde IDE den de girilebilir.

System.Environment Sınıfı: Bu sınıf static bir sınıftır yani hep static elemanlardan oluşmaktadır. Bu sınıfta hem çalışmakta olan uygulamaya yönelik hm de bazı sisteme yönelik işlemler yapan elemanlar bulunmaktadır. Sınıfın faydalı tipik elemanları şunlardır.

- Yukarıda söylenildiği gibi komut satırı argümanlarını alan property ve fonksiyonlar vardır.
- Prosesin çevredeğişkenlerini get eden ve set eden fonksiyonlar vardır.
- Gerek konsol gerekse GUI uygulamalarında programı sonlandırmak için Exit fonksiyonu kullanılabilir.
- Sistemdeki tüm sürücülerin listesi GetLogicalDrives fonksiyonuyla elde edilebilir.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Drawing;

namespace ConsoleApp
{
   class Program
   {
      public static int Main()
```

```
        {
            foreach (string s in Environment.GetLogicalDrives())
                Console.WriteLine(s);

            return 0;
        }
    }

}
```

- Sınıfın CurrentDirectory isimli property elemanı prosesin aktif çalışma dizinini alıp set etmekte kullanılır.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Drawing;

namespace ConsoleApp
{
    class Program
    {
        public static int Main()
        {
            string cmd;

            for (; ; )
            {
                Console.Write("{0}>", Environment.CurrentDirectory);
                cmd = Console.ReadLine();
            }

            return 0;
        }
    }

}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Drawing;
using System.IO;

namespace ConsoleApp
{
    class Program
    {
        public static int Main()
        {
            string cmd;
```

```
            string [] cmdParams;

            for (; ; )
            {
                Console.Write("{0}>", Environment.CurrentDirectory);
                cmd = Console.ReadLine();
                cmdParams = cmd.Split(' ', '\t');
                if (cmdParams.Length == 0)
                    continue;
                switch (cmdParams[0])
                {
                    case "cd":
                        try
                        {
                            Environment.CurrentDirectory = cmdParams[1];
                        }
                        catch (DirectoryNotFoundException e)
                        {
                            Console.WriteLine("System cannot find specified directory.\n");
                        }
                        break;
                    case "exit":
                        goto EXIT;

                }
            }

        EXIT:

            return 0;
        }
    }

}
```

- Ağdaki bilgisayar ismi MachineName propertysi ile elde edilebilir.
- Sınıfın diğer elemanları MSDN dökümanlarından izlenmelidir.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Drawing;
using System.IO;

namespace ConsoleApp
{
    class Program
    {
        public static int Main()
        {
            string cmd;
            string [] cmdParams;
```

```
        for (; ; )
        {
            Console.Write("{0}>", Environment.CurrentDirectory);
            cmd = Console.ReadLine();
            cmdParams = cmd.Split(' ', '\t');
            if (cmdParams.Length == 0)
                continue;
            switch (cmdParams[0])
            {
                case "cd":
                    try
                    {
                        Environment.CurrentDirectory = cmdParams[1];
                    }
                    catch (DirectoryNotFoundException e)
                    {
                        Console.WriteLine("System cannot find specified directory.\n");
                    }
                    break;
                case "exit":
                    goto EXIT;

            }
        }

    EXIT:

        Console.WriteLine("Processor Count: {0}", Environment.ProcessorCount);

        return 0;
    }
  }
}
```

Programın Sonlandırılması: Bir C# programının sonlandırılması için birkaç yöntem kullanılabilir.

1. Uygulama ister Console isterse GUI olsun Environment.Exit fonksiyonu sonlandırmada kullanılabilir. Ancak GUI uygulamalarının bu fonksiyonla sonlandırılması tavsiye edilen bir durum değildir. Çünkü bu program arka planda ExitProces API fonksiyonunu çağırmaktadır. Çıkış veya sonlandırma işlemi ani bir biçimde mesaj döngüsü dikkate alınmadan yapılır.
2. Anımsanacağı gibi programın normal sonlanması anapencerenin kapatılması nedeniyle Appilication.Run fonksiyonundan çıkılması ve Main fonksiyonunun bitmesiyle olmaktadır. O halde ana çıkış programın anapenceresini kapatmaya çalışılarak yapılan çıkıştır. Bu yolla yine FormClosing mesajı oluşturulur ve çıkış duruma göre yine engellenebilir.
3. Prosesin sonlandırılması ayrıca Appilication sınıfının Exit fonksiyonuyla da yapılabilir. Aslında bu yöntem ikinci yöntemi kapsamaktadır. Ancak bu fonksiyon çağrıldığında her ana pencere için Form Closing mesajı gönderilir. Aynı zamanda her thread aynı yöntemle sonlandırılmaktadır. Bu yöntem programın çok fazla ana penceresi varsa ve programımız özellikle GUI threadler içeriyorsa tavsiye edilebilir.

Anahtar Notlar: Microsoftun Framework SDK sı içinde aşağı seviyeli iki önemli araç ilasm.exe ve ildasm.exe programlarıdır. İlasm programı sembolik Makine dilinde yazılmış olan programları arakoda dönüştürür. İldasm ise bunun tam tersini yapar.

İlasm ve idasm programlarının dışında ayrıca sdk içinde bulunmayan Decompiler programlarıda mevcuttur. Decompiler programları arakod içeren exe. Ya da dll dosyalarını C# kaynak kaduna dönüştürebilmektedir. En çok kullanılan Decompiler programlar Salamander ve Dis# programlarıdır.

Process Kavramı: Program kavramı daha çok programın kaynak kodunu ya da çalıştırılabilir dosyayı belirtmektedir. Bir program çalıştırıldığında artık ona process denir. İşletim sistemi prgramı çalıştırdığında yani onu process haline getirdiğinde çeşitli tablolar kurarak çalışmakta olan programı izlemektedir.

İşletim sistemi düzeyinde proces gibi çeşitli kaynakların yönetimi için System.Diagnostics isim alanı içindeki sınıflar kullanılmaktadır. İşletim sistemi düzeyinde çalışmakta olan programlar process sınıfıyla temsil edilmektedir. Process sınıfı türünden bir nesne çalışmakta olan bir programın bilgilerini içermektedir. Process sınıfının static GetProcesses isimli fonksiyonu o anda çalışmakta olan tüm programlara ilişkin bilgileri Process sınıfı türünden bir diziye yerleştirir ve o diziyi verir.
public static Process[] GetProcesses()

Process sınıfının ProcessName isimli property elemanı çalışmakta olan programın ismini elde etmek için kullanılır.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Diagnostics;

namespace ProcessSample
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }

      private void button1_Click(object sender, EventArgs e)
      {
         foreach (Process p in Process.GetProcesses())
            listBox1.Items.Add(p.ProcessName);
      }
   }
}
```

Anahtar Notlar: .net için usenet haber gurupları microsoft.public.dotnet dizininin altındaki guruplardır.
Process sınıfının pek çok faydalı fonksiyonu ve elemanı vardır. Örneğin Process.CloseMainWindow isimli fonksiyon process bir GUI programına ilişkinse onun anapenceresini kapatmaya çalışır.

CloseMainWindow fonksiyonu programın ana penceresine WM_CLOSE mesajı göndermektedir. Yani yaratılan etki tamamen sanki form sınıfının Close fonksiyonunun çağrılması gibi olmaktadır.

Bir GUI programının dışarıdan sonlandırılmasına ilişkin en doğal yöntem önce onun mesaj döngüsü yoluyla normal bir biçimde sonlandırılmasına çalışmaktır. Anımsanacağı gibi bu işlem CloseMainWindow fonksiyonuyla gerçekleştirilebilir. Daha önceden de belirtildiği gibi bu mesajın gönderilmesi adeta anpencerenin çarpı düğmesine tıklanması etkisi yaratmaktadır. Bu işlem sonucunda eğer program sonlanmazsa artık gerçekten biz killfonksiyonunu çaırarak vzorla sonlandırma yapabiliriz. Şüphesiz kill fonksiyonu ancak yetkili olduğumuz prosesleri sonlandırabilir. Prosesin sonlanmış olup olmadığı bool türden HasExited property si ile belirlenebilir. Windows un kendisinin yaptığı gibi genel eğilim Close işleminden sonra 5 saniye civarında beklemektir. Bu bekleme işlemi WaitForExit fonksiyonlarıyla yapılabilir.

public bool WaitForExit(int milliseconds)

fonksiyon ilgili proses sonlanana kadar ya da en fazla parametresiyle belirtilen milisaniye kadar çağıran thread i bloke eder.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Diagnostics;

namespace ProcessSample
{
   public partial class Form1 : Form
   {
      private Process[] m_processes;
      public Form1()
      {
         InitializeComponent();
      }

      private void button1_Click(object sender, EventArgs e)
      {
         listBox1.Items.Clear();
         m_processes = Process.GetProcesses();

         foreach (Process p in m_processes)
            listBox1.Items.Add(p.ProcessName);


      }

      private void listBox1_DoubleClick(object sender, EventArgs e)
      {
         Process p = m_processes[listBox1.SelectedIndex];
```

```
            try
            {
               p.CloseMainWindow();

               if (!p.WaitForExit(5000))
               {
                  p.Kill();
               }

               button1_Click(null, null);
            }
            catch (Exception ex)
            {
               MessageBox.Show(ex.Message);
            }


         }
      }
   }
```

Process sınıfı yalnızca çalışan prosesleri kontrol etmekte değil aynı zamanda yeni proses oluşturmakta yani yeni program çalıştırmakta da kullanılabilir. Bir programı çalıştırabilmek için yapılacak şey static Start fonksiyonunu çağırmaktır. Pekçok start fonksiyonu vardır. En tipik olanı aşağıdakidir.

public static ;Process Start(string FileName)

Fonksiyon parametre olarak bir dosya ismi istemektedir. Dosya olarak .exe dosya verilebilir ya da ilişkilendirilmiş bir dosya verilebilir. Örneğin biz parametre olarak ".doc" uzantılı bir dosya verirsek Start fonksiyonu word programını çalıştırıp bu dosyayı açacaktır.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Diagnostics;

namespace ProcessStart
{
   public partial class Form1 : Form
   {
      private Process m_proc;

      public Form1()
      {
         InitializeComponent();
```

```
        }

        private void button1_Click(object sender, EventArgs e)
        {
            OpenFileDialog ofd = new OpenFileDialog();

            if (ofd.ShowDialog() == DialogResult.OK)
            {
                try
                {
                    m_proc = Process.Start(ofd.FileName);
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message);
                }
            }
        }
    }
}
```

static Start fonksiyonu yerine static olmayan Start fonksiyonu da kullanılabilir. Fakat bunun için dosya isminin ProcessName elemanına girilmesi gerekmektedir. Diğer bir start fonksiyonu 2 parametre alan fonksiyondur.

public static Process Start(string fileName, string arguments)
public static Process Start(processStartInfo, startInfo)

En geniş Start fonksiyonu şüphesiz process Start info parametreli fonksiyondur. ProcessStartInfo sınıfı çeşitli belirlemeleri set etmek için kullanılan pek çok elemana sahiptir. Yani programcı bu sınıf türünden nesnenin içini doldurur. Onu Start programına parametre olarak geçer. ProcessStartInfo sınıfının CreateNoWindow isimli property elemanı Console uygulamalarında Console penceresinin açılmamasını sağlamaktadır.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Diagnostics;
using System.IO;

namespace BasicIDE
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
```

```
        }

        private void compileToolStripMenuItem_Click(object sender, EventArgs e)
        {
            saveToolStripMenuItem_Click(null, null);

            try
            {
                ProcessStartInfo psi = new ProcessStartInfo(
                    @"C:\Windows\Microsoft.NET\Framework\v2.0.50727\csc.exe", "test.cs");

                psi.CreateNoWindow = true;
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }

        }

        private void saveToolStripMenuItem_Click(object sender, EventArgs e)
        {
            try
            {
                File.WriteAllText("test.cs", textBox1.Text);
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
    }
}
```

Bir Console programını çalıştırıp onun ekrana yazdıklarını elde etmek için sırasıyla şu işlemler yapılmalıdır:

1. ProcessStartInfo türünden bir sınıf nesnesi yaratılır. Çalıştırılacak programın ismi fileName property sinne ve komut satırı argümanları arguments propertysine girilir. Fakat zaten bu iki değeri set eden iki parametreli bir başlangıç fonksiyonu da vardır.
2. ProcessStartInfo nesnesinin CreatNoWindow propertysi false olarak girilir. Ayrıca yine sınıfın UseShellExecute propertysine false girilir.
3. Şimdi stdout dosyasının yönlendireleceğini belirtmek için RedirectStandartOutput property si true yapılmalıdır.
4. Process sınıfının standartOutputisimli property elemanı bize bir StreamReader nesnesi verir. Biz bu StreamReader nesnesinden okuma yaptığımızda aslında çalıştırdığımız programın ekrana yazdırdığı bilgileri okumuş olacağız.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
```

```
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Diagnostics;
using System.IO;

namespace BasicIDE
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }

      private void compileToolStripMenuItem_Click(object sender, EventArgs e)
      {
         saveToolStripMenuItem_Click(null, null);

         try
         {
            ProcessStartInfo psi = new ProcessStartInfo(
               @"C:\Windows\Microsoft.NET\Framework\v2.0.50727\csc.exe", "test.cs");

            psi.CreateNoWindow = true;
            psi.UseShellExecute = false;
            psi.RedirectStandardOutput = true;

            Process p = Process.Start(psi);

            StreamReader sr = p.StandardOutput;

            textBox2.Text = "";

            string str;
            while ((str = sr.ReadLine()) != null)
            {
               textBox2.Text += str + "\r\n";
            }

         }
         catch (Exception ex)
         {
            MessageBox.Show(ex.Message);
         }

      }

      private void saveToolStripMenuItem_Click(object sender, EventArgs e)
      {
         try
```

```
            {
                File.WriteAllText("test.cs", textBox1.Text);
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        private void runToolStripMenuItem_Click(object sender, EventArgs e)
        {
            try
            {
                Process.Start("test.exe");
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
    }
}
/*
class App
{
    public static void Main()
        {
            System.Console.WriteLine("kjhkjhkjhkjhkj");

            System.Console.ReadLine();asd
        }

}
*/
```

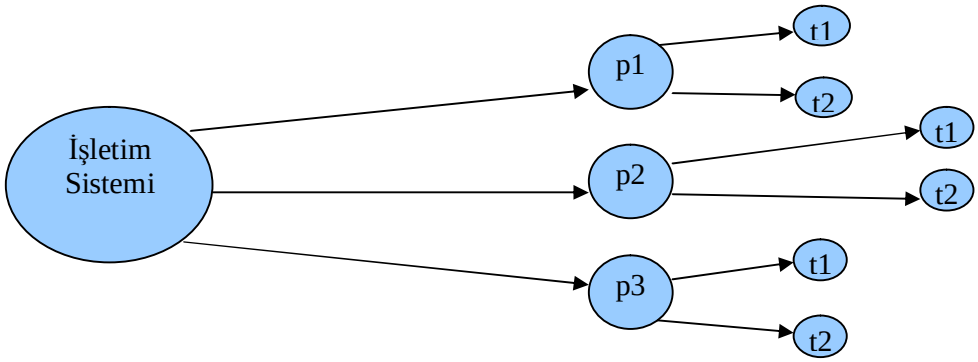
Process in Modül Listesi: exe, dll gibi process alanına bağımsız yüklenebilen dosyalara modül denilmektedir. (Modül terimi sistem programlama terminolojisine ilişkindir. .net teki assembly terimi tam olarak buradaki modül anlamına gelmemektedir.) İşte exe, dll gibi modüller ProcessModule sınıfıyla temsil edilmiştir. ProcessModul sınıfının FileName isimli property elemanı modul dosyasının tüm yol ifadesiyle ModulName isimli property elemanı bunun yalnızca ismini elde etmek için kullanılır. Process sınıfının Modules isimli property elemanı bir Collection sınıf türündendir. Process in tüm modüllerini vermektedir.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Diagnostics;
```

```
namespace ProcessSample
{
   public partial class Form1 : Form
   {
      private Process[] m_processes;
      public Form1()      {
         InitializeComponent();
      }

      private void button1_Click(object sender, EventArgs e)
      {
         listBox1.Items.Clear();
         m_processes = Process.GetProcesses();

         foreach (Process p in m_processes)
            listBox1.Items.Add(p.ProcessName);


      }

      private void listBox1_DoubleClick(object sender, EventArgs e)
      {
         Process p = m_processes[listBox1.SelectedIndex];

         try
         {
            p.CloseMainWindow();

            if (!p.WaitForExit(5000))
            {
               p.Kill();
            }

            button1_Click(null, null);
         }
         catch (Exception ex)
         {
            MessageBox.Show(ex.Message);
         }


      }

      private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
      {
         listBox2.Items.Clear();
         Process p = m_processes[listBox1.SelectedIndex];

         foreach (ProcessModule pm in p.Modules)
         {
            listBox2.Items.Add(pm.ModuleName);
```

```
        }
    }
  }
}
```

Thread İşlemleri: Multiprocessing işletim sistemlerinde çizelgeleme işlemi zaman paylaşımlı olarak yapılmaktadır. Yani işletim sistemi bir process i belirli bir süre çalıştırır sonra onun çalışmasına ara verir diğerine geçer ve bu biçimde zaman paylaşımlı bir çalışma söz konusudur.



Bir process in parçalı çalışma süresine quanta süresi denirWindows sistemlerinde tipik olarak 20ms, 60 ms, 120ms değerleri kullanılmaktadır. Bir programın bağımsız olarak çizelgelenen her farklı akışına thread denilmektedir. Örneğin iki threadli bir process de bağımsız iki akış söz konusu olabilir. Thread mekanizmasına sahip olan windows, Unix/Linux gibi sistemler thread temelinde çizelgeleme uygulamaktadır. Yani her bir quanta süresi dolduğunda bir thread in çalışmasına arar verilir. Yeni bir threade geçilir. Böylece bir process in threadleri sanki bağımsız programlarmış gibi birlikte çalışabilmektedir.

Threadler Neden Kullanılır?

1. Threadler arkaplan olaylarını etkin bir biçimde izlenmesi için kullanılabilir. Örneğin asıl akış yoluna devamederken diğer bir thread akışı bir portu kontrol edebilir.

ThreadConsole1 yerleştir.

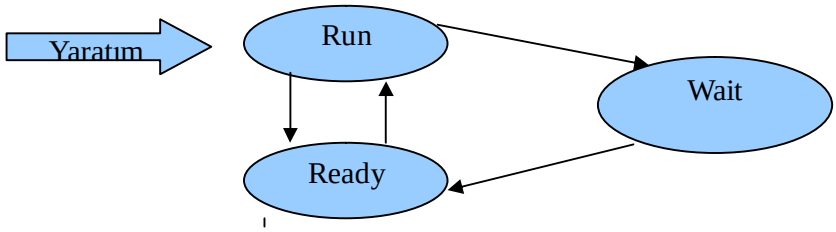
2. Threadler bizim toplamda daha fazla CPU zamanı elde etmemizi sağlayabilir. Örneğin programımızı Birkaç thread ile düzenlersek birim zamanda programımızın toptan çalıştırdığı kod miktarı fazlalaşır.
3. Threadler sayesinde paralel programlama yapabiliriz. Eğer makinamızda birden fazla işlemci Ya da işlemci çekirdeği varsa gerçekten programımızın farklı akışları aynı anda çalışabilir.

Bloke Kavramı ve Threadlerin Bloke Edilmesi: Klavye, dosya, network işlemleri ve buna benzer işlemler uzun süre beklemeye yol açmaktadır. İşte bu tür olaylarda işletim sistemi thread igeçici olarak çizelge dışına çıkararak arka planda olayı izlemektedir. Böylece dışsal olayı bekleyen thread boşuna CPU zamanı harcamaz. Arka plan olay gerçekleştiğinde işletim sistemi bunu belirler ve thread i yenideen çizelgeye koyar. İşte thread in bu biçimde geçici olarak çizelge dışına çıkartılmasına thread in bloke edilmesi diyoruz. Böylece aslında sistemde yüzlerce thread çalışıyor olabildiği halde bunlardan çok azı gerçekten çizelgede aktif durumdadır. Örneğin bir thread dosyadan bir okuma yapacak olsun. Tipik olarak işletim sistemi thread i çizelge dışına çıkararak bekletir. Bu sırada diğer threadler çalıştırılır. Disk işlemi gerçekleşip de okuma bitirilince thread yeniden çizelgeye konulur ve Read fonksiyonu geri döner. Okuma işlemini yapmak isteyen thread

için yine bir değişiklik yoktur. O yine beklemektedir.

GUI programlama modelinde anımsanacağı gibi bir GUI uygulaması yaşamını Application.Run fonksiyonunda yani mesaj döngüsünde geçirir. İşte Application.Run fonksiyonu eğer mesaj kuyruğunda mesaj bulamazsa yani kuyruk boşsa blokeye yol açmaktadır. Böylece bir pencereye hiç mesaj gitmiyorsa muhtemelen pencereye ilişkin threadde CPU zamanı harcamıyor durumdadır.

Bir thread in yaşam döngüsü şöyledir.



.nette Thread İşlemleri: Thread işlemleri System.Threading isimalanındaki sınıflarla gerçekleştirilmektedir. Thread kavramı Thread sınıfıyla temsil edilmiştir. Thread nesnesi aşağıdaki başlangıç fonksiyonuyla yaratılabilir.

public Thread(ParameterizedThreadStart start)

ParameterizedThreadStart şöyle bir delegedir.
public delegate void ParameterizedThreadStart(Object obj)
Görüldüğü gibi Thread fonksiyonu olacak fonksiyonun geri dönüş değeri void paramtresi object türünden olmalıdır.

Thread t = new Thread(new ParameterizedThreadStart(ThreadProc));

Burada ThreadProc fonksiyonunun geri dönüş değeri void parametresi object türden olmalıdır. Bu fonksiyon thread akışının başlatılacağı fonksiyondur. Aslında Framework 2.0 dan önce maalesef aşağıdaki biçimde olan başlangıç fonksiyonu kullanılmaktaydı: *public Thread(ThreadStart start)* ThreadStart isimli delege şöyledir: *public delegate ThreadStart()*

Bir thread nesnesi yaratıldığında thread yaratılır fakat henüz çalışmıyor durumdadır. Thread in çalışması Start fonksiyonlarıyla sağlanır. İki Start fonksiyonu vardır:
public void Start(Onject parameter) Bu fonksiyon eğer thread parameterizedThreadStart delegesi ile yaratılmışsa kullanılır. Aşağıdaki ise ThreadStart delegesi ile yaratılmışsa kullanılır.
public void Start ()

Bu durumda aşağıdaki gibi yaratılıp çağrılabilir:

Thread t = new Thread(new ParameterizedThreadStart(ThreadProc));
t.Start(obj);

Threadlerin stack leri tamamen birbirinden ayrılmıştır. Yani her thread yerel değişkenlerin kendine özgü bir kopyasını kullanır. Böylece farklı iki thread aynı fonksiyon üzerinde ilerlerken aslında yerel değişkenlerin farklı kopyalarını kullanıyor durumdadır.

```
using System;
using System.Threading;
```

```
namespace CSD
{
   class App
   {
      public static void Main()
      {
         Thread t = new Thread(new ParameterizedThreadStart(ThreadProc));

         t.Start(null);

         Foo("Main thread");
      }

      private static void ThreadProc(object o)
      {
         Foo("Other thread");
      }

      private static void Foo(string s)
      {
         for (int i = 0; i < 10; ++i)
         {
            Console.WriteLine("{0}: {1}", s, i);
            Thread.Sleep(1000);
         }
      }

   }
}
```

Ancak sınıfın statik veri elemanlarının toplam bir kopyası vardır ve her thread aynı kopya üzerinde çalışır. Benzer biçimde Heap alanı da ortak paylaşılmaktadır.

```
using System;
using System.Threading;

namespace CSD
{
   class App
   {
      private static Thread [] m_threads;

      public static void Main()
      {
         m_threads = new Thread[10];

         for (int i = 0; i < 3; ++i)
         {
            m_threads[i] = new Thread(new ParameterizedThreadStart(ThreadProc));
            m_threads[i].Start(i);
         }
```

```
            Console.ReadLine();
        }

        private static void ThreadProc(object o)
        {
            for (int i = 0; ; ++i)
            {
                Console.WriteLine("{0}.Thread: {1}", o.ToString(), i);
                Thread.Sleep(1000);
            }
        }


    }
}
```

Anahtar Notlar: Sistemde hanggi proseslerin bulunduğunu o proseslerin hangi threadlere sahip olduğunu spy ++ programı ile öğrenebiliriz.

Threadlerin Sonlanması: Threadlerin birkaç biçimde sonlanabilir.

1. Thread fonksiyonu sonlandığında thread de otomatik sonlanır.
2. Herhangi bir thread çalışırken thread sınıfının Abort fonksiyonuyla sonlandırılabilir. Aslında Abort fonksiyonu thread üzerinde ThreadAbortException isimli bir Exception oluşturur. Bu Exception ele alınmasdığı için bu thread sonlandırılmaktadır.

```
usinng  System;
using System.Threading;

namespace CSD
{
    class App
    {
        private static Thread [] m_threads;

        public static void Main()
        {
            m_threads = new Thread[10];

            for (int i = 0; i < 3; ++i)
            {
                m_threads[i] = new Thread(new ParameterizedThreadStart(ThreadProc));
                m_threads[i].Start(i);



            }

            Console.ReadLine();

            m_threads[2].Abort();
            Console.ReadLine();
        }
```

```
        private static void ThreadProc(object o)
        {
            for (int i = 0; i < 10; ++i)
            {
                try
                {
                    Console.WriteLine("{0}.Thread: {1}", o.ToString(), i);
                    Thread.Sleep(1000);
                }
                catch (Exception e)
                {
                    Console.WriteLine(e);
                }



            }
        }


    }
}
```

3.

Abort fonksiyonu uygulandığında thread üzerinde ThreadAbortException isimli Exception oluşur. Bu Exception try catch bloğu ile yakalanabilir eğer catch bloğunda ResertAbort isimli fonksiyon çağrılmazsa catch bloğunun sonunda otomatik yeniden throw işlemi yapılır. Böylece thread yine sonlandırılmış olur. Eğer Abort işlemine karşı threadimizin sonlandırılmasını istemiyorsak static ResertAbort fonksiyonu çağrılmalıdır.

```
using System;
using System.Threading;

namespace CSD
{
    class App
    {
        private static Thread [] m_threads;

        public static void Main()
        {
            m_threads = new Thread[10];

            for (int i = 0; i < 3; ++i)
            {
                m_threads[i] = new Thread(new ParameterizedThreadStart(ThreadProc));
                m_threads[i].Start(i);



            }
```

```
            Console.ReadLine();

            m_threads[2].Abort();
            Console.ReadLine();
        }

        private static void ThreadProc(object o)
        {
            for (int i = 0; i < 10; ++i)
            {
                try
                {
                    Console.WriteLine("{0}.Thread: {1}", o.ToString(), i);
                    Thread.Sleep(1000);
                }
                catch (Exception e)
                {
                    Console.WriteLine(e);
                    Thread.ResetAbort();
                }



            }
        }


    }
}
```

4. Threadler önplan(forground) ve arkaplan(background) olmak üzere ikiye ayrılır. Threadin önplan Ya da arkaplan olmasınının tek etkisi sonlanma kısmında ortaya çıkmaktadır. Prosesin son önplan threadi sonlandığında tüm arkaplan threadlerde otomatik olarak sonlandırılır. Böylece proses sonlandırılmış olur. Thread yaratıldığında default olarak önplan dır. Ana threadde önplandır.

```
using System;
using System.Threading;

namespace CSD
{
    class App
    {
        private static Thread [] m_threads;

        public static void Main()
        {
            m_threads = new Thread[10];

            for (int i = 0; i < 3; ++i)
            {
                m_threads[i] = new Thread(new ParameterizedThreadStart(ThreadProc));
```

```
            m_threads[i].Start(i);
            m_threads[i].IsBackground = true;
        }

    }

    private static void ThreadProc(object o)
    {
        for (int i = 0; i < 10; ++i)
        {
            try
            {
                Console.WriteLine("{0}.Thread: {1}", o.ToString(), i);
                Thread.Sleep(1000);
            }
            catch (Exception e)
            {
                Console.WriteLine(e);
                Thread.ResetAbort();
            }



        }
    }


  }
}
```

Ana threadin diğer threadlerden hiçbir farkı yoktur. Yani ana thread sonlandığı halde diğer threadler çalışmaya devam ediyor olabilir. Prosesin son threadi de sonlandığında program otomatik olarak sonlanır.

Threadlerin stackleri tamemen ayrılmış olmasına karşın heap alanı ortak kullanılmaktadır. Bu nedenle sınıfın ortak veri elemanını ortaklaşa bir biçimde kullanabilmektedirler.

Threadlerin Sonlanmasının Beklenmesi: Thread sınıfının Join isimli fonksiyonları bir thread sonlanana kadar fonksiyonu çağıran thread i bekletmektedir. Parametresiz Join fonksiyonu thread sonlanana kadar bekleme yapar. Int parametreleri Join fonksiyonu ve TimeSpan parametreli join fonksiyonu belli bir zamanaşımı parametresi de almaktadır.

```
using System;
using System.Threading;

namespace CSD
{
  class App
  {
    private static Thread m_thread;

    public static void Main()
    {
```

```
            m_thread = new Thread(new ParameterizedThreadStart(ThreadProc));
            m_thread.Start("other thread");
            m_thread.IsBackground = true;

            m_thread.Join();

            Console.WriteLine("ends...");

        }

        private static void ThreadProc(object o)
        {
            for (int i = 0; i < 10; ++i)
            {
                Console.WriteLine("{0}.Thread: {1}", o.ToString(), i);
                Thread.Sleep(1000);
            }
        }


    }
}
```

GUI ve Worker Threadler: Microsoft threadleri GUI ve Worker Threadler olmak üzere ikiye ayırmaktadır. Eğer bir threadde en az bir pencere yaratılmışsa bu tür threadlere GUI thread denilmektedir. Eğer Threadde hiçbir pencere yaratılmamışsa bunlara worker threadler denir. Windows sistemlerinde her thread için ayrı bir mesaj kuyruğu oluşturulduğu için yeni bir pencere yaratıldığında o threadde yeniden mesaj döngüsüne girmek gerekir. Yani yeni yaratılan threadde Appilication.Run fonksiyonunun çağrılması gerekir.

Form1.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Threading;

namespace GUIThread
{
    public partial class Form1 : Form
    {
        private Thread m_thread;

        public Form1()
        {
            InitializeComponent();
        }
```

```
        private void button2_Click(object sender, EventArgs e)
        {
            for (long i = 0; i < 10000000000; ++i)
                ;
        }

        private void button1_Click(object sender, EventArgs e)
        {
            m_thread = new Thread(new ParameterizedThreadStart(ThreadProc));
            m_thread.Start();

        }

        private void ThreadProc(object o)
        {
            Form2 f = new Form2();

            Application.Run(f);



        }
    }
}
```

Form2.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace GUIThread
{
    public partial class Form2 : Form
    {
        public Form2()
        {
            InitializeComponent();
        }
    }
}
```

Bu durumda örneğin bir GUI thread tipik olarak şöyle oluşturulmalıdır.

```
private void ThreadProc(object o)
{
        Application.Run(new myForm());
}
```

Görüldüğü gibi bu thread kodu tamemen main fonksiyonunda olduğu gibidir.

.nette pekçok sınıf kendi içinde zaten bir thread açmaktadır. Örneğin System.Threading isim alanındaki Timer sınıfı kendi içinde bir thread açar ve delege fonksiyonunu bu threadle çağırır. Buna benzer bir timer sınıfı şöyle yazılabilir.

ThreadSample2
Form1.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Threading;

namespace ThreadSample
{
   public partial class Form1 : Form
   {
      private Thread m_thread;
      private bool m_bFlag;
      private CSD.Timer m_timer;
      private int m_i;

      public Form1()
      {
         InitializeComponent();
      }

      private void button1_Click(object sender, EventArgs e)
      {
         m_timer = new CSD.Timer(1000);
         m_timer.Start(null);

         m_timer.Tick += new CSD.TimerProc(m_timer_Tick);

      }

      void m_timer_Tick(object o)
      {
         label1.Text = m_i.ToString();
         ++m_i;
      }

   }
}
```

Timer.cs

```
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Text;
using System.Threading;

namespace CSD
{
   delegate void TimerProc(object o);

   class Timer : IDisposable
   {
      private Thread m_thread;
      private int m_period;

      public event TimerProc Tick;

      public Timer(int period)
      {
         m_period = period;

         m_thread = new Thread(new ParameterizedThreadStart(timerThreadProc));
         m_thread.IsBackground = true;
      }

      public void Start(object o)
      {
         m_thread.Start(o);
      }

      public void Resume()
      {
         m_thread.Resume();
      }

      public void Suspend(object o)
      {
         m_thread.Suspend();
      }

      public void Dispose()
      {
         m_thread.Abort();
      }

      private void timerThreadProc(object o)
      {
         for (; ; )
         {
            Thread.Sleep(m_period);
            Tick(o);
         }
      }
   }
```

```
}

ThreadSample3
Form1.cs
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Threading;

namespace ThreadSample
{
   public partial class Form1 : Form
   {
      private Thread m_thread;
      private bool m_bFlag;
      private CSD.Timer m_timer;
      private int m_i;

      public Form1()
      {
         InitializeComponent();
      }

      private void button1_Click(object sender, EventArgs e)
      {
         m_timer = new CSD.Timer(1000);
         m_timer.Start(null);

         m_timer.Tick += new CSD.TimerProc(m_timer_Tick);

      }

      void m_timer_Tick(object o)
      {
         label1.Text = m_i.ToString();
         ++m_i;
      }

   }
}
Timer.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
```

```
namespace CSD
{
   delegate void TimerProc(object o);

   class Timer : IDisposable
   {
      private Thread m_thread;
      private int m_period;

      public event TimerProc Tick;

      public Timer(int period)
      {
         m_period = period;

         m_thread = new Thread(new ParameterizedThreadStart(timerThreadProc));
         m_thread.IsBackground = true;
      }

      ~Timer()
      {
         m_thread.Abort();
         GC.SuppressFinalize(this);
      }

      public void Start(object o)
      {
         m_thread.Start(o);
      }

      public void Resume()
      {
         m_thread.Resume();
      }

      public void Suspend(object o)
      {
         m_thread.Suspend();
      }

      public void Dispose()
      {
         m_thread.Abort();
      }

      private void timerThreadProc(object o)
      {
         for (; ; )
         {
            Thread.Sleep(m_period);
            Tick(o);
         }
```

```
    }
  }
}
```

Threadlerin Senkronizasyonu: Threadler konusunun önemli bir bölümünü thread senkronizasyonu içermektedir. Birden fazla thread ortak bir amaç için çalışırken kimi zaman birbirlerini beklemeleri gerekir.

Kritik Kod Bloklarının Oluşturulması: Threadlerin çalışmasına ara verilmesi preemptive bir biçimde gerçekleştirilmektedir. Bir thread henüz çıkmadan başka bir threadin girmemesi gerekn kod bloklarına kritik kod blokları denilmektedir. Kritik kod blokları özellikle ortak kaynaklara erişilirken dikkat edilmesi gerekn bir konudur. Örneğin iki thread ortak bir ArrayList te eleman insert ediyor olsun bir thread işlemini bitirmeden arada kesilirse diğer threadin obür thread çıkana kadar onu beklemesi gerekir.

Kritik kodların manuel bir biçimde oluşturulması mümkün değildir. Bunlar için özel fonksiyonlar kullanılmaktadır. Örneğin aşağıdaki manuel flag yöntemi garanti bir yöntem değildir.

```
flag = false;

while(flag)
        ;
                //Kod burada kesilirse ne olu?
flag = true;
...
                // Kritik kod
flag = false;
```

Kritik kod oluşturmak için Monitor sınıfının Enter ve Exit fonksiyonları kullanılmaktadır.

```
Monitor.Enter(...);

...             //Kritik kod


Monitor.Exit(...);
```

Critical Section1 yerleştir.

Monitor sisteminde threadlerden biri Monitor.Enter fonksiyonundan girdiğindehenüz kritik koddan çıkmadan diğer bir thread kritik koda girmeye çalışırsa Enter fonksiyonunun içerisinde blokeye yol açar. Kritik kodun içindeki thread Exit fonksiyonunu gördüğünde Enter dolayısıyla bakleyen thread içeriye girer.

Kritik kod örneği gerekliliğini anlatan örnek:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Collections;
```

```
namespace CriticalSection
{
   class Program
   {
      private static Random m_rand;

      static void Main(string[] args)
      {
         m_rand = new Random();

         Thread t = new Thread(new ParameterizedThreadStart(ThreadProc));
         t.Start(null);

         for (int i = 0; i < 10; ++i)
            Test();

         t.Join();



      }

      public static void ThreadProc(object o)
      {
         for (int i = 0; i < 10; ++i)
            Test();
      }

      public static void Test()
      {
         Monitor.Enter(m_rand);
         Console.WriteLine("-------");
         Console.WriteLine("1. Adım");
         Thread.Sleep(m_rand.Next(100));
         Console.WriteLine("2. Adım");
         Thread.Sleep(m_rand.Next(100));
         Console.WriteLine("3. Adım");
         Thread.Sleep(m_rand.Next(100));
         Console.WriteLine("4. Adım");
         Thread.Sleep(m_rand.Next(100));
         Console.WriteLine("5. Adım");
         Thread.Sleep(m_rand.Next(100));
         Console.WriteLine("-------");
         Monitor.Exit(m_rand);
      }
   }
}
```

Eğer birden fazla thread Enter fonksiyonu dolayısıyla kritik koda girmeden bekliyorsa kritik kod içindeki thread kritik koddan çıktığında bekleyen hangi threadin gireceği konusunda garanti bir

belirleme yoktur. Fakat adil bir sistem uygulanmaktadır. Enter ve Exit fonksiyonları object türünden parametre almaktadır. Fonksiyonların object parametresi kritik kodu betimlemektedir. Örneğin kodumuzda iki farklı yerde Monitor.Enter ve Monitor.Exit ile oluşturulmuş kritik kod olsun. Threadlerden biri bunlardan birine girmişse diğerinin diğerine girebilmesi istenebilir. Çünkü bunlar farklı konulara ilişkin kritik kodlar olabilir. İşte biz Enter fonksiyonuna aynı object değerini verirsek aynı konuya ilişkin kritik kod oluşturmuş oluruz. Bu durumda bir thread bir kritik koda girdiğinde diğeri de diğerine giremez.

Kritik kod içinde bir exception oluşursa akış başka bir bölgeye atlar. İşte çıkarken kilidi açmak gerekebilir. Bu nedenle Exit işleminin Finally bloğunda yapılması ıygun olur.

```
try
{
        Monitor.Enter(o);


        .... Kritik Kod


}

finally
{
        Monitor.Exit(o);
}
```

Monitor.Enter ve Exit işlemini yapan C# da Lock deyimi vardır. Lock deyiminin genel biçimi şöyledir.

```
lock(<ifade>)
        <deyim>
```

Lock deyimi tamamen aşağıdaki ile eşdeğerdir.

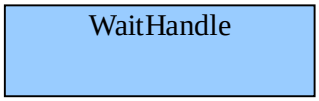
```
Monitor.Enter(<ifade>)

try
{
        <deyim>
}
finally
{
        Monitor.Exit(<ifade>);
}
```

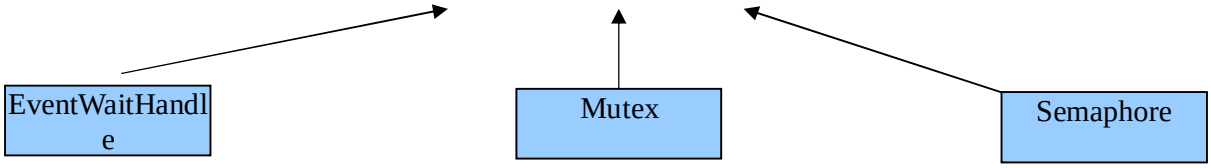
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Collections;

namespace CriticalSection
```

```
{
   class Program
   {
      private static ArrayList m_al;
      private static Random m_rand;

      static void Main(string[] args)
      {
         m_rand = new Random();
         m_al = new ArrayList();

         Thread t = new Thread(new ParameterizedThreadStart(ThreadProc));
         t.Start(null);

         for (int i = 0; i < 10000; ++i)
         {
            lock (m_al)
            {
               m_al.Insert(0, i);
            }
         }

         t.Join();

         Console.WriteLine("Ok = {0}", m_al.Count);


      }

      public static void ThreadProc(object o)
      {
         for (int i = 0; i < 10000; ++i)
         {
            lock (m_al)
            {
               m_al.Insert(0, i);
            }
         }
      }


   }
}
```

Kernel Senkronizasyon Nesneleri ile Thread Senkronizosyonu: Event, Mutex, Semaphore gibi Senkronizsayon nesnelerine windows da kernel senkronizasyon nesneleri denilmektedir.

Tüm senkronizasyon nesnelerine ilişkin sınıflar WaitHandle isimli sınıftan türemiştir. WaitHandle abstract bir sınıftır. Bu sınıftan EventWaitHandle, Mutex ve Semaphore sınıfları türetilmiştir.





Her senkronizasyon nesnesinin açık (signaled) ve kapalı(nonsingaled) olmak üzere iki durumu vardır. WaitHandle sınıfının WaitOne fonksiyonları eğer nesne açıksa geçiş yapar kapalıysa threadi bloke ederek çizelge dışına çıkarır. Nesnelerin açık ya da kapalı olma durumu ilgili nesneye göre belirlenmektedir.

EventWaitHandle Nesnesi: EventWaitHandle nesnesi bir olay gerçekleşene kadar threadi blokede bekletmek için kullanılır. Nesne aşağıdaki başlangıç fonksiyonu ile yaratılabilir.

public EventWaitHandle(bool initialState, EventResetMode mode)

Fonksiyonun birinci parametresi nesnenin başlangıçtaki durumunu belirtir. True ise nesne açık False ise kapalıdır. İkinci parametre EventResetMode isimli Enum türündendir. AutoReset ve ManuelReset biçiminde iki elemanı vardır. Event nesnesi Manuel ise Wait fonksiyonlarına geçiş yapıldığı zaman Resetleme işlemi programcının sorumluluğundadır. AutoReset durumunda ise geçiş yapıldığında nesne otomatik olarak kapalı duruma geçer. Event nesnesi tipik olarak şöyle kullanılır.

1. EventWaitHandle türünden bir nesne sınfın veri elemanı olarak bildirilir. Böylece her nesne erişebilir. Nesne işin başında kapalı ve AutoReset biçiminde yaratılabilir.
2. Bekleme yapmak isteyen thread Event nesnesi ile WaitOne fonksiyonunu çağırır.
3. Diğer thread işini bitirince EventWaitHandle sınıfının Set fonksiyonu çağrılır. Set fonksiyonu nesneyi açık duruma getirmektedir. Benzer biçimde kapalı duruma getirmek içinde Reset fonksiyonu vardır. Event AutoReset ise nesne otomatik olarak yeniden kapalı duruma geçer.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Threading;

namespace EventSample
{
   public partial class Form1 : Form
   {
      private Thread m_thread;
      private EventWaitHandle m_event;

      public Form1()
      {
         InitializeComponent();
```

```
            m_event = new EventWaitHandle(false, EventResetMode.AutoReset);

        }

        private void button1_Click(object sender, EventArgs e)
        {
            m_thread = new Thread(new ParameterizedThreadStart(ThreadProc));
            m_thread.Start();

        }

        private void button2_Click(object sender, EventArgs e)
        {
            m_event.Set();
        }

        private void ThreadProc(object o)
        {
            m_event.WaitOne();

            MessageBox.Show("Ok");

        }

    }
}
```

Üretici Tüketici Problemi: Üretici tüketici problemi gerçek uygulamalarda çok sık karşılaşılan tipik bir problemdir. Bu problemde bir üretici ve bir de tüketici thread vardır. Üretici thread bir faaliyet sonucunda bir değer elde eder ve bu değeri bir değişkene yerleştirir. Tüketici threadde bu değeri alarak kullanır. Üretici thread tüketici thread henüz almadan yeni bir değeri yerleştirmemeli tüketici thread de üretici thread yeni bir değeri koymadan ikinci kez almamalıdır.

Üretici Tüketici probleminde karşılan tipik sorun kodu:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Threading;

namespace Producer_Consumer
{
    public partial class Form1 : Form
    {
        private Thread m_producer;
        private Thread m_consumer;
        private Random m_rand;
        private int m_shared;
```

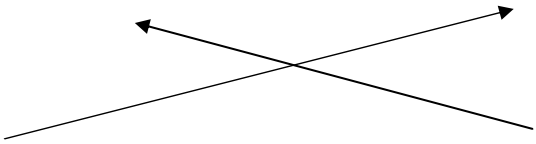
```
    public Form1()
    {
        InitializeComponent();

        m_rand = new Random();

        m_producer = new Thread(new ParameterizedThreadStart(ProducerThreadProc));
        m_producer.Start(null);
        m_producer.IsBackground = true;

        m_consumer = new Thread(new ParameterizedThreadStart(ConsumerThreadProc));
        m_consumer.Start(null);
        m_consumer.IsBackground = true;
    }

    private void ProducerThreadProc(object o)
    {
        int i = 0;

        for (; ; )
        {
            Thread.Sleep(m_rand.Next(200));
            m_shared = i;
            if (i == 50)
                break;
            ++i;
        }
    }

    private void ConsumerThreadProc(object o)
    {
        for (; ; )
        {
            Thread.Sleep(m_rand.Next(200));
            listBox1.Items.Add(m_shared);
            if (m_shared == 50)
                break;
        }
    }

  }
}
```

ÜreticiTüketici Probleminin EventWaitHandle Nesneleri ile Çözümü: Üretici-tüketici problemi Event nesneleri ile aşağıdaki gibi tipik olarak çözülebilir.

EventWaitHandle m_producerEvent = new EventWaitHandle(false, EventResetMode AutoReset);
EventWaitHandle m_consumerEvent = new EventWaitHandle(true, EventResetMode AutoReset);

Üretici	Tüketici

for(;;)
{
m_consumer EventWaitOne();
//set value
	for(;;)
	{
	m_producer EventWaitOne();
	//set value



Düzeltilmiş program:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Threading;

namespace Producer_Consumer
{
   public partial class Form1 : Form
   {
      private Thread m_producer;
      private Thread m_consumer;
      private EventWaitHandle m_eventProducer;
      private EventWaitHandle m_eventConsumer;
      private Random m_rand;
      private int m_shared;

      public Form1()
      {
         InitializeComponent();

         m_rand = new Random();

         m_eventConsumer = new EventWaitHandle(true, EventResetMode.AutoReset);
         m_eventProducer = new EventWaitHandle(false, EventResetMode.AutoReset);

         m_producer = new Thread(new ParameterizedThreadStart(ProducerThreadProc));
         m_producer.Start(null);
         m_producer.IsBackground = true;

         m_consumer = new Thread(new ParameterizedThreadStart(ConsumerThreadProc));
         m_consumer.Start(null);
         m_consumer.IsBackground = true;

      }
```

```
    private void ProducerThreadProc(object o)
    {
        int i = 0;

        for (; ; )
        {
            m_eventConsumer.WaitOne();
            Thread.Sleep(m_rand.Next(200));
            m_shared = i;
            if (i == 50)
                break;
            ++i;
            m_eventProducer.Set();
        }
    }

    private void ConsumerThreadProc(object o)
    {
        for (; ; )
        {
            m_eventProducer.WaitOne();
            Thread.Sleep(m_rand.Next(200));
            listBox1.Items.Add(m_shared);
            if (m_shared == 50)
                break;
            m_eventConsumer.Set();

        }
    }

  }
}
```

Semaphore Nesneleri: Semaphore nesneleri tipik olarak bir vkritik koda en fazla n tane threadin girmesini sağlamak için kullanılmaktadır. Semaphore nesneleri sayaçlıdır. Eğer Semaphore sayacı 0 dan büyükse nesne açık durumda 0 a eşitse kapalı durumdadır. Semaphore işlemleri Semaphore sınıfıyla temsil edilmiştir. Tipik olarak başlangıç fonksiyonu şöyledir.

public Semaphore(int initialCount, int maximumCount)

Fonksiyonun birinci parametresi semaphore sayacının başlangıç değeri, 2. parametresi maksimum değeridir. Semaphore sayacı maksimum değerden daha fazla artmamaktadır. Semaphore nesnesi ile kritik kod şöyle oluşturulur.

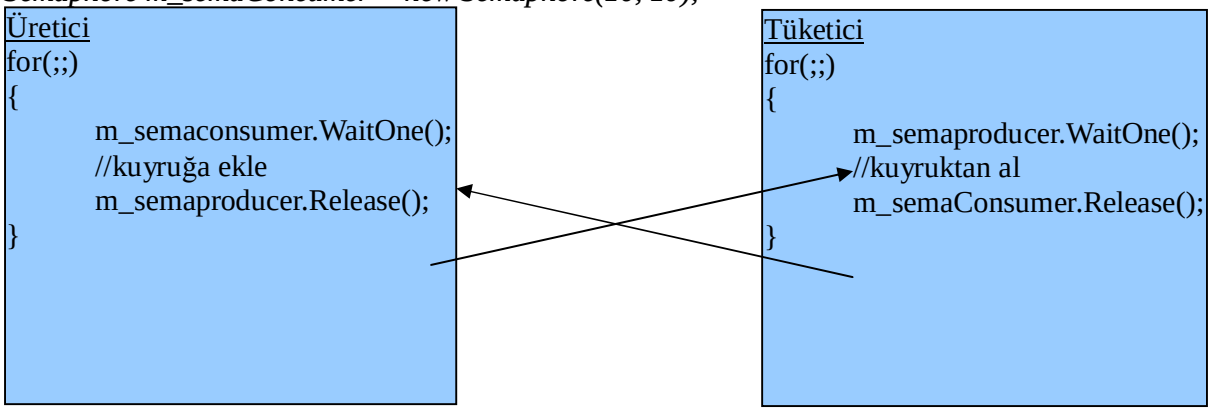
```
m_sema.WaitOne();

//...
.... Kritik Kod


m.sema.Release();
```

Wait fonksiyonları kullanıldığında eğer semaphore sayacı 0 dan büyükse geçiş yapılır ve wait fonksiyonları sayacı otomatik olarak 1 düşürür. Realese fonksiyonu semaphore sayacını bir arttırmaktadır. Böylece yukarıda kritik kod en fazla n kişinin girebileceği bir kodu belirtmektedir.

Üretici Tüketici Problemini Kuyruklu Versiyonu: Üretici tüketici probleminin tekli versiyonunda bir üretici bir tüketici vardır ve ortada paylaşılan alan tek elemanlıktır. Bu tür sistemler üretici ve tüketici bakımından çok fazla beklemeye yol açmaktadır. Halbuki ortadaki paylaşılan n elemanlı olsa toplam bekleme zamanı daha az olur. Ancak kuyruk tam dolu olduğunda Ya da tam boş olduğunda bekleme gerçekleşir.

Üretici tüketici probleminin bu tür n li çözümü için tipik olarak iki semaphore kullanılır. Başlangıçta tüketici semaphor u n de üretici semaphore u sıfırdadır. Üretici thread tüketici semaphore nu kullanarak tüketici threadde üretici semaphore nu kullarak bekleme yapar.

Semaphore m_semaProducer = new Semaphore(0, 10);
Semaphore m_semaConcumer = new Semaphore(10, 10);



Şüphesiz burada kullanılan kuyruk sistemininde ayrıca korunması gerekir.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Threading;
using System.Collections;

namespace Producer_Consumer
{
   public partial class Form1 : Form
   {
      private Thread m_producer;
      private Thread m_consumer;
      private Semaphore m_semaProducer;
      private Semaphore m_semaConsumer;
```

```
private Random m_rand;
private Queue m_queue;

public Form1()
{
   InitializeComponent();

   m_rand = new Random();

   m_queue = new Queue();
   m_semaConsumer = new Semaphore(10, 10);
   m_semaProducer = new Semaphore(0, 10);

   m_producer = new Thread(new ParameterizedThreadStart(ProducerThreadProc));
   m_producer.Start(null);
   m_producer.IsBackground = true;

   m_consumer = new Thread(new ParameterizedThreadStart(ConsumerThreadProc));
   m_consumer.Start(null);
   m_consumer.IsBackground = true;

}

private void ProducerThreadProc(object o)
{
   int i = 0;

   for (; ; )
   {
      m_semaConsumer.WaitOne();
      Thread.Sleep(m_rand.Next(200));
      lock (m_queue)
      {
         m_queue.Enqueue(i);
      }
      if (i == 50)
         break;
      ++i;
      m_semaProducer.Release();
   }
}

private void ConsumerThreadProc(object o)
{
   int val;

   for (; ; )
   {
      m_semaProducer.WaitOne();
      Thread.Sleep(m_rand.Next(200));

      lock (m_queue)
```

```
            {
               val = (int)m_queue.Dequeue();
            }

            if (val == 50)
               break;
            listBox1.Items.Add(val);
            m_semaConsumer.Release();

         }
      }

   }
}
```

Mutex Nesnelerinin Kullanımı: Neredeyse tüm işletim Sisteminde varolan senkronizasyon nesnesidir. Mutex nesnesinin sahipliği bir thread tarafından alınır ve sahipliği ancak almış olan thread verebilir. Mutex nesnesi mutex sınıfıyla temsil edilmiştir. Mutex nesnesi yaratılırken mutaex sınıfının başlangıçları yoluyla nesnenin başlangıçtaki durumu yani sahipliğinin alınıp alınmayacağı belirlenebilir.

public Mutex (bool initiallyOwner)

Fonksiyonun parametresi mutex nesnesinin başlangıçtaki sahipliğinin alınıp alınmayacağına ilişkindir. Nesnenin sahipliği ReleaseMutex fonksiyonuyla bırakılır.

public void ReleaseMutex()

Wait fonksiyonları eğer mutex in sahipliğini eğer başka bir thread almışsa beklemeye yol açar değilse nesne açık durumdadır ve sahiplik alınır. Mutex nesnesi ile kritik kod şöyle oluşturulur.

```
m_mutex.WaitOne();

... Kritik Kod


m_mutex.ReleaseMutex();
```

Şüphesiz Mutex sınıfı da WaitHandle sınıfından türetildiği için Dispose fonksiyonuna sahiptir ve kullanımdan sonra Dispose fonksiyonunun çağrılması uygun olur.

Aynı thread windows sistemlerinde mutex nesnesinin sahipliğini yeniden alabilir. Fakat aldığı kadar ReleaseMutex uygulamalıdır. Örneğin biz mutex nesnesi ile üretici tüketici problemini çözemeyiz. Çünkü başka bir threadin sahipliğini almış olduğu mutex nesnesini bırakamayız.

Kernel Senkronizasyon Nesnelerinin Prosesler Arası Kullanımı: Event, Mutex, Semaphore nesneleri istenilirse isimli bir biçimde prosesler arasında da kullanılabilir. EventWaitHandle, Semaphore ve Mutex Sınıflarının ayrıca String parametreli başlangıç fonksiyonları da vardır. İki ayrı proses ortak bir isim üzerinden anlaşırsa ikisi de aynı isimli aynı türden nesneyi yarattığında aslında bunlar aynı nesnedir. Prosesler arası haberleşme mümkün olur.

Örneğin bir proses EventWaitHandle türünden nesneyiaşağıdaki gibi yaratmış olsun:
m_event = new EventWaitHandle(false, EventResetMode.AutoReset, "CSD1993);

Şimdi başka bir programında Event nesnesini yukarıdaki gibi aynı ismi vererek yarattığını düşünelim Bu durumda bu proses yeni bir Event nesnesi yaratmayacaktır. İsimler aynı olduğu için yaratılmış olanı açacaktır. Diğer bir nokta aynı ismi vererek açan diğer prosesin farklı bir modda nesneyi açmak istemesi durumunda nesne açılacak fakat programcının belirttiği modda değil ilk prosesin modunda açılmış olacaktır.

Asenkron Fonksiyonlar: Normal fonksiyon çağrıları senkrondur. Asenkron fonksiyon çağırma denilince thread yaratılarak fonksiyon çağrımı anlaşılmalıdır. Fonksiyon çağırma işlemi devam ettiği halde fonksiyondan geri dönülür. Asenkron çağrılar aslında tüm delegelere özgüdür.

Process1/program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace Process1
{
   class Program
   {
      private static EventWaitHandle m_event;

      static void Main(string[] args)
      {
         m_event = new EventWaitHandle(false, EventResetMode.AutoReset, "CSD1993");
         Console.WriteLine("Waiting process1");
         m_event.WaitOne();
         Console.WriteLine("ok...");
         //...

      }
   }
}
```

Process2/program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace Process2
{
   class Program
   {
      private static EventWaitHandle m_event;

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

```
            m_event = new EventWaitHandle(false, EventResetMode.AutoReset, "CSD1993");
            Console.WriteLine("Press any key to release other process");
            Console.ReadLine();
            m_event.Set();
            //...

        }
    }
}
```

Biz bir delege referansıyla fonksiyon çağırma operatörü kullanarak fonksiyonu çağırdığımızda aslında C# derleyicisi bu işlemi Delegate sınıfının Invoke fonksiyonuyal yapmaktadır.
Örneğin:

```
Proc p = new  Proc(Foo)

p.Invoke();
```

Biz çağırmayı aşağıdaki gibi de yapabiliriz.

```
p();
```

Delege yoluyla çağırma işlemi Invoke fonksiyonuyla değilde BeginInvoke fonksiyonuyla yapılırsa çağırma asenkron olur. Yani çağırma sırasında bir thread yaratılır ve fonksiyon o thread tarafından çağrılır.

BeginInvoke fonksiyonunun ilk n tane parametresi delege bildirimindeki parametrik yapıyla tamamen aynıdır. Son iki parametresi sırasıyla AsynCallback sınıfı türünden ve object türünden parametrelerdir. AsynCallback parametresi null geçilebilir. AsynCallback bir delegedir. Bu delegede verilen fonksiyon işlem bittiğinde çağrılacak fonksiyondur. Null geçilirse işlem bitince herhangi bir fonksiyonun çağrılmayacağı sonucu çıkarılır. BeginInvoke fonksiyonunun son object parametresi programcı tarafından ileride ele alınacağı gibi elde edilebilecek bir değerdir. Bu parametre de null geçilebilir.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Drawing;
using System.IO;
using System.Threading;

namespace ConsoleApp
{
   class Program
   {
      public static void Main(string[] str)
      {
         Proc p = new Proc(Foo);

         p.Invoke(10);

         Console.WriteLine(".....");
```

```
            Console.ReadLine();
        }


        public static void Foo(int n)
        {

            for (int i = 0; i < n; ++i)
            {
                Console.WriteLine("Foo: {0}", i);
                Thread.Sleep(1000);
            }
        }
    }

    delegate void Proc(int a);
}
```

O halde biz bir fonksiyonun başka bir thread tarafından çalıştırılmasını istiyorsak hiç thread yaratma zahmetine girmeden fonksiyonu delegeye yerleştirip BeginInvoke fonksiyonunu çağırabiliriz.

Fonksiyonları asenkron çağırdıktan sonra bazen biraz paralel ilerleyip sonra ofonksiyonun bitmesini bekleyebiliriz. İşte bu işlem EndInvoke fonksiyonuyla yapılmaktadır. BeginInvoke bize IAsyncResult türünden bir arayüz referansı verir. EndInvoke fonksiyonu da BeginInvoke fonksiyonu tarafından verilen bu arayüz referansını istemektedir.

```
Proc p =new Proc(Foo);
IAsyncResult iresult;

iresult = p.BeginInvoke(10, null, null)
//...
p.EndInvoke(iresult);
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Drawing;
using System.IO;
using System.Threading;

namespace ConsoleApp
{
    class Program
    {
        public static void Main(string[] str)
        {
            Proc p = new Proc(Foo);

            p.BeginInvoke(10, null, null);
```

```
            Console.WriteLine(".....");
            Console.ReadLine();
        }


        public static void Foo(int n)
        {

            for (int i = 0; i < n; ++i)
            {
                Console.WriteLine("Foo: {0}", i);
                Thread.Sleep(1000);
            }
        }
    }

    delegate void Proc(int a);

}
```

EndInvoke fonksiyonu aynı zamanda çağrılan asenkron fonksiyonun geri dönüş değeri ile geri dönmektedir. Yani ilgili fonksiyonun bitmesi beklenirken aynı zamanda onun gerei dönüş değeri de elde edilmektedir.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Drawing;
using System.IO;
using System.Threading;

namespace ConsoleApp
{
    class Program
    {
        public static void Main(string[] str)
        {
            Proc p = new Proc(Foo);
            IAsyncResult iresult;

            iresult = p.BeginInvoke(10, null, null);
            //....
            int val = p.EndInvoke(iresult);

            Console.WriteLine("Result: {0}", val);
        }

        public static int Foo(int n)
        {
            for (int i = 0; i < n; ++i)
            {
```

```
            Console.WriteLine("Foo: {0}", i);
            Thread.Sleep(1000);
        }

        return 100;
    }
  }

  delegate int Proc(int a);
}
```

BeginInvoke fonksyonunun AsynCallback delege parametresi işlem bitince çaprılacak fonksiyonu belirtir. Bu delegenin parametrik yapısı şöyledir.
public delegate void AsynCallback(IAsyncResult ar)
Fonksiyona geçirilecek vAsyncResult parametresi bize çağrılmış olan fonksiyon hakkında bilgi vermektedir. Bu arayüzün object türünden AsynState property si BeginInvoke fonksiyonuna geçilecek son parametreyi belirtir. Böylece biz farklı Asenkron fonksiyonlar için aynı bitiş fonksiyonlarını verebiliriz. Bu parametre belki de yeni bir işlemin başlatılması için bize yardımcı olabilir.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Drawing;
using System.IO;
using System.Threading;

namespace ConsoleApp
{
  class Program
  {
    public static void Main(string[] str)
    {
      Proc p = new Proc(Foo);
      IAsyncResult iresult;

      iresult = p.BeginInvoke(10, new AsyncCallback(CompletionProc), 500);
      Console.ReadLine();


    }

    private static void CompletionProc(IAsyncResult ia)
    {
      int val = (int) ia.AsyncState;
      Console.WriteLine("Param: {0}", val);
    }

    public static int Foo(int n)
    {
      for (int i = 0; i < n; ++i)
```

```
            {
                Console.WriteLine("Foo: {0}", i);
                Thread.Sleep(1000);
            }

            return 100;
        }
    }

    delegate int Proc(int a);
}

 using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Drawing;
using System.IO;
using System.Threading;

namespace ConsoleApp
{
    class Program
    {
        private static Proc m_proc;

        public static void Main(string[] str)
        {
            m_proc = new Proc(Foo);
            IAsyncResult iresult;

            iresult = m_proc.BeginInvoke(3, new AsyncCallback(CompletionProc), null);
            Console.ReadLine();
        }

        private static void CompletionProc(IAsyncResult ia)
        {
            //...
            Console.WriteLine("Proc Completed");

            m_proc.BeginInvoke(3, new AsyncCallback(CompletionProc), null);
        }

        public static int Foo(int n)
        {
            for (int i = 0; i < n; ++i)
            {
                Console.WriteLine("Foo: {0}", i);
                Thread.Sleep(1000);
            }

            return 100;
```

```
        }
    }

    delegate int Proc(int a);
}
```

Threadlere Özgü Alanlar(Thread Local Storage): Bazen bir değişkenin threade özgübir kopyası kullanılmak istenebilir yani paylaşılan bir nesnenin her thread için farklı bir kopyası olsun istenebilir. Bu konuya windows sistemlerinde "Thread Local Storage" denilmektedir.

Threade özgü alanları oluşturmak için Thread sınıfının AllocateDataSlot , AllocateNamedDataSlot , FreeNamedDataSlot fonksiyonları kullanılır. AllocateDataSlot fonksiyonu parametresi şöyledir.
public static LocalDataStoreSlot AllocateDataSlot()
Fonksiyon bize LocalDataStoreSlot isimli bir sınıf türünden nesne verir. AllocateNamedDataSlot fonksiyonu da aşağıdaki gibidir.
public static LocalDataStoreSlot AllocateNamedDataSlot(string name)
Daha sonra elde edilen bu nesne Thread sınıfının SetData ve GetData fonksiyonlarıyla kullanılan
public static void SetData(LocalDataStoreSlot slot, Object data)
public static object GetData(LocalDataStoreSlot slot)
Nihayet threade özgü oluşturulan bu alan FreeNamedSlot fonksiyonuyla yok edilir.
public static void FreeNamedDataSlot(string name)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Drawing;
using System.IO;
using System.Threading;

namespace ConsoleApp
{
    class Program
    {
        private static Thread m_thread1;
        private static Thread m_thread2;
        private static LocalDataStoreSlot m_slot;


        public static void Main(string[] str)
        {
            m_thread1 = new Thread(new ParameterizedThreadStart(ThreadProc));
            m_thread2 = new Thread(new ParameterizedThreadStart(ThreadProc));
            //...
            m_slot = Thread.AllocateDataSlot();

            m_thread1.Start(100);
            m_thread2.Start(200);


            m_thread1.Join();
            m_thread2.Join();
```

```
        }

        public static void ThreadProc(object o)
        {
            Thread.SetData(m_slot, o);

            Foo();
            //...
        }


        private static void Foo()
        {
            int val = (int)Thread.GetData(m_slot);

            Console.WriteLine(val);
        }
    }
}
```

.Net te daha sonraları TLS konusu daha basitleştirilmiştir. Öyleki Sınıfını Statik veri elemanı ThreadStaticAttribute sınıfıyla özniteliklendirilrse bu TLS tahsisatları otomatik yapılacaktır.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Drawing;
using System.IO;
using System.Threading;

namespace ConsoleApp
{
    class Program
    {
        private static Thread m_thread1;
        private static Thread m_thread2;
        [ThreadStatic]
        private static int m_val;

        public static void Main(string[] str)
        {
            m_thread1 = new Thread(new ParameterizedThreadStart(ThreadProc));
            m_thread2 = new Thread(new ParameterizedThreadStart(ThreadProc));
            //...

            m_thread1.Start(100);
            m_thread2.Start(200);


            m_thread1.Join();
```

```
            m_thread2.Join();

        }

        public static void ThreadProc(object o)
        {
            m_val = (int)o;

            Foo();
            //...
        }


        private static void Foo()
        {
            Console.WriteLine(m_val);
        }
    }
}
```

Çok İşlemcili Sistemler İçin Önlemler: Çok işlemcili ya da çok çekirdekli sistemlerde çok seyrek olarak karşılaşılsada önemli bir problem vardır. Birden fazla threadin paylaşılan ortak bir değişkene erişmesi durumunda threadler den biri yazma yapıyorsa eğer threadlerfarklı işlemcilere atanmışsa ve işlem tesadüfen aynı anda gerçekleşmişse bozuk değer oluşabilmektedir. Çok işlemcili sistemlerde bu bozulma çok seyrek olsada bir olasılık olarak her zaman söz konusudur. Bunu dikkate almayan programlar böcekli kabul edilmektedir. Örneğin threadlerden biri m_val değişkenine değer yazarken diğeri de bunu okuyor olsun. Bu iki thread tesadüfen paralel çalışan iki farklı işlemciye ya da çekirdeğe atanmış olabilir ve bu okuma yazma işlemleri tesadüfen aynı anda gerçekleşmiş olabilir. İşte bu durumda bozulma oluşabilir.

Bu tür durumlarda bu kadar seyrek oluşma olasılığı olan bir durum için monitor sınıfıyla ya da Lock deyimiyle senkronizasyon uygulamaya çalışmak kötü bir tekniktir. Intel işlemcilerinde bir Makine komutunu bu anlamda atomik yapan özel komutlar vardır. C# da Volatile belirleyicisi CLR sistemine ilgili değişkenin threadler arasında paylaşıldığını ve çok işlemci için tehdit oluştuğunu belirtmektedir. Bu durumda çok işlemcili sistemler için threadler arasında paylaşılacak değişkenlerin kesinlikle volatile bildirilmesi gerekir.

```
private volatile int  m_val;
```

Thread Güvenli Fonksiyonlar: Aynı anda birden fazla thread tarafından çalıştırıldığı halde soruna yol açmayan fonksiyonlara Thread güvenli fonksiyonlar diyoruz. Genel olarak sınıfın veri elemanlarını kullanan fonksiyonlar(yani yerel değişkenlerden başka değişkenleride kullanan fonksiyonlar) thread güvenli değildir.

Bir fonksiyonun thread güvenli olup olmadığını önceden bilmek önemlidir. .net sınıf sisteminde genel olarak sınıfların static fonksiyonları thread güvenlidir. Fakat static olmayan fonksiyonlar thread güvenli değildir. Bir fonksiyonu thread güvenli yapabilmek için o fonksiyonun kullandığı sınıfın veri elemanlarını ThreasStatic özniteliği ile özniteliklendirerek o veri elemanını diğer threadlerden ayırmak gerekir.

Thread Güvenli Collection Nesneleri: .net sınıf sisteminde diğer sınıflarda olduğu gibi sınıfların yalnızca static fonksiyonları thread güvenlidir. Collection sınıflar genel olarak threadler arası

kullanımı yoğun sınıflardır. Şüphesiz bütün bir fonksiyonu dışarıdan lock işlemi ile kilitlemek yerine içeriden gerekli küçük bölümleri kilitlemek daha etkin kullanıma yol açar. Bu nedenle performans dikkate alındığında Collection sınıfların daha verimli bir biçimde thread güvenli yaplması önemlidir. (Yani biz thread güvenli olmayan Collection sınıfların fonksiyonlarını dışarıdan kritik kod içine alıp kullanabiliriz. Fakat bu fonksiyonların kendi içinde kritik kod oluşturması şüphesiz daha verimlidir.)

Her Collection sınıfın Synchronized isimli bir static fonksiyonu vardır. Bu fonksiyon parametre olarak ilgili collection türünden bir nesne alır ve bize aynı collection türünden fakat thread güvenli olan yeni bir nesne verir.

Örneğin:

```
ArrayList m_a;
//...
m_a = ArrayList.Synchronized(new ArrayList());
```

Artık burada geri verilen nesnesyi thread güvenli olarak kullanabiliriz.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Threading;
using System.Collections;

namespace Producer_Consumer
{
   public partial class Form1 : Form
   {
      private Thread m_producer;
      private Thread m_consumer;
      private Semaphore m_semaProducer;
      private Semaphore m_semaConsumer;
      private Random m_rand;
      private Queue m_queue;

      public Form1()
      {
         InitializeComponent();

         m_rand = new Random();
         m_queue = Queue.Synchronized(new Queue());

         m_queue = new Queue();
         m_semaConsumer = new Semaphore(10, 10);
         m_semaProducer = new Semaphore(0, 10);

         m_producer = new Thread(new ParameterizedThreadStart(ProducerThreadProc));
```

```
            m_producer.Start(null);
            m_producer.IsBackground = true;

            m_consumer = new Thread(new ParameterizedThreadStart(ConsumerThreadProc));
            m_consumer.Start(null);
            m_consumer.IsBackground = true;

        }

        private void ProducerThreadProc(object o)
        {
            int i = 0;

            for (; ; )
            {
                m_semaConsumer.WaitOne();
                Thread.Sleep(m_rand.Next(200));
                m_queue.Enqueue(i);
                if (i == 50)
                    break;
                ++i;
                m_semaProducer.Release();
            }
        }

        private void ConsumerThreadProc(object o)
        {
            int val;

            for (; ; )
            {
                m_semaProducer.WaitOne();
                Thread.Sleep(m_rand.Next(200));

                val = (int)m_queue.Dequeue();
                if (val == 50)
                    break;
                listBox1.Items.Add(val);
                m_semaConsumer.Release();

            }
        }

    }
}
```

Resource Use(Kaynak Kullanımı): Pek çok programda onlarca küçük jpg, bmp gibi görüntü dosyaları waw gibi ses dosyaları kullanılmaktadır. Bu dosyaların programın çalışma zamanı sırasında yüklenerek işlerlik kazanması güvenli bir yaklaşım değildir. Bu küçük dosyalardan biri bile silinse program muhtemelen çalışmayacaktır. İşte bu tür durumlarda onlarca küçük dosyanın kurulum sırasında başka bir makinaya çekilmesi yerine bu dosyaların exe dosya içine gömülmesi çok daha etkin bir yöntemdir. İşte Kaynak(Resource) konusu bununla ilgilidir.

Herhangi bir dosya bir kaynak biçiminde Assembly içine gömülebilir. Bunun için csc.exe programında komut satırında /resource seçeneği kullanılmaldır. /resource seçeneğiningenel biçim aşağıdaki gibidir.
/resource:<dosya ismi>

csc.exe derleyicisi belirtilen dosyayı bir kaynak olarak assembly dosyasına gömmektedir. Bu tür kaynaklara aynı zamanda gömülü kaynaklar embedded resources ya da manifest resources denilmektedir. Assembly dosyası içine gömdüğümüz kaynaklar program içinden çekilerek kullanılabilir. Bunun için şu adımlar izlenmelidir:

1. Öncelikle kaynağın içinde bulunduğu Assembly nin Assembly nesnesini elde etmek gerekir. Bu işlem Assembly sınıfının GetXXXAssembly isimli static fonksiyonları ile yapılabilir. Örneğin Assembly sınıfının GetExacutingAssembly fonksiyonu bize çalışmakta olan kodun içinde bulunduğu Assemblynin Assembly nesnesini vermektedir.
2. Assembly sınıfının GetManifestResourceStream fonksiyonları kullanılarak kaynak bilgileri elde edilebilir. Bu fonksiyonlar bize bir stream nesnesi vermektedir. Bizbu stream nesnesinden okuma yaptığımızda aslında assembly içine gömülmüş kaynaktan okuma yapmış oluruz

```
Assembly a = Assembly.GetExecutingAssembly();
try
{
Stream s = a.GetMaifestResourceStream("message.txt");
StreamReader sr = new StreamReader(s);
Console.WriteLine(sr.ReadToEnd());
        }
        catch (Exception e)
        {
                Console.WriteLine(e.Message);
        }
```

Visual Studio IDE si kullanılarak bir kaynağı manifest kaynak olarak Assembly ye gömmek için ilgili dosya projeye eklenir sonra fareyle sağ tuşa basılıp propertys menüsüne gelinir Bulid Action EmbeddedResourse biçiminde seçilir.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace ResourceSample
{
   public partial class Form1 : Form
   {
      private Image m_image;

      public Form1()
      {
         InitializeComponent();
```

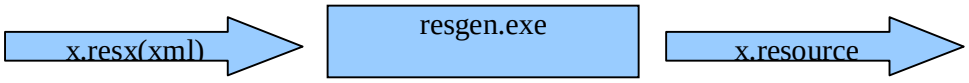
```
            try
            {
                m_image = new Bitmap(@"E:\DotNetAppBasic\ResourceSample\forest.jpg");

            }

            catch (Exception e)
            {
                MessageBox.Show(e.Message);
            }
        }

        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;

            g.DrawImage(m_image, this.ClientRectangle);
        }
    }
}

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Reflection;

namespace ResourceSample
{
    public partial class Form1 : Form
    {
        private Image m_image;

        public Form1()
        {
            InitializeComponent();

            try
            {
                Assembly a = Assembly.GetExecutingAssembly();
                m_image = new Bitmap(a.GetManifestResourceStream("ResourceSample.Forest.jpg"));
            }

            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
```

```
    private void Form1_Paint(object sender, PaintEventArgs e)
    {
        Graphics g = e.Graphics;

        g.DrawImage(m_image, this.ClientRectangle);
    }
  }
}
```

XML Temelli Kaynaklar: Kaynak kullanımının temellerini yukarıda açıkladık. Ancak Microsoft Visual Studio 2005 ile birlikte özellikle IDE üzerinde kaynak işlemlerini daha kolay yapabilmek için XML tabanlı kaynak kullanımını tasarlamıştır.

XML tabanlı kaynak kullanımının anahtar noktaları şunlardır.

- İrili ufaklı tüm kaynaklar farklı dosyalar biçiminde değil bir XML dosyası biçiminde text tabanlı olarak ifade edilir.
- Oluşturulan bu kaynak XML kaynak dosyası resgen.exe denilen özel bir programla derlenir ve bir .resource dosyası elde edilir.



Elde edilen bu .resource dosyası Assembly dosyasına yukarıda açıklandığı gibi gömülür. Buradaki .xml dosya formatı ve .resource dosya formatı microsoft tarafından açıklanmıştır. Fakat programcının bunları bilmesi gerekmektedir.

- .net kütüphanesinde ResourceManager isimli sınıf kullanılarak bu Assembly ye gömülmüş olan .Resource kaynaklar elde edilebilir.

Visual Studio 2005 IDE si ile birlikte kaynak kullanımı iyice basitleştirilmiştir. Yukarıda açıkladığımız xml tabanlı kaynak projeye eklenmiş olan .resx uzantılı dosyadır. Visual Studio IDE si eğer istenirse Assembly içine gömülmüş olan .resource dosyasının içinden ilgili kaynağı alan property lere sahip bir kod dosyasını bizim için de oluşturmaktadır. Bu dosya IDE de x.resx dosyası için x.Designer.cs dosyasıdır. Şüphesiz programcı x.Designer.cs dosyasının içindeki kodlar yerine kaynağı ResourceManager sınıfını kullanarak doğrudan alabilir.

Bu durumda kaynak işlemlerini IDE kullanarak pratik bir biçimde gerçekleştirmek için projeye resx dosyası eklenir(Zaten proje açıldığında projede böyle bir dosya vardır) resx dosyasına görsel editörle kayna eklenir. Proje build yapıldığında bu .resx dosyası IDE tarafından resgen.exe kullanılarak .resource formatına dönüştürülüp Assembly ye eklenir. Bizim yapacağımız tek şey o kaynak için üretilmiş property yi kullanmaktır.

x.Designer.cs içindeki sınıf projenin isim alanı.properties diye isimlendirilen bir isim alanına yerleştirilmiş Resources isimli bir sınıftır. Kaynağı alan propertyler statik propertylerdir. O halde örneğin projemizin isim alanı CSD ise Message isimli bir kaynağı elde eden property nin nitelikli ismi CSD.Properties.Resources.Message biçimindedir.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
```

```
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Reflection;

namespace ResourceSample
{
   public partial class Form1 : Form
   {
      private Image m_image;

      public Form1()
      {
         InitializeComponent();

         this.Text = ResourceSample.Properties.Resources.Message;

         try
         {
            Assembly a = Assembly.GetExecutingAssembly();
            m_image = new Bitmap(a.GetManifestResourceStream("ResourceSample.Forest.jpg"));
         }

         catch (Exception e)
         {
            Console.WriteLine(e.Message);
         }
      }

      private void Form1_Paint(object sender, PaintEventArgs e)
      {
         Graphics g = e.Graphics;

         g.DrawImage(m_image, this.ClientRectangle);
      }
   }
}
```

.net ortamında bu IDE nin x.Designer.cs dosyası içinde bize property ile sunduğu çalışma tarzına "typed resource" da denilmektedir.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Reflection;

namespace ResourceSample
```

```
{
   public partial class Form1 : Form
   {
      private Image m_image;

      public Form1()
      {
         InitializeComponent();

         m_image = ResourceSample.Properties.Resources.Forest;

      }

      private void Form1_Paint(object sender, PaintEventArgs e)
      {
         Graphics g = e.Graphics;

         g.DrawImage(m_image, this.ClientRectangle);
      }
   }
}
```

XML tabanlı kaynak kullanımını daha yönetilebilir yapmak için Visual Studio IDE si uygulama için toplam 1 tane olan genel bir "resources.resx" isimli bir kaynak dosyası ile her form için ayrı bir .resx kaynak dosyası oluşturmaktadır. Programcının eğer ilgili kaynak belirli bir forma yönelik ise o kaynağı o formun resx dosyasına koyması tavsiye edilmektedir. Fakat dikkat edileceği gibi uygulama genelindeki resources.resx dosyası için işi kolaylaştıran Resources.Designer.cs dosyası üretildiği halde formlar için böyle bir .cs dosyası üretilmemektedir. Yani Form a ilişkin resx içindeki kaynaklar ResourceManager sınıfı kullanılarak elde edilmelidir. Ancak programcı isterse form a ilişki resx dosyasının üzereine gelip properties menüsünden Custom Tool ResxFileCodeGenerator girerse bu resx dosyası içinde x.Designer.cs dosyası üretilir.

Elimizde varolan bir dosyayı XML tabanlı kaynağa IDE yoluyla ekleyebiliriz.Bunun için Solution Explorer da .resx uzantılı dosyaya çift tıklanır ve burada Edd Resource/Add Existinf File yapılır. .resx uzantılı XML tabanlı kaynak dosyasına eklenen dosyalar eğer resim dosyası ise kaynak editör bunun için image türünden property yazar.

Eğer biz kaynağı uygulamanın kaynak dosyasına yani resources.resxe yerleştirmişsek kaynak yukarıdada belirtildiği gibi properties isim alanındaki Resources sınıfının statik propertyleri yoluyla çekilir. Yok eğer Formun kaynak dosyasına eklemişsek zaten kaynağı elde eden property formun içine yazılmaktadır.

Ayrıca IDE de istenirse kaynağa eklenmiş olan dosyanın Assembly dosyasına gömülmemeside sağlanabilir. Bunun için ilgili kaynağın üzerine gelinip Properties menüsünden Copy to output file seçeneğinde copy always seçilir.

IDE bazı kontrollerin image propertyleri söz konusu olduğunda zaten XML tabanlı .resx dosyalarına otomatik başvurmaktadır. Zaten import seçeneği hem kaynağa ekleme hem de kullanma işlemlerini bir arada yapmaktadır.

Anahtar Notlar: Microsoft eski windows sürümlerinde oyun kartlarına ilişkin oyunlarda kullandığı kartları Cards.dll isimli bir dosyada toplamıştır. İstersek bu dll içinde ki bu oyun kartlarını .bmp

olarak save edebiliriz.

Programa İlişkin Ayarlar(Application Settings): Pek çok uygulama sonlandırıldıktan sonra saklanması gereken pek çaok ayar bilgileri kullanmaktadır. Kullanıcı çeşitli bileşenlerin renklerini değiştirebilir Ya da bir open diaalog penceresi sonlandırıldığı yerden yeniden açılmak istenebilir. Bu tür ayar bilgilerinin saklanması için iki yöntem kullanılır.

1. Windows un hem kendi ayarlarının hemde programcının ayarlarının saklandığı registry dosyasını kullanmak.
2. Tamamen bu amaçla oluşturulmuş bir data dosyası kullanmak.

.net 2005 ile birlikte uygulamaya ilişkin ayar bilglerinin yine xml tabanlı dosyalarda saklanması sağlanmaktadır. Sihirbazla proje oluşturulduğunda Ide otomatik olarak Settings.setting isimli bir xml dosyası ile Settings.Designer.cs isimli bir property dosyası oluşturmaktadır. Uygulamaya ilişkin ayarlar IDE yoluyla Menu kullanılarak oluşturulabilir. Projeye birden fazla settings eklenebilir. Proje built yapıldığında tüm bu settings dosyaları birleştirilir ve .exe.config dosyası oluşturulur. Bu dosyada kurulum sırasında hedef bilgisayara yüklenmelidir. .exe.config uzantılı bu dosya bir xml data dosyasıdır. Fakat bizim oluşturacağımız sıradan bir data dosyasından farkı bunun IDE destekli bir biçimde pratik kullanılabilmesidir.

Uygulamaya ilişkin ibr ayar oluşturabilmek için Solution Explorer da setting dosyası üzerine çift tıklanır. Önümüze çıkan girdiğimiz tüm menüde her ayar elemanı için Settings.Designer.cs dosyasında bir property oluşturulmaktadır.

IDE Settings.Designer.cs dosyası içerisinde default isminde ayarları temsil eden bir sınıf geri döndüren Settings isimli sınıfın bir property sini oluşturur. O halde örneğin message isminde bir ayar elemanı oluşturmuş olalım. Bunun program içindeki kullanımı şöyle olacaktır.

string msg = Properties.Settings.Default.Message;

Burada properties kaynak konusunda da gördüğümüz isim alanıdır. Settings IDE tarafından oluşturulmuş bir sınıftır. Default bu sınıfın static bir property sidir ve bize Settings sınıfı türünden bir nesne vermektedir. Message ise bizim ayar elemanı için oluşturulan bir propertydir.

Ayrıca ayar elemanları Scope bakımından User ve Application olmak üzere ikiye ayrılmıştır. Bir ayar scope bakımından user durumundaysa kullanıcıya özgüdür. Yani her kullanıcı login olduğunda kendine özgü ayarları kullanır. Fakat ayar scope bakımından Application ise kullanıcıdan bağımsız olarak yani her kullanıcı için aynı olacak biçimde etki gösterir.

Ayrıca settings sınıfının (Bu sınıf ApplicationSettingsBase sınıfından türetilmiştir.) bazı faydalı fonksiyonları vardır.

Reload Fonksiyonu ayar propertylerini .exe.config dosyasından yükler. Reset fonksiyonu propertylere default değerlerini yükler. (Yani IDE ekranındaki elle girilen değerler) ve nihayet Save fonksiyonu da property içindeki değerleri .exe.config dosyasına geri yükler.

Bunların dışında programcı ayarlara değer girilirken ya da ayarlar save edilirken çeşitli kontrollerin yapılmasını isteyebilir. Bunun için Settings sınıfının (ApplicationSettingsBase sınıfının) çeşitli event elemanları kullanılır.

Application ayarları normal olarak read only dir. Fakat user ayarları read/write biçimindedir. Dolayısıyla her user için program çalışırken ayrı bir ayar dosyası vardır ve bu ayar dosyası

\users\kullanıcı adı\Local Settings\içerisindedir.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace SampleSettings
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();

         this.Left = Properties.Settings.Default.FormX;
         this.Top = Properties.Settings.Default.FormY;

      }

      private void Form1_Click(object sender, EventArgs e)
      {


      }

      private void Form1_FormClosing(object sender, FormClosingEventArgs e)
      {
         Properties.Settings.Default.FormX = this.Left;
         Properties.Settings.Default.FormY = this.Top;

         Properties.Settings.Default.Save();

      }


   }
}
```

Anahtar Notlar: Users ayarlarını property yoluyla değiştirdiğimizde otomatik bir biçimde bu ayarlar kalıcı hale gelmemektedir. Ayarları kalıcı hale getirebilmek için Save fonksiyonunu çağrılması gerekir. Bu fonksiyonun çağrılması için uygun bir yer FormClosing mesajıdır.

Form Editör Kullanarak Otomatik Ayar Oluşturma: .net in kontrollerine ilişkin ayarlar form editör kullanarak veri bağlama(data binding) özelliği de kullanılarak kolay bir biçimde kalıcı hale getirilebilmektedir. Bunu sağlamak için Form editörde kontrol üzerine gelinir. Properties menüsüne geçilir ve orada Application Setting kısmına gelinir. Burada hazır bir biçimde Location ve test bölümü vardır fakat farklı bölümleri biz oluşturabiliriz.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace SampleSettings
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();

      }

      private void Form1_Click(object sender, EventArgs e)
      {


      }

      private void Form1_FormClosing(object sender, FormClosingEventArgs e)
      {
         Properties.Settings.Default.Save();

      }

      private void button1_Click(object sender, EventArgs e)
      {
         ColorDialog cd = new ColorDialog();

         if (cd.ShowDialog() == DialogResult.OK)
         {
            this.BackColor = cd.Color;
         }
      }


   }
}
```

Burada new seçeneği ile biz bir property için yeni bir kalıcı ayar bağlantısı yapabiliriz. Bu bağlantıyı görsel olarak yaptıktan sonra IDE arka planda diğer bütün ayarlamaları yapmaktadır. Fakat yine save işlemi programcı tarafından yapılmalıdır.

```
using System;
using System.Collections.Generic;
```

```
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace SampleSettings
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();

      }

      private void Form1_Click(object sender, EventArgs e)
      {


      }

      private void Form1_FormClosing(object sender, FormClosingEventArgs e)
      {
         Properties.Settings.Default.Save();

      }

      private void button1_Click(object sender, EventArgs e)
      {
         ColorDialog cd = new ColorDialog();

         if (cd.ShowDialog() == DialogResult.OK)
         {
            this.BackColor = cd.Color;
         }

         this.label1.Text = "Max Score: 1234";
      }


   }
}
```

Öznitelikler(Attributes): Öznitelikler sınıf gibi yapı gibi, veri elemanı, property gibi, fonksiyon gibi pek çok programlama elemanına iliştirilebilir. İliştirme işlemi köşeli parantezler içinde ilgili elemanın öncesinde gerçekleştirilir. **Örneğin:**

```
[My Atrribute]
class Sample
{
```

```
        //...
}
```

İlgili elemana iliştirdiğimiz öz nitelik sınıfsal bir biçimde metadata olarak Assembly dosyasına yerleştirilir.

Köşeli parantezler içinde iliştirdiğimiz özniteliklerin Öznitelik sınıflarına ilişkin olması gerekir. System isim alanındaki Attribute isimli sınıftan türetilen sınıflara öznitelik sınıfları denilir.
Örneğin:

```
class MyAttribute : Attribute
{
        //...
}
[MyAttribute]
class Sample
{
        //...
}
```

Attribute sınıfı Abstract bir sınıftır. Dolayısıyla bu sınıftan türetme yapmazsak nesne yaratamayız. Bir elemana bir öznitelik iliştirildiğinde derleyici öznitelik sınıfı türünden bir nesne yaratır. Bu nesnenin bilgilerini metadata olarak Assembly dosyasına yerleştirir. Köşeli parantezler içinde aslında bir başlangıç fonksiyonu çağırma ifadesi yer alır. Öznitelik nesnesi bu başlangıç fonksiyonuyla yaratılır. Örneğin:

```
[MyAttribute("This is a test")]
class Sample
{
        //...
}
```

Burada MyAttribute sınıfının string parametreli başlangıç fonksiyonu çağrılarak öznitelik taratılacak. Nesnenin dataları da metadata olarak Assmebly dosyasına gömülecektir. Öznitelik sınıf isminden sonra hiç fonksiyon parantesi açılmassa nesne için default başlangıç fonksiyonu çağrılır. Yani:

```
[MyAttribute]
```

ile

```
[MyAttribute()]
```

aynı anlamdadır. Şüphesiz bir programlama elemanına birden fazla öznitelik nesnesi atanabilir. Sintaks olarak bu nesneler aralarına virgül konularak aynı köşeli parantez içerisinde Ya da ayrı ayrı köeli parantezler içerisinde belirtilebilir.
Örneğin:

```
[MyAttribute, YourAttribute]
class Sample
{
        //...
}
```

ya da aynı şeyi şöyle de yapabiliriz.

```
[MyAttribute]
[YourAttribute]
class Sample
{
```

```
        //...
}
```
aynı anlamdadır.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace AttributeSample
{
   class Program
   {
      static void Main(string[] args)
      {
      }
   }


   [
   MyAttribute
      (

      "This is a test")]  [YourAttribute(100)]class Sample
   {
      [YourAttribute(100)]
      private int m_a;

   }

   class MyAttribute : Attribute
   {
      private string m_msg;

      public MyAttribute(string msg)
      {
         m_msg = msg;
      }

      public string Msg
      {
         get { return m_msg; }
         set { m_msg = value; }
      }
   }

   class YourAttribute : Attribute
   {
      private int m_no;

      public YourAttribute(int no)
      {
         m_no = no;
```

```
        }

        public int No
        {
            get { return m_no; }
            set { m_no = value; }
        }
    }
}
```

Bir bildirime iliştirilen öznitelik nesnesi programın çalışma zamanı sırasında programcı tarafından ya da CLR tarafından elde edilebilir.

Bunun için sırasıyla şu işlemler yapılır.

1. Bu aşamada öznitelik nesnesinin iliştirildiği programlama elemanının içinde bulunduğu Assembly ye ilişkin Assembly nesnesi elde edilir. Bu işlem Assembly sınıfının GetAssembly fonksiyonuyla, GetExecutingAssembly fonksiyonuyla ya da GetCallingAssembly fonksiyonuyla gerçekleştirilebilir.
2. Öznitelik bilgileri elde edilecek türün type nesnesi elde edilir. Type nesnesi GetType fonksiyonuyla ya da Typeof operatörü ile elde edilebilir.
3. Type sınıfının GetCustomAttributes fonksiyonları bize tür e iliştirilmiş bütün öznitelik nesnelerini bize vermektedir.

public abstract Object[] GetCustomAttributes(bool inherit) fonksiyonun parametresi true girilirse türün taban sınıflarına ilişkin öznitelik nesneleri de elde edilir. Fonksiyonun geri dönüş değeri aslında metadata olara Assembly e iliştirilmiş olan öznitelik nesnelerini vermektedir.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace AttributeSample
{
    class Program
    {
        static void Main(string[] args)
        {
            Type t = typeof(Sample);

            foreach (object o in t.GetCustomAttributes(false))
            {
                MyAttribute ma = o as MyAttribute;

                if (ma != null)
                {
                    Console.WriteLine(ma.Msg);
                }
            }


        }
```

```
}


[MyAttribute("This is a test")]
[YourAttribute(100)]
class Sample
{
   //...
}

class MyAttribute : Attribute
{
   private string m_msg;

   public MyAttribute(string msg)
   {
      m_msg = msg;
   }

   public string Msg
   {
      get { return m_msg; }
      set { m_msg = value; }
   }
}

class YourAttribute : Attribute
{
   private int m_no;

   public YourAttribute(int no)
   {
      m_no = no;
   }

   public int No
   {
      get { return m_no; }
      set { m_no = value; }
   }
}
```

Örneğin Sample sınıfına ilişkin bu sınıfa iliştirilmiş olan MyAttribute türünden öznitelik nesnelerini şöyle elde edebiliriz.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace AttributeSample
{
```

```
class Program
{
   static void Main(string[] args)
   {
      Type t = typeof(Sample);

      foreach (object o in t.GetCustomAttributes(false))
      {
         MyAttribute ma = o as MyAttribute;

         if (ma != null)
         {
            Console.WriteLine(ma.Msg);
         }

         YourAttribute ya = o as YourAttribute;

         if (ya != null)
         {
            Console.WriteLine(ya.No);
         }
      }


   }
}


[MyAttribute("This is a test")]
[YourAttribute(100)]
class Sample
{
   //...
}

class MyAttribute : Attribute
{
   private string m_msg;

   public MyAttribute(string msg)
   {
      m_msg = msg;
   }

   public string Msg
   {
      get { return m_msg; }
      set { m_msg = value; }
   }
}

class YourAttribute : Attribute
```

```
    {
        private int m_no;

        public YourAttribute(int no)
        {
            m_no = no;
        }

        public int No
        {
            get { return m_no; }
            set { m_no = value; }
        }
    }
}
```

Görüldüğü gibi program çalışırken biz nesneye iliştirilmiş olan öznitelik nesnelerini elde edebilmekteyiz. Type sınıfının aşağıdaki gibi bir GetCustomAttribute fonksiyonu da vardır. *public abstract Object[] GetCuıstomAttribute(Type attributeType, bool inherit)* Bu

fonksiyonun birinci parametresi öznitelik sınıfına ilişkin Type nesnesidir. Yani biz bu fonksiyonla örneğin bir sınıfa iliştirilmiş yalnızca MyAtrribute türünden nesneleri elde edebiliriz. Örneğin:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace AttributeSample
{
    class Program
    {
        static void Main(string[] args)
        {
            Type t = typeof(Sample);

            foreach (MyAttribute ma in t.GetCustomAttributes(typeof(MyAttribute), false))
            {
                Console.WriteLine(ma.Msg);
            }


        }
    }


    [MyAttribute("This is a test")]
    [YourAttribute(100)]
    class Sample
    {
        //...
    }
```

```
   class MyAttribute : Attribute
   {
      private string m_msg;

      public MyAttribute(string msg)
      {
         m_msg = msg;
      }

      public string Msg
      {
         get { return m_msg; }
         set { m_msg = value; }
      }
   }

   class YourAttribute : Attribute
   {
      private int m_no;

      public YourAttribute(int no)
      {
         m_no = no;
      }

      public int No
      {
         get { return m_no; }
         set { m_no = value; }
      }
   }
}
```

Diğer programlama elemanlarına iliştirilmiş olan öznitelik nesnelerinin nasıl elde edileceği Reflection konusu içerisinde ele alınacaktır.

Öznitelik sınıflarına ilişkin iki tür parametre bilgisi vardır.

1. Konumsal(positional) parametreler
2. İsimli(named) parametreler

Öznitelik sınıfının başlangıç fonksiyonlarında bulunan parametrelere konumsal parametreler denir ve bu parametreler eğer o başlangıç fonksiyonu kullanılıyorsa kesinlikle bulundurulmak zorundadır. Öznitelik sınıflarının public veri elemanları ve propertyleri isimli parametrelerdir. Bunlar konumsal parametrelerden sonra isim = değer(values) biçiminde belirtilirler.

Örneğin:

```
class MyAttribute : Attribute
{
        private string m_msg;
        private int m_no;

        public MyAttribute(string msg)
```

```
        {
                //...
        }

        public int no
        {
        get{ return m_no}
        set { m_no = value;}
        }
        //...
}
```

Burada string parametresi konumsal no parametresi ise isimseldir. İsimli parametreler hiç belirtilmeyebilir. Fakat belirtilecekse konumsal parametrelerden sonra isim = value biçiminde belirtilmelidir.

Örneğin:

```
[MyAttribute ("This is a test", No = 123)]
class Sample
{
        //...
}
```

İsimli parametreler birden fazla bulunabilir. Bunların hangi sırada belirtildiğinin bir önemi yoktur.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace AttributeSample
{
   class Program
   {
      static void Main(string[] args)
      {
         Type t = typeof(Sample);

         foreach (MyAttribute ma in t.GetCustomAttributes(typeof(MyAttribute), false))
         {
            Console.WriteLine("{0} {1}", ma.Msg, ma.No);
         }


      }
   }


   [MyAttribute("This is a test", No = 123)]
   class Sample
   {
      //...
   }

   class MyAttribute : Attribute
```

```
    {
        private string m_msg;

        public string Msg
        {
            get { return m_msg; }
            set { m_msg = value; }
        }
        private int m_no;

        public MyAttribute(string msg)
        {
            m_msg = msg;
        }

        public int No
        {
            get { return m_no; }
            set { m_no = value; }
        }


        //...

    }


}
```

Öznitelik sınıflarının isimlerinin Attribute son ekiyle oluşturulması zorunlu değildir fakat gelenekseldir. Örneğin MyAtrribute gibi. C# standartlarına göre bir öznitelik sınıfı eğer XXXAttribute biçiminde isimlendirilmişse biz onu XXX biçiminde kullanabiliriz. Örneğin SerializableAttribute öznitelik sınıfını biz şöyle kullanabiliriz: [Seriliazable]
Şüphesiz bu kullanım yalnızca [] içinde kullanılabilir. Fakat XXXAttribute yerine öznitelik sınıfları yalnızca XXX biçiminde de isimlendirilebilir. Bu durumda sınıf ismi doğal olarak XXX biçiminde kullanılır.

Hem XXX biçiminde hem de XXXAttribute isminde iki öznitelik sınıfı bir arada bulunabilir. Bu sınıfların birarada bulunması değil köşeli parantez içinde XXXAttribute ün XXX biçiminde kullanılması soruna yol açar. Fakat [] içinde XXXAttribute şeklinde kullanım soruna yol açmaz.

.Net sınıf kütüphanesinde çok sayıda öznitelik sınıfıda bulunmaktadır. Programcının kütüphane içindeki programlama elemanlarını incelerken bu elemanlara iliştirilmiş öznitelik sınıflarına dikkat etmesi tavsiye edilir. Çünkü bu öznitelik nesneleri igii programlama elemanı hakkında ipuçları vermektedir.

Öznitelik Bilgileri Neden Kullanılır?: Öznitelik bilgileri bir programlama elemanı hakkında ek birtakım bilgiler vermektedir. Öznitelik nesneleri derleyici tarafından CLR tarafından, programcının kendisi tarafından kullanılıyor olabilir. Bir elemana öznitelik iliştirmenin bir anlamı vardır. Örneğin bir Enum türüne FlagsAttribute isimli öznitelik iliştirilirse bu Enum türünün elemanlarının “|” operatörü ile kombine edilebileceği anlamı çıkar. Programcı enum türüne bunun iliştirildiğini gördüğünde hemen bu bilgiyi edinmiş olur. Üstelik bu bilgi IDE gibi programlar tarafından da

faydalı amaçlarla kullanılabilir.

C# derleyicisi de bazı özniteliklere daha derleme aşamasında özel önem vermektedir. Örneğin biz bir sınıf ya da yapıya Serliazable özniteliğini iliştirirsek daha derleme aşamasında derleyici sınıfın serihalegetirilebilme özelliğini kontrol eder.

Bir türe ya da programlama elemanına iliştirdiğimiz öznitelik onun belirli bir özelliğe uygunluğunu da belirtiyor olabilir. Böylece bu özelliğe uygunluk programın çalışma zamanı sırasında denetlenebilmektedir. Bir program bir türe bir özniteliğin iliştirilmiş olduğunu gördüğünde ona özel bir işlem uygulayabilir.

Öznitelik Sınıflarına Uygulanan Öznitelik Sınıfları: Öznitelik sınıflarına uygulanan en önemli öznitelik sınıfı AttributeUsageAttribute sınıfıdır. Bu öznitelik sınıfının başlangıç fonksiyonu yani konumsal parametresi AttributeTarget isimli bir Enum türündendir. Bu parametre ilgili özniteliğin hangi program elemanlarına uygulanabileceğini belirtmektedir.

Örneğin:

```
[AttributeUsage(AttributeTargets.Class)]
calss MyAttribute : Attribute
{
        //....
}
```

Burada MyAttribute isimli öznitelik sadece sınıflara uygulanır.
AttributeUsage Attribute sınıfının AllowMultiple isimli bool türden isimli parametresi ilgili özniteliğin birden fazla kez aynı elemana uygulanıp uygulanmayacağını belirtir. Normal olarak aynı öznitelik aynı elemana birden fazla kez uygulanamaz.

```
[AttributeUsage(AttributeTargets.Class, AllowMultiple = true)]
calss MyAttribute : Attribute
{
        //....
}
```

Burada artık biz MyAttribute isimli özniteliği aynı elemana birden fazla kez uygulayabiliriz.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace AttributeSample
{
   [My]
   [My]
   class Program
   {

      static void Main(string[] args)
      {

      }
   }
```

```
   [AttributeUsage(AttributeTargets.Class, AllowMultiple = true)]
   class MyAttribute : Attribute
   {
      //...
   }
}
```

Yansıtma (Reflection) İsimli: Bir .net programında kullandığımız sınıf, yapı gibi tüm türlere ilişkin bilgiler, tüm fonksiyonlara ilişkin bilgiler, tüm veri elemanlarına ve diğer programlama elemanlarına ilişkin bilgiler derleme sırasında derleyici tarafından Assembly dosyasının (PE Format) metadata bölümüne yazılmaktadır. Aynı zamanda .net sınıf sistemi içinde bu bilgilerin elde edilmesini ağlayan sınıflarda vardır. İşte bir Assembly içindeki Metadata ları alarak üzerinde işlem yapmamaya yansıtma işlemleri denilmektedir. Bu sınıflar System.Reflection isim alanı içindedir.

Yansıtma işlemlerinin ilk adımı ilgili Assemblye ilişkin bilgilerin elde edilmesidir. Bir Assembly dosyasının bütünü daha önce belirtildiği gibi Assembly sınıfıyla temsil edilmektedir. İlgili Assembly için Assembly nesnesinin elde edilmesi Assembly sınıfınınGetAssembly, GetCallingAssembly, GetExacutingAssembly, GetEntryAssembly fonksiyonlarıyla elde edilir.

İlgili Assemly nin bilgileri elde edildikten sonra sıra bu Assembly içindeki türlerin bilgilerin elde edilmesine3 gelmiştir. Bunlara; Sınıf, enum, delege, yapı, fonksiyon, Arayüz, tür (Type) denilmektedir. Sistem her bir tür için bir type nesnesi oluşturmaktadır. İşte Assembly sınıfının aşağıdaki GetTypes fonksiyonu o Assembly içindeki tüm türleri bir Type dizisi olarak vermektedir.

```
public virtual Type [] GetTypes()
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Reflection;

namespace ReflectionSample
{
   class Program
   {
      int a;

      static void Main(string[] args)
      {
         Assembly a = Assembly.GetExecutingAssembly();

         foreach (Type type in a.GetTypes())
            Console.WriteLine(type.Name);
      }
   }

   struct X
   {
      //...
   }
```

```
    interface Y
    {

    }

    class Z
    {
    }
}
```

Asswmbly içindeki tüm türlere ilişkin Type nesnelerini elde ettikten sonra artık sıra bu türlerin içindeki elemanları elde etmeye gelmiştir.
(DeAssembler programı Reflector programı bedava)

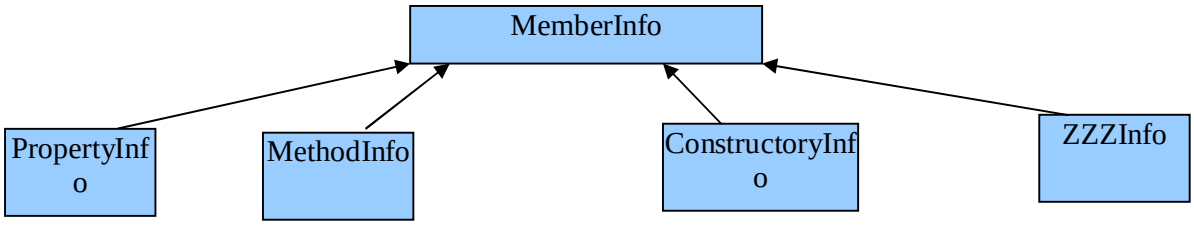
Type sınıfının IsXXX isimli propertyleri ilgili türün XXX türü olup olmadığını belirlemekte kullanılır. Örneğin IsClass ilgili türün bir sınıf olup olmadığını IsValueType bir yapı olup olmadığını anlamakta kullanılır.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Reflection;

namespace ReflectionSample
{
    class Program
    {
        static void Main(string[] args)
        {
            Assembly a = Assembly.GetExecutingAssembly();

            foreach (Type type in a.GetTypes())
            {
                Console.Write(type.Name + ":");
                if (type.IsClass)
                    Console.Write("Class");
                else if (type.IsEnum)
                    Console.Write("Enum");
                else if (type.IsInterface)
                    Console.Write("Interface");
                else if (type.IsValueType)
                    Console.Write("Struct");
                Console.WriteLine();

            }
        }
    }

    struct X
    {
        //...
    }
```

```
    interface Y
    {

    }

    class Z
    {
    }
}
```

Type sınıfının bir grup GetYYY fonksiyonu vardır. Buradaki YYY tür içerisindeki hangi elemanlara ilişkin bilgilerin alınacağını belirtir. Örneğin GetConstructors tüm başlangıç fonksiyonlarını, GetField tüm veri elemanlarını, GetEvent tüm Event elemanlarını GetMethod tüm fonksiyonları almak için kullanılır.

Bir tür içindeki tüm programlama elemanları ZZZInfo isimli sınıflarla temsil edilmiştir. Burada ZZZ türün içindeki ilgili elemanı belirtmektedir. Örneğin MethodInfo gibi PropertyInfo gibi. Bu sınıfların hepsi MemberInfo isimli sınıftan türetilmiştir.



Örneğin bir sınıfın tüm fonksiyonlarına ilişkin bilgileri şöyle elde edebiliriz.

```
Assembly a = Assembly.GetExacutingAssembly();
Type t = a.GetType(ReflectionSample.Program);

foreach (MethodInfo mi int t.GetMethods())
        Console.WriteLine(mi.Name)
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Reflection;

namespace ReflectionSample
{
    class Program
    {
        static void Main(string[] args)
        {
            Assembly a = Assembly.GetExecutingAssembly();
            Type t = a.GetType("ReflectionSample.Program");
```

```
            foreach (MethodInfo mi in t.GetMethods())
                Console.WriteLine(mi.Name);

        }
    }

    struct X
    {
        //...
    }


    interface Y
    {

    }

    class Z
    {
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Reflection;

namespace ReflectionSample
{
    class Program
    {
        public static void Main(string[] args)
        {
            Assembly a = Assembly.GetExecutingAssembly();
            Type t = a.GetType("ReflectionSample.Program");
            MethodInfo mi = t.GetMethod("Foo");

            foreach (ParameterInfo pi in mi.GetParameters())
                Console.WriteLine("{0} {1}", pi.ParameterType.Name, pi.Name);


        }

        public void Foo(int a, long b)
        {

        }
    }

    struct X
```

```
    {
        //...
    }


    interface Y
    {

    }

    class Z
    {
    }
}
```

Artık Assmebly sınıfından başlayarak programa ilişkin tüm detaylar tek tek elde edilebilir. Zaten DisCharp gibi Reflector gibi Salamandergibi DeCompilerler .net in bu yansıtma sınıflarını kullanarak yaılmışlardır.

MDI(Multiple Document Interface) Uygulamaları: Eskiden MDI uygulamaları çok yoğun olarak kullanılıyordu. Gerçektende Microsoft un Excel, Word gibi programları MDI biçiminde desgn edilmişti. Fakat daha sonra MDI uygulamalarının popülaritesi azalmıştı. Microsoft un kendisi de daha sonraları MDI uygulamları yerine bir programı birden fazla kez çalıştırma yöntemini uygulamaya başlamıştır. Microsoft son zamanlarda MDI uygulamaları yerine kendi tipik programlarını her bir dökuman penceresinin anapencere olduğu Multiform olarak tasarlamaktadır. Yani örneğin Microsoft un word programında her file/new yaptığımızda yeni bir word çalıştırılmaz fakat yeni bir word formu oluşturulur. Bu kullanım MDI uygulamalarındaki faydaya benzer bir fayda oluşturmaktadır. Fakat burada sözü edilen MDI uygulamaları böyle değildir. Fakat yine de popülaritesi düşmesine karşın IDE ler gibi tipik programlar yine de MDI uygulamalarını kullanmaktadır.

Tipik bir MDI uygulamasında API terminolojisi ile konuşulursa 3 tür pencere vardır. Programın ana penceresine Frame penceresi(Frame window) denir. Frame penceresinin çalışma alanını kaplayan gri renkte bir pencere alanı daha oluşturulmuştur ve buna da Client penceresi denir. Nihayet kullanıcının etkileştiği asıl işlevsel pencerelere ise document penceresi (document windows) denilmektedir. .Net altında programcının Client penceresi ile doğrudan bir etkileşimi yoktur. Fakat arka planda Client penceresi Frame penceresi ile documnet pencereleri arasında arayüz oluşturma görevindedir.

Tipik bir MDI uygulamasında document pencerelerini idare etmek için bir menü çubuğunda Window popup bulunmaktadır. Tabi bunun bulunması zorunlu değildir.

.Nette hem Frame penceresi hem de document pencererleri yine form sınıfından türetilmiş sınıflarla temsil edilmektedir. MDI uygulaması için iki şeyin yapılması yeterlidir.

1. Programın ana penceresini temsil eden(Yani Frame penceresi olacak olan) form sınıfının bool türden IsMDI property si true yapılmalıdır. Bu property true yapıldığında arka planda otomatik olarak ana pnecere bir frame penceresi yapılır ve gri renkteki Client penceresi yaratılır.
2. Document pencereleri de form sınıfından türetilmiş sınıflarla temsil edilir. Bir document penceresi yaratılacağı zaman tek yapılacak bu form sınıfı türünden bir nesne yaratmak ve

Form sınıfının MDIParent simli property elemanına frame penceresine ilişkin form referansını atamaktır.

Form1.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace MdiSample
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }

      private void newToolStripMenuItem_Click(object sender, EventArgs e)
      {
         Form2 f = new Form2();
         f.MdiParent = this;
         f.Visible = true;
      }
   }
}
```

Form2.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace MdiSample
{
   public partial class Form2 : Form
   {
      public Form2()
      {
         InitializeComponent();
      }
   }
}
```

*F*orm sınıfının MdiChildren isimli property elemanı bize bir form dizisi olarak yaratılmış olan tüm document pencerelerini verir. Biz eğer bütün docment pencereleri üzerinde işlem yapmak istersek

bu formları dolaşabiliriz.

Anahtar Notlar: Visual Studio add new iteim seçeneği ile MdiParent seçilirse Window menüsüne de sahip olan basit bir iskelet MDI programı yazmaktadır. Bunun için boş bir windows form uygulaması yaratılarak Ana form silinip bunun yerine yeni bir MDIParent form projeye eklenebilir. Burada Form editörün oluşturduğu kodlarda document penceresi olarak doğrudan form sınıfı kullanılmıştır. Şüphesiz programcının bunun yerine kendi formunu kullanması anlamlı olur.

MDIParent1.cs Wizardın kendi yazdığı

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace MdiWizard
{
    public partial class MDIParent1 : Form
    {
        private int childFormNumber = 0;

        public MDIParent1()
        {
            InitializeComponent();
        }

        private void ShowNewForm(object sender, EventArgs e)
        {
            Form childForm = new Form();
            childForm.MdiParent = this;
            childForm.Text = "Window " + childFormNumber++;
            childForm.Show();
        }

        private void OpenFile(object sender, EventArgs e)
        {
            OpenFileDialog openFileDialog = new OpenFileDialog();
            openFileDialog.InitialDirectory =
Environment.GetFolderPath(Environment.SpecialFolder.Personal);
            openFileDialog.Filter = "Text Files (*.txt)|*.txt|All Files (*.*)|*.*";
            if (openFileDialog.ShowDialog(this) == DialogResult.OK)
            {
                string FileName = openFileDialog.FileName;
            }
        }

        private void SaveAsToolStripMenuItem_Click(object sender, EventArgs e)
        {
            SaveFileDialog saveFileDialog = new SaveFileDialog();
```

```
            saveFileDialog.InitialDirectory =
Environment.GetFolderPath(Environment.SpecialFolder.Personal);
            saveFileDialog.Filter = "Text Files (*.txt)|*.txt|All Files (*.*)|*.*";
            if (saveFileDialog.ShowDialog(this) == DialogResult.OK)
            {
                string FileName = saveFileDialog.FileName;
            }
        }

        private void ExitToolsStripMenuItem_Click(object sender, EventArgs e)
        {
            this.Close();
        }

        private void CutToolStripMenuItem_Click(object sender, EventArgs e)
        {
        }

        private void CopyToolStripMenuItem_Click(object sender, EventArgs e)
        {
        }

        private void PasteToolStripMenuItem_Click(object sender, EventArgs e)
        {
        }

        private void ToolBarToolStripMenuItem_Click(object sender, EventArgs e)
        {
            toolStrip.Visible = toolBarToolStripMenuItem.Checked;
        }

        private void StatusBarToolStripMenuItem_Click(object sender, EventArgs e)
        {
            statusStrip.Visible = statusBarToolStripMenuItem.Checked;
        }

        private void CascadeToolStripMenuItem_Click(object sender, EventArgs e)
        {
            LayoutMdi(MdiLayout.Cascade);
        }

        private void TileVerticalToolStripMenuItem_Click(object sender, EventArgs e)
        {
            LayoutMdi(MdiLayout.TileVertical);
        }

        private void TileHorizontalToolStripMenuItem_Click(object sender, EventArgs e)
        {
            LayoutMdi(MdiLayout.TileHorizontal);
        }

        private void ArrangeIconsToolStripMenuItem_Click(object sender, EventArgs e)
```

```
        {
            LayoutMdi(MdiLayout.ArrangeIcons);
        }

        private void CloseAllToolStripMenuItem_Click(object sender, EventArgs e)
        {
            foreach (Form childForm in MdiChildren)
            {
                childForm.Close();
            }
        }
    }
}
```

Form sınıfının LayoutMdi isimli fonksiyonu document pencerelerini düzenlemek için kullanılır. Bu fonksiyon MdiLayout isimli bir enum parametresi almaktadır. Bu Enum türünün Cascade, TileHorizantal, TileVertical, ArrangeIcon isimli elemanları vardır.

Form1.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace MdiSample
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void newToolStripMenuItem_Click(object sender, EventArgs e)
        {
            Form2 f = new Form2();
            f.MdiParent = this;
            f.Show();

        }

        private void cascadeToolStripMenuItem_Click(object sender, EventArgs e)
        {
            this.LayoutMdi(MdiLayout.Cascade);
        }

        private void tileHorizontalToolStripMenuItem_Click(object sender, EventArgs e)
        {
```

```
            this.LayoutMdi(MdiLayout.TileHorizontal);
        }

        private void tileVerticalToolStripMenuItem_Click(object sender, EventArgs e)
        {
            this.LayoutMdi(MdiLayout.TileVertical);
        }

        private void arrangeIconsToolStripMenuItem_Click(object sender, EventArgs e)
        {
            this.LayoutMdi(MdiLayout.ArrangeIcons);
        }


    }
}
```

Form2.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace MdiSample
{
    public partial class Form2 : Form
    {
        public Form2()
        {
            InitializeComponent();
        }
    }
}
```

MDI uygulamalarının çoğunda bir dökuman açıldığında aynı zamanda bu dökuman ismi window menüsünün altında da görüntülenir. Bu klasik bir uygulamadır. Manuel olarak yapılabilir fakat .net bunu kolaylaştırıcı basit bir yöntem sunmaktadır. MenuStrip sınıfının MDIWindowListItem isimli property elemanı ToolStripMenuItem sınıfı türündendir. Bu eleman bir popup girilirse yukarıda açıkladığımız dökuman penceresi listelemesi otomatik yapılır. Bu otomatize etme ileminde menu separatörü de otomatik yerleştirilmektedir. Fakat maalesef tüm dökuman pencereleri kapatıldığında bu separatör hala kalmaktadır. Bu işlemi biz popup menünün DropDownOpening ve DropDownClosed Eventlerinde manuel bir biçimde sağlayabiliriz.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
```

```
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace CSD
{
   public partial class MDINotepadForm : Form
   {
      private int m_count;

      public MDINotepadForm()
      {
         InitializeComponent();

         m_count = 1;
      }

      private void newToolStripMenuItem_Click(object sender, EventArgs e)
      {
         DocumentForm df = new DocumentForm();
         df.MdiParent = this;
         df.Text = "Document " + m_count.ToString();
         ++m_count;


         df.Show();

      }

      private void tileHorizontalToolStripMenuItem_Click(object sender, EventArgs e)
      {
         this.LayoutMdi(MdiLayout.TileHorizontal);
      }

      private void tileVerticalToolStripMenuItem_Click(object sender, EventArgs e)
      {
         this.LayoutMdi(MdiLayout.TileVertical);
      }

      private void cascadeToolStripMenuItem_Click(object sender, EventArgs e)
      {
         this.LayoutMdi(MdiLayout.Cascade);
      }

      private void arrangeIconsToolStripMenuItem_Click(object sender, EventArgs e)
      {
         this.LayoutMdi(MdiLayout.ArrangeIcons);
      }

      private void windowToolStripMenuItem_DropDownOpening(object sender, EventArgs e)
      {
         if (MdiChildren.Length == 0)
```

```
windowToolStripMenuItem.DropDownItems[windowToolStripMenuItem.DropDownItems.Count -
1].Visible = false;
        }
    }
}

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace CSD
{
    public partial class MDINotepadForm : Form
    {
        private int m_count;

        public MDINotepadForm()
        {
            InitializeComponent();

            m_count = 1;
        }

        private void newToolStripMenuItem_Click(object sender, EventArgs e)
        {
            DocumentForm df = new DocumentForm();
            df.MdiParent = this;
            df.Text = "Document " + m_count.ToString();
            ++m_count;


            df.Show();

        }

        private void tileHorizontalToolStripMenuItem_Click(object sender, EventArgs e)
        {
            this.LayoutMdi(MdiLayout.TileHorizontal);
        }

        private void tileVerticalToolStripMenuItem_Click(object sender, EventArgs e)
        {
            this.LayoutMdi(MdiLayout.TileVertical);
        }
```

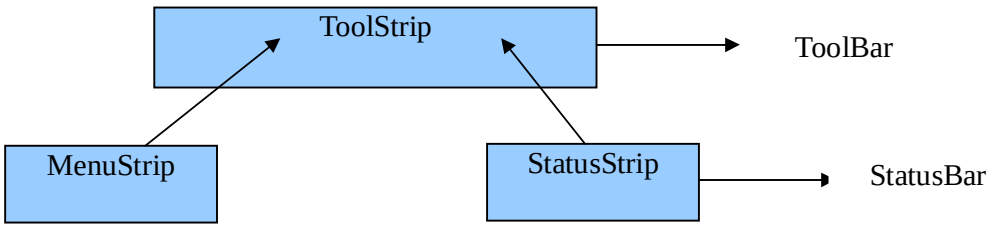
```
    private void cascadeToolStripMenuItem_Click(object sender, EventArgs e)
    {
        this.LayoutMdi(MdiLayout.Cascade);
    }

    private void arrangeIconsToolStripMenuItem_Click(object sender, EventArgs e)
    {
        this.LayoutMdi(MdiLayout.ArrangeIcons);
    }

    private void windowToolStripMenuItem_DropDownOpening(object sender, EventArgs e)
    {
        if (MdiChildren.Length == 0)

windowToolStripMenuItem.DropDownItems[windowToolStripMenuItem.DropDownItems.Count -
1].Visible = false;
    }

    public void saveToolStripMenuItem_Click_1(object sender, EventArgs e)
    {
        DocumentForm df = (DocumentForm)this.ActiveMdiChild;

        if (df.DocPath == null)
        {


            SaveFileDialog sfd = new SaveFileDialog();
            sfd.Filter = "Text Files(*.txt)|*.txt|All Files(*.*)|*.*";

            if (sfd.ShowDialog() == DialogResult.OK)
            {
                df.SaveAs(sfd.FileName);
                MDINotepadForm_MdiChildActivate(null, null);

                df.Text = Path.GetFileName(sfd.FileName);
            }
        }
        else
        {
            df.Save();
            MDINotepadForm_MdiChildActivate(null, null);
        }
    }

    public void MDINotepadForm_MdiChildActivate(object sender, EventArgs e)
    {
        DocumentForm df = (DocumentForm)this.ActiveMdiChild;

        if (df != null)
        {

            saveToolStripButton.Enabled = df.Modified;
```

```
            saveToolStripMenuItem.Enabled = df.Modified;
        }
    }

    private void openToolStripMenuItem_Click(object sender, EventArgs e)
    {
        OpenFileDialog ofd = new OpenFileDialog();
        ofd.Filter = "Text Files(*.txt)|*.txt|All Files(*.*)|*.*";

        if (ofd.ShowDialog() == DialogResult.OK)
        {
            DocumentForm df = new DocumentForm();
            df.MdiParent = this;
            df.Show();

            df.Open(ofd.FileName);
            MDINotepadForm_MdiChildActivate(null, null);
            df.Text = Path.GetFileName(ofd.FileName);
        }
    }
  }
}
```

MDI Uygulamaları ve Menu Birleştirme İşlemleri: MDI uygulamalarının çoğunda Frame penceresinin anamenüsü hiç dökuman açık değilken ya da en azından bir dökuman açıkken farklılık göstermektedir. Örneğin bir MDI editör uygulamasında hiçbir dökuman açık değilken ana menüdeki edit popup ının bulunması anlamsızdır. Bazen dökuman pencereleri aktifken başka popup larada elemanlar eklenmektedir. Bu işlemler manuel olarak yapılabilir fakat .net kütüphanesi bu işlemleri otomatize edecek bir mekanizma sunmaktadır.

Yukarıda anlatılan işlemleri otomatik yapmak için ana menü ye ek olarak bir de dökuman pencerelerine menü yerleştirilir. Menu birleştirme işlemleri için hem ana menünün hemde dökuman penceresenin menüsünün MenuStrip sınıfına ilişkin bool türden AllowMerge property elemanı true yapılmalıdır. Bundan sonra birleştirme işleminin nasıl yapılacağı belirlenir. ToolStripMenuItem sınıfının MergeAction isimli property elemanı MergeAction isimli Enum türündendir. Burada ilgili popup ın ya da menü elemanının anamenüye nasıl ekleneceği belirtilmektedir. Append sona ekleme anlamına gelir. Insert belirli bir indexe ekleme anlamına gelir. Bunun için MergeIndex property sine bakılmaktadır. Replace anamenüdeki ile yer değiştirme anlamına gelir. Remove uyuşan elemanın silineceği anlamına gelir MatchOnly seçeneğinde bir işlem yapılmaz.

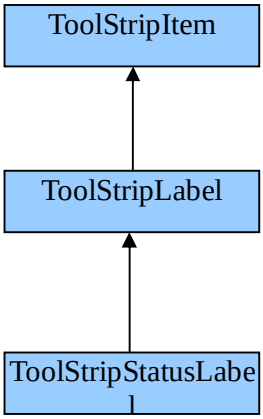
Durum Çubuğu Kontrolü: Pek çok uygulamada ana pencerenin altında programın o andaki durumu hakkında bilgi veren Durum Çubuğu Kontrolü(StatesBar) bulunur. FrameWork 2.0 ile birlikte durum çubuğu kontrolü de bir şerit kontrolü olarak ele alınmaktadır.



Durum Çubuğu kontrolleri Bölümlerden oluşur. Bu bölümlere eleman yerleştirilebilir.

Durum Çubuğu kontrolü tipik olarak şöyle kullanılır:

1. StatusStrip sınıfı türünden bir nesne yaratılır. Control Default aşağı yuvalanmıştır.
2. Tıpkı menulerde yaptığımız gibi durum çubuğu elemanlarını statusStrip sınıfının ItemsCollection elemanına ekleyebiliriz. Görülüdüğü gibi durum çubuğu tamamen ana menuye benzemektedir. Anamenuye PopUp menuler eklenirken Durum çubuğuna tipik olarak StatusLabel,progressBar, DropDownButton ve SplitButton nesneleri eklenebilir. Şüphesiz durum çubuğundaki en tipik eleman StatusLabel elemanıdır. Programcı tipik olarak bu bölüme bir yazı yerleştirir.StatusLabel ToolStripStatusLabel sınıfı ile temsil edilmiştir. Bu sınıf ToolStripStatusLabel sınıfından türetilmiştir. O da ToolStripItem Sınıfından türetilmiştir.



Sınıfın Text propertysi burada görüntülenecek yazıyı belirtir.

Durum Çubuğu elemanları için en önemli özellik bölümün yerleşimine ilişkindir. ToolStripStatusLabel sınıfının Size elemanı ile genişlik ayarlanabilir. bölüm Default durumda AutoSize biçimindedir. Eğer genişlik istenildiği gibi ayarlanacaksa önce AutoSize propertysi False yapılmalıdır. Çünkü Default olarak bu true durumdadır. Bu durumda bölüm otomatik olarak oraya yerleştirilen yazının genişliği kadar genişliğe sahip olabilir. Genellikle DurumÇubuğunun son elemanı geri kalan tüm bölgeyi kaplamaktadır. Bunu sağlamak için ToolStripStatus sınıfnın bool türden Spring propertysi true yapılmalıdır. Bölümlerin default olarak sınır çizgileri yoktur. Fakat toolStripStatusLabel sınıfının BorderStyle propertysi ile sınır çizgileri istenildiği gibi ayarlanabilir. Fakat bu property sınır çizgilerinin çıkmasına yol açmaz. Öncelikle sınıfın borderSides Propertysi ile sınır çizgilerinin çizilmesini eklemek gerekir. BorderStyle BorderSides ile kenar çizgilere izin verilmişse etkili olur. Pek çok programda borderSides ile bölümün dört tarafına sınır çizgileri yerleştirilmekte borderStyle Sunken yapılmaktadır.

ListView Kontrolü: ListView kontrolü sütunlardan ve satırlardan oluşmaktadır. Aslında bu kontrolün ikonik kullanımı da mevcuttur. Fakat burada klasik details kullanımı ele alınmaktadır. Bu klasik kullanım için Listwiev türünden nesne yaratıldıktan sonra öncelikle kontrolün view proğertysi details biçime alınmalıdır. Çünkü bu property default durumda largealcon biçimindedir. Öncelikle kontrole sütunlar eklenmelidir. Herbir sütun ColumnHeader sınıfıyla temsil edilmektedir. Programcı ColumnHeader nesnesini yaratıp Listview sınıfının Columns Propertysiyle belirtilen Collectiona ekler. Columns isimli property elemanı ColumnHeaderCollection isimli bir sınıf türündendir. Bu sınıfın Collectiona ekleme yapan ColumnHeader parametreli ve String Parametreli Add fonksiyonları vardır. String parametreli Add fonksiyonları ColumnHeader nesnesini kendi içerisinde yaratıp ekleme yapmaktadır. Bu durumda Kontrole bir sütun eklemenin iki yolu olur.

Birincisi column Header nesnesini yaratarak eklemektir.

```
public partial class Form1 : Form
{
	private ListView m_lv;

	 public Form1()
	{
		InitializeComponent();

		 private ListView m_lv;
		 //...
		m_lv = new ListView();
		m_lv.Dock = DockStyle.Fill;
		m_lv.View = View.Details;

		 ColumnHeader ch = new ColumnHeader();
		ch.Text = "Adı Soyadı";

		m_lv.Columns.Add(ch);

		this.Controls.Add(m_lv);
	}
}
```

İkinci yol ise şu şekilde olur:

this.Controls.Add("Add Soyad");

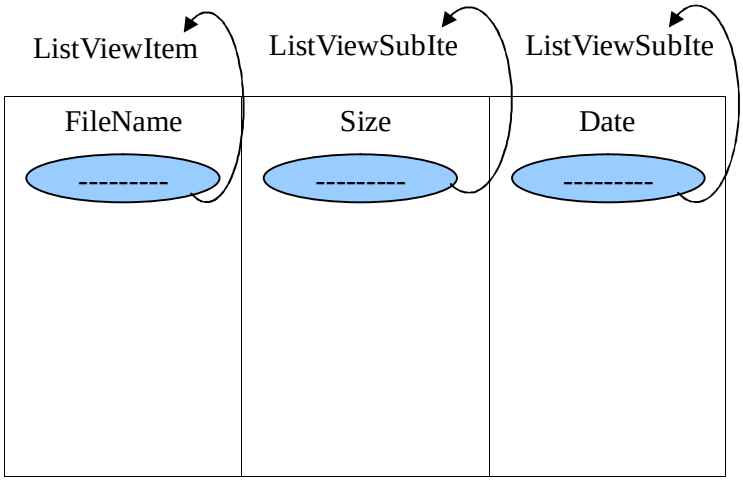
ColumnHeader sınıfının Width property elemanı Sütunun genişliğini belirlemekte kullanılır.

```
private ListView m_lv;

	public Form1()
	{
		InitializeComponent();

		m_lv = new ListView();
		m_lv.Dock = DockStyle.Fill;
		m_lv.View = View.Details;

		m_lv.Columns.Add("File Name");
```

```
            m_lv.Columns[0].Width = 100;

            m_lv.Columns.Add("Size");
            m_lv.Columns[1].Width = 100;

            m_lv.Columns.Add("Date");
            m_lv.Columns[2].Width = 100;

            this.Controls.Add(m_lv);
        }
```

Column sınıfının TextAllign propertysi ile sütun başlık yazısının hizalanması sağlanabilir.

Sütunlar eklendikten sonra artık sıra satırların eklenmesine gelmiştir. Satırlar Listview item nesneleri ile belirtilmiştir. Listview kontrolünde aslında bir anahtar sütun vardır. Diğer sütunlar o anahtar sütundaki bilgilerin detaylarını içermektedir. ListviewItem nesneleri ListviewSubItem nesnelerini tutmaktadır.



Görüldüğü gibi Listview kontrolü ListviewItem nesnelerini tutan Collection gibidir. ListviewItem ise ListviewSubItem nesnelerini tutar. ListViewItem da ListviewSubItem da sütunlarla temsil edilir.

Listview sınıfının Items isimli property elemanı ListviewItem isimli Collection sınıf türündendir. Bu collection sınıf ListviewItem nesnelerini tutmaktadır. O halde özetle özetle Listview sınıfının iki önemli Collection elemanı vardır. Columns sütunları,items satırları tutmakta kullanılır.

ListviewItem sınıfının SubItems isimli property elemanı ise ListViewSubItems nesnelerini tutan bir collection türündendir. O halde bir satıra eklemek için şu uzun yöntem kullanılarbilir:

1. ListviewItem nesnesi yaratılır.
2. ListviewSubItem nesneleri yaratılır.

3. ListviewSubItem nesneleri ListviewItem sınıfının SubItems propertysi yoluyla ListviewItem nesnesine eklenir.
4. ListviewItem nesnesi ise Listview sınıfının Items propertysi yoluyla Listview nesnesine eklenir.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace ListViewSample
{
   public partial class Form1 : Form
   {
      private ListView m_lv;

      public Form1()
      {
         InitializeComponent();

         m_lv = new ListView();
         m_lv.Dock = DockStyle.Fill;
         m_lv.View = View.Details;

         m_lv.Columns.Add("File Name");
         m_lv.Columns[0].Width = 100;

         m_lv.Columns.Add("Size");
         m_lv.Columns[1].Width = 100;


         m_lv.Columns.Add("Date");
         m_lv.Columns[2].Width = 100;


         ListViewItem lvi = new ListViewItem("A.DAT");
         ListViewItem.ListViewSubItem lvsi1 = new ListViewItem.ListViewSubItem();
         lvsi1.Text = "1234323";
         ListViewItem.ListViewSubItem lvsi2 = new ListViewItem.ListViewSubItem();
         lvsi2.Text = "10/12/2008";

         lvi.SubItems.AddRange(new ListViewItem.ListViewSubItem[] { lvsi1, lvsi2 });
         m_lv.Items.Add(lvi);

         this.Controls.Add(m_lv);
      }
   }
}
```

Wizard ile yazılan kod:

```
namespace ListViewWizard
{
   partial class Form1
   {
      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.IContainer components = null;

      /// <summary>
      /// Clean up any resources being used.
      /// </summary>
      /// <param name="disposing">true if managed resources should be disposed; otherwise,
false.</param>
      protected override void Dispose(bool disposing)
      {
         if (disposing && (components != null))
         {
            components.Dispose();
         }
         base.Dispose(disposing);
      }

      #region Windows Form Designer generated code

      /// <summary>
      /// Required method for Designer support - do not modify
      /// the contents of this method with the code editor.
      /// </summary>
      private void InitializeComponent()
      {
         System.Windows.Forms.ListViewItem listViewItem1 = new
System.Windows.Forms.ListViewItem(new string[] {
         "A.DAT",
         "1234354",
         "10/12/2008"}, -1);
         this.listView1 = new System.Windows.Forms.ListView();
         this.columnHeader1 = new System.Windows.Forms.ColumnHeader();
         this.columnHeader2 = new System.Windows.Forms.ColumnHeader();
         this.columnHeader3 = new System.Windows.Forms.ColumnHeader();
         this.SuspendLayout();
         //
         // listView1
         //
         this.listView1.Columns.AddRange(new System.Windows.Forms.ColumnHeader[] {
         this.columnHeader1,
         this.columnHeader2,
         this.columnHeader3});
         this.listView1.Dock = System.Windows.Forms.DockStyle.Fill;
         this.listView1.Items.AddRange(new System.Windows.Forms.ListViewItem[] {
```

```
            listViewItem1});
            this.listView1.Location = new System.Drawing.Point(0, 0);
            this.listView1.Name = "listView1";
            this.listView1.Size = new System.Drawing.Size(625, 264);
            this.listView1.TabIndex = 0;
            this.listView1.UseCompatibleStateImageBehavior = false;
            this.listView1.View = System.Windows.Forms.View.Details;
            //
            // columnHeader1
            //
            this.columnHeader1.Text = "File Name";
            this.columnHeader1.Width = 100;
            //
            // columnHeader2
            //
            this.columnHeader2.Text = "Size";
            this.columnHeader2.Width = 100;
            //
            // columnHeader3
            //
            this.columnHeader3.Text = "Date";
            this.columnHeader3.Width = 100;
            //
            // Form1
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(625, 264);
            this.Controls.Add(this.listView1);
            this.Name = "Form1";
            this.Text = "Form1";
            this.ResumeLayout(false);

        }

        #endregion

        private System.Windows.Forms.ListView listView1;
        private System.Windows.Forms.ColumnHeader columnHeader1;
        private System.Windows.Forms.ColumnHeader columnHeader2;
        private System.Windows.Forms.ColumnHeader columnHeader3;
    }
}
```

ListView kontrolüne satır eklemenin kolay bir yolu da vardır. Anımsanacağı gibi ListView sınıfının satırları tutan Items property si ListView.ListViewItemCollection sınıfı türündendir. Bizdaha önce bu sınıfın ListViewItem parametreli ya da string parametreli Add fonksiyonlarını kullanmıştık. İşte ListViewItem sınıfının köşeli panatez parametreli bir başlangıç fonksiyonu da vardır. Programcı bu

başlangıç fonksiyonuna 0. sütundan başlayarak tüm sütunlara ilişkin satır yazılarını verir. Fonksiyon bu yazılardan hareketle ListViewSubItem nesnelerini oluşturarak eklemeleri yapar. Örneğin Bir dizin içindeki tüm dosya bilgilerini bir ListView kontrolüne şöyle aktarabiliriz.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace ListViewSample
{
   public partial class Form1 : Form
   {
      private ListView m_lv;

      public Form1()
      {
         InitializeComponent();

         m_lv = new ListView();
         m_lv.Dock = DockStyle.Fill;
         m_lv.View = View.Details;


         m_lv.Columns.Add("File Name", 200);
         m_lv.Columns.Add("Size", 200);
         m_lv.Columns.Add("Date", 200);

         foreach (string file in Directory.GetFiles(@"c:\windows"))
         {
            FileInfo fi = new FileInfo(file);
            m_lv.Items.Add(new ListViewItem(
                     new string [] {fi.Name, fi.Length.ToString(), fi.CreationTime.ToString()}));
         }

         this.Controls.Add(m_lv);
      }
   }
}
```

ListView sınıfının bool türden FullRowSelect property elemanı satırın bütünüyle seçilmesini sağlar. Propertinin default değeri false dur.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
```

```
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace ListViewSample
{
   public partial class Form1 : Form
   {
      private ListView m_lv;

      public Form1()
      {
         InitializeComponent();

         m_lv = new ListView();
         m_lv.Dock = DockStyle.Fill;
         m_lv.View = View.Details;
         m_lv.FullRowSelect = true;


         m_lv.Columns.Add("File Name", 200);
         m_lv.Columns.Add("Size", 200);
         m_lv.Columns.Add("Date", 200);

         foreach (string file in Directory.GetFiles(@"c:\windows"))
         {
            FileInfo fi = new FileInfo(file);
            m_lv.Items.Add(new ListViewItem(
              new string [] {fi.Name, fi.Length.ToString(),
                        fi.CreationTime.ToString()}));
         }
         this.Controls.Add(m_lv);
      }
   }
}
```

ListWiew sınıfının bool türden MultiSelect property si çoklu seçime olanak sağlamaktadır. Bu property nin de default durumu true dur.

Listview sınıfının SelectedItems isimli property elemanı, seçilmiş elemanları ListViewItem nesneleri tutan collection biçiminde verir. Yine ListView sınıfının SelectedIndices isimli property elemanı bize seçilmiş olan elemanların indeks numaralarını int türden bir collection da toplayarak verir.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
```

```
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace ListViewSample
{
   public partial class Form1 : Form
   {
      private ListView m_lv;

      public Form1()
      {
         InitializeComponent();

         m_lv = new ListView();
         m_lv.Bounds = new Rectangle(0, 0, 500, 400);
         m_lv.View = View.Details;
         m_lv.FullRowSelect = true;


         m_lv.Columns.Add("File Name", 200);
         m_lv.Columns.Add("Size", 200);
         m_lv.Columns.Add("Date", 200);

         foreach (string file in Directory.GetFiles(@"c:\windows"))
         {
            FileInfo fi = new FileInfo(file);
            m_lv.Items.Add(new ListViewItem(
              new string [] {fi.Name, fi.Length.ToString(),
                        fi.CreationTime.ToString()}));
         }
         this.Controls.Add(m_lv);
      }

      private void button1_Click(object sender, EventArgs e)
      {
         string s = "";
         foreach (ListViewItem lvi in m_lv.SelectedItems)
         {
            s += lvi.Text + "\n";
         }

         MessageBox.Show(s);
      }
   }
}
```

ListView sınıfının bool türden CheckBoxes isimli property elemanı her satırda bir de seçenek kutusu çıkartılmasını sağlar. Default biçim false biçimindedir. Bu biçimde elemanların Checked yapılması ile seçilmesi aynı anlamda değildir. İstersek biz ayrıca CheckedItems ve CheckedIndecis propertyleri ile Checked yapılmış elemanların listesini alabiliriz.

ListView sınıfının bool türden GridLines isimli property elemanı kontrolde ızgara çizgilerinin çıkmasını sağlar. Bu property nin de default durumu false biçimindedir.

Programcı isterse bir satırın tamamının zemin rengini ve yazı rengini değiştirebilir. Bunun için ListViewItem sınıfının BackColor ve ForeColor property elemanları kullanılmaktadır.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace ListViewSample
{
   public partial class Form1 : Form
   {
      private ListView m_lv;

      public Form1()
      {
         InitializeComponent();

         m_lv = new ListView();
         m_lv.Bounds = new Rectangle(0, 0, 500, 400);
         m_lv.View = View.Details;
         m_lv.FullRowSelect = true;
         m_lv.CheckBoxes = true;
         m_lv.GridLines = true;


         m_lv.Columns.Add("File Name", 200);
         m_lv.Columns.Add("Size", 200);
         m_lv.Columns.Add("Date", 200);

         foreach (string file in Directory.GetFiles(@"c:\windows"))
         {
            FileInfo fi = new FileInfo(file);


            ListViewItem lvi = new ListViewItem(new string [] {fi.Name, fi.Length.ToString(),
                     fi.CreationTime.ToString()});

            m_lv.Items.Add(lvi);
            if (Path.GetExtension(fi.Name).ToLower() == ".exe")
               lvi.ForeColor = Color.Red;
```

```
        }
        this.Controls.Add(m_lv);
    }

    private void button1_Click(object sender, EventArgs e)
    {
        string s = "";
        foreach (int index in m_lv.SelectedIndices)
        {
            s += index.ToString()+ "\n";
        }

        MessageBox.Show(s);
    }
  }
}
```

ListView sınıfının çeşitli faydalı Event Elemanları vardır. ListView sınıfının ColumnClick isimli event elemanı bir sütun başlığına tıklandığında tetiklenir. Oluşan mesajda biz tıklanmış olunan sütunun numarasınıda elde edebiliriz. Control ün ItemActivate isimli Event elemanı bir eleman seçildiğinde tetiklenir. Fakat active etme işleminin tek tıklamaylamı yoksa çift tıklamaylamı yapılacağı sınıfın Activation isimli property elemanıyla belirlenmektedir. Ayrıca tıpkı ListBox kontrolünde olduğu gibi bu kontrolünde Changed isimli event elemanı vardır. Bu event seçili eleman değiştiğinde tetiklenmektedir.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace ListViewSample
{
  public partial class Form1 : Form
  {
    private ListView m_lv;

    public Form1()
    {
      InitializeComponent();

      m_lv = new ListView();
      m_lv.Bounds = new Rectangle(0, 0, 500, 400);
      m_lv.View = View.Details;
      m_lv.FullRowSelect = true;
      m_lv.GridLines = true;
      m_lv.SelectedIndexChanged += new EventHandler(m_lv_SelectedIndexChanged);
```

```
            m_lv.Columns.Add("File Name", 200);
            m_lv.Columns.Add("Size", 200);
            m_lv.Columns.Add("Date", 200);

            foreach (string file in Directory.GetFiles(@"c:\windows"))
            {
               FileInfo fi = new FileInfo(file);


               ListViewItem lvi = new ListViewItem(new string [] {fi.Name, fi.Length.ToString(),
                           fi.CreationTime.ToString()});

               m_lv.Items.Add(lvi);
               if (Path.GetExtension(fi.Name).ToLower() == ".exe")
                  lvi.ForeColor = Color.Red;


            }
            this.Controls.Add(m_lv);
         }

         void m_lv_SelectedIndexChanged(object sender, EventArgs e)
         {
            string s = "";
            foreach (int index in m_lv.SelectedIndices)
            {
               s += index.ToString() + "\n";
            }

            MessageBox.Show(s);
         }

          private void button1_Click(object sender, EventArgs e)
         {
            string s = "";
            foreach (int index in m_lv.SelectedIndices)
            {
               s += index.ToString()+ "\n";
            }

            MessageBox.Show(s);
         }
      }
   }
```

Sütun başlıklarına tıklandığında ilgili sütuna göre sıraya dizme çok uygulanan bir yöntemdir. Fakat bu işlem otomatik yapılmamaktadır. Bu işlemi gerçekleştirmek için şu işlemler uygulanmalıdır.

1. Sıraya dizme işlemi ListView sınıfının sort fonksiyonuyla yapılmaktadır. Fakat programcının daha önce bazı hazırlıklar yapması gerekebilir. Programcı bu sort fonksiyonunu şüphesiz ColumnClick eventi oluştuğunda çağıracaktır.
2. Programcının Icomparer arayüzünü destekleyen bir sınıf oluşturması gerekir. Sonra bu sunuf türünden bir nesne yaratarak ListView sınıfının ListViewItemSorter property sine atama

yapması gerekir. ICompare arayüzünün Compare isimli tek bir fonksiyonu vardır. *int Compare(Object x, Object y)* Sort fonksiyonu aslında sıraya dizme sırasında karşılaştırma yaparken iki ListView nesnesini bu fonksiyona parametre yapmaktadır. Yani programcı Icompare arayüzü destekleyen bu sınıfta bu fonksiyonu yazarken Fonksiyonun iki parametresinde ListViewItem nesnelerinin olduğunu bilmelidir. Bu fonksiyon öyle bir biçimde yazılmalıdırki Birinci yazı ikinci yazıdan büyükse pozitif herhangi bir değere küçükse negatif herhangi bir değere ve eşitse sıfır değerine geri dönsün.

3. Programcının ilgili sütuna göre sıraya dizme işlemi yapabilmesi için Compare fonksiyonunu o sütuna göre yazması gerekir. O halde tipik olarak Icomparer arayüzünü destekleyen sınıfa sütun numarası veri elemanı olarak geçirilebilir. Compare fonksiyonu da bu veri elemanına bakarak çalışabilir.
4. ListView sınıfının Sorting isimli property elemanı büyükten küçüğemi, yoksa küçükten büyüğemi sıraya dizileceğini belirtir.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.Collections;

namespace ListViewSample
{
   public partial class Form1 : Form
   {
      private ListView m_lv;

      public Form1()
      {
         InitializeComponent();

         m_lv = new ListView();
         m_lv.Bounds = new Rectangle(0, 0, 600, 400);
         m_lv.View = View.Details;
         m_lv.FullRowSelect = true;
         m_lv.GridLines = true;
         m_lv.ColumnClick += new ColumnClickEventHandler(m_lv_ColumnClick);


         m_lv.Columns.Add("File Name", 200);
         m_lv.Columns.Add("Size", 200);
         m_lv.Columns.Add("Date", 200);

         foreach (string file in Directory.GetFiles(@"c:\windows"))
         {
            FileInfo fi = new FileInfo(file);


            ListViewItem lvi = new ListViewItem(new string[] {fi.Name,  fi.Length.ToString(),
```

```
                    fi.CreationTime.ToString()});

            m_lv.Items.Add(lvi);
            if (Path.GetExtension(fi.Name).ToLower() == ".exe")
                lvi.ForeColor = Color.Red;


        }
        this.Controls.Add(m_lv);
    }

    void m_lv_ColumnClick(object sender, ColumnClickEventArgs e)
    {
        m_lv.ListViewItemSorter = new MyComparer(e.Column);
        m_lv.Sort();
    }

    private void button1_Click(object sender, EventArgs e)
    {
        string s = "";
        foreach (int index in m_lv.SelectedIndices)
        {
            s += index.ToString()+ "\n";
        }

        MessageBox.Show(s);
    }
}

class MyComparer : IComparer
{
    private int m_cno;
    private static int ms_oldcno;
    private static bool ms_sortOrder = true;

    public MyComparer(int cno)
    {
        if (cno != ms_oldcno)
        {
            ms_sortOrder = false;

        }
        else
            ms_sortOrder = !ms_sortOrder;

        m_cno = cno;
        ms_oldcno = cno;
    }
     public int Compare(object x, object y)
    {
        ListViewItem lvi1;
        ListViewItem lvi2;
```

```
            if (ms_sortOrder)
            {
               lvi1 = (ListViewItem)y;
               lvi2 = (ListViewItem)x;
            }

            else
            {
               lvi1 = (ListViewItem)x;
               lvi2 = (ListViewItem)y;
            }

            switch (m_cno)
            {
               case 0:
                  return string.Compare(lvi1.SubItems[m_cno].Text, lvi2.SubItems[m_cno].Text);
               case 1:
                  return int.Parse(lvi1.SubItems[m_cno].Text) - int.Parse(lvi2.SubItems[m_cno].Text);
               case 2:
                  DateTime dt1 = DateTime.Parse(lvi1.SubItems[m_cno].Text);
                  DateTime dt2 = DateTime.Parse(lvi2.SubItems[m_cno].Text);

                  return DateTime.Compare(dt1, dt2);
            }

            return 0;
         }
      }
   }
```

Anahtar Notlar: Tıpkı int , long gibi temel yapıların olduğu gibi DateTime yapısının da parse fonksiyonu vardır. Bu fonksiyon bizden tarih ve zaman bilgisini yazı olarak alır ve onu DateTime yapısına dönüştürür. Ayrıca iki DateTime yapısını karşılaştırarak pozitiflik ve negatiflik durumuna göre int değere geri dönen DateTime yapısının static Compare fonksiyonu da vardır. DateTime yapısının Parse fonksiyonunun yazıyı hangi formatta alacağına ilişkin detaylı bilgi MSDN dökumanlarından izlenebilir.

.Nette Veri Tabanı İlemleri: Bilgileri belirli bir düzene göre tutan ve üzerinde bazı işlemlerin yapılmasına olanak sağlayan bir ya da birden fazla dosyaya veri tabanı denilmektedir. Veri tabanı işlemleri temel olarak üç yöntemle gerçekleştirilebilir.

1. İşlemler doğrudan aşağı seviyeli dosya sınıfları ve fonksiyonları kullanılarak gerçekleştirilebilir. Örneğin biz birkaç yüz öğrencinin bilgilerini tutacaksak ve karmaşıkta bir sorgulama yapmayacaksak bu manuel yöntemi uygulayabiliriz.
2. Programcı özel yazılmış veri tabanı kütüphanelerini kullanabilir. Özellikle 90 lı yıllara kadarki dönem içinde bu tür çözümler kullanılıyordu. Yani sözkonusu olan kütüphanede kayıt ekleyen, kayıt silen, arama yapmayı sağlayan fonksiyonlar vardı. Örneğin: Ctrieve, DbVista, Btrieve ve DB kütüphaneleri yoğun olarak kullanılmıştır. Veri tanabı kütüphaneleri Veri Tabanı Yönetim Sistemlerine göre(VTYS) çok daha hızlı çözüm sunmaktadır. Fakat bu kütüphaneler esnek değildir ve data formatına sıkıca bağlanır. Bu tür kütüphaneler VTYS nin sunduğu pek çok kolaylığı sunmamaktadır.
3. Programcı tüm veri tabanı işlemlerini ismine VTYS(Database Managment System) denilen bir yazılıma bırakarak herşeyi o yazılıma yaptırabilir. Günümüzde ağırlıklı olarak kullanılan

yöntem budur. .Nette de temel olarak bu yöntemin kullanılmasına yönelik çözüm sunulmuştur.

Veri Tabanı Yönetim Sistemleri(VTYS): Veri tabanı yönetim sistemleri veri tabanı işlemlerini yüksek seviyeli bir biçimde gerçekleştirmek için kullanılan özel yazılımlardır. Bu tür yazılımlar tipik olarak işlemlerin dosya kısmını kendi üzerlerine alıp kullanıcının aşağı seviye dosya formatından bağımsız bir biçimde veri tabanına erişmesini mümkün hale getirmektedir. Kendi içlerinde pek çok aracı içermektedir. Pek çok VTYS inde yönetim işlemi komut satırında ya da görsel olarak gerçekleştirilebilir. Verilerin gösterimine ilişkin çeşitli araçlar bulunabilir ve adeta programcı herhangi bir programlama diline gereksinim duymadan veri tabanı işlemlerini gerçekleştirebilir. Bu tür yazılımlar veri bütünlüğünü korumak için özel mekanizmalara sahiptir ve güvenilir bir ortam sunmaktadır. VTYS lerinin çoğu yine ya hep ya hiç biçiminde transaction(Bir gurup işlemi ya hep ya hiç biçiminde yapmak) işlemlerini sağlamaktadır.

VTYS leri tipik olarak Client ve Server çalışan sistemlerdir. Veri tanabı yönetim sisteminden hizmet alan programlar client programlardır. VTYS leri kendisi server sistemlerdir. Client Server ile bağlantı sağlar ve ondan bazı işlemleri yapmasını ister. Server bu işlemleri Client için yapar sonuçları client a gönderir. Şüphesiz veri tanabı yönetim sistemleri aynı anda yakın ya da uzak pek çok client a hizmet verebilmektedir.

VTYS lerinden en çok kullanılanları Oracle firmasından oracle Microsoft firmasından SQL Server IBM in DB2 su Sybase, Progres, Informix

Bedava olanlardan en çok tercih edileni MySQL dir.

SQL kullanıcının VTYS iş yaptırmak için kullandığı standart bir sorgulama dilidir. Örneğin kullanıcı veri tabanına bir kayıt eklemek istediğinde bunu VTYS mine bir SQL komutu olarak iletir. VTYS bu komutu yorumlayarak kullanıcının isteğini yerine getirir. SQL asıl işi yapan dil değildir. Gerçek veri tabanı işlemleri C de ve C++ yapılan kodlar la yapılmaktadır. SQL yalnızca istek bildirmekte kullanılan bir dildir. VTYS lerinin gerçek veri tabanı işlemlerini yapan bölümlerine motor kısım denir. Bugün VTYS lerinin hemen hepsi standart SQL komutlarını desteklemektedir. SQL bu nedenle standart bir arayüz olarak kullanılmaktadır.

VTYSlerini Programlama Yoluyla Kullanılması: Daha önceden de belirtildiği gibi VTYS leri kendi içinde pek çok aracı içeren büyük yazılımlardır. Bunların Etkin kullanılması ve yönetimi için özel eğitimlere gereksinim duyulmaktadır. Bir VTYS konusunda uzmanlaşmış kişiye Veritabanı Yöneticisi(Database Administrator) denilmektedir.

Fakat veri tabanı yönetim sistemlerinin tipik diğer bir kullanımıda başka bir programlama dilinden ya da ortamdan onlara iş yaptırmak amaçlıdır. Örneğin bir C Ya da C# programcısı VTYS ine bağlantı kurup kayıt ekleme, kayıt silme, sorgulama gibi işlemleri ona yaptırabilir. Kursumuzda bu VTYS leri .netten bu amaçla kullanılacaktır. Böylesi bir kullanım için temel bir SQL bilgisi yeterlidir. VTYS yöneticisi seviyesinde bir bilgiye gereksinim yoktur.

.Netin Veri Tabanı Sınıfları ve ADO.Net(Active Data Object): Microsoft .net ortamı içinde VTYS iyle bağlantı kurmakta kullanılan ona SQL de komut göndermekte kullanılan, sonuçları almakta kullanılan bir grup sınıf bulundurmaktadır. VTYS iyle kurulan ilişki ADO teknolojine dayandığı için bu sınıflara ADO.Net sınıfları denilmektedir. Veritabanı işlemlerine yönelik sınıfların ismi System.Data isim alanında ya da bu isim alanının içindeki isim alanları içindedir.

Microsoft farklı VTYS lerini benzer bir mantık içinde kullanmayı hedefleyen bir tasarımı yeğlemiştir. Yani VTYS değişse de bizim konuya yaklaşımımız kullandığımız sınıflar ve

fonksiyonlar birbirine çok benzemektedir. Böylece daha kolay bir öğrenme sağlamaktadır.

İlişkisel Veritabanı Kavramı: Veritabanlarının yüksek seviyeli bir biçimde eleanlınmasına yönelik çeşitli veri tabanı modelleri önewrilmiştir. Bunlardan en çok tercih edilen ve kullanılan İlişkisel(Relational) veritabanı modelidir. İlişkisel veritabanında bir veritabanı tablolar biçimindedir. Bir tablo sütunlardan ve satırlardan oluşur. Sütunlara alan(field) satırlara kayıt(record) denilmektedir. Tabloların ve sütunların birer ismi vardır. Sütunlarda bulunan bilgilerin türleri vardır. Bir vertaban pek çok tablodan oluşabilir. Şüphesiz veritabanındaki tablo ilişkisi kullanıcı algısına yöneliktir. Bu durum diskteki fiziksel model hakkında bir fikir veremez. VTYS bu tabloların oluşturulması ve yönetiminden sorumludur.

Veritabanı işlemlerinin anahtar noktalarından biri sorgulama(query) işlemidir. Sorgulama belirli koşulları sağlayan kayıtlar üzerinde işlem yapmaktır. Örneğin "Eskişehirde doğup yaşı 21 den büyük olanların listesinin alınması"

Temel SQL Komutları: SQL komutları çeşitli alt kategorilerde incelenebilir. ADO.Net programcısı için temelSQL bilgisi yetrerlidir. SQL komutlarının genel biçimi karışık olmasına karşın burada temel kullanımlar üzerinde durulacaktır. Komutlardan bazıları kayıt geri döndürmez, bazıları kayıt geri döndürür. Örneğin Select komutu bize bir takım kayıtlar geri döndürmektedir. Fakat örneğin Delete komutu yalnızca iş yapmaktadır.

SELECT Komutu: Select komutu belirli koşulu sağlayan kayıtların elde edilmesi için kullanılmaktadır. Select komutuyla belirli bir tablodan ya da tablolardan belirli sütunlara ilişkin belirli koşulları sağlayan kayıtlar elde edilebilir. Bir SQL komutu çeşitli cümleciklerin bir araya gelmesinden oluşturulur. Bu cümleciklerin en çok kullanılanı WHERE cümleciğidir.

Anahtar Notlar: SQL standarlara göre büyük küçük harf duyarlılığı olan bir dil değildir. Fakat SQL sözcüklerinin büyük harflerle yazılması geleneksel ve iyi bir tekniktir.

SELECT anahtar sözcüğünü tablodaki hangi sürunların elde edileceğine ilişkin bir liste izler. Yani biz koşulu sağlayan kayıtların yalnızca belirli sütunlarını elde edebiliriz. Burada yıldız "*" tüm sütunlar anlamına gelmektedir. Sonra from anahtar sözcüğü ve hangi tablolar üzerinde işlem yapılacağına ilişkin tablo listesi gelir. Nihayet koşul için bir WHERE cümleciği getirilebilir. O halde SELECT komutunun tipik biçimi şöyledir:
SELECT <sütun listesi> FROM <Tablo Listesi> WHERE<Koşul>

Buradaki kullanım genel biçim değil tipik biçimdir. Örneğin:
SELECT PersonName, PersonNo FROM Person
Burada koşul belirtilmediği için Person tablosundaki tüm kayıtları isim ve numaraları elde edilmiştir. Örneğin:
*SELECT * FROM Person WHERE PersonName='Kaan Aslan'* Burada Person tablosundaki ismi Kaan Aslan olan tüm kayıtlar yani satırlar elde edilmektedir. Koşul kısmı mantıksal operatörlerle detaylandırılabilir. Örneğin:
*SELECT * FROM Person WHERE Age>20 AND BirthPlace = 'Eskişehir'*
LIKE operatörü Joker karekterlerin kullanılmasına olanak sağlamaktadır.
Örneğin: *SELECT * FROM Person WHERE Age>20 AND BirthPlace = 'Eskişehir' AND NAME LIKE'A%'* Burada ismi A ile başlayan yaşı 20 den büyük olanların listesi elde edilmektedir.
SELECT komutuna başka cümleciklerde eklenebilir. Örneğin ORDER BY<Sütun Listesi> cümleciği elde edilen kayıtların hangi sütunlara göre sıralı olacağını belirlemektedir. İlk belirtilen sütun aynı ise diğer sütunlar etkili olur. Örneğin:
*SELECT * FROM Person WHERE PersonName LIKE 'A%' ORDER BY PersonNo* Burada ismi A ile başlayanların listesi valınmaktadır fakat bu liste numara sırasına göre dizilmiştir.

INSERT INTO Komutu: Bu komut veritabanına kayıt eklemek için kullanılır.
Tipik biçimi şöyledir.
INSERT INTO <Tablo İsmi>(<Sütun Listesi>) VALUES (<Değer Listesi>)
Örneğin: *INSERT INTO Person (PersonName, PersonNo) VALUES ('Kaan Aslan, 120)*
Tablo isminden sonra parantez açılmazsa tüm sütunlar anlaşılır. Sütun isimlerinin tablodaki sıraya göre olması gerekmez. Fakat değerlerin sırasıyla sütunlara karşılık gelmesi gerekmektedir.

DELETE Komutu: Delete komutu tipik olarak belirli koşulları sağlayan kayıtların silinmesi amacıyla kullanılmaktadır. Komutların tipik biçimi şöyledir.
DELETE FROM <Table Listesi> WHERE <Koşul>
Örneğin: *DELETE FROM Person WHERE Name='Kaan Aslan' AND BirthPlace = 'Eskişehir'*
UPDATE Komutu: UPDATE komutu koşulu sağlayan kayıtlardaki bazı sütunların değiştirilmesi için kullanılır. Örneğin tm Kaan Aslan isimlerini Kaan Arslan yapmak için Update komutunu kullanabiliriz. UPDATE komutunun tipik biçimi şöyledir.
UPDATE <Toblo Listesi> SET<Sütun = Değer Listesi> WHERE <Koşul>
Örneğin: *UPDATE Person SET PersonName= 'Kaan Aslan' WHERE PersonName = 'Kaan Arslan'*

Şüphesiz UPDATE ile birden fazla sütun üzerinde değişiklik yapılabilir.
Örneğin: *UPDATE Person SET PersonName= 'Kaan Aslan' , BirthPlace='Eskişehir' WHERE PersonName = 'Kaan Arslan'*

CREATE DATABASE Komutu: Sıfırdan yeni bir Veritabanı yaratmak için kullanılır. Tipik biçimi şöyledir. *CREATE DATABASE <İsim> Örneğin: CREATE DATABASE Personel_Takip*

CREATE TABLE Komutu: Bir veritabanı tablosu oluşturmak için kullanılır. Bu komut veritabanına yeni bir tablo yaratmak için kullanılır. Tablo yaratılırken tabloya bir isim verilir. Sonra tablodaki tüm sütun isimleri ve bunların türleri belirtilir. SQL de standart bazı türler vardır. Fakat Veritabanı Yönetim Sistemine bağlı olarak başka ek türlerde söz konusu olabilir. Tipik türler şunlardır.

INT	: Tam sayı Türü
DEC	: Gerçel Sayı turu
CHAR	: n tane karakter türü
VARCHAR	: değişken karakter
DATE	: Tarih
DATETIME	: Tarih Zaman

CREATE TABLE komutunun tipik kullanımı şöyledir.
CREATE TABLE <Tablo İsmi> (<İsim ve tür listesi>)
Örneğin: *CREATE TABLE Person(PersonName VARCHAR(32), PersonNo INT)*

USE Komutu: USE komutu zaten yaratılmış olan bir veritabanını kullanmak için kullanılır.
Genel biçimi şöyledir: *USE <Veritabanı İsmi>* Örneğin: *USE Personel_Takip*

Anahtar Notlar: Normal olarak SQL komutları “;” ile sonlandırılır. .Nette de komut satırında da aynı satır üzerine birden fazla komut verilebilir. Fakat ileride görüleceği gibi .Nette tek bir komutu “;” ile sonlandırmak zorunda değiliz.
ADO.net Bağlantılı ve Bağlantısız Sınıflar: Ado.net bağlantısız sınıfları veritabanı tablolarını bellekte temsil etmek için kullanılan sınıflardır. Bu sınıfları kullanabilmek için VTYS ile bağlantı kurmuş olmamız gerekmez. Halbuki bağlantılı sınıflar VTYS ile iş yaptırmak için kullanılan sınıflardır.
Veritabanı Tedarikçi Sınıfları: Ado.net sınıf sisteminde Veritabanı ne biçimde olursa olsun

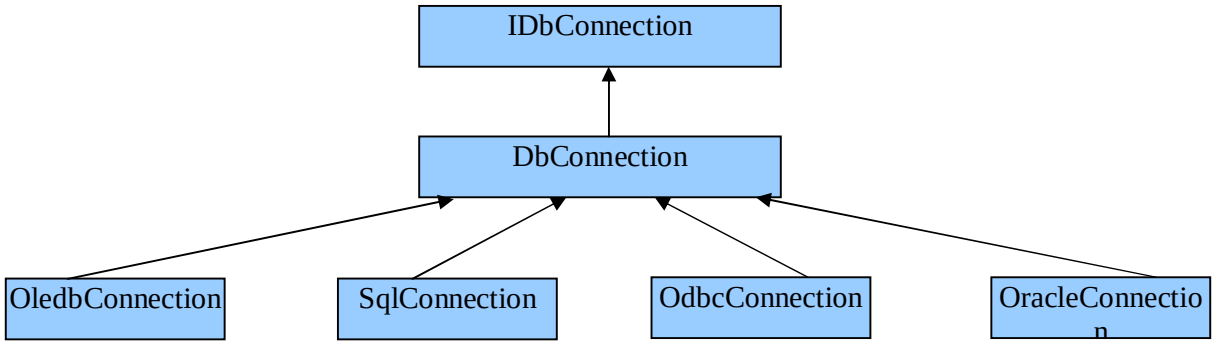
işlemler benzer biçimde yapılmaktadır. Veritabanı ynetimsistemleri için farklı tedarikçi sınıflar vardır. Bu tedarikçi sınıflar ortak aratüzlweri destekledikleri için benzer kullanımlara sahiptir. Microsoft sqlserver içim sqlXXX biçiminde isimlendirilmiş tedarikçi sınıfları, Access jet motoru için, Excel jetmotoru için Mysql için genel amaçlı OLEDBXXX tedarikçi sınıflarını, ODBC bağlantısı için ODBCXXX tedarikçi sınıflarını Oracle işlemleri için OracleXXX tedarikçi sınıflarını bulundurmaktadır.

Örneğin biz Access veritabanı ile OLEDB tedarikçi sınıflarını kullanarak işlemlerini yapabiliriz. Bağlantı için OLEDBCollection sınıflarını kullanırız. Fakat VTYS SQL Server olsaydı SQLConnection sınıfını kullanacaktık. Bu durumda örneğin biz örneklerimizi OledbXXX isi, ODBCxxx biçiminde diğer tedarikçi versiyonları da vardır ve kullanımlar birbirine benzer.

Oledb ve odbc yöntemleri genel yöntemlerdir. Pek çok VTYS üzerinde bu tedarikçi sınıflar yoluyla işlem yapılabilir. Örneğin biz odbc tedarikçi sınıfları ile Oracle işlemleri yapabiliriz. Çünkü odbc genel bir yöntemdir. Fakat oracle tedarikçi sınıfları bu veritabanı için özel düzenlenmiş vve daha hızlı sonuç verir. Benzer biçimde SqlServer veritabanı içinde sqlXXX tedarikçi sınıflarını kullanmak yerinde olur. Access ve MySQL VTYS ler için Oledb tedarikçi sınıfları kullanılabilir.

VTYSleri ile Bağlantının Kurulması: VTYS client server biçimide çalışan sistemlerdir. Dolayısıyla onlara iş yaptırabilmek için önce bir bağlantının sağlanması gerekir. Bunun için tedarikçiYYY olmak üzere YYYConnection sınıfları kullanılır. Yani SqlServer için SqlConnection, Oledb için OledbConnection Odbc için OdbcConnection, Oracle için OracleConnection sınıfları kullanılmalıdır.

Connection sınıfları DbConnection isimli bir Abstract sınıftan türetilmiştir. Bu abstrac sınıfta IDbConnection arayüzünü desteklemektedir.



Görüldüğü gibi tedarikçi sınıfların ortak fonksiyonları DbConnection sınıfında toplanmıştır. Buradaki en genel işlemler IDbConnection arayüzüyle temsil edilmektedir.

VTYS ne bağlanırken Client programın bağlantının niteliği hakkında Server a çeşitli bilgiler iletmesi gerekmektedir. Ado.nette bu bilgiler string türünden ConnectionString denilen property ile temsil edilmiştir. ConnectionString property si IdbConnection arayüzünün bir propertysidir. Dolayısıyla tüm tedarikçi sınıflarda bu propertynin bulunması zorunludur. Bağlantı yazısının formatı VTYS inden VTYS ine farklılık gösternektedir. Yazı genel olarak “;” ile ayrılmış bölümlerden oluşur. Bu “;” arasındaki bölümler x=y biçimde ifadelerden oluşmaktadır. Fakat kesin format karışık olabilmektedir ve bunun için dökumanlara başvurulmalıdır.
Örneğin Access jetmoturuna bağlanmak için tipik olarak aşağıdaki gibi bir bağlantı yazısı oluşturulur.
“Provider=Microsoft.Jet.OLEDB.4.0; Data Source = c:\bin\test.mdb”
Görüldüğü gibi Oledb tedarikçisinde bağlantı yazısı içinde bir provider bölümü olmak zorundadır.

Çünkü Oledb genel bir tedarikçidir. Ve Oledb sınıflarıyla Access, Oracle, Sql gibi VTYS leri ile işlem yapılabilir. Data Source Access veritabanı için ilgili mdb dosyasının yol ifadesini içerir.

YYYConnection sınıflarının default başlangıç fonksiyonları ve String parametreli başlangıç fonksiyonları vardır. Default başlangıç fonksiyonları ile nesne yaratılırsa bağlantı yazısı daha sonra ConnectionString property si ile girilmelidir. Ya da programcı bağlantı yazısını Oledb connection sınıfının başlangıç fonksiyonunda da belirtebilir.

Bağlantı yazısı programın içinde string olarak verilebileceği gibi bir kaynak biçiminde Application Setting içinde Ya da Registery de saklanabilir.

Programcı bağlantı yazısını set ettikten sonra gerçek anlamda bağlantıyı YYYConnection sınıflarının Open fonksiyonuyla kurar. Openfonksiyonu IDbConnection arayüzünden gelmektedir. Bağlantının mümkün olduğu kadar kısa süre içinde kapatılması iyi bir tekniktir. Gerektiğinde yeniden Open işlemi yapılabilir. Open ile bağlantının kurulması sırasında pek çok nedenden dolayı başarısızlık söz konusu olabilir. Programcının Open fonksiyonunu try catch blokları içinde çağırması uygun olur. Bağlantıyı kapatmak için Close fonksiyonu kullanılır. Bu fonksiyonda IDbConnection arayüzünden gelmektedir.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data;
using System.Data.OleDb;

namespace DataBaseSample
{
   public partial class Form1 : Form
   {
      private OleDbConnection m_dbConnection;

      public Form1()
      {
         InitializeComponent();

         m_dbConnection = new OleDbConnection();
         m_dbConnection.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0; "+
            @"Data Source=E:\DotNetApp-1317\DataBaseSample\Person.mdb";
      }

      private void button1_Click(object sender, EventArgs e)
      {
         try
         {
            m_dbConnection.Open();
         }
         catch (Exception ex)
```

```
            {
                MessageBox.Show(ex.Message);
            }
        }
    }
}
```

YYYConnection sınıflarının State isimli property elemanları IdbConnection arayüzünden gelmektedir. Bu property bağlantı durumunu belirten bir Enum türündendir.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data;
using System.Data.OleDb;

namespace DataBaseSample
{
    public partial class Form1 : Form
    {
        private OleDbConnection m_dbConnection;

        public Form1()
        {
            InitializeComponent();

            m_dbConnection = new OleDbConnection();
            m_dbConnection.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0; "+
                @"Data Source=E:\DotNetApp-1317\DataBaseSample\Person.mdb";
        }

        private void button1_Click(object sender, EventArgs e)
        {
            MessageBox.Show(m_dbConnection.State.ToString());

            try
            {
                m_dbConnection.Open();

                //...

                MessageBox.Show(m_dbConnection.State.ToString());


                m_dbConnection.Close();
            }
            catch (Exception ex)
```

```
        {
            MessageBox.Show(ex.Message);
        }
    }
  }
}
```

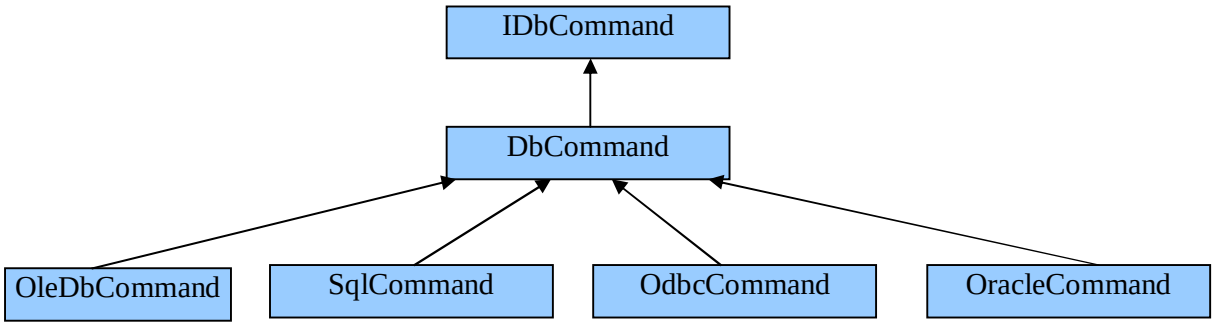
YYYConnection sınıflarının ConnectionTimeout isinli property elemanları IDbConnection arayüzünden gelmektedir. OleDB için zamanaşımı default 15 sn'dir. Yani 15 sn'yeye kadar bağlantı sağlanamazsa Exception oluşur.

YYYConnection sınıflarının diğer elemanları MSDN dökumanlarından izlenebilir.

VTYS'ine SQL Komutlarının Uygulanması: Daha önceden de belirtildiği gibi VTYS i SQL Arayüzüyle yönetilmektedir. .net programcısı tipik olarak yaptırmak istediği işlemleri SQL komutlarına dönüştürür ve bunları Ado.net sınıflarıyla VTYS ine gönderir.

SQL komutlarından bazıları örneğin tipik olarak Select komutu bize kayıtlar geri döndürür. Bu komutlar uygulandığında VTYS nin verdiği kayıtları bizim bir yerde saklamamız gerekir. Kayıtların elde edilerek saklanması için özel bağlantısız sınıflar kullanılmaktadır. Bu konuya daha sonra değinilecektir. O halde şimdilik biz Insert, Update, Delete gibi kayıt geri döndürmeyen komutları uygulayacağız.

Bir Sql komutu YYYCommandsınıfı türünden bir nesne ile temsil edilmektedir. YYYCommand sınıfları DbCommand isimli bir Abstract sınıftan türetilmiştir. Bu sınıfta IDbCommand arayüzünü desteklemektedir.



Programcı örneğin Oledb sınıfı türünden bir nesne yaratır ve Sql komutunu bu nesneye yerleştirir. YYYCommand sınıflarının String türünden CommandText isimli property elemanları IdbCommand arayüzünden gelmektedir ve Sql komutunu temsil etmektedir. Aslında YYYCommand sınıflarının String parametreli başlangıç fonksiyonları da vardır. Sql komutunu CommandText propertysine yerleştirmek yerine bu başlangıç fonksiyonuna parametre olarak vermekte aynı anlamdadır.

CommandText oluşturulduktan sonra artık sıra Sql komutlarının uygulanmasına gelmiştir. YYYCommand nesneleri Connection nesnesini de kullanmaktadır. Bu nedenle programcının YYYcommand sınıflarının Connection propertylerine bağlantı nesnelerini de vermesi gerekir. Örneğin:

```
OleDbCommand dbCommand = new OleDbCommand;
```

dbCommand.Connection = m_dbConnection;
dbCommand.CommandText = "INSERT INTO Info(PersonName, PersonNo) VALUES ('Bob Marley', 632)";

YYYCommand sınıflarının aslında Connection değerini de parametre olarak alan başlangıç fonksiyonu da vardır. Yani yukarıdaki kod şöylede yazılabilirdi.
OleDbCommand dbCommand = new OleDbCommand("INSERT INTO Info(PersonName, PersonNo) VALUES ('Bob Marley', 632)", m_dbConnection);

Nihayet SQL komutu YYYCommand sınıflarının ExacuteNonQuery fonksiyonuyla çalıştırılır.(Fonksiyonun yanlış isimlendirildiğini Microsoft kabul etmektedir.) Fonksiyon Int geri dönüş değerine sahiptir ve işlemden etkilenen kayıt sayısını geri döndürmektedir. Komutun işletilmesi sırasında Excepiton oluşabilir. ExacuteNonQuery fonksiyonu uygulanmadan önce bağlantı Open fonksiyonuyla açılmalıdır.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data;
using System.Data.OleDb;

namespace DataBaseSample
{
   public partial class Form1 : Form
   {
      private OleDbConnection m_dbConnection;

      public Form1()
      {
         InitializeComponent();

         m_dbConnection = new OleDbConnection();
         m_dbConnection.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0; "+
            @"Data Source=E:\DotNetApp-1317\DataBaseSample\Person.mdb";
      }

      private void button1_Click(object sender, EventArgs e)
      {
         MessageBox.Show(m_dbConnection.State.ToString());

         try
         {
            m_dbConnection.Open();

            OleDbCommand dbCommand = new OleDbCommand("INSERT INTO Info(PersonName,
PersonNo) VALUES('Bob Marley', 632)", m_dbConnection);
```

```
            dbCommand.ExecuteNonQuery();
            m_dbConnection.Close();
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }
  }
}

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data;
using System.Data.OleDb;

namespace DataBaseSample
{
  public partial class Form1 : Form
  {
    private OleDbConnection m_dbConnection;

    public Form1()
    {
      InitializeComponent();

      m_dbConnection = new OleDbConnection();
      m_dbConnection.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0; "+
        @"Data Source=E:\DotNetAppBasic\DataBaseSample\Person.mdb";
    }

    private void button1_Click(object sender, EventArgs e)
    {

      try
      {
        m_dbConnection.Open();

        OleDbCommand dbCommand = new OleDbCommand(
   "DELETE FROM Info WHERE PersonNo > 750", m_dbConnection);

        int result = dbCommand.ExecuteNonQuery();

        MessageBox.Show(result.ToString());

        m_dbConnection.Close();
```

```
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }
  }
}
```

SQL Komutlarında Parametre Kullanımı: Şüphesiz veritabanı uygulamalarında insert into, update, slect gibi SQL komutları program içinde static bir biçimde oluşturulmaz kullanıcının girdiği bilgiler kullanılarak oluşturulur.

Bu durumda sorgulama ekranlarından alınan bilgiden SQL cümlelerinin oluşturulması gerekir. Ancak bu yazıların oluşturulması çok kolay değildir.

Anahtar Notlar: Bir form ilk kez açılırken form sınıfına Load isimli mesaj gönderilir. Bazı programcılar form açıldığında yapılacak ilk işlemleri bu Load eventinde gerçekleştirmektedir. Load Eventi oluştuğunda pencere yaratılmıştır. Fakat henüz görünür durumda değildir.

Örnek: Windows forms kullanarak programatik komut satırı oluşturupprograma kendisi kayıt eklemek ve silmek

AddRecordForm.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace CSD
{
  public partial class AddRecordForm : Form
  {
    public AddRecordForm()
    {
      InitializeComponent();
    }

    private void button1_Click(object sender, EventArgs e)
    {
      try
      {
        int val = int.Parse(textBox2.Text);
      }
      catch (Exception ex)
      {
        MessageBox.Show(ex.Message);
        textBox2.Focus();
```

```
            textBox2.SelectAll();
            return;
        }

        this.DialogResult = DialogResult.OK;
    }

    private void button2_Click(object sender, EventArgs e)
    {
        this.DialogResult = DialogResult.Cancel;
    }

    public string PersonName
    {
        get { return textBox1.Text; }
    }

    public int PersonNo
    {
        get
        {
            return int.Parse(textBox2.Text);
        }
    }

  }
}
```

DataBaseMainForm.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data;
using System.Data.OleDb;

namespace CSD
{
  public partial class Form1 : Form
  {
    private OleDbConnection m_dbConnection;

    public Form1()
    {
      InitializeComponent();

    }

    private void addRecordToolStripMenuItem_Click(object sender, EventArgs e)
```

```
    {
        AddRecordForm arf = new AddRecordForm();

        if (arf.ShowDialog() == DialogResult.OK)
        {
            OleDbCommand dbCommand = new OleDbCommand();
            dbCommand.Connection = m_dbConnection;
            dbCommand.CommandText = "INSERT INTO Info(PersonName, PersonNo) VALUES('"
+
                arf.PersonName + "'," + arf.PersonNo.ToString() + ")";

            try
            {
                m_dbConnection.Open();

                dbCommand.ExecuteNonQuery();

                m_dbConnection.Close();

            }

            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }



        }
    }

    private void Form1_Load(object sender, EventArgs e)
    {
        m_dbConnection = new OleDbConnection();
        m_dbConnection.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0; " +
            @"Data Source=E:\DotNetAppBasic\DataBaseRecordProc\Person.mdb";
    }

    private void deleteRecordToolStripMenuItem_Click(object sender, EventArgs e)
    {
        DeleteRecordForm drf = new DeleteRecordForm();

        if (drf.ShowDialog() == DialogResult.OK)
        {
            OleDbCommand dbCommand = new OleDbCommand();
            dbCommand.Connection = m_dbConnection;
            dbCommand.CommandText = "DELETE FROM Info WHERE PersonName = '" +
                drf.PersonName + "' AND PersonNo = " + drf.PersonNo.ToString();

            try
            {
                m_dbConnection.Open();
```

```
            int result = dbCommand.ExecuteNonQuery();
            m_dbConnection.Close();

            if (result == 0) {
               MessageBox.Show("Cannot find record");
               return;
            }

         }

         catch (Exception ex)
         {
            MessageBox.Show(ex.Message);
         }
      }
   }
}
```

DataBaseDeleteForm.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace CSD
{
   public partial class DeleteRecordForm : Form
   {
      public DeleteRecordForm()
      {
         InitializeComponent();
      }

      private void button1_Click(object sender, EventArgs e)
      {
         try
         {
            int val = int.Parse(textBox2.Text);
         }
         catch (Exception ex)
         {
            MessageBox.Show(ex.Message);
            textBox2.Focus();
            textBox2.SelectAll();
            return;
         }

         this.DialogResult = DialogResult.OK;
```

```
    }

    private void button2_Click(object sender, EventArgs e)
    {
        this.DialogResult = DialogResult.Cancel;
    }

    public string PersonName
    {
        get { return textBox1.Text; }
    }

    public int PersonNo
    {
        get
        {
            return int.Parse(textBox2.Text);
        }
    }

  }
}
```

SQL komutlarında parametre kullanmak için önce komut yazısı parametreyi belirten bir yer tutucuyla oluşturulur. Bu yer tutucu Oledb tedarikşi sınıflarında "?" karakteri SQL Server ve ORACLE tedarikçi sınıflarında Add parametre ismi şeklindedir. Örneğin Oledb için parametrik bir INSERT INTO komutu şöyle yazılır.

INSERT INTO Info(PersonName, PersonNo) VALUES(?, ?)

Sql Server ve Oracle tedarikçilerinde yertutucu yalnızca bir soru işareti karekteri değildir bir isim de içermektedir. Örneğin:*INSERT INTO Info(PersonName, PersonNo) VALUES(@Name, @No)*

Yertutucular yerine yerleştirilecek olan yazılar YYYParameter isimli sınıf nesneleri ile temsil edilir. Programcı herbir yer tutucu için bir YYYParameter nesnesi yaratır ve bu nesneleri YYYCommand sınıfının parameter isimli collection elemanına ekler. Parameter isimli collection eleman YYYParameterCollection isimli bir sınıf türündendir. OleDb tedarikçi sınıflarında yertutucu yalnızca bir ssoru işaretiyle belirlendiği için bu collectiona belirleme sırası önemlidir. Halbuki Sql Server ve Oracle tedarikçi sınıflarında yertutucularla eşlendirmeler sıraya göre değil isme göre yapılır.

YYYParameter sınıfının ParameterName isimli property elemanı yertutucu parametrenin ismini, Value isimli property elemanı yertutucu elemanın değerini belirtir. Value propertysi object türünden olduğu için bizim bu value propertysi içindeki değerin türünü de ayrıca belirtmemiz gerekir. Bunun için YYYParameter sınıfının DbType isimli property elemanı kullanılır. Aslında YYYParameter sınıfının çeşitli başlangıç fonksiyonları bu değerleri parametre olarak almaktadır. OleDb tedarikçisinde ParamterName kullanılmamaktadır boş geçilebilir.

Örneğin OleDb için bir parametre işlemi Aşağıdaki şekilde yapılabilir

AddForm.cs

```
using System;
using System.Collections.Generic;
```

```
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace CSD
{
    public partial class AddRecordForm : Form
    {
        public AddRecordForm()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            try
            {
                int val = int.Parse(textBox2.Text);
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
                textBox2.Focus();
                textBox2.SelectAll();
                return;
            }

            this.DialogResult = DialogResult.OK;
        }

        private void button2_Click(object sender, EventArgs e)
        {
            this.DialogResult = DialogResult.Cancel;
        }

        public string PersonName
        {
            get { return textBox1.Text; }
        }

        public int PersonNo
        {
            get
            {
                return int.Parse(textBox2.Text);
            }
        }

    }
```

```
}
MainForm.cs
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data;
using System.Data.OleDb;

namespace CSD
{
   public partial class Form1 : Form
   {
      private OleDbConnection m_dbConnection;

      public Form1()
      {
         InitializeComponent();

      }

      private void addRecordToolStripMenuItem_Click(object sender, EventArgs e)
      {
         AddRecordForm arf = new AddRecordForm();

         if (arf.ShowDialog() == DialogResult.OK)
         {
            OleDbCommand dbCommand = new OleDbCommand();
            dbCommand.Connection = m_dbConnection;
            dbCommand.CommandText = "INSERT INTO Info(PersonName, PersonNo) VALUES(?,
?)";

            OleDbParameter param1 = new OleDbParameter(null, arf.PersonName);
            OleDbParameter param2 = new OleDbParameter(null, arf.PersonNo.ToString());

            dbCommand.Parameters.AddRange(new OleDbParameter[] { param1, param2 });


            try
            {
               m_dbConnection.Open();

               dbCommand.ExecuteNonQuery();

               m_dbConnection.Close();

            }
```

```
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }



        }
    }

    private void Form1_Load(object sender, EventArgs e)
    {
        m_dbConnection = new OleDbConnection();
        m_dbConnection.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0; " +
            @"Data Source=E:\DotNetAppBasic\DataBaseRecordProc\Person.mdb";
    }

    private void deleteRecordToolStripMenuItem_Click(object sender, EventArgs e)
    {
        DeleteRecordForm drf = new DeleteRecordForm();

        if (drf.ShowDialog() == DialogResult.OK)
        {
            OleDbCommand dbCommand = new OleDbCommand();
            dbCommand.Connection = m_dbConnection;
            dbCommand.CommandText = "DELETE FROM Info WHERE PersonName = '" +
                drf.PersonName + "' AND PersonNo = " + drf.PersonNo.ToString();

            try
            {
                m_dbConnection.Open();
                int result = dbCommand.ExecuteNonQuery();
                m_dbConnection.Close();

                if (result == 0) {
                    MessageBox.Show("Cannot find record");
                    return;
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
    }
  }
}
```

DeleteForm.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
```

```
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace CSD
{
    public partial class DeleteRecordForm : Form
    {
        public DeleteRecordForm()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            try
            {
                int val = int.Parse(textBox2.Text);
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
                textBox2.Focus();
                textBox2.SelectAll();
                return;
            }

            this.DialogResult = DialogResult.OK;
        }

        private void button2_Click(object sender, EventArgs e)
        {
            this.DialogResult = DialogResult.Cancel;
        }

        public string PersonName
        {
            get { return textBox1.Text; }
        }

        public int PersonNo
        {
            get
            {
                return int.Parse(textBox2.Text);
            }
        }

    }
}
```

YYYParameterCollection sınıfının çeşitli Add fonksiyonları kendi içinde YYYParameter nesnelerini oluşturup zaten eklemektedir. Bu durumda yukarıdaki parametre oluşturma işlemi şöylede yapılabilirdi.

AddRecordForm.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace CSD
{
   public partial class AddRecordForm : Form
   {
      public AddRecordForm()
      {
         InitializeComponent();
      }

      private void button1_Click(object sender, EventArgs e)
      {
         try
         {
            int val = int.Parse(textBox2.Text);
         }
         catch (Exception ex)
         {
            MessageBox.Show(ex.Message);
            textBox2.Focus();
            textBox2.SelectAll();
            return;
         }

         this.DialogResult = DialogResult.OK;
      }

      private void button2_Click(object sender, EventArgs e)
      {
         this.DialogResult = DialogResult.Cancel;
      }

      public string PersonName
      {
         get { return textBox1.Text; }
      }

      public int PersonNo
      {
```

```
            get
            {
                return int.Parse(textBox2.Text);
            }
        }

    }
}

MainRecordForm.cs
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data;
using System.Data.OleDb;

namespace CSD
{
    public partial class Form1 : Form
    {
        private OleDbConnection m_dbConnection;

        public Form1()
        {
            InitializeComponent();

        }

        private void addRecordToolStripMenuItem_Click(object sender, EventArgs e)
        {
            AddRecordForm arf = new AddRecordForm();

            if (arf.ShowDialog() == DialogResult.OK)
            {
                OleDbCommand dbCommand = new OleDbCommand();
                dbCommand.Connection = m_dbConnection;
                dbCommand.CommandText = "INSERT INTO Info(PersonName, PersonNo) VALUES(?,
?)";
                dbCommand.Parameters.Add(null, arf.PersonName);
                dbCommand.Parameters.Add(null, arf.PersonNo.ToString());

                try
                {
                    m_dbConnection.Open();

                    dbCommand.ExecuteNonQuery();

                    m_dbConnection.Close();
```

```
            }

            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }



        }
    }

    private void Form1_Load(object sender, EventArgs e)
    {
        m_dbConnection = new OleDbConnection();
        m_dbConnection.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0; " +
            @"Data Source=E:\DotNetAppBasic\DataBaseRecordProc\Person.mdb";
    }

    private void deleteRecordToolStripMenuItem_Click(object sender, EventArgs e)
    {
        DeleteRecordForm drf = new DeleteRecordForm();

        if (drf.ShowDialog() == DialogResult.OK)
        {
            OleDbCommand dbCommand = new OleDbCommand();
            dbCommand.Connection = m_dbConnection;
            dbCommand.CommandText = "DELETE FROM Info WHERE PersonName = '" +
                drf.PersonName + "' AND PersonNo = " + drf.PersonNo.ToString();

            try
            {
                m_dbConnection.Open();
                int result = dbCommand.ExecuteNonQuery();
                m_dbConnection.Close();

                if (result == 0) {
                    MessageBox.Show("Cannot find record");
                    return;
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
    }
}
```

DeleteRecordForm.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace CSD
{
    public partial class DeleteRecordForm : Form
    {
        public DeleteRecordForm()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            try
            {
                int val = int.Parse(textBox2.Text);
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
                textBox2.Focus();
                textBox2.SelectAll();
                return;
            }

            this.DialogResult = DialogResult.OK;
        }

        private void button2_Click(object sender, EventArgs e)
        {
            this.DialogResult = DialogResult.Cancel;
        }

        public string PersonName
        {
            get { return textBox1.Text; }
        }

        public int PersonNo
        {
            get
            {
                return int.Parse(textBox2.Text);
            }
        }
```

```
    }
}
```

Yukarıdaki uygulama Sql Server da yapılsaydı isimli bir arama söz konusu olacaktı. Bu durumda ekleme işleminin sırası önemli değildir.

```
SqlCommand dbCommand = new SqlCommand();
dbCommand.Connection = m_Conection;
dbCommand.CommandText = “INSERT INTO Info(PersonName, PersonNo) VALUES(@Name, @No)”;
dbCommand.Parameter.Add(Add(“No”, arf.PersonNo.ToString());
dbCommand.Parameter.Add(Add(“ame”, arf.PersonName);
```

ADO.NET’in Bağlantısız Sınıfları:

Bağlantısız sınıfları veri tabanı bağlantı işlemlerinden bağımsız veri saklama işlemleridir.

DataTable Sınıfı:

Datatable sınıfı veri tabanı tablosunun bellekte temsil eden temel bir sınıftır. SELECT komutuyla vtys den elde edilen bilgiler bit datatable içerisinde geçici olarak saklanır. Ayrıca ADO.NET sisteminde bir datatable nesnesi ile gerçek veri tabanına bir aktarımda sağlanır. Yani biz bir datatable üzerindeki bilgilerde değişikli yapıp bunu veri tabanına yansıtabiliriz.

DataGridWiev isimli kontrol temel olarak bir veri tabanı tablosunu görüntülemek için kullanımaktadır. Bu kontrol oldukça detaylıdır ve daha sonra ele alınacaktır. Fakat en basit kullanımı kontrolün DataSource isimli elemanına bir DataGridWiev atanmasıdır. Bu durumda dataGridWiev kontrolü bu veri tabanı tablosunu görüntüler.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;


namespace DataTableSample
{

   public partial class Form1 : Form
   {
      private DataGridView m_dgv;

      public Form1()
      {
         InitializeComponent();

         m_dgv = new DataGridView();
         m_dgv.Dock = DockStyle.Fill;

         DataTable dt = new DataTable("File");
```

```
            dt.Columns.Add("PersonName", typeof(string));
            dt.Columns.Add("PersonNo", typeof(long));

            foreach (string file in Directory.GetFiles(@"c:\windows"))
            {
               FileInfo fi = new FileInfo(file);

               dt.Rows.Add(fi.Name, fi.Length);

            }

            m_dgv.DataSource = dt;

            this.Controls.Add(m_dgv);
        }
    }
}
```

Bir DataTable nesnesi DataTable sınıfının default başlangıç fonksiyonu ile yada string parametreli başlangıç fonksiyonu ile de yaratılabilir. Tablonun bir ismi vardır. Bu TableName property si ile belirlenir. Başlangıç fonksiyonun daki string bu ismi belirtmektedir.

Tablo yaratıldıktan sonra tabloya sutunlar eklenmelidir. Daha sonra da satırlar eklenir. Aslında bu ekleme işlemi ListWiev kontrolündeki mantığa çok benzemektedir. DataTale sınıfının sütünları DataColumn isimli sınıfla temsil edilmektedir. O halde programcı her sutun için bir DataColumn nesnesi oluşturup ColumnsCollection elemanına ekler.

DataTable nesnesinin bir sutunu için en önemli iki bilgi sutun ismi ve türüdür. İsim string ile tür ise Type sınıfı ile belirtilir. DataColumn sınıfınıı başlangıç fonksiyonları ları ise bu iki özellik hemen set edilebilir. Yada sınfının ColumnName ve DataType property leri kullanılır. Örneğin;

DataTable dt = new DataTable (“ Person”);

DataColumn dt1=new DataColumn(“PersonName”, typeof(string));

DataColumn dt2=new DataColumn(“PersonNo”, typeof(int));

dt.Column.AddRange(new DataColumn[] {dt1,dt2});

Aslında DataColumn nesnelerini ayrı ayrı yaratıp eklemektense tıpkı ListView kontrolünde oluğu gibi daha pratik bir yoluda vardır. DataTable sınıfının Columns isimli property elemanı DataColumnsCollection siimli bir Collection sınıf türündendrir.

public DataColumnCollection columns (get;)

Bu Collection sınıfın string ve type parametreli Add fonksiyonları zaten kendi içerisinde DataColumn nesnesini kendisi yaratıp eklemektedir. O halde yukarıdaki ekleme işlemi daha basit olarak şöyle yapılabilir.

DataTable dt = new DataTable(“Person”);

dt.Columns.Add(“PersonName”, typeof(string));

dt.Columns.Add(“PersonNo”, typeof(int));

DataColumn sınıfının başka önemli elemanlarıda vardır. Örneğin MaxLenght isimli property elemanı kolondaki maximum karakter sayısını sınırlandırmada kullanılabilir. Ordinal isimli

property eleman sutunun sura numarasını elde etmekte kullanılabilir. Column Mapping isimli property elemanı ileride ele alınacaktır.

Bir veri tabanını tablosunun satrıları DataRow sınıfı ile temsil edilmektedir. Yani her bir satır aslında bir DataRow nesnesidir.

Aslında bir DataRow nesnesi new operatörü ile sıfırdan yaratılmaz. Çünkü DataRow nesnesinin içeriği sutun özelliğine göre configure edilmiş olmasıdır. Bu nedenle yeni bir DataRow nesnesi DataTable sınıfının new Row fn ile yaratılır. Örneğin ;

DataRow dr = dt.new Row();

Bu biçimde bir DataRow nesnesi yaratıldığında programcı bunun içini doldurup DataTable nesnesine eklemelidir. new Row fonksiyonu yanlızca bize o sütün yapısına ilişkin bir DataRow nesnesi oluşturmaktadır. DataTable sınıfının Rows isimli collection elemanı DataRow collection isimli bir collection sınıf türündendir. Bu collection DataRow nesnelerini tutmaktadır. Örneğin;

```
DataRow dr = dt.NewRow();
//….
dt.Rows.Add(dr);
```

Şimdi DataRow nesnesinin içinin doldurulmasına sıra gelmiştir. Bunu sağlamanın en pratik yolu indeksleyicileri kullanmaktır. Temel olarak üç tür indeksleyici vardır. DatColumm parametreli, string parametreli ve int parametreli. String parametreli indeksleyici kolon ismini almaktadır. int parametreli indeksleyici kolon indeks numarasını alır. Ve DataColumn parametreli indeksleyici DataColumn nesnesini alarak ilgili satırın

```
DataRow dr = dt.NewRow();
dr[“PersonName”] = “Kaan Aslan”;
dr[“PersonNo”] = 123;
dt.Rows.Add(dr);
```

Anahtar Notlar: DataGridView kontolü üzerinde ekleme, silme ve güncelleme yapıldığında ileride ele alınacağı gibi bu ilgili satırlara yansıtılmaktadır. Yani DataGridView sadece gösterme amaçlı değil aynı zamanda günceleme amaçlıda kullanılmaktadır.

Satır ekleme işlemi *DataRowCollection sınıfının ParamsObject[] Add* fonksiyonuylada yapılmaktadır.

DataRow sınıfı aslında bir satırın birkaç versiyonunu bir arada tutmaktadır. Sınıfın RowState isimli property elemanı DataRowState isimli enum türündendir. Bu enum türünün anlamları ve DataRow nesnesinin içinde bulunduğu durum şunlardan biri olabilir.

Detached : Eğer DataRow nesnesi bu durumdaysa tabloya bağlanmamış durumdadır . Örneğin NewRow ile yaratılmıştır fakat henüz tabloya eklenmemiştir. Yada tablodan çıkartılmış durumdadır.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;


namespace DataTableSample
{

   public partial class Form1 : Form
   {
      private DataGridView m_dgv;

      public Form1()
      {
         InitializeComponent();

         m_dgv = new DataGridView();
         m_dgv.Dock = DockStyle.Fill;

         DataTable dt = new DataTable("File");

         dt.Columns.Add("PersonName", typeof(string));
         dt.Columns.Add("PersonNo", typeof(long));

         foreach (string file in Directory.GetFiles(@"c:\windows"))
         {
            FileInfo fi = new FileInfo(file);

            dt.Rows.Add(fi.Name, fi.Length);

         }

         DataRow dr = dt.NewRow();

         MessageBox.Show(dr.RowState.ToString());

         m_dgv.DataSource = dt;

         this.Controls.Add(m_dgv);
      }
   }
}
```

Added : DataRow nesnesi yaratılmış ve tabloya eklenmiştir. Fakat AcceptChanges fonksiyonu henüz çağrılmamıştır.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;


namespace DataTableSample
{

   public partial class Form1 : Form
   {
      private DataGridView m_dgv;

      public Form1()
      {
         InitializeComponent();

         m_dgv = new DataGridView();
         m_dgv.Dock = DockStyle.Fill;

         DataTable dt = new DataTable("File");

         dt.Columns.Add("PersonName", typeof(string));
         dt.Columns.Add("PersonNo", typeof(long));

         foreach (string file in Directory.GetFiles(@"c:\windows"))
         {
            FileInfo fi = new FileInfo(file);

            dt.Rows.Add(fi.Name, fi.Length);

         }

         DataRow dr = dt.NewRow();

         dt.Rows.Add(dr);

         MessageBox.Show(dr.RowState.ToString());

         m_dgv.DataSource = dt;

         this.Controls.Add(m_dgv);
      }
   }
}
```

Unchanged : DatatRow nesnesi tabloya eklenmiş durumdadır. Ve AcceptChanges nesnesi

çağrılmıştır ancak nesne üzerinde henüz bir değişiklik yapılmamıştır.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;


namespace DataTableSample
{

   public partial class Form1 : Form
   {
      private DataGridView m_dgv;

      public Form1()
      {
         InitializeComponent();

         m_dgv = new DataGridView();
         m_dgv.Dock = DockStyle.Fill;

         DataTable dt = new DataTable("File");

         dt.Columns.Add("FileName", typeof(string));
         dt.Columns.Add("FileSize", typeof(long));

         foreach (string file in Directory.GetFiles(@"c:\windows"))
         {
            FileInfo fi = new FileInfo(file);

            dt.Rows.Add(fi.Name, fi.Length);

         }

         DataRow dr = dt.NewRow();

         dt.Rows.Add(dr);

         dr["FileName"] = "X.DAT";
         dr["FileSize"] = 5353535;

         dt.AcceptChanges();

         MessageBox.Show(dr.RowState.ToString());

         m_dgv.DataSource = dt;
```

```
            this.Controls.Add(m_dgv);
        }
    }
}
```

Modified : En son AcceptChanges fonksiyonu çağrıldıktan sonra nesne üzerinde değişiklik yapılmıştır. Fakat henüz yeniden AcceptChanges fonks,yonu çağrılmamıştır.

Deleted : Satır delete fonksiyonu ile silinmiştir.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;


namespace DataTableSample
{

    public partial class Form1 : Form
    {
        private DataGridView m_dgv;

        public Form1()
        {
            InitializeComponent();

            m_dgv = new DataGridView();
            m_dgv.Dock = DockStyle.Fill;

            DataTable dt = new DataTable("File");

            dt.Columns.Add("FileName", typeof(string));
            dt.Columns.Add("FileSize", typeof(long));

            foreach (string file in Directory.GetFiles(@"c:\windows"))
            {
                FileInfo fi = new FileInfo(file);

                dt.Rows.Add(fi.Name, fi.Length);

            }

            DataRow dr = dt.NewRow();
            // Detached

            dt.Rows.Add(dr);
            // Added
```

```
            dr["FileName"] = "X.DAT";
            dr["FileSize"] = 5353535;
            // Added

            dt.AcceptChanges();
            // Unchanged

            dr["FileName"] = "Y.DAT";
            // Modified

            dt.AcceptChanges();
            // Unchanged

            dr.Delete();

            MessageBox.Show(dr.RowState.ToString());

            m_dgv.DataSource = dt;

            this.Controls.Add(m_dgv);
        }
    }
}
```

Şimdi bir DataRow nesnesine işlemler uygulayarak onun durumunun nasıl değiştiğini değerlendirelim.

1 – DataRow nesnesini NewRow ile yaratalım şu anda **Detached** durumdadır.

2 – DataRow nesnesini DataTable ye ekleyelim. Nesne şimdi **Added** durumdadır.

3 – Henüz eklediğimiz DataRow üzerinde değişiklik yapalım. AcceptChanges nesnesi çağrılmadığı için nesne henüz **Added** durumundadır.

4 – Şimdi DataTable sınıfının AcceptChanges fonksiyonu çağıralım. Artık DataRow nesnesi UnChanged duruma gelmiştir.

5 – Şimdi DataRow üzerinde herhangi bir değişiklik yapalım. Artık nesne **Modified** durumdadır.

6 – Şimdi yeniden DataTable sınıfının AcceptChanges fn çağıralım. Nesne yeniden Unchanged duruma geçmiştir.

7 – Şimdi DataRow nesnesini DataRow sınıfının **Delete** fn ile silelim. Nesne **Deleted** durumdadır. Aynı zamanda tablodan çıkartılmıştır. Biz eğer ilgili satırı DataRow sınıfının ilgili fn ile değilde Rows Collection unun Remove fonksiyonu ile silmeye çalışsaydık bu durumda nesnenin tabloyla hiçbir ilişkisi kalmayacaktı bu durumda nesne deleted değil detached duruma gelecekti. Görüldüğü gibi DataRow sınıfının Delete fonksiyonu silme yapmıştır fakat tablo ilişkisini henüz kesmemiştir. Örneğin ;

```
DataRow dr = dt.NewRow();
 // Detached
dt.Rows.Add(dr);
 // Added
 dr["FileName"] = "X.DAT";
 dr["FileSize"] = 5353535;
 // Added
dt.AcceptChanges();
// Unchanged
dr["FileName"] = "Y.DAT";
// Modified
dt.AcceptChanges();
 // Unchanged
dr.Delete();
//Deleted
dt.Rows.Remove(dr);
 //Detached
MessageBox.Show(dr.RowState.ToString());
m_dgv.DataSource = dt;
```

DataRow Nesnesinin Değişik Versiyonları: Anımsanacağı gibi DataRow nesneleri DataTable nesnesinin Rows Colleciton yoluyla tabloya eklenmektedir. Ayrıca biz DataRow sınıfının çeşitli indeksleyicileriyle ilgili sutun elemanlarına ulaşmıştık. İşte DataRow sınıfının ayrıca tüm tek parametreli indeksleyicelerinin yanı sıra iki parametreli indeksleyicileride vardır. Bu indeksleyiclerin parametreli yine DataRow string yada int türündendir. Fakat ikinci parametreleri DataRowVorsion isimli bir enum türündendir. Aslında bir DataRow nesnesinin duruma göre dört farklı versiyonu tutulmaktadır. Bu versiyonlar şunlardır :

Current : Bu versiyon DataRow nesnesinin son durumunu veren versiyondur. Zaten tek parametreli indexleyiciler bu versiyonu vermektedir.

Orginal : Orginal versiyondan amaç bir önceki durumunu tutmaktadır. Örneğin biz Unchanged durumdaki nesneyi değiştirip Modified duruma sokmuş olalım. Current versiyon bize değişmiş olan durumu verir. Halbuki Orginal versiyon değişimden önceki değeri bize verecektir. Her ne zaman AcceptChanges uygulansa Orginal versiyon tamamen Current versiyonla aynı yapılmaktadır. Yani nesne UnChanged durumdaysa zaten Orginal versiyonla Current version aynı durumdadır. Ayrıca nesnenin orginal versiyonu newRow ile nesne yaratılıp yaratılmaz oluşturulmaz tabloya ekleme yapıldığında da oluşturulmaz AcceptChanges fn çağrıldıktan sonra oluşturulur. DataRow nesnesi Delete fn ile silindiğinde onun Current versiyonu yok olur fakat Orginal versiyonu kalır.

DataSet Sınıfı:
Anımsanacağı gibi DataTable sınıfı bir veritabanı tablosunu bellekte temsil etmektedir. DataSet

sınıfı ise bir grup DataTable nesnesini tutmak için düşünülmüştür. Yani DaSet DataTable lardan oluşmaktadır. Bir DataSet nesnesi sınıfın default başlangıç fonksiyonuyla yaratılabilir. DataSet nesnelerinin de birer ismi vardır. İsim DataSetName property si ile temsil edilmektedir.
public string DataSetName { get; set;}
DataSet sınıfının Tables isimli Collection elemanı DataTableCollection isimli Collection türündendir ve DataTable nesnelerini tutmaktadır.

Select Komutuyla Veritabanından Bilgilerin Elde Edilmesi: Bilindiği gibi Slect SQL komutu kayıt geri döndüren bir komuttur. Geri döndürülen kayıtlar bir DataTable ya da bir DataSet içindeki DataTable içerisine yerleştirilir.

Sorgulama sonucunda elde edilen kayıtların DataTable ya da DataSet içine yerleştirilmesi YYYDataAdapter sınıflarıyla gerçekleştirilmektedir. YYYDataAdapter nesneleri bir YYYCommand nesnesi alır. YYYDataAdapter nesnesi şöyle kullanılmaktadır.

1. YYYDataAdapter nesnesi sınıfın herhangi bir başlangıç fonksiyonuyla yaratılabilir.
2. Programcının YYYDataAdapter nesnesine en azından bir Select SQL komutu vermesi gerekir. Bunu yapmanın tipik yolu Select komutu için bir tane YYYCommand nesnesi oluşturmak ve bu Command nesnesini YYYDataAdapter sınıfının SelectCommand propertysine yerleştirmektir. Zaten sınıfın YYYCommand parametreli bir başlangıç fonksiyonuda vardır. Diğer bir seçenek Select SQL komutunun komut yazısını YYYDataAdapter sınıfının başlangıç fonksiyonunda belirtmek olabilir. Bu durumda başlangıç fonksiyonu YYYCommand nesnesini oluşturup yine sınıfın SelectCommand propertysini yerleştirir. O halde YYYCommand sınıfları YYYConnection nesnesini YYYDataAdapter sınıflarıda YYYCommand nesnelerini istemektedir.
3. SQL Select komutunun uygulanarak bunun bir DataTable ya da DataSet nesnesini içine sonuçları yerleştirebilmesi için YYYDataAdapter sınıflarının Fill fonksiyonları kullanılır. Örneğin bir Fill fonksiyonunun parametresi şöyledir.

*public int Fill(DataTable dataTable)*diğer bir Fill fonksiyonuda şöyledir.
public override int Fill(DataSet dataSet) Bu biçimde elde edilen tabloların isimleri sırasıyla Table, Table1, Table2, ... biçimindedir. Fakat programcı isterse istenilen isimli bir DataTable da yaratabilir. Bunun için aşağıdaki Fill fonksiyonu kullanılır.
public int Fill(DataSet dataSet, string srcTable)

AddRecord.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace CSD
{
   public partial class AddRecordForm : Form
   {
      public AddRecordForm()
      {
         InitializeComponent();
```

```
        }

        private void button1_Click(object sender, EventArgs e)
        {
            try
            {
                int val = int.Parse(textBox2.Text);
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
                textBox2.Focus();
                textBox2.SelectAll();
                return;
            }

            this.DialogResult = DialogResult.OK;
        }

        private void button2_Click(object sender, EventArgs e)
        {
            this.DialogResult = DialogResult.Cancel;
        }

        public string PersonName
        {
            get { return textBox1.Text; }
        }

        public int PersonNo
        {
            get
            {
                return int.Parse(textBox2.Text);
            }
        }

    }
}
```

MainRecordForm.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data;
using System.Data.OleDb;

namespace CSD
{
```

```
public partial class Form1 : Form
{
    private OleDbConnection m_dbConnection;

    public Form1()
    {
        InitializeComponent();

    }

    private void addRecordToolStripMenuItem_Click(object sender, EventArgs e)
    {
        AddRecordForm arf = new AddRecordForm();

        if (arf.ShowDialog() == DialogResult.OK)
        {

            OleDbCommand dbCommand = new OleDbCommand();
            dbCommand.Connection = m_dbConnection;
            dbCommand.CommandText = "INSERT INTO Info(PersonName, PersonNo) VALUES(?,
?)";
            dbCommand.Parameters.Add("Name", arf.PersonName);
            dbCommand.Parameters.Add("No", arf.PersonNo.ToString());

            try
            {
                m_dbConnection.Open();

                dbCommand.ExecuteNonQuery();

                m_dbConnection.Close();

            }

            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }



        }
    }

    private void Form1_Load(object sender, EventArgs e)
    {
        m_dbConnection = new OleDbConnection();
        m_dbConnection.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0; " +
            @"Data Source=E:\DotNetAppBasic\DataBaseRecordProc\Person.mdb";
    }

    private void deleteRecordToolStripMenuItem_Click(object sender, EventArgs e)
```

```
        {
            DeleteRecordForm drf = new DeleteRecordForm();

            if (drf.ShowDialog() == DialogResult.OK)
            {
                OleDbCommand dbCommand = new OleDbCommand();
                dbCommand.Connection = m_dbConnection;
                dbCommand.CommandText = "DELETE FROM Info WHERE PersonName = '" +
                    drf.PersonName + "' AND PersonNo = " + drf.PersonNo.ToString();

                try
                {
                    m_dbConnection.Open();
                    int result = dbCommand.ExecuteNonQuery();
                    m_dbConnection.Close();

                    if (result == 0) {
                        MessageBox.Show("Cannot find record");
                        return;
                    }
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message);
                }
            }
        }

        private void findToolStripMenuItem_Click(object sender, EventArgs e)
        {
            FindRecordForm frf = new FindRecordForm(m_dbConnection);

            if (frf.ShowDialog() == DialogResult.OK)
            {


            }
        }
    }
}
```

DeleteRecordForm.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace CSD
{
```

```
    public partial class DeleteRecordForm : Form
    {
        public DeleteRecordForm()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            try
            {
                int val = int.Parse(textBox2.Text);
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
                textBox2.Focus();
                textBox2.SelectAll();
                return;
            }

            this.DialogResult = DialogResult.OK;
        }

        private void button2_Click(object sender, EventArgs e)
        {
            this.DialogResult = DialogResult.Cancel;
        }

        public string PersonName
        {
            get { return textBox1.Text; }
        }

        public int PersonNo
        {
            get
            {
                return int.Parse(textBox2.Text);
            }
        }

    }
}
```

FindRecordForm.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
```

```
using System.Windows.Forms;
using System.Data.OleDb;

namespace CSD
{
   public partial class FindRecordForm : Form
   {
      private OleDbConnection m_connection;

      public FindRecordForm()
      {
         InitializeComponent();
      }

      public FindRecordForm(OleDbConnection dbConnection) : this()
      {
         m_connection = dbConnection;
      }

      private void button1_Click(object sender, EventArgs e)
      {

         try
         {
            OleDbCommand dbCommand = new OleDbCommand("SELECT PersonName, PersonNo
FROM Info WHERE " + textBox1.Text, m_connection);
            OleDbDataAdapter da = new OleDbDataAdapter(dbCommand);
            DataSet ds = new DataSet();

            da.Fill(ds);
            dataGridView1.DataSource = ds.Tables["Table"];

         }
         catch (Exception ex)
         {
            MessageBox.Show(ex.Message);
         }



      }

      private void button2_Click(object sender, EventArgs e)
      {
         this.DialogResult = DialogResult.OK;
      }

   }
}
```

Fill fonksiyonlarının geri dönüş değerleri DataSet ve DataTable içine yerleştirilen satır sayısını belirtir. Fill fonksiyonları otomatik olarak kendi içlerinde bağlantıyı açarak SelectCommand

propertysinde belirtilen SQL komutunu uygular. Sonucuda ilgili DataSet ya da DataTable içine yerleştirir. Örneğin bir Fill işlemi şöyle yapılabilir.

```
OleDbCommand dbCommand = new OleDbCommand("SELECT PersonName, PersonNo FROM
Info WHERE PersonNo>300"
OleDbAdapter da = new OleDbAdapter(dbCommand);
DataSet ds = new DataSet();
da.Fill(ds);
dataGridView1.DataSource = ds.Tables["Table"];
```

Örneğimizde tablo ismi verilmediği için Fill fonksiyonu default olarak Table isimli bir tabloya yerleştirme yapmıştır. Ama istenirse aşağıdaki gibi bizim istediğimiz bir tablo ismiyle yarleştirme yapılır.

```
DataSet ds = new DataSet();
da.Fill(ds, "info")
```

Fill fonksiyonu bir DataTable da verilebilirdi. Bu durumda yerleştirme doğrudan o DataTable nesnesine yapılacaktı.

```
OleDbCommand dbCommand = new OleDbCommand("SELECT PersonName, PersonNo FROM
Info WHERE PersonNo>300"
OleDbAdapter da = new OleDbAdapter(dbCommand);
DataTable dt = new DataTable();
da.Fill(dt);
dataGridView1.DataSource = dt;
```

DataSet ya da DataTable içindeki bilgilerin Veritabanına Yansıtılması: Veritabanına sorgulama yapıp sonuçları bir DataSet ya da DataTable a yerleştirdikten sonra bu nesneler üzerinde işlemler yaparsak bu işlemden otomatik olarak veritabanı etkilenmez. İşte bu nesneler üzerindeki satırlarda değişiklik yaptığımızda bu değişikliklerin veritabanına yansıtılması için bazı şeylerin bilinmesi gerekmektedir. Öncelikle yapılan değişikliklerin yansıtılması YYYDataAdapter sınıflarının Update fonksiyonlarıyla gerçekleştirilmektedir. Bu sınıfların pek çok Update fonksiyonu vardır. Fakat en tipik olan DataSet parametreli ve DataTable parametreli Update fonksiyonlarıdır. Ancak Update fonksiyonlarını kullanmadan önce bazı hazırlıkların da yapılması gerekmektedir. Ado.Net içinde işlemleri kolaylaştıran pek çok yardımcı sınıf düşünülmüştür. Fakat programcının daha aşağı seviyeli mekanizmaları bilmesinde fayda vardır.

Update fonksiyonunu çağırmadan önce programcının bazı command nesnelerini hazırlaması gerekir. Aslında YYYDataAdapter sınıflarının yalnızca SelectCommand isimli bir Command property si yoktur. Aynı zamanda UpdateCommand, InsertCommand, DeleteCommand propertyleri de vardır. Gerçektende SelectCommand propertysi en önemli property olduğu için başlangıç fonksiyonunda da belirtilmiştir. O halde YYYDataAdapter sınıflarının bu command nesneleri programcı tarafından oluşturulmuş olmalıdır. Fill fonksiyonları SelectCommand propertysi ile belirtilen command nesnesini uygular. Eğer biz yalnızca Fill işlemi uygulayacaksak YYYDataAdapter sınıfının yalnızca SelectCommand propertysinin set edilmiş olması yeterlidir. Fakat eğer biz Update işlemi uygulayacaksak InsertCommand, UpdateCommand, DeleteCommand nesnelerinin de oluşturulmuş olması gerekmektedir. Çünkü Update fonksiyonları ileride açıklanacağı gibi gerektiğinde bu Command nesnelerini kullanmaktadır. Yani görüldüğü gibi Fill fonksiyonları yalnızca SelectCommand nesnesini kullanırken Update fonksiyonları InsertCommand, DeleteCommand ve UpdateCommand nesnelerini de kullanmaktadır.

YYYDataAdapter sınıflarının Fill fonksiyonları kabaca önce bağlantıyı açar. Sonra SelectCommand nesnesi ile SQL komutunu uygular. Elde edilen satırlarıda DataTable ya da DataSet nesnesine

yerleştirir ve tabi bağlantıyı kapatır.
Peki Update fonksiyonları ne yapmaktadır? İşte Update fonksiyonları DataTable içindeki tüm satırları ya da DataSet içindeki DataTable nesnelerinin tüm satırlarını tek tek inceler bu satırların RowState propertylerine bakar. Eğer RowState propertysi Deleted ise o satır için DeleteCommand isimli Command nesnesini, RowState Edit ise Insertcommand command nesnesini ve RowState Changed ise UpdateCommand nesnesini ExacuteNonQuery ile uygular. Örneğin Update fonksiyonu bir satırın "Edit" olduğunu gördüğünde o satırı gerçekten veritabanına eklemek için InsertCommand ile belirtilen komutu uygular. Update fonksiyonu benzer biçimde bir satırın Deleted olduğunu gördüğünde o satırı gerçekten silmek için DeleteCommand nesnesi ile belirtilen SQL komutunu uygular.

Update fonksiyonları yalnızca belirtilen command nesnelerini uygular. Bu command nesnelerinin oluşturularak ilgili propertylere set edilmiş olması gerekir. Bu da programcının bu işlemleri yerine getirecek SQL komutlarını hazırlamış olmasını gerektirir.

Ado.net içinde ilgili command nesnelerini otomatik oluşturmak içinde mekanizmalar vardır. Programcı mekanizmaları kullanabilir. Fakat bu nesnelerin manuel oluşturulması esneklik sağlamaktadır.

Command Nesnelerinin Otomatik Oluşturulması: Command nesnelerinin otomatik oluşturulması oldukça kolaydır. Tek yapılacak şey bir YYYCommandBuildir nesnesi oluşturmak ve bu nesneyi YYYDataAdapter nesnesi ile ilişkilendirmektir. YYYCommandBuildir sınıfının YYYDataAdapter isimli başlangıç fonksiyonu bunun için kullanılabilir. Ya da nesne sınıfın default başlangıç fonksiyonuyla oluşturulup YYYDataAdapter nesnesi sınıfın DataAdapter property sine girilebilir. Görüldüğü gibi YYYCommandBuilder sınıfı yalnızca YYYDataAdapter nesnesini kullanmaktadır.

Örneğin otomatik komut üretmesi şöyle yapılabilir.
AddRecordsForm.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace CSD
{
   public partial class AddRecordForm : Form
   {
      public AddRecordForm()
      {
         InitializeComponent();
      }

      private void button1_Click(object sender, EventArgs e)
      {
         try
         {
            int val = int.Parse(textBox2.Text);
```

```
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
                textBox2.Focus();
                textBox2.SelectAll();
                return;
            }

            this.DialogResult = DialogResult.OK;
        }

        private void button2_Click(object sender, EventArgs e)
        {
            this.DialogResult = DialogResult.Cancel;
        }

        public string PersonName
        {
            get { return textBox1.Text; }
        }

        public int PersonNo
        {
            get
            {
                return int.Parse(textBox2.Text);
            }
        }

    }
}
```

DataBaseMainForm.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data;
using System.Data.OleDb;

namespace CSD
{
    public partial class Form1 : Form
    {
        private OleDbConnection m_dbConnection;

        public Form1()
        {
            InitializeComponent();
```

```
    }

    private void addRecordToolStripMenuItem_Click(object sender, EventArgs e)
    {
        AddRecordForm arf = new AddRecordForm();

        if (arf.ShowDialog() == DialogResult.OK)
        {

            OleDbCommand dbCommand = new OleDbCommand();
            dbCommand.Connection = m_dbConnection;
            dbCommand.CommandText = "INSERT INTO Info(PersonName, PersonNo) VALUES(?,
?)";
            dbCommand.Parameters.Add("Name", arf.PersonName);
            dbCommand.Parameters.Add("No", arf.PersonNo.ToString());

            try
            {
                m_dbConnection.Open();

                dbCommand.ExecuteNonQuery();

                m_dbConnection.Close();

            }

            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }



        }
    }

    private void Form1_Load(object sender, EventArgs e)
    {
        m_dbConnection = new OleDbConnection();
        m_dbConnection.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0; " +
            @"Data Source=D:\Ders\IlerCSharp\DataBaseRecordProc8\Person.mdb";
    }

    private void deleteRecordToolStripMenuItem_Click(object sender, EventArgs e)
    {
        DeleteRecordForm drf = new DeleteRecordForm();

        if (drf.ShowDialog() == DialogResult.OK)
        {
            OleDbCommand dbCommand = new OleDbCommand();
            dbCommand.Connection = m_dbConnection;
```

```
            dbCommand.CommandText = "DELETE FROM Info WHERE PersonName = '" +
               drf.PersonName + "' AND PersonNo = " + drf.PersonNo.ToString();

            try
            {
               m_dbConnection.Open();
               int result = dbCommand.ExecuteNonQuery();
               m_dbConnection.Close();

               if (result == 0) {
                  MessageBox.Show("Cannot find record");
                  return;
               }
            }
            catch (Exception ex)
            {
               MessageBox.Show(ex.Message);
            }
         }
      }

      private void findToolStripMenuItem_Click(object sender, EventArgs e)
      {
          FindRecordForm frf = new FindRecordForm(m_dbConnection);

         if (frf.ShowDialog() == DialogResult.OK)
         {


         }
      }
   }
}
DeleteRecordsForm.cs
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace CSD
{
   public partial class DeleteRecordForm : Form
   {
      public DeleteRecordForm()
      {
         InitializeComponent();
      }
```

```
        private void button1_Click(object sender, EventArgs e)
        {
            try
            {
                int val = int.Parse(textBox2.Text);
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
                textBox2.Focus();
                textBox2.SelectAll();
                return;
            }

            this.DialogResult = DialogResult.OK;
        }

        private void button2_Click(object sender, EventArgs e)
        {
            this.DialogResult = DialogResult.Cancel;
        }

        public string PersonName
        {
            get { return textBox1.Text; }
        }

        public int PersonNo
        {
            get
            {
                return int.Parse(textBox2.Text);
            }
        }

    }
}
```

FindRecordsForm.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.OleDb;

namespace CSD
{
    public partial class FindRecordForm : Form
    {
```

```
private OleDbConnection m_connection;
private DataTable m_dt;

public FindRecordForm()
{
   InitializeComponent();
}

public FindRecordForm(OleDbConnection dbConnection) : this()
{
   m_connection = dbConnection;
}

private void button1_Click(object sender, EventArgs e)
{

   try
   {
      OleDbCommand dbCommand = new OleDbCommand(
            "SELECT * FROM Info WHERE " +
            textBox1.Text, m_connection);
      OleDbDataAdapter da = new OleDbDataAdapter(dbCommand);
      m_dt = new DataTable();
      da.Fill(m_dt);

      dataGridView1.DataSource = m_dt;

   }
   catch (Exception ex)
   {
      MessageBox.Show(ex.Message);
   }
}

private void button2_Click(object sender, EventArgs e)
{
   this.DialogResult = DialogResult.OK;
}

private void button3_Click(object sender, EventArgs e)
{
   try
   {
      OleDbCommand dbCommand = new OleDbCommand();
      dbCommand.CommandText = "SELECT * FROM Info WHERE " +
            textBox1.Text;
      dbCommand.Connection = m_connection;
      OleDbDataAdapter da = new OleDbDataAdapter(dbCommand);
      OleDbCommandBuilder cb = new OleDbCommandBuilder(da);

      //...
      da.Update(m_dt);
```

```
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }
  }
}
```

YYYCommandBuilder nesnesinin sınıfın veri elemanı biçimnde bildirilmesi gerekmez. Çünkü bu nesne yaratıldığında kendi referansını Event mekanizması için YYYDataAdapter nesnesine yazdığı için çöp toplayıcı nesneyi toplamaz.

Anahtar Notlar: Pekçok diyalog penceresinde Enter tuşuna basıldığında tipik bazı işlemler yapılır. Yani Enter tuşu basıldığındaçoğu kez programcı mevcut bir düğmeye basılmış etkisi istemektedir. Bu durum manuel olarak sağlanabilir. Örneğin KeyDown mesajları işlenebilir. Ancak bu temayla çok sık karşılaşıldığı için basit bir yöntem de sunulmuştur. Form sınıfının AcceptButton isimli property elemanı bu amaçla kullanılır. Benzer biçimde Esc tuşu için CancelButton propertysi de vardır.

Command Nesnelerinin Manuel Oluşturulması: Command nesnelerinin manuel üretilmesi mümkündür. Burada command nesneleri ayrı ayrı ele alınacaktır.

Insert Command Nesnesinin Manuel Üretilmesi: Programcı InsertCommand nesnesini oluşturuken komut yazısını parametrik bir biçimde(Örneğin OleDb tedarikçisi sözkonusu ise “?” ile)oluşturur. Tabi Update fonksiyonu ilgili satırın durumunu Edit gördüğünde bu komutu uygulamadan önce komut parametrelerini yerleştirmelidir. Komut parametrelerinin nasıl yerleştirileceği yani satırın hangi elemanları olarak yerleştirileceğini programcı belirler.

Programcı InsertCommand nesnesini şöyle oluşturur.

1. YYYCommand nesnesi yaratarak Insert Into sql komutunu CommandText biçiminde girer.
2. InsertInto komutundaki bilgiler parametrik olmalıdır. Programcı her parametre için bir YYY parameter nesnesi oluşturur. Bunu YYYCommand nesnesinin Parameter isimli Collectionına atar. Programcı parametrenin Value değerini set etmez fakat satırın hangi sütununa ve satırın hangi versiyonuna ilişkin olduğunu belirtir. Bunun için SourceColumn ve SourceVersion propertylerini kullanır.
3. YYYDataAdapter sınıfının Update fonksiyonu uygulandığında Update fonksiyonu satırı Edit gördüğünde InsertCommand nesnesini kullanarak ExacuteNonQuery yapmak ister. Fakat önce komutun parametrelerini yerleştirmelidir. Bunun için Nesnenin Parameters Collectionını inceler. Her bir YYYParameter nesnesinin value değerini o nesnede belirtildiği biçimde ilgili satırdan alarak set eder.

Örneğin bir InsertCommand nesnesi üreten fonksiyon şöyle yazılabilir.

DataBaseMainForm

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
```

```
using System.Data;
using System.Data.OleDb;

namespace CSD
{
   public partial class Form1 : Form
   {
      private OleDbConnection m_dbConnection;

      public Form1()
      {
         InitializeComponent();

      }

      private void addRecordToolStripMenuItem_Click(object sender, EventArgs e)
      {
         AddRecordForm arf = new AddRecordForm();

         if (arf.ShowDialog() == DialogResult.OK)
         {

            OleDbCommand dbCommand = new OleDbCommand();
            dbCommand.Connection = m_dbConnection;
            dbCommand.CommandText = "INSERT INTO Info(PersonName, PersonNo) VALUES(?,
?)";
            dbCommand.Parameters.Add("Name", arf.PersonName);
            dbCommand.Parameters.Add("No", arf.PersonNo.ToString());

            try
            {
               m_dbConnection.Open();

               dbCommand.ExecuteNonQuery();

               m_dbConnection.Close();

            }

            catch (Exception ex)
            {
               MessageBox.Show(ex.Message);
            }



         }
      }

      private void Form1_Load(object sender, EventArgs e)
      {
         m_dbConnection = new OleDbConnection();
```

```
            m_dbConnection.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0; " +
                @"Data Source=E:\DotNetAppBasic\DataBaseRecordProc\Person.mdb";
        }

        private void deleteRecordToolStripMenuItem_Click(object sender, EventArgs e)
        {
            DeleteRecordForm drf = new DeleteRecordForm();

            if (drf.ShowDialog() == DialogResult.OK)
            {
                OleDbCommand dbCommand = new OleDbCommand();
                dbCommand.Connection = m_dbConnection;
                dbCommand.CommandText = "DELETE FROM Info WHERE PersonName = '" +
                    drf.PersonName + "' AND PersonNo = " + drf.PersonNo.ToString();

                try
                {
                    m_dbConnection.Open();
                    int result = dbCommand.ExecuteNonQuery();
                    m_dbConnection.Close();

                    if (result == 0) {
                        MessageBox.Show("Cannot find record");
                        return;
                    }
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message);
                }
            }
        }

        private void findToolStripMenuItem_Click(object sender, EventArgs e)
        {
             FindRecordForm frf = new FindRecordForm(m_dbConnection);

            if (frf.ShowDialog() == DialogResult.OK)
            {


            }
        }
    }
}
```

FindRecordForm.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
```

```
using System.Windows.Forms;
using System.Data.OleDb;

namespace CSD
{
   public partial class FindRecordForm : Form
   {
      private OleDbConnection m_connection;
      private DataTable m_dt;

      public FindRecordForm()
      {
         InitializeComponent();
      }

      public FindRecordForm(OleDbConnection dbConnection) : this()
      {
         m_connection = dbConnection;
      }

      private void button1_Click(object sender, EventArgs e)
      {

         try
         {
            OleDbCommand dbCommand = new OleDbCommand(
                  "SELECT * FROM Info WHERE " +
                  textBox1.Text, m_connection);
            OleDbDataAdapter da = new OleDbDataAdapter(dbCommand);
            m_dt = new DataTable();
            da.Fill(m_dt);

            dataGridView1.DataSource = m_dt;

         }
         catch (Exception ex)
         {
            MessageBox.Show(ex.Message);
         }
      }

      private void button2_Click(object sender, EventArgs e)
      {
         this.DialogResult = DialogResult.OK;
      }

      private void button3_Click(object sender, EventArgs e)
      {
         try
         {
            OleDbCommand dbCommand = new OleDbCommand();
            dbCommand.CommandText = "SELECT * FROM Info WHERE " +
```

```
                textBox1.Text;
            dbCommand.Connection = m_connection;
            OleDbDataAdapter da = new OleDbDataAdapter(dbCommand);
            da.InsertCommand = CreateInsertCommand();

            //...
            da.Update(m_dt);
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }

    private OleDbCommand CreateInsertCommand()
    {
        OleDbCommand dbCommand = new OleDbCommand();
        dbCommand.Connection = m_connection;
        dbCommand.CommandText = "INSERT INTO Info(PersonName, PersonNo) VALUES(?,
?)";

        OleDbParameter dbParam;

        dbParam = new OleDbParameter(null, OleDbType.VarChar, 0, "PersonName");
        dbParam.SourceVersion = DataRowVersion.Current;
        dbCommand.Parameters.Add(dbParam);

        dbParam = new OleDbParameter(null, OleDbType.Integer, 0, "PersonNo");
        dbParam.SourceVersion = DataRowVersion.Current;
        dbCommand.Parameters.Add(dbParam);

        return dbCommand;
    }


  }
}
```

AddRecordForm.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace CSD
{
  public partial class AddRecordForm : Form
  {
```

```
        public AddRecordForm()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            try
            {
                int val = int.Parse(textBox2.Text);
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
                textBox2.Focus();
                textBox2.SelectAll();
                return;
            }

            this.DialogResult = DialogResult.OK;
        }

        private void button2_Click(object sender, EventArgs e)
        {
            this.DialogResult = DialogResult.Cancel;
        }

        public string PersonName
        {
            get { return textBox1.Text; }
        }

        public int PersonNo
        {
            get
            {
                return int.Parse(textBox2.Text);
            }
        }

    }
}
```

DeleteRecordForm.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
```

```
namespace CSD
{
   public partial class DeleteRecordForm : Form
   {
      public DeleteRecordForm()
      {
         InitializeComponent();
      }

      private void button1_Click(object sender, EventArgs e)
      {
         try
         {
            int val = int.Parse(textBox2.Text);
         }
         catch (Exception ex)
         {
            MessageBox.Show(ex.Message);
            textBox2.Focus();
            textBox2.SelectAll();
            return;
         }

         this.DialogResult = DialogResult.OK;
      }

      private void button2_Click(object sender, EventArgs e)
      {
         this.DialogResult = DialogResult.Cancel;
      }

      public string PersonName
      {
         get { return textBox1.Text; }
      }

      public int PersonNo
      {
         get
         {
            return int.Parse(textBox2.Text);
         }
      }

   }
}
```

DeleteCommand Nesnesinin Oluşturulması: DeleteCommand Nesnesi de InsertCommand nesnesi gibi oluşturulur. Programcının sql komutundaki WHERE koşuluna tüm sütunları parametre olarak kullnaması tavsiye edilir. Satır Deleted durumuna geldiğinde satırın Orginal versiyonu durmaktadır. DeleteCommand nesnesi üreten fonksiyon şöyle yazılabilir.

UpdateCommand Nesnesinin Oluşturulması: UpdateCommand nesnesinin oluşturulmasında daha fazla parametreye gereksinim duyulur. Çünkü satırın hem Orginal versiyonu hem de Current versiyonu kullanılır. UpdateCommand nesnesi üreten bir fonksiyon şöyle yazılabilir.

FindRecordForm.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.OleDb;

namespace CSD
{
   public partial class FindRecordForm : Form
   {
      private OleDbConnection m_connection;
      private DataTable m_dt;

      public FindRecordForm()
      {
         InitializeComponent();
      }

      public FindRecordForm(OleDbConnection dbConnection) : this()
      {
         m_connection = dbConnection;
      }

      private void button1_Click(object sender, EventArgs e)
      {

         try
         {
            OleDbCommand dbCommand = new OleDbCommand(
                  "SELECT * FROM Info WHERE " +
                  textBox1.Text, m_connection);
            OleDbDataAdapter da = new OleDbDataAdapter(dbCommand);
            m_dt = new DataTable();
            da.Fill(m_dt);

            dataGridView1.DataSource = m_dt;

         }
         catch (Exception ex)
         {
            MessageBox.Show(ex.Message);
         }
      }
```

```
    private void button2_Click(object sender, EventArgs e)
    {
        this.DialogResult = DialogResult.OK;
    }

    private void button3_Click(object sender, EventArgs e)
    {
        try
        {
            OleDbCommand dbCommand = new OleDbCommand();
            dbCommand.CommandText = "SELECT * FROM Info WHERE " +
                textBox1.Text;
            dbCommand.Connection = m_connection;
            OleDbDataAdapter da = new OleDbDataAdapter(dbCommand);
            da.InsertCommand = CreateInsertCommand();
            da.DeleteCommand = CreateDeleteCommand();
            da.UpdateCommand = CreateUpdateCommand();


            //...
            da.Update(m_dt);
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }

    private OleDbCommand CreateInsertCommand()
    {
        OleDbCommand dbCommand = new OleDbCommand();
        dbCommand.Connection = m_connection;
        dbCommand.CommandText = "INSERT INTO Info(PersonName, PersonNo) VALUES(?,
?)";

        OleDbParameter dbParam;

        dbParam = new OleDbParameter(null, OleDbType.VarChar, 0, "PersonName");
        dbParam.SourceVersion = DataRowVersion.Current;
        dbCommand.Parameters.Add(dbParam);

        dbParam = new OleDbParameter(null, OleDbType.Integer, 0, "PersonNo");
        dbParam.SourceVersion = DataRowVersion.Current;
        dbCommand.Parameters.Add(dbParam);

        return dbCommand;
    }

    private OleDbCommand CreateDeleteCommand()
    {
        OleDbCommand dbCommand = new OleDbCommand();
```

```
            dbCommand.Connection = m_connection;
            dbCommand.CommandText = "DELETE FROM Info WHERE PersonName = ? AND
PersonNo = ?";

            OleDbParameter dbParam;

            dbParam = new OleDbParameter(null, OleDbType.VarChar, 0, "PersonName");
            dbParam.SourceVersion = DataRowVersion.Original;
            dbCommand.Parameters.Add(dbParam);

            dbParam = new OleDbParameter(null, OleDbType.Integer, 0, "PersonNo");
            dbParam.SourceVersion = DataRowVersion.Original;
            dbCommand.Parameters.Add(dbParam);

            return dbCommand;
        }

        private OleDbCommand CreateUpdateCommand()
        {
            OleDbCommand dbCommand = new OleDbCommand();
            dbCommand.Connection = m_connection;
            dbCommand.CommandText =
              "UPDATE Info SET PersonName = ?, PersonNo = ? WHERE PersonName = ? AND
PersonNo = ?";

            OleDbParameter dbParam;

            dbParam = new OleDbParameter(null, OleDbType.VarChar, 0, "PersonName");
            dbParam.SourceVersion = DataRowVersion.Current;
            dbCommand.Parameters.Add(dbParam);

            dbParam = new OleDbParameter(null, OleDbType.Integer, 0, "PersonNo");
            dbParam.SourceVersion = DataRowVersion.Current;
            dbCommand.Parameters.Add(dbParam);

            dbParam = new OleDbParameter(null, OleDbType.VarChar, 0, "PersonName");
            dbParam.SourceVersion = DataRowVersion.Original;
            dbCommand.Parameters.Add(dbParam);

            dbParam = new OleDbParameter(null, OleDbType.Integer, 0, "PersonNo");
            dbParam.SourceVersion = DataRowVersion.Original;
            dbCommand.Parameters.Add(dbParam);

            return dbCommand;
        }


    }
}
DataBaseMainForm.cs
using System;
using System.Collections.Generic;
```

```
using System.ComponentModel;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data;
using System.Data.OleDb;

namespace CSD
{
   public partial class Form1 : Form
   {
      private OleDbConnection m_dbConnection;

      public Form1()
      {
         InitializeComponent();

      }

      private void addRecordToolStripMenuItem_Click(object sender, EventArgs e)
      {
         AddRecordForm arf = new AddRecordForm();

         if (arf.ShowDialog() == DialogResult.OK)
         {

            OleDbCommand dbCommand = new OleDbCommand();
            dbCommand.Connection = m_dbConnection;
            dbCommand.CommandText = "INSERT INTO Info(PersonName, PersonNo) VALUES(?,
?)";
            dbCommand.Parameters.Add("Name", arf.PersonName);
            dbCommand.Parameters.Add("No", arf.PersonNo.ToString());

            try
            {
               m_dbConnection.Open();

               dbCommand.ExecuteNonQuery();

               m_dbConnection.Close();

            }

            catch (Exception ex)
            {
               MessageBox.Show(ex.Message);
            }



         }
```

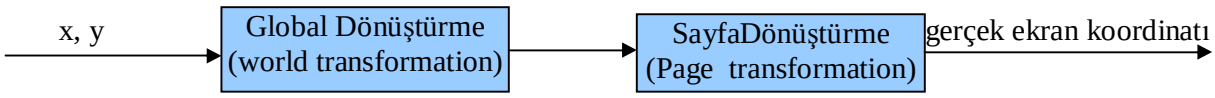
```
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            m_dbConnection = new OleDbConnection();
            m_dbConnection.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0; " +
                @"Data Source=E:\DotNetAppBasic\DataBaseRecordProc\Person.mdb";
        }

        private void deleteRecordToolStripMenuItem_Click(object sender, EventArgs e)
        {
            DeleteRecordForm drf = new DeleteRecordForm();

            if (drf.ShowDialog() == DialogResult.OK)
            {
                OleDbCommand dbCommand = new OleDbCommand();
                dbCommand.Connection = m_dbConnection;
                dbCommand.CommandText = "DELETE FROM Info WHERE PersonName = '" +
                    drf.PersonName + "' AND PersonNo = " + drf.PersonNo.ToString();

                try
                {
                    m_dbConnection.Open();
                    int result = dbCommand.ExecuteNonQuery();
                    m_dbConnection.Close();

                    if (result == 0) {
                        MessageBox.Show("Cannot find record");
                        return;
                    }
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message);
                }
            }
        }

        private void findToolStripMenuItem_Click(object sender, EventArgs e)
        {
             FindRecordForm frf = new FindRecordForm(m_dbConnection);

            if (frf.ShowDialog() == DialogResult.OK)
            {


            }
        }
    }
}
```

Bu durumda Update işlemide şöyle uygulanır.

```
OleDbCommand dbCommand = new OleDbCommand(
```

```
        "SELECT * FROM Info WHERE " +
        textBox1.Text, m_connection);
    OleDbDataAdapter da = new OleDbDataAdapter(dbCommand);
    m_dt = new DataTable();
    da.Fill(m_dt);

    dataGridView1.DataSource = m_dt;
```

Çizim İşlemlerinde Koordinat Dönüştürmesi: Çizim işlemlerinde default koordinat sistemindeki birim pixel cinsindendir. Orjin noktası çalışma alanının solüst köşesidir ve x ekseni sağa doğru y ekseni aşağıya doğru akmaktadır. Bu koordinat sistemi bazı uygulamalarda zorluklara yol açmaktadır. Çizim fonksiyonlarına girdiğimiz değerler, koordinat değerleri iki dönüştürmeye sokulduktan sonra gerçek piksel değerlerine dönüştürülmektedir.



Görüldüğü gibi çizim fonksiyonlarına girdiğimiz x, y değerleri önce global dönüştürmeye sokulmakta sonra bu dönüştürülmüş değerler sayfa dönüştürülmesine sokulmaktadır. Global dönüştürme üzerinde etkili olan fonksiyonlar TranslateTransform, RotateTransform ve ScaleTransform fonksiyonlarıdır. Sayfa dönüştürmesi üzerinde PageUnit ve PageScale propertyleri etkili olmaktadır.

Graphics sınıfının ScaleTransform fonksiyonunun parametrik yapısı şöyledir.
public void ScaleTransform(float sx, float sy)

Fonksiyonun parametreleri çizim fonksiyonlarına girdiğimiz değerlerin hangi değerlerle çarpılacağını belirtir. Örneğin eksenlerin yönlerini değiştirmek istersek -1 değerini verebiliriz.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace TransformSample
{
   public partial class Form1 : Form
   {
      private float m_scaleX, m_scaleY;

      public Form1()
      {
         InitializeComponent();
```

```
            m_scaleX = 1;
            m_scaleY = 1;
        }



        private void panel1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;

            g.TranslateTransform(100, 100);
            g.ScaleTransform(m_scaleX, m_scaleY);


            g.DrawRectangle(Pens.Red, 100, 100, 100, 100);
        }

        private void toolStripLabel1_Click(object sender, EventArgs e)
        {
            m_scaleX += 1;
            m_scaleY += 1;

            panel1.Invalidate();


        }

        private void toolStripLabel2_Click(object sender, EventArgs e)
        {
            if (m_scaleX > 1)
            {
                m_scaleX -= 1;
                m_scaleY -= 1;
            }

            panel1.Invalidate();
        }
    }
}
```

TranslateTransform fonksiyonu orjin noktasını kaydırmak için kullanılmaktadır.
public void TranslateTransform(float dx, float dy)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
```

```
namespace TransformSample
{
   public partial class Form1 : Form
   {
      private float m_scaleX, m_scaleY;

      public Form1()
      {
         InitializeComponent();


         m_scaleX = 10;
         m_scaleY = -10;
      }



      private void panel1_Paint(object sender, PaintEventArgs e)
      {
         Graphics g = e.Graphics;

         g.TranslateTransform(panel1.Width / 2, panel1.Height / 2);
         g.ScaleTransform(m_scaleX, m_scaleY);

         g.DrawLine(Pens.Red, 0, panel1.Height / 2, 0, -panel1.Height / 2);
         g.DrawLine(Pens.Red, panel1.Width / 2, 0, -panel1.Width / 2, 0);

         List<PointF> points = new List<PointF>();

         for (double x = -2 * Math.PI; x <= 2 * Math.PI; x += 0.1)
         {
            points.Add(new PointF((float) x, (float) Math.Sin(x)));
         }

         PointF [] pts = points.ToArray();

         g.DrawLines(Pens.Blue, pts);




      }

      private void toolStripLabel1_Click(object sender, EventArgs e)
      {
         m_scaleX += 1;
         m_scaleY -= 1;

         panel1.Invalidate();
```

```
        }

        private void toolStripLabel2_Click(object sender, EventArgs e)
        {
            if (m_scaleX > 1)
            {
                m_scaleX -= 1;
                m_scaleY += 1;
            }

            panel1.Invalidate();
        }
    }

    class MyPanel : Panel
    {
        public MyPanel()
        {
            this.ResizeRedraw = true;
        }
    }
}
```

Kalem gibi çizim nesneleri default olarak ScaleTransform işleminden etkilenmektedir. Örneğin biz ScaleTransform fonksiyonuyla çarpansal değerler verdiğimizde kalemin kalınlıkları da duruma göre artmaktadır. Bu durumda kalemin kalınlığı manuel olarak ayarlanabilir.

Pen sınıfının ScaleTransform ve RotateTransform fonksiyonları kaleme özgü transform değerleri alabilir. Böylece örneğin programcı global düzeyde ScaleTransform uyguladıktan sonra kalem nesnesi için yeniden transform fonksiyonu uygulayabilir.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace TransformSample
{
    public partial class Form1 : Form
    {
        private float m_scaleX, m_scaleY;
        private Pen m_bluePen;
        private Pen m_redPen;

        public Form1()
        {
            InitializeComponent();
```

```
    m_scaleX = 15;
    m_scaleY = -20;

    m_bluePen = new Pen(Brushes.Blue);
    m_redPen = new Pen(Brushes.Red);
}



private void panel1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;

    g.TranslateTransform(panel1.Width / 2, panel1.Height / 2);
    g.ScaleTransform(m_scaleX, m_scaleY);

    m_redPen.ScaleTransform(1 / m_scaleX, 1 / m_scaleY);
    g.DrawLine(m_redPen, 0, panel1.Height / 2, 0, -panel1.Height / 2);
    g.DrawLine(m_redPen, panel1.Width / 2, 0, -panel1.Width / 2, 0);

    List<PointF> points = new List<PointF>();

    for (double x = -2 * Math.PI; x <= 2 * Math.PI; x += 0.1)
    {
        points.Add(new PointF((float) x, (float) Math.Sin(x)));
    }

    PointF [] pts = points.ToArray();

    m_bluePen.ScaleTransform(1 / m_scaleX, 1 / m_scaleY);
    g.DrawLines(m_bluePen, pts);




}

private void toolStripLabel1_Click(object sender, EventArgs e)
{
    m_scaleX += 1;
    m_scaleY -= 1;

    panel1.Invalidate();


}

private void toolStripLabel2_Click(object sender, EventArgs e)
{
    if (m_scaleX > 1)
    {
        m_scaleX -= 1;
```

```
            m_scaleY += 1;
        }

        panel1.Invalidate();
    }
  }

  class MyPanel : Panel
  {
    public MyPanel()
    {
      this.ResizeRedraw = true;
    }
  }
}
```

Pen sınıfının ResetTransform fonksiyonu kalem üzerine uygulanan Transform işlemlerini reset eder.

RotateTransform fonksiyonu eksen döndürmesi yapar. Fonksiyonun parametresi derece cinsinden açı almaktadır.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace TransformSample
{
  public partial class Form1 : Form
  {
    private float m_scaleX, m_scaleY;
    private Pen m_bluePen;
    private Pen m_redPen;
    private float m_rotation;

    public Form1()
    {
      InitializeComponent();

      m_scaleX = 15;
      m_scaleY = -20;
      m_rotation = 0;

      m_bluePen = new Pen(Brushes.Blue);
      m_redPen = new Pen(Brushes.Red);
    }
```

```
private void panel1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;

    g.RotateTransform(m_rotation);
    g.TranslateTransform(panel1.Width / 2, panel1.Height / 2);
    g.ScaleTransform(m_scaleX, m_scaleY);

    m_redPen.ScaleTransform(1 / m_scaleX, 1 / m_scaleY);
    g.DrawLine(m_redPen, 0, panel1.Height / 2, 0, -panel1.Height / 2);
    g.DrawLine(m_redPen, panel1.Width / 2, 0, -panel1.Width / 2, 0);

    List<PointF> points = new List<PointF>();

    for (double x = -2 * Math.PI; x <= 2 * Math.PI; x += 0.1)
    {
        points.Add(new PointF((float) x, (float) Math.Sin(x)));
    }

    PointF [] pts = points.ToArray();


    m_bluePen.ScaleTransform(1 / m_scaleX, 1 / m_scaleY);
    g.DrawLines(m_bluePen, pts);




}

private void toolStripLabel1_Click(object sender, EventArgs e)
{
    m_scaleX += 1;
    m_scaleY -= 1;

    panel1.Invalidate();


}

private void toolStripLabel2_Click(object sender, EventArgs e)
{
    if (m_scaleX > 1)
    {
        m_scaleX -= 1;
        m_scaleY += 1;
    }

    panel1.Invalidate();
}
```

```
        private void toolStripLabel3_Click(object sender, EventArgs e)
        {
            m_rotation += 10;

            panel1.Invalidate();
        }
    }

    class MyPanel : Panel
    {
        public MyPanel()
        {
            this.ResizeRedraw = true;
        }
    }
}
```

Eğik yazı yazdırma işlemleri bu fonksiyon sayesinde yapılabilmektedir.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace TransformSample
{
    public partial class Form1 : Form
    {
        private float m_scaleX, m_scaleY;
        private Pen m_bluePen;
        private Pen m_redPen;
        private float m_rotation;

        public Form1()
        {
            InitializeComponent();

            m_scaleX = 15;
            m_scaleY = -20;
            m_rotation = 0;

            m_bluePen = new Pen(Brushes.Blue);
            m_redPen = new Pen(Brushes.Red);
        }
```

```
private void panel1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;

    g.RotateTransform(m_rotation);

    StringFormat sf = new StringFormat();
    sf.Alignment = StringAlignment.Center;
    sf.LineAlignment = StringAlignment.Center;
    g.DrawString("This is a test", this.Font, Brushes.Red, panel1.ClientRectangle, sf);

    //g.TranslateTransform(panel1.Width / 2, panel1.Height / 2);
    //g.ScaleTransform(m_scaleX, m_scaleY);
    //m_redPen.ScaleTransform(1 / m_scaleX, 1 / m_scaleY);
    //g.DrawLine(m_redPen, 0, panel1.Height / 2, 0, -panel1.Height / 2);
    //g.DrawLine(m_redPen, panel1.Width / 2, 0, -panel1.Width / 2, 0);

    //List<PointF> points = new List<PointF>();

    //for (double x = -2 * Math.PI; x <= 2 * Math.PI; x += 0.1)
    //{
    //    points.Add(new PointF((float) x, (float) Math.Sin(x)));
    //}

    //PointF [] pts = points.ToArray();


    //m_bluePen.ScaleTransform(1 / m_scaleX, 1 / m_scaleY);
    //g.DrawLines(m_bluePen, pts);







}

private void toolStripLabel1_Click(object sender, EventArgs e)
{
    m_scaleX += 1;
    m_scaleY -= 1;

    panel1.Invalidate();


}

private void toolStripLabel2_Click(object sender, EventArgs e)
{
    if (m_scaleX > 1)
```

```
            {
               m_scaleX -= 1;
               m_scaleY += 1;
            }

            panel1.Invalidate();
         }

         private void toolStripLabel3_Click(object sender, EventArgs e)
         {
            m_rotation += 10;

            panel1.Invalidate();
         }
      }

      class MyPanel : Panel
      {
         public MyPanel()
         {
            this.ResizeRedraw = true;
         }
      }
   }
```

Ardışıl yapılan Scale ve Rotate işlemleri kümülatif etki yaratmaktadır. Örneğin biz iki kez üst üste 10 ar derecelik RotateTransform uygulamış olalım. Toplam 20 derecelik bir döndürme işlemi yapılmış olur. Benzer biçimde biz önce 2 çarpansal değeriyle sonra 4 çarpansal değeriyle ScaleTransform işlemi yapmış olalım. Toplamda 8 çarpansal değere karşılık gelmektedir.

Eksen döndürmeleri matriksel biçimde ***transformasyon matriksi*** verilerekte yapılabilir. (Programing Microsoft Windows with C# MS Press)

Global dönüştürmeden elde edilen değerler sayfa dönüştürmesine sokulur. Graphics sınıfının PageUnit isimli property elemanı GraphicsUnit isimli enum türündendir. Bu enum türünün pixel, inch, milimeter gibi elemanları vardır. Bu property global dönüştürme sonucunda elde edilen değerin birimini belirlemektedir. Örneğin biz 0,0 koordinatından 100, 100 koordinatına bir doğru çizmiş olalım. ScaleTransform değeri +2, +2 biçiminde olsun. Bu durumda global dönüştürme sonucunda 0, 0 ve 200, 200 değerleri elde edilir. Buradaki 0,0 ve 200, 200 ün birimi Sayfa dönüştürmesi aşamasında PageUnit propertysine bakılarak belirlenmektedir. Default PageUnit pixel durumundadır. Örneğin biz onu milimeter biçimine çekersek buradaki 200, 200 değeri 200 mm anlamına gelir.

Eğer çizim işlemleri yazıcı için yapılıyorsa mm ve inch büyüklükleri tamamen gerçekçi biçimde oluşturulur. Yani biz 10mm bir çizimi yazıcıya yapsak cetvelle ölçtüğümüzde gerçekten 10 mm çıkmaktadır. Peki ekrandaki durum nasıl olacaktır.? Ekran büyüklükleri değiştiği için pixel boyutlarını önceden bilmek çok zordur. Bu durumda bu mm ve inch gibi değerler ekran sözkonusu olduğunda göreli değerler olur. Sistemdeki default font un punto büyüklüğü etkili olmaktadır. Vista sistemlerinde default font 8 puntodur. 1 punto 1/72 inch dir. 72 punto 1 inch yapmaktadır. Bu durumda biz ekranda PageUnit değerini inch e çekersek ve 1 inch uzunluğunda bir doğru çizmeye çalışırsak 1 inch de 72 pixel olduğu için ve sistemdeki default font 8 punto olduğu için 1 inchlik

uzunluk 72*8 pixele karşılık gelecektir. mm değeride inch değerine göre ayarlanmıştır.

Grafik sınıfının PageScale isimli property elemanı bir birimin kaç PageUnit değerine ilişkin olduğunu belirtir. Örneğin normal dönüştürme sonrasında 100 değeri elde edilmiş olsun. PageUnit property sinin mm olduğunu düşünelim. PageScale de 2 olsun. İşte global dönüştürme sırasında elde edilen 100 değeri 200mm anlamına gelecektir. *public float PageScale{ get; set}*

PageScale propertysinin tipik kullanımı şöyledir. Örneğin biz 0-10000 aralığında değerler elde edelim. Fakat bunu 300 lük bir çözünürlük içinde yorumlamak isteyelim. Bu durumda PageScale değerini 300/10000 yaparız. Böylece 0 ile 10000 arasındaki değeri çizimde kullanabiliriz. Bu property nin default değeri 1 biçimindedir.

ImageList Sınıfı: ImageList sınıfı aynı boyutta resimleri tutan bir collection sınıftır. .net te pek çok karmaşık kontrol ImageList sınıfını kullanmaktadır. ImageList sınıfı türünden bir nesne sınıfın default başlangıç fonksiyonuyla yaratılabilir. ImageList sınıfının Images isimli collection elemanı ImageList.ImageCollection isimli collection bir sınıf türündendir ve resimleri tutar. ImageList sınıfının ImageSize isimli property elemanı Size türdendir ve resimlerin boyutunu belirtmektedir. Collection a atanan tüm resimler otomatik olarak bu boyuta gerektirir. Bu propertynin default değeri 16 *16 dır. Maksimum değeri 256 *256 olabilir.

Sınıfın ColorDepth isimli property elemanı ColorDepth isimli enum türündendir ve resimlerin renk çözünürlüğünü belirtmektedir. Default 8 bit renk çözünürlüğündedir. Sınıfın Draw isimli fonksiyonları doğrudan graphics nesnesi alarak resmi çizer.

Anahtar Notlar: .net içinde Managed exe ve dll ler içindeki kaynakları tek tek alıp save edebilecek bir program yazabiliriz. Fakat Unmanaged exe ve dll lerdeki kaynakları API fonksiyonlarını çağırmadan elde edebilecek bir mekanizma yoktur. Ancak Unmanaged exe ve dll ler içindeki kaynakları alıp bunu birer dosya biçiminde diske saklayan pek çok basit yazılımı bedava biçimde internette bulunabilmektedir.

TreeView Kontrolü: Bu kontrol altlık üstlük ilişkisi içinde olan ağ aç yapılarını görüntülemek amacıyla kullanılır. TreeView sınınıfıyla temsil edilmektedir. Nesne sınıfın Default başlangıç fonksiyonuyla yaratılabilir. Control deki her elemana bir düğüm denir. Düğümler de TreeNode isimli bir sınıfla temsil edilmiştir.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace ImageListSample
{
   public partial class Form1 : Form
   {
      ImageList m_il;

      public Form1()
      {
```

```
            InitializeComponent();

            m_il = new ImageList();
            m_il.ImageSize = new Size(64, 64);
            m_il.ColorDepth = ColorDepth.Depth24Bit;
        }

        private void button1_Click(object sender, EventArgs e)
        {
            try
            {
                foreach (string file in Directory.GetFiles(
                        @":\Users\pc3\Desktop\ResourceExtract\ChessFigures", "*.bmp",
                        SearchOption.TopDirectoryOnly))
                {
                    m_il.Images.Add(new Bitmap(file));
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }

            this.Invalidate();
        }

        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;

            if (m_il.Images.Count == 0)
                return;


            int x , y;


            for (int i = 0; i < 10; ++i)
            {
                y = i * m_il.ImageSize.Height;

                x = 0;
                for (int k = 0; k < 10; ++k)
                {
                    x = k * m_il.ImageSize.Width;

                    m_il.Draw(g, new Point(x, y), 100+ i * 10 + k);
                }
            }
        }
    }
}
```

TreeView sınıfının Nodes isimli collection elemanı kök elemanları tutmak için kullanılır. TreeNode sınıfının Nodes isimli collection elemanı ise TreeNode nesnelerini tutan bir collectiondır. Bu collection eleman düğümün alt düğümlerini tutmaktadır. Yani TreeView kontrolü düğümlerden düğümlerde alt düğümlerden oluşmaktadır. TreeNode sınıfının Parent isimli elemanı üst düğümün TreeNode referansını Text isimli property elemanı düğümün yazısal ismini tutmaktadır. Düğümün tüm alt düğümleri ChildNode collection elemanı ile elde edilebilir.

Tıpkı bazı kontrollerde olduğu gibi TreeNode sınıfının Nodes isimli collection elemanının ilişkin olduğu TreeNode collection sınıfının String parametreli add fonksiyonları kendi içinde TreeNode nesnesini yaratıp eklemeyi yapmaktadır.

TreeView sınıfının SelectedNode isimli property elemanı o anda seçili olan TreeNode elemanını vermektedir. Eğer hiçbir eleman seçili değilse null referans elde edilir.

TreeNode sınıfının string türünde FullPath isimli property elemanı düğümün yerini tersbölü karekterleriyle bize vermektedir. Yine TreeNode sınıfının ImageIndex isimli property elemanı o düğüm için görüntülenecek resmin ImageList içindeki indeks numarasını belirtir.

TreeView sınıfının ImageList isimli property elemanı tüm düğümler için resimleri tutan ImageList nesnesi almaktadır. Yani programcı tüm düğümlerin resimlerini bir ImageList e yerleştirmeli sonra TreeNode nesnesinin ImageIndex propertysi ile resmi belirlemelidir. Programcı tüm ağacı işin başında özyinelemeli olarak oluşturabilir. Fakat bazı durumlarda tüm ağacın oluşturulması zaman kaybına yol açabilmektedir. Örneğin dizin ağacının TreeView kontrolünde gösterilmesi verilebilir. Bu durumda ileride ele alınacağı gibi gösterim Exspend mesajlarında ayrı ayrı yapılabilir.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace TreeViewSample
{
   public partial class Form1 : Form
   {
      private TreeView m_tv;

      public Form1()
      {
         InitializeComponent();

         m_tv = new TreeView();
         m_tv.Size = new Size(300, 300);



         m_tv.Parent = this;
      }
```

```
        private void button1_Click(object sender, EventArgs e)
        {
            foreach (string drive in Directory.GetLogicalDrives())
            {
                TreeNode tn = new TreeNode(drive);
                m_tv.Nodes.Add(tn);
                AddNode(tn);
            }
        }

        private void AddNode(TreeNode tn)
        {
            string path = tn.FullPath;

            foreach (string dir in Directory.GetDirectories(tn.FullPath))
            {
                try
                {
                    TreeNode stn = new TreeNode(Path.GetFileName(dir));
                    tn.Nodes.Add(stn);
                    AddNode(stn);
                }
                catch (Exception ex)
                {
                }
            }
        }
    }
}
```

TreeView sınıfının BeforeExpand isimli event elemanı alt düğümleri olan düğüme tıklandığında açım işlemi yapılmadan önce tetiklenir. Benzer biçimde AfterExpand açım işlemi yapıldıktan sonra tetiklenmektedir. Benzer biçimde bir alt ağacı kapatırken de BeforeCollapse ve AfterCollapse event elemanları da vardır.

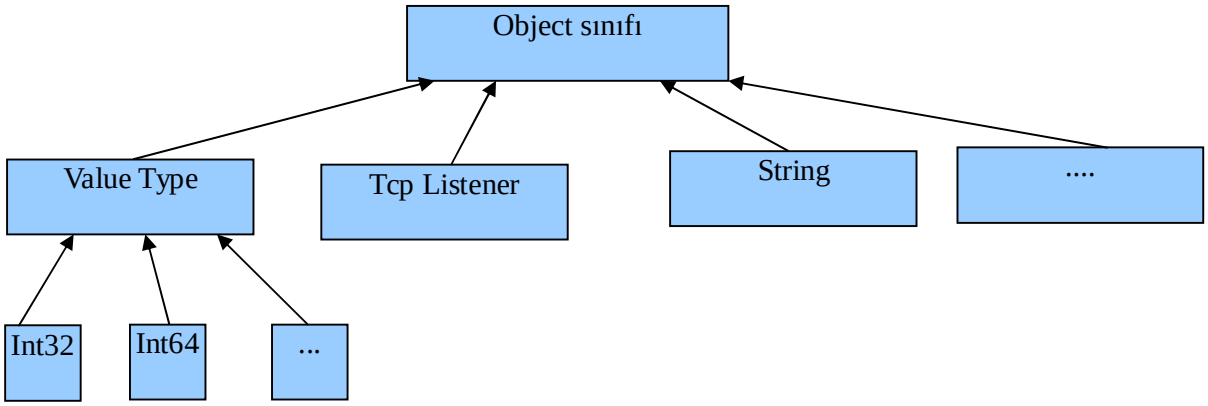
Örneğin biz daha etkin bir biçimde dizin ağacını TreeView da görüntülemek isteyelim bunun için işin başında her sürücüdeki kök dizindeki dizinler bulunur ve ağaca eklenir. Bu durumda kökteki sürücülerde + sembolü çıkar. Programcı BeforeExpand mesajını işleyerek ilgili dizine ilişkin alt düğme tıklandığında o dizinin içindeki dizinlerin altdizinlerini bularak ağaca ekler.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace TreeViewSample
```

```
{
   public partial class Form1 : Form
   {
      private TreeView m_tv;

      public Form1()
      {
         InitializeComponent();

         m_tv = new TreeView();
         m_tv.Size = new Size(300, 300);
         m_tv.BeforeExpand += new TreeViewCancelEventHandler(m_tv_BeforeExpand);




         m_tv.Parent = this;
      }

      void m_tv_BeforeExpand(object sender, TreeViewCancelEventArgs e)
      {
         foreach (TreeNode tn in e.Node.Nodes)
            AddNode(tn);
      }

      private void button1_Click(object sender, EventArgs e)
      {

         foreach (string drive in Directory.GetLogicalDrives())
         {
            try
            {
               TreeNode tn = new TreeNode(drive);
               m_tv.Nodes.Add(tn);
               AddNode(tn);
            }
            catch (Exception ex)
            {
            }
         }
      }

      private void AddNode(TreeNode tn)
      {
         string path = tn.FullPath;

         if (tn.Nodes.Count != 0)
            return;

         try
         {
```

```
            foreach (string dir in Directory.GetDirectories(tn.FullPath))
            {
               try
               {
                  TreeNode stn = new TreeNode(Path.GetFileName(dir));
                  tn.Nodes.Add(stn);
               }
               catch (Exception ex)
               {
               }
            }
         }
         catch (Exception ex)
         {
         }
      }
   }
}
```

Genericler: Yazılım mühendisliğinde türden bağımsız işlemlerin yapılabilmesi hep bir sorun biçiminde algılanmış ve de çeşitli çözümler üretilmeye çalışılmıştır. Sözgelimi Eyfel ve Ada dilleri Bu dillerde generticler konusu adreslenmeye çalışılmıştır. Zira C++ daysa template dediğimiz konu başlığı bu problemin adreslenmesine yöneliktir. C# da ise 2.0 versiyonuna kadar dilin sintaksına entegreedilmiş bir çözüm bulunmamaktaydı. 2005 de çıkan 2.0 versiyonunda ise Generic ismiyle bu problem adreslenmeye çalışılmıştır. C# 2.0 dan önce bu problem .net sınıf sisteminin kendi iç dinamiklerine bırakılmış ve de bu sınıf sisteminin mutlak taban sınıfı durumundaki object sınıfının tür uyuşum özelliği ile giderilmeye çalışılmıştır.



Ancak bu tekniğin yarattığı bir takım anomaliler ve komplikasyonlar söz konusudur. Bu noktadan sonra odağında bu mekanizmanın yer aldığı yapıları inceleyip sotunları ortaya çıkaracak ve de genericlerle bu sorunların çözümünü göreceğiz.
Klasik stack uygulaması türden bağımsız olarak object nesnesi kullanılarak aşağıdaki gibi yapılabilir.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
namespace MyStack
{
   class MyStack
   {
      private object[] arr;
      private int cursor = 0;

      public MyStack(int size)
      {
         arr = new Object[size];
         cursor = 0;
      }

      public void Push(object val)
      {
         arr[cursor++] = val;
      }

      public object Pop()
      {
         return arr[--cursor];
      }
   }



   class Program
   {
      static void Main(string[] args)
      {
         MyStack stk = new MyStack(10);

         stk.Push("aykut");

         string s = (string) stk.Pop();

         Console.WriteLine(s);

      }
   }
}
```

Bu örnekte nesne implicit olarak alınan herhangi bir tip neticede object e dönüşüyor. Bu da geri planda ekstra işlemler yapıyor.

Aşağıdaki gibi değer türüne ilişkin bir değeri stackte int push edersek burada int türü obje türüne dönüşüyor. İnt bir değer tür ve stackte yer alır. Ama objeler nesne yani obje türünden olmasından dolayı heapte bir yerde tututluyor bu dönüşüm boxing(kutulama) olarak biliniyor. Bu durumda hem tür dönüşümü hem bellekte yer değişişimi söz konusu tersi durumda(pop) unboxing gerçekleşiyor. Obje türünden değer türüne bir dönüşüm oluyor vebellek adresleri de değişiyor. Heapten Stake bir yer değişimi... Bu durumda CPU için fazladan işlem yükü getiriyor.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace MyStack
{
   class MyStack
   {
      private object[] arr;
      private int cursor = 0;

      public MyStack(int size)
      {
         arr = new Object[size];
         cursor = 0;
      }

      public void Push(object val)
      {
         arr[cursor++] = val;
      }

      public object Pop()
      {
         return arr[--cursor];
      }
   }

   class Submarine
   {



   }

   class Program
   {
      static void Main(string[] args)
      {
         MyStack stk = new MyStack(10);

         int i = 9;

         stk.Push(i);

         int s = (int) stk.Pop();

         Console.WriteLine(s);

      }
   }
```

```
}
```

Submarine türünden bir nesne yaratıp bunu object türüne imlicit dönüştürdüğümüzde bir boxing söz konusu değil ikiside referans türüne ilişkin fakat unboxsing durumunda aşağıdaki örnekte olduğu gibi pop işleminde nesneyi stringe dönüştürüp bir string nesnesine atadığımızda pop submarine verdiğini bildiğim halde bunu string e dönüştürüp bunu bir string nesnesine atadığımızda bu açık hata derleme zamanında hata vermiyor fakat runtime da bu exceptiona yol açıyor ve program çöküyor. Bu bir programcının yapmaması gerekn bir hata.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace MyStack
{
   class MyStack
   {
      private object[] arr;
      private int cursor = 0;

      public MyStack(int size)
      {
         arr = new Object[size];
         cursor = 0;
      }

      public void Push(object val)
      {
         arr[cursor++] = val;
      }

      public object Pop()
      {
         return arr[--cursor];
      }
   }

   class Submarine : Object
   {
      private string name;

      public string Name
      {
         get { return name; }
         set { name = value; }
      }
      private string no;

      public string No
      {
```

```
            get { return no; }
            set { no = value; }
        }

        public Submarine(string name, string no)
        {
            this.name = name;
            this.no = no;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            MyStack stk = new MyStack(10);

            Submarine sub = new Submarine("Burakreis", "S-356");

            stk.Push(sub);

            string s = (string) stk.Pop();


        }
    }
}
```

Bu yapının getirdiği problemler kısa olarak şöyle anlatılabilir. Bu tarzda tasarlanmış stack sistemine referans türe ilişkin elemanlar push edildiğinde ilgili türden obje türüne otomatik tür dönüştürmesi oluşmakta bu da belirli bir performans kaybı olarak hissedilmektedir.
İkinci olarak Stackte kullanılan elemanlar şayet değer türünde elemanlarsa bu durumda hem tür dönüştürmesi yapılmalı hemde puch işleminde stack ten heap e (boxing) pop işleminde ise heapten stacke(unboxing) veri taşınması yaşanmaktadır. Boxing, Unboxing adıyla anılan bu işlemler oluşmasını istemediğimiz türden işlemlerdir. Üçüncü olarak yukarıdaki örnek te submarin türünden bir eleman stacke push edildiğinde pop ile alındığında yanlış bir türe dönüştürülerek alınırsa yaşanacak olan hata derleme zamanında değil çok daha sonra ve de sinsice çalışma zamanında bir exception biçiminde ortaya çıkacaktır. Kısaca bu mekanizma bug oluşumuna müsait kötü bir mekanizmadır.

Bunun çözümü C++ Template olarak bilinen Genericlerdir.

Yukarıdaki örneğin generic olarak tasarımı aşağıdaki gibidir. Öncelikle bu süreç içinde hiçbir tür ismi obje de dahil olarak geçmemelidir. Bu durum tanımlanırken değil yaratılırken de olmamalıdır.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
namespace MyStack
{
    class MyStack<T>
    {
        private T[] arr;
        private int cursor = 0;

        public MyStack(int size)
        {
            arr = new T[size];
            cursor = 0;
        }

        public void Push(T val)
        {
            arr[cursor++] = val;
        }

        public T Pop()
        {
            return arr[--cursor];
        }
    }

    class Submarine : Object
    {
        private string name;

        public string Name
        {
            get { return name; }
            set { name = value; }
        }
        private string no;

        public string No
        {
            get { return no; }
            set { no = value; }
        }

        public Submarine(string name, string no)
        {
            this.name = name;
            this.no = no;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
```

```
            MyStack<string> stk = new MyStack<string> (10);

            stk.Push("kaan");

            Console.WriteLine(stk.Pop());

        }
    }
}
```

Bir tür jeneric olarak tasarlanacaksa örenğin bir sınıf: Sınıf isminden sonra açılıp kapatılan açısal parantezler içine bir yer tutucu karakter kullanıp artık o sınıfın içinde herhangi bir özel tür belirtmeyiz. Sınıf bildiriminde açısal parantezler içine yazılan yer tutucu daha sonra osınıf türünden bir nesne yaratılırken özgün bir tür ismiyle değiştirilecek ve de o tür ismi artık geçerli olacaktır. Generic olarak tasarlanmış yapılarda yukarıda bahsedilen üç sorun yaşanmayacaktır.

Aşağıdaki örnekte stack in büyüme faktörü de tutuluyor. (Array.Resize la bu iş çok kolay)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace MyStack
{
    class MyStack<T>
    {
        private T[] arr;
        private int cursor = 0;
        private int growth_Factor;

        public MyStack(int size, int GrowthFactor)
        {
            arr = new T[size];
            cursor = 0;
            growth_Factor = GrowthFactor;
        }

        public void Push(T val)
        {
            if (cursor >= arr.Length)
            {
                Array.Resize(ref arr, arr.Length * growth_Factor);
            }
            else
            {
                arr[cursor++] = val;
            }
        }

        public T Pop()
        {
            return arr[--cursor];
```

```
        }
    }

    class Submarine : Object
    {
        private string name;

        public string Name
        {
            get { return name; }
            set { name = value; }
        }
        private string no;

        public string No
        {
            get { return no; }
            set { no = value; }
        }

        public Submarine(string name, string no)
        {
            this.name = name;
            this.no = no;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            MyStack<Submarine> stk = new MyStack<Submarine> (10, 2);

            stk.Push(new Submarine("Anafartalar", "S-300" ));

            Submarine ship =  stk.Pop();

        }
    }
}
```

Transaction Nedir? Birbirleriyle mantıksal Ya da işlevsel anlamda ilişkili olan bir Ya da daha fazla sayıdaki işlemi tek ve tümleşik bir yapı altında toplayan ve de bunların atomik davranmasını sağlayan kavramdır. Bu kavram dahaçok veritabanı programcılığının bir terimidir. Bunu anlamak için ACIT doktirinini bilmek gerekir
A: Atomicity
C:Consistency
I: Isolation
D:Durability

Bu dört kavram olmadan transaction anlaşılmaz.

Atomicity: Atomiklikçok önemli bir kavramdır. Bunun tek cümleyle özeti ya hep ya hiç tir. Transaction işlemlerini Transaction Boundery denilen bir çerçeve içinde gerçekleşir. Örnek olarak bir EFT de bu tümlesik yapı içinde birinci işlem. Hesaptan paranın çekilişi hedef hesaba paranın yatırılışı, EFT Onay ve Dekont Basımı yani bir EFT işlemi 4 alt işlemden oluşuyor. Bu işlemin atomik olması gerekiyor. Yani bu işlemlerin hiçbirinde hata olmaması gerekir. Örneğin Kaynak hesaptan parayı çekelim ama hedef hesaba para yatırılamadı. Bu durumda transaction işlemi atomik olarak bu işlem gerçekleşmemiş sayılır. Transaction işlemi başarıyla sonuçlanana kadar bu işlemler havada kalır hepsi başarılı olursa transaction başarılı olmuş olur. Atomiklik işte budur.

Consistency: Tutarlılık; birtransaction işlemi tutarlı olması gerekir. Diyelimki 1000 ytlilik hesapta 200 ytl havale ettik ama hesapta 800 kalması gerekir. 900 kalıyorsa bu tutarsızlıktır.

Isolation: Bir transaction bir sistem üzerinde çalışırken aynı anda çalışmakta olan diğer transactionlardan sistemi izole eder. Bir transaction banka hesabımızdan 1000 ytl nin Gelir vergisini hesaplayacak bu hesap işlemi bitene kadar bu hesap üzerinde başka bir transaction bir işlem yapamamalı. Bu izolasyon durumu diğer transactionlara izin vermez ve onları başarısız kılar. Fakat bu transaction durumunda kilitin esnekliği ayarlanabilir.

Durability: Kararlılık, Sürdürülebilirlik; Transaction üzerinde çalıştığı işlemde bir problem oldu ve işlem gerçekleşemedi. Bu ilkeye göre bir problem halinde yapılan işlemin geri alınabilmesi ve son halinde kalabilmesi gerekir.

Bir transaction ACID ilkelerini tam olarak yerine getirilebiliyorsa o transaction doğru tasarlanmış demektir.

Gerçekte veritabanı özelinde transaction işlemi nasıl yapılıyor. Veritabanlarında 2 ayrı türde transaction işlemi var.

1. Manuel Transaction veya Local Transactionlar
2. Otomotik Transaction veya Global Transaction lar

Manuel transactionları biz yönetiyoruz. Otamatik transactionlar birden fazla network sistemi trafından yönetiliyor.

Manuel Transactionları ADO.NET, T-SQL lar yönetir.
Otomatik Transactionlar COM+ yönetiyor Windows NT den önce MTS: Microsoft Transaction Server dı.

ADO.NET te Transactionlar model olarak nasıl yönetiliyor? Transaction yönetimi connection nesnesiyle başlıyor. Ado.net öncesindeconnection nesnesinin transaction ynetme fonksiyonları vardı. Ama Ado.net Connection nesnesi modeli böyle değil. Connection nesnesi yratıldıktan sonra XXXConnection sınıfının Begin Transaction (); ile Transaction işlemi başlar. Isolation nesnesinin yumuşaklık sertliğini ayarlamak için BeginTransaction(Isolation Level) Isolation Level enumı kullanılır. Connection nesnesinin yönetimi için
XXXTransaction nesnesini yöneten yeni bir nesnenin yönetimi için Begin Transaction yarattığı nesne refaransı kullanılır.
XXXTransaction trs = con.BeginTransaction(-);

Daha sonra Binding işlemeleri ve Exacute işlemleri yapılır. Exacute işlemlerin genelde try Catch işlemleri içinde yapılır. Biz bir transaction içine bind ettikten sonra komutumuzu yapılan hiçbir işlem veritabanına yansımaz. Bu Execute işlemi içinde transaction işleminin kalıcı olması için Commit işleminin bitmesi gerekir. Yani Exception başarılı olmaması gerekir. Eğer Exception işlemi gerçekleşirse bütün transactionlar RollBack işlemi ile geri alınır.

```
trs.RollBack();
TransactionSmp1(Kötü tasarım"Hakanın yüzünden oldu:)") yerleştir. Normalde try catch ile
yapılmalı...
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.SqlClient;

namespace TransactionSmp
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }

      private void button1_Click(object sender, EventArgs e)
      {
         SqlConnection cnn = new SqlConnection();
         cnn.ConnectionString = "server=PC6; database=CSDSite; integrated security=true";
         cnn.Open();

         // Transactionı başlat
         SqlTransaction trs = cnn.BeginTransaction();

         SqlCommand cmd = cnn.CreateCommand();
         cmd.CommandText = "Delete from Egitmenler where EgitmenID=11";

         SqlCommand cmd2 = cnn.CreateCommand();
         cmd2.CommandText = "Delete from Kurslar where KursID=50";


         // Transactiona komutların bind edilişi, transaction boundary belirlenmesi
         cmd.Transaction = trs;
         cmd2.Transaction = trs;

            if (cmd.ExecuteNonQuery() != 0)
            {
               if (cmd2.ExecuteNonQuery() != 0)
               {
                  trs.Commit();
               }
               else
               {
                  trs.Rollback();
               }
```

```
            }
            else
            {
               trs.Rollback();
            }

         cnn.Close();


      }
   }
}
```

Biriş uygulamasının yapması gereken işlemler

1. Verinin kullanıcı sunumu / alınması
2. İş kurallarına göre verinin işlenmesi
3. Veritabanı işlemleri

Çok katmanlı mimari: **Çok katmanlı programlama mimarisini öğren**
Yukarıdaki iş uygulamaları çok katmanlı mimaride şöyle katmanlaştırılabilir.

1. Presentation Layer
2. Bussines Layer
3. Data Layer

DAL Yerleştir. Data Access Layer (Çok katmanlı yapı)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using System.Data.SqlClient;
using System.Data.Common;

namespace DAL
{
   public abstract class CGenelDBIslemleri
   {
      public abstract DataTable GetDataTable(string qry, DbParameter[] prs);

      public abstract DataSet GetDataSet(string qry, DbParameter[] prs);

      public abstract int QueryExecute(string qry, DbParameter[] prs);

      // ...
   }

   public class CSQLHelper : CGenelDBIslemleri
   {
      private string m_CnnStr = "";

      private SqlConnection m_Cnn = null;
      private SqlCommand m_Cmd = null;
```

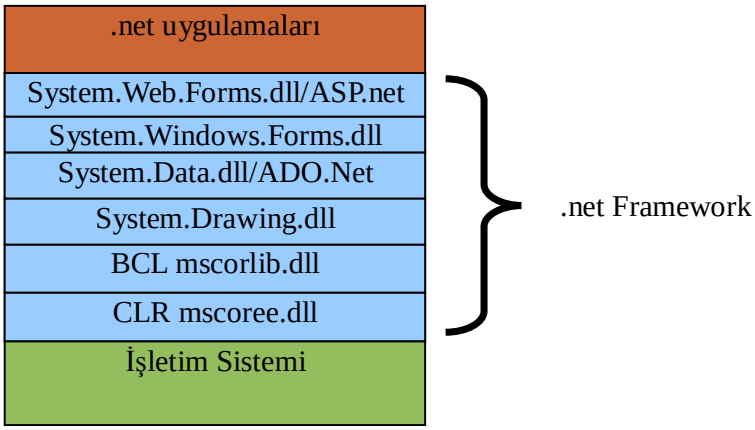
```
public CSQLHelper(string cnnStr)
{
   m_CnnStr = cnnStr;
}

private void Open()
{
   try
   {
      m_Cnn = new SqlConnection(m_CnnStr);
      m_Cnn.Open();
   }
   catch (SqlException exc)
   {
      throw exc;
   }
}

private void Close()
{
   if (m_Cnn.State != ConnectionState.Closed)
   {
      m_Cnn.Close();
   }
}

public override DataTable GetDataTable(string qry, DbParameter[] prs)
{
   Open();

   m_Cmd = new SqlCommand(qry, m_Cnn);

   if (prs != null)
   {
      foreach (SqlParameter p in prs)
      {
         m_Cmd.Parameters.Add(p);
      }
   }

   SqlDataAdapter a = new SqlDataAdapter(m_Cmd);

   DataTable t = new DataTable();

   a.Fill(t);

   Close();

   return t;
}
```

```
        public override DataSet GetDataSet(string qry, DbParameter[] prs)
        {
            Open();

            m_Cmd = new SqlCommand(qry, m_Cnn);

            if (prs != null)
            {
                foreach (SqlParameter p in prs)
                {
                    m_Cmd.Parameters.Add(p);
                }
            }

            SqlDataAdapter a = new SqlDataAdapter(m_Cmd);

            DataSet t = new DataSet ();

            a.Fill(t);

            Close();

            return t;
        }

        public override int QueryExecute(string qry, DbParameter[] prs)
        {
            Open();

            m_Cmd = new SqlCommand(qry, m_Cnn);

            if (prs != null)
            {
                foreach (SqlParameter p in prs)
                {
                    m_Cmd.Parameters.Add(p);
                }
            }

            int i = m_Cmd.ExecuteNonQuery();

            Close();

            return i;
        }

    }
}
```

Setup: Program debug ile yazıldıktan sonra Release le tekrar build edilir. (.net te setup uygulamasını araştır...)

ASP.Net: Sunucu taraflı çalışan dinmaik web uygulamalarını geliştirmekte kullanılan bir net teknolojisinin ismidir.

ASP.net .net teknolojileri içinde nasıl bir konuma sahiptir. Bunu anlmak için .net framework ü bilmek gerekir.



.net Framework demek Visula Studio demek değil. Diğer programlama dilleriylede .net kütüphanesi kullanarak programlar yazılabilir.