.NET Ortamında GUI Programlama

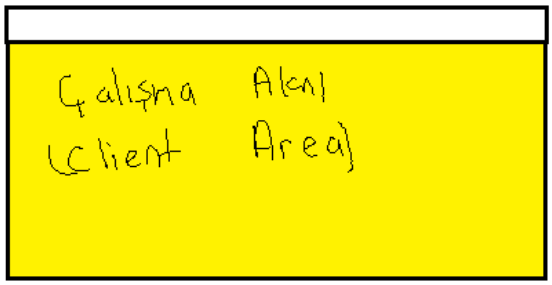
.NET'te Pencereli GUI programları oluşturabilmek için iki framework aktif olarak kullanılmaktadır: Forms ve WPF. Forms kütüphanesi Mono'da Mobil aygıtlarda benzer biçimde kullanılır. WPF daha modern ve daha hızlıdır. Ancak daha az taşınabilirdir. Genel olarak Forms kütüphanesi daha fazla platform tarafından (örneğin Mono ve Windows Mobile) desteklenmektedir. Ancak WPF daha modern bir ortamdır.

Pencere Kavramı ve Pencere Terminolojisi

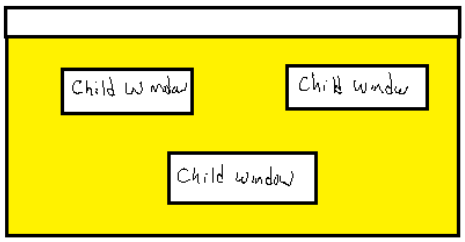
Ekranda bağımsız olarak kontrol edilebilen dikdörtgensel bölgelere pencere denilmektedir.

Masaüstüne doğrudan açılan pencerelere "ana pencereler (top level windows)" denilmektedir. .NET'te ana pencerelere "Form" da denilmektedir. Ana pencereler genellikle (fakat zorunlu değil) bir başlık (caption) kısmına sihiptir. Bunların genellikle kalın bir sınır çizgileri olur. Başlığın sağ üst kısmında pencereyi konumlandırmak için üç küçük simge (icon) bulunmaktadır. Başlığın sol üst kısmında ise genellikle fare olmadan pencere işlemlerinin yapılmasını sağlayan bir sistem menüsü bulunmaktadır.

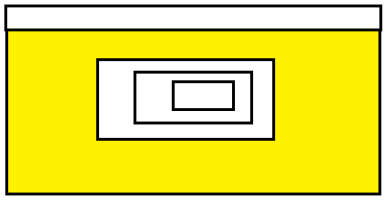
Bir pencerenin pencere başlığının ve sınır çizgilerinin içerisinde kalan alana "çalışma alanı (client area)" denilmektedir.



Bir pencerenin çalışma alanında bulunan ve oradan dışarı çıkamayan pencerelere ise alt pencereler (child windows) denir. Alt pencerelerin genellikle başlık kısmı bulunmaz. Ancak alt pencereler ana pencereler gibi başlık kısımlarına da ship olabilirler.



Alt pencerelerin de alt pencereleri söz konusu olabilir:



Her alt pencerenin bir "üst penceresi (parent window)" vardır. Aynı üst pencereye sahip olan alt pencerelere "kardeş pencereler (sibling windows)" denilmektedir.

Aslında masaüstü penceresi (desktop) da bir penceredir ve bu pencere ana pencereler için üst pencere görevindedir. Dolayısıyla ana pencereler aslında kardeş pencerelerdir.

Bazı pencereler hem ana pencere gibi hem de alt pencere gibi davranmaktadır. Bunlar doğrudan masaüstüne açılırlar. Her zaman üst pencerelerinin üzerinde görüntülenirler ve bunların üst pencereleri minimize edildiğinde onlar da minimize edilirler. Bu tür pencerelere "sahiplenilmiş pencereler (owned windows)" ya da halk arasında "diyalog pencelereri" denilmektedir.

Bir GUI programda görmüş olduğumuz düğmeler (buttons), edit alanları (text boxes), listeleme kutuları (list boxes) gibi görsel öğeler hep birer penceredir. Bu görsel öğeler sıradan bir alt pencerenin içinin uygun biçimde dekore edilmesiyle oluşturulurlar. Yani aslında düğme diye birşey yoktur. İçi boş bir pencere düğme görüntüsü verecek biçimde boyanmıştır. Bu görsel öğelerin pek çoğu .NET sınıf kütüphensinde hazır olarak bulunur. Örneğin Button sınıfı düğmeyi, TextBox sınıfı edit alanını temsil etmektedir. Bir düğme çıkartmak için yapılacak tek şey bir Button nesnesi yaratmaktır.

.NET'te Pencere Sınıf Hiyararişisi

Pencereler ister ana pencere olsun isterse alt pencere olsun birtakım ortak özelliklere sahiptir. Örneğin hepsinin bir zemin rengi, bir betimleyici yazısı, bir konumu, bir büyüklüğü vs. bulunur. İşte pencerelerin ortak özellikleri .NET kütüphanesinde Control isimli bir sınıfta toplanmıştır. Diğer sınıflar Control sınıfından türetilmiş durumdadır. Form isimli sınıf ana pencereyi temsil eder. Form sınıfı da dolaylı olarak Control sınıfından türetilmiştir. Control abstract bir sınıf değildir. Control sınıfı türünden bir nesne yaratılırsa sınır çizgileri olmayan, içi boş en yalın pencere elde edilir.

İskelet GUI Programı

Masaüstüne içi boş bir ana pencere çıkartan elet iskelet bir C# programı şöyle yazılır:

1) Önce boş bir proje (empty project) oluşturulur ve bir kaynak dosya projeye eklenir.

2) System.dll, System.Windows.Forms.dll ve System.Drawing.dll dosyalarına referans edilir.

3) İskelet program aşağıdaki gibi yazılır:

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

namespace BUUBM
{
classApp
    {
       public static void Main()
        {
       Application.Run(new MainForm());
        }
    }

       Class MainForm : Form
    {
       public MainForm()
        {
       //...
        }
    }
}
```

4) Proje seçeneklerine gelinip "Output Type" "Windows Appliaction" olarak seçilir.

İskelet programda Form sınıfından MainForm isimli bir sınıf türetilmiştir. Bu sınıf türünden nesne yaratılarak Application.Run metoduna parametre yoluyla geçirilmiştir.

Mesaj Tabanlı Programlama Modeli

Windows ve diğer GUI tabanlı işletim sistemleri mesaj tabanlı programlama modelini kullanmaktadır. Bu modelde programı ilgilendiren her türlü olaya "mesaj (message)" denilmektedir. Birtakım girdileri bu modelde programcı elde etmeye çalışmaz. Bunu bizzat işletim sisteminin kendisi elde ederek programcıya verir. İşletim sistemi bir pencereyi ilgilendiren bir olay tespit ettiğinde bu olayı bir mesaj formuna dönüştürerek ismine "mesaj kuyruğu (message queue)" denilen bir kuyruk sistemine yazar. İşletim sistemi yalnızca mesajı tespit edip kuyruğa eklemektedir. Bu mesaj kuyruğuna bakıp sıradaki mesajı oradan alarak işlemek programcının sorumluluğundadır. Programcı kuyrukta sıradaki mesajı alır, bu mesajın neden kuyruğa bırakıldığına bakar ve uygun işlemleri yapar. Bir GUI programının ömrü "kuyruktaki sırada bulunan mesajı al, incele ve gerekeni yap" biçiminde bir döngüde geçmektedir. Bu döngüye programın mesaj "döngüsü (message loop)" denir. C gibi aşağı seviyeli dlllerde bu döngü bizzat programcı tarafından oluşturulmaktadır. Ancak .NET'te Application sınıfının static Run metodu bizim için bu işlemi yapmaktadır. Yani bir GUI programının yaşamı Run metodundaki mesaj döngüsünde geçmektedir.

Pekiyi bir GUI programı nasıl sonlanmaktadır. İşte kullanıcı pencerenin sağ üst köşesindeki X tuşuna bastığında (ya da ALT + F4 tuşuna bastığında) Windows kuyruğa WM_CLOSE isimli mesajı bırakır. Application.Run metodu bu mesajı aldığında mesaj döngüsünden çıkar ve geri döner. Böylece Main metodu sona erer ve program da bitmiş olur.

Mesaj kuyrukları Windows'ta thread'e özgüdür. Yani her thread'in ayrı bir mesaj kuyruğu vardır. Bir thread'in yarattığı bütün pencerelere ilişkin mesajlar aynı kuyruğa yazılmaktadır. Pekiyi mesaj kuyruğunda hiç mesaj yoksa Application.Run ne yapar? Windows'ta bu durumdaki thread'ler işlemci zamanı harcamadan dndurulup bloke edilmektedir. Ta ki mesaj kuyruğuna bir mesaj gelene kadar. Yani kuyruğunda mesaj olmayan bir thread boşa CPU kaynağı harcamamaktadır.

İskelet GUI Programının Açıklaması

İskelet programda Form sınıfı doğrudan değil, türetilerek kullanılmıştır. Bunun amacı Form özelliğine sahip sınıfa birtakım eklemeler yapılmasına olanak sağlamaktır. Application sınıfının üç static Run metodu vardır. İskelet programda aşağıdaki Run metodu kullanılmıştır:

```
public static void Run(Form mainForm)
```

Bu Run metodu aldığı formu görünür yapar. Normal olarak Form nesnesi yaratıldığında ana pencere yaratılmıştır fakat görünür değildir. Bir ana pencereyi görünür hale getirmek için Form sınıfının Show metodu çağrılabilir ya da eşdeğer olarak Form sınıfının bool türden Visible property'sine true atanabilir. Örneğin:

```
MainForm m1 = newMainForm();
f1.Show();
```

Application sınıfının Run metodu bizden aldığı Form nesnesini zaten kendisi görünür hale getirmektedir. Run metoduna geçirdiğimiz Form nesnesinin diğer bir önemi de "hangi ana pencere kapatıldığında mesaj döngüsünden çıkılacağını tespit etmektir. Programın birden fazla ana penceresi olabilir. Bu durumda biz Run metoduna hangi Form nesnesini geçirmişsek o kapatıldığında Run metodu sonlanacaktır.

Pixel Kavramı

Bugün kullandığımız grafik ekranlarda görüntülenecek en küçük birim bir noktadır. Buna "pixel (picture element)" denilmektedir. Çözünürlük (resolution) ekranın o anda kaç pixel'lik bir matris gibi davrandığını belirtir. Örneğin 1920X1080 çözünürlükte ekran her satırında 1920 pixel bulunan 1080 satırdan oluşmaktadır. Pixeller bağımsız olarak kontrol edilebilen noktalardır. Yani her pixel 16 milyon renkten bir tanesiyle renklendirilebilmektedir. Ekrandaki herşey aslında bu pixel'llerin biraraya getirilmesiyle oluşturulur. Örneğin bir resim dosyası aslında pixel'lerin hangi renklerde gösterileceğini belirten bir formata sahiptir. Çözünürlük yükseltilir, ekran boyu sabit kalırsa pixel'ler küçülür. Böylece tüm görüntü küçülmüş olur. Yani çözünürlük artırıldıkça ekrana daha fazla görüntü daha küçük olarak sığabilir. Şüphesiz görüntünün net biçimde algılanması çözürlükle ilgilidir. Çözünürlük yükseldikçe pixellerin arası daha fazla dolar, dolayısıyla daha ayrıntılı ve net bir görüntü elde edilir. Tabii belli, bir çöznürlük farklı boyutlardaki monitörlerde aynı netliğe karşı gelmez. Bu nedenle inch başına düşen pixel sayısı da önemli bir parametredir. Örneğin küçük bir

monitördeki 1920x1080 çöznürlük daha net bir görüntünün algılanmasına yol açacaktır. İnch başına düşen pixel sayısına dpi (dot per inch) denilmektedir.

Form uygulamalarında temel birim pixel'dir. Örneğin biz bir düğmenin genişliğini ve yüksekliğini hep pixel cinsinden belirleriz.

Point, Size ve Rectangle Yapıları

GUI programlamada Point, Size ve Rectangle isimli yapılar çok sık kullanılmaktadır. Bunlar System.Drawing.dll içerisindedir ve System.Drawing isim alanında bulunurlar.

Point yapısı ekrandaki bir pixel'in koordinatlarını tutmak için düşünülmüştür. Point yapısının başlangıç metodu tutulacak noktayı bizden alır. Örneğin:

```
using System;
using System.Drawing;

namespace BUUBM
{
    class Program
    {
        static void Main(string[] args)
        {
            Point pt = new Point(100, 200);
            Console.WriteLine(pt.ToString());

        }
    }
}
```

Yapının X ve Y read/write property elemanları bu değerleri almamızı ve değiştirmemizi sağlar. Örneğin:

```
using System;
using System.Drawing;

namespace BUUBM
{
    class Program
    {
        static void Main(string[] args)
        {
            Point pt = new Point();

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

            Console.WriteLine("({0}, {1})", pt.X, pt.Y);
        }
    }
}
```

Yapının Offset isimli metodu noktayı deltax, deltay kadar kaydırır.

```
using System;
using System.Drawing;

namespace BUUBM
{
    class Program
    {
        static void Main(string[] args)
        {
            Point pt = new Point(100, 200);

            pt.Offset(2, -4);
```

```
            Console.WriteLine("({0}, {1})", pt.X, pt.Y);    // (102, 196)
        }
    }
}
```

Point yapısının iki Point'i == ve != ile karşılaştıran operatör metotları vardır. Örneğin:

```
using System;
using System.Drawing;

namespace BUUBM
{
    class Program
    {
        static void Main(string[] args)
        {
            Point pt1 = new Point(10, 20);
            Point pt2 = new Point(10, 20);

            Console.WriteLine(pt1 == pt2 ? "Evet" : "Hayır");
        }
    }
}
```

Size isimli yapı Point yapısına benzemektedir. Fakat bu yapının amacı genişlik-yükseklik bilgisi tutmaktır. Size yapısının Width ve Height isimli iki read/write property elemanı vardır. Yapının başlangıç metodu bizden genişlik ve yüksekliği alır. Örneğin:

```
using System;
using System.Drawing;

namespace BUUBM
{
    class Program
    {
        static void Main(string[] args)
        {
            Size sz = new Size(100, 200);

            Console.WriteLine(sz);
            Console.WriteLine("{0}, {1}", sz.Width, sz.Height);

        }
    }
}
```

İki Size nesnesi yapının + operatör metoduyla toplanıp - operatör metoduyla çıkartılabilir. Örneğin:

```
using System;
using System.Drawing;

namespace BUUBM
{
    class Program
    {
        static void Main(string[] args)
        {
            Size sz1 = new Size(10, 20);
            Size sz2 = new Size(30, 40);
            Size sz3;

            sz3 = sz1 + sz2;

            Console.WriteLine(sz3);
        }
```

```
    }
}
```

Yine Size yapısının da iki Size nesnesini karşılaştıran == ve != operatör metotları vardır. Point yapısının bir Point ile Size nesnesini toplayan ve çıkartan operatör metotları vardır. Örneğin:

```
using System;
using System.Drawing;

namespace BUUBM
{
    class Program
    {
        static void Main(string[] args)
        {
            Point pt = new Point(10, 20);
            Size sz = new Size(3, 5);
            Point result;

            result = pt + sz;
            Console.WriteLine(result);
        }
    }
}
```

Rectangle isimli yapı dikdörtgensel bir bölgenin koordinatlarını tutmak için düşünülmüştür. Yapının dört int parametreli başlangıç metodu bizden sırasıyla sol üst köşe ve genişlik yükseklik değerlerini ister.

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

Örneğin:

```
using System;
using System.Drawing;

namespace BUUBM
{
    class Program
    {
        static void Main(string[] args)
        {
            Rectangle rect = new Rectangle(10, 10, 100, 200);
            Console.WriteLine(rect);
        }
    }
}
```

Rectangle yapısının Point ve Size parametreli bir başlangıç metodu daha vardır:

```
public Rectangle(Point location, Size size)
```

Örneğin:

```
using System;
using System.Drawing;

namespace BUUBM
{
    class Program
    {
        static void Main(string[] args)
        {
            Rectangle rect = new Rectangle(new Point(10, 10), new Size(100, 200));
            Console.WriteLine(rect);

        }
```

```
    }
}
```

Rectangle yapısının X ve Y property'leri read/write property'lerdir. Dikdörtgenin bize sol-üst köşe koordinatlarını verir. Left ve Top property'leri de yine sol-üst köşe koordinatlarını bize vermektedir. Fakat bu property'ler read-only'dir. Yapının Width ve Height read/write property'leri dikdörtgenin genişlik ve yükseklik bilgilerini bize verir. Location isimli read/write property sol-üst köşe koordinatını bize Point olarak vermektedir. Size isimli read/write property bize dikdörtgenin genişlik ve yüksekliğini Size yapısı olarak verir. Right ve Bottom property'leri read-only'dir ve bize sağ-alt köşe koordinatlarını verir. Örneğin:

```
using System;
using System.Drawing;

namespace BUUBM
{
    class Program
    {
        static void Main(string[] args)
        {
            Rectangle rect = new Rectangle(new Point(10, 10), new Size(100, 200));

            Console.WriteLine("{0}, {1}, {2}, {3}", rect.X, rect.Y, rect.Width, rect.Height);
            Console.WriteLine("{0}, {1}", rect.Location, rect.Size);
        }
    }
}
```

Rectangle yapısının Offset isimli metodu dikdörtgeni deltax, deltay kadar ötelemekte kullanılır. Örneğin:

```
using System;
using System.Drawing;

namespace BUUBM
{
    static void Main(string[] args)
    {
            Rectangle rect = new Rectangle(new Point(10, 10), new Size(100, 200));

            rect.Offset(8, -2);
            Console.WriteLine(rect);
        }

    }
}
```

Yapının Inflate metotları dikdörtgeni iki ucundan açmak ya da büzmek için kullanılır. Örneğin:

```
using System;
using System.Drawing;

namespace BUUBM
{
    static void Main(string[] args)
        {
            Rectangle rect = new Rectangle(10, 10, 100, 100);

            rect.Inflate(5, 5);
            Console.WriteLine(rect);

        }

    }
}
```

Yapının Contains isimli metotları "hit testing" yapmak için kulanılır.

Point yapısının, Size yapısının ve Rectangle yapısının çeşitli operatör metotları vardır. Bunlar kullanım sırasında gerektiğinde açıklanacaktır.

Renk Kavramı ve Color Yapısı

Bilgisayar ekranındaki her pixel yaklaşık 16 milyon renkten birisi olacak biçimde renklendirilebilir. Eskiden CRT monitörler çok yaygın kullanılıyordu. Bugün artık LCD monitörler yaygın kullanılmaktadır. Eskiden CRT monitörlerde bir pixel'i oluşturmak için 3 elekton tabancası kullanılıyordu. Bunlar Red, Green ve Blue tabancalarıydı. Bu üç tabanca belli bir pixel'e belli bir yoğunlukta ışıtıldığında 16 milyon renkten biri elde ediliyordu. Bugünkü LCD sistemlerinde de benzer mantık kullanılmaktadır. Biz bir rengi bilgisayar sisteminde Kırmızı, Yeşil ve Mavinin tonal birleşimleriyle ifade etmekteyiz. Bu tonal birleşimler [0, 255] arası değer alabilmektedir. Böylece toplam $2^8 * 2^8 * 2^8 = 2^{24}$ renk elde etmek mümkündür. Örneğin Red = 255, Green = 0 ve Blue = 0 tonal bileşenler kullanılırsa tam kırmızı bir renk elde edilir. Ya da örneğin Red = 255, Green = 255 ve Blue = 0 tonal birleşimlerinden sarı renk elde edilir.

Aslında modern grafik kartlarında rengi oluşturan dördüncü bir bileşen de vardır. Buna alfa bileşeni ya da alfa kanalı (alpha channel) denilmektedir. Alfa kanalı saydamlık (transparency) belirtir. Saydamlık bir pixel diğerinin üzerine getirildiğinde arkasının görülmesi anlamındadır. Alfa faktörü 255 ise tam saydamsız durum söz konusudur, 0 ise tam saydamlık söz konusudur.

.NET'te renkler Color isimli bir yapıyla temsil edilmektedir. Bir Color nesnesi yapının başlangıç metoduyla değil yapının FromArgb static metotlarıyla oluşturulur.

```
public static Color FromArgb(int red, int green,int blue)
public static Color FromArgb(int alpha, int red, int green, int blue)
```

Örneğin:

```
Color c = Color.FromARgb(255, 0, 0);
```

Tabii bu biçimde renkleri oluşturmak oldukça zahmetlidir. Bunun için Color yapısına çok sayıda ilgili renkten Color nesnesi veren static read-only property elemanı yerleştirilmiştir. Örneğin Color.Red bize kırmızı bir Color nesnesi, Color.Yellow sarı bir Color nesnesi verir.

Color yapısının R, G, B, A isimli static olamayan read-only property elemanları bize ilgili rengin bileşenlerini verir.

Alt Pencerelerin Oluşturulması

Ana pencerelerin (yani Form penceresinin) içi genellikle boş olmaz. Onların içerisinde düğme gibi, edit alanları gibi çeşitli alt pencereler bulunur. Ana pencere genellikle içerisinde bu alt pencerelerle birlikte açılır. Ana pencere ile bu alt pencereler arasındaki ilişki içerme (composition) ilişkisidir. O halde bir alt pencere yaratımı tipik olarak şöyle yapılmalıdır: Ana pencerenin veri elemanlı olarak alt pencere referansı bildirilir. Bunun yaratımı ana pencerenin başlangıç metodunda yapılır.

Pekiyi bir alt pencerenin hangi pencerenin alt penceresi olduğu nasıl belirlenmektedir? Bunun iki yolu vardır:

1) Üst pencerenin Control sınıfından gelen Controls collection elemanına alt pencerenin Control referansını eklemek. Control sınıfınn Controls isimli collection property'si Control.ControlCollection isimli bir sınıf türündendir. Bu sınıf IList arayüzünü desteklemektedir. Örneğin:

```
mainForm.Controls.Add(button);
```

2) Alt pencere nesnesinin Control sınıfından gelen Parent property'sine üst pencere referansını atamak. Örneğin:

```
button.Parent = mainForm;
```

Yukarıdaki iki biçim eşdeğerdir. Zaten Parent property'sinin set bölümündeüst pencerenin Controls property'sine ekleme yapılmıştır. Yani Control sınıfının Parent property'si şöyle yazılmıştır:

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

Bir alt pencere Control sınıfından gelen Dispose metoduyla yok edilebilir. Örneğin bir düğmeyi yok edecek olalım:

```
button:Dispose();
```

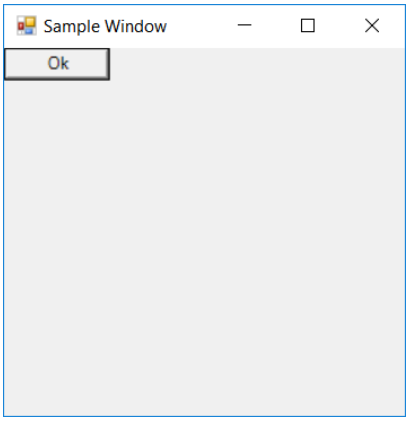
Artık bu düğme alt penceresi hiç yaratılmamış gibi olacaktır.

Control Sınıfının Önemli Property Elemanları

Her pencerenin bir pencere yazısı vardır. Örneğin düğmenin pencere yazısı düğme üzerindeki yazıdır. TextBox'ın pencere yazısı onun içerisindeki girilmiş olan yazıdır. Form'un pencere yazısı başlık çubuğundaki yazıdır. Pencerelerin pencere yazıları Control sınıfının read/write Text property'si ile temsil edilmektedir. Örneğin:

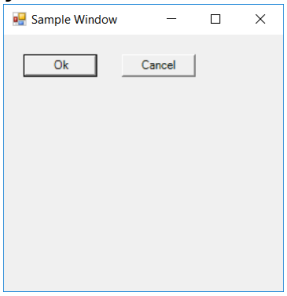
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WinFormsApp1
{
    public partial class Form1 : Form
    {
        private Button m_buttonOk;
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            this.Text = "Sample Window";
            m_buttonOk = new Button();
            m_buttonOk.Parent = this;
            m_buttonOk.Text= "OK";
        }
    }
}
```

Burada ana pencerenin başlık yazısı “Sample Window” biçiminde, düğmenin başlık yazısı da “Ok” biçiminde belirlenmiştir:



Her pencerenin bir konumu vardır. Konumda ana pencereler için orijin noktası masaüstünün sol-üst köşesi alt pencereler için orijin noktası ise onun üst penceresinin çalışma alanının sol-üst köşesidir. Control sınıfının Point türünden Location isimli read/write property elemanı pencerenin sol köşesinin bulunacağı koordinatı belirtir. Yani pencere sol-üst köşesi o koordinatta olacak biçimde konumlandırılır.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WinFormsApp1
{
    public partial class Form1 : Form
    {
        private Button m_buttonOk;
        private Button m_buttonCancel;

        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            this.Text = "Sample Window";
            m_buttonOk = new Button();
            m_buttonOk.Parent = this;
            m_buttonOk.Text= "OK";
            m_buttonOk.Location = new Point(20, 20);
            this.Controls.Add(m_buttonOk);

            m_buttonCancel = new Button();
            m_buttonCancel.Parent = this;
            m_buttonCancel.Text= "Cancel";
            m_buttonCancel.Location = new Point(120, 20);
        }
    }
}
```



Anahtar Notlar: GUI arayüzü tasarlarken neyin nereye yerleştirileceği konusunda tereddütler olabilir. Bunun için programcının beğendiği programların arayüzlerini incelemesi ve tasarımını onlara benzetmesi tavsiye edilir. Siz de artık GUI uygulamalarını incelerken onun tasarımına dikkat etmelisiniz. Oralardan güzel görünüm hakkında notlar almalısınız.

Control sınıfının Left ve Top property'leri read/write property'lerdir. Bunlar pencerenin sol-üst köşe koordinatlarını alıp değiştirmekte kullanılabilir. (Şüphesiz Location property'si de aynı amaçla kullanılabilir). Örneğin:

```
ctrl.Location = new Point(x, y);
```

ile,

```
ctrl.Left = x;
ctrl.Top = y;
```

eşdeğer etkiye yol açar. Control sınıfının read-only Right ve Bottom property'leri bize pencerenin sağ-alt köşesine ilişkin koordinatları verir. Fakat bu property'ler read-only olduğu için biz bunlara değer atayamayız.

Control sınıfının Width ve Height isimli read/write property elemanları pencerenin genişlik ve yüksekliğini ayarlamakta kullanılabilir. Control sınıfının Size isimli read/write property elemanı Size isimli yapı türündendir. Tek hamlede bu property yoluyla genişlik yükseklik ayarlamasını yapabiliriz. Yani:

```
ctrl.Size = new Size(w, h);
```

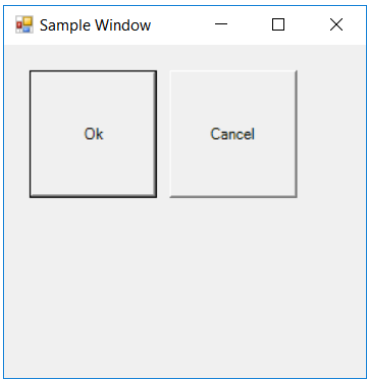
ile,

```
ctrl.Width = w;
ctrl.Height = h;
```

eşdeğer işlevelere sahiptir. Örneğin:

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

namespace BUUBM
{
    public partial class Form1 : Form
    {
        private Button m_buttonOk;
        private Button m_buttonCancel;

        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            this.Text = "Sample Window";

            m_buttonOk = new Button();
            m_buttonOk.Text = "Ok";
            m_buttonOk.Location = new Point(20, 20);
            m_buttonOk.Size = new Size(100, 100);
            this.Controls.Add(m_buttonOk);

            m_buttonCancel = new Button();
            m_buttonCancel.Parent = this;
            m_buttonCancel.Text = "Cancel";
            m_buttonCancel.Size = new Size(100, 100);
            m_buttonCancel.Location = new Point(130, 20);
```

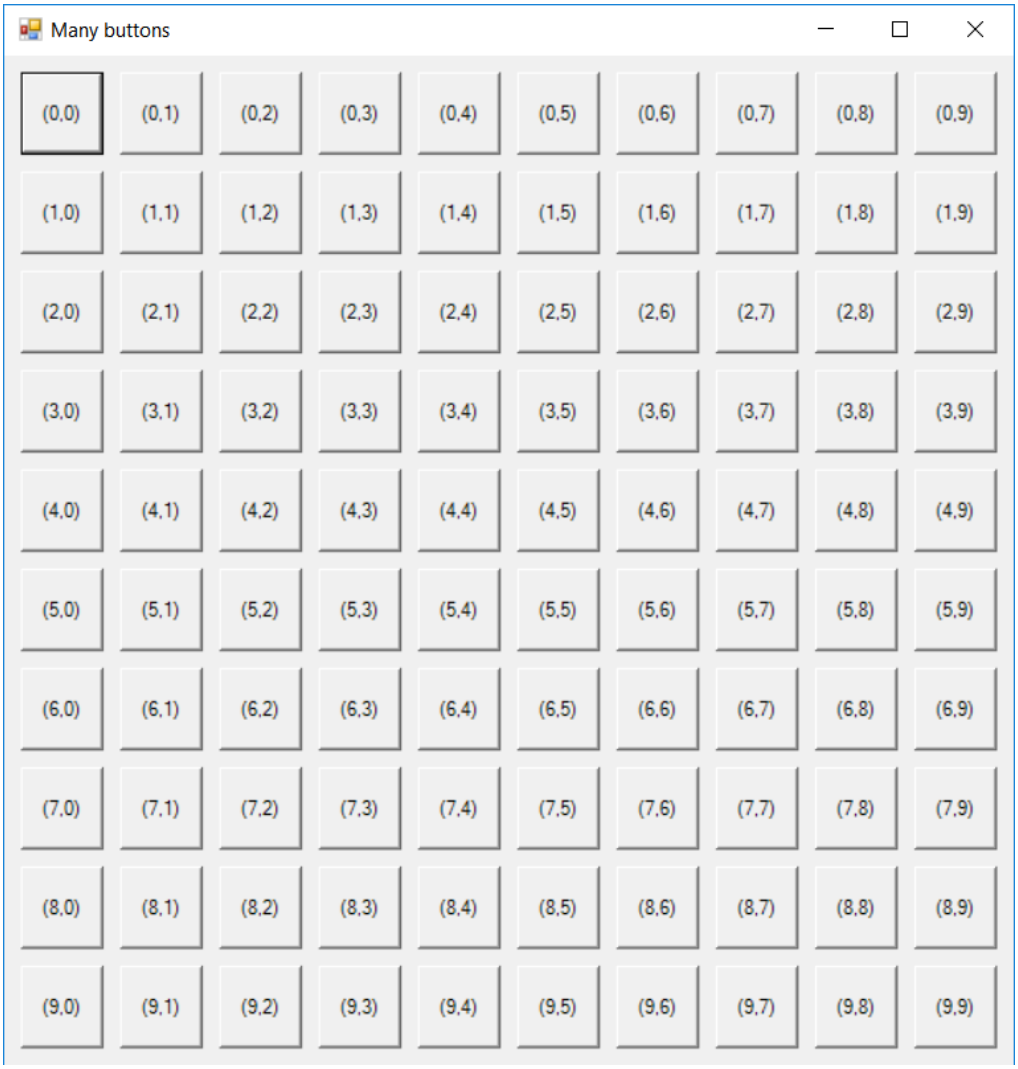
```
        }
    }
}
```



Anahtar Notlar: Application sınıfının static EnableVisualStyles isimli metodu kontrollerin yeni stilde (Windows Vista, 7, 8 stili) görüntülenmesine yol açar. Programın başında bir kez bu çağrıyı yaparsak kontroller yeni stilde görüntülenebilir.

Örneğin:

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

namespace BUUBM
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            this.Text = "Many buttons";
            this.ClientSize = new Size(610, 610);

            int x;
            int y = 10;
            for (int i = 0; i < 10; ++i)
            {
                x = 10;
                for (int k = 0; k < 10; ++k)
                {
                    Button button = new Button();
                    button.Text = string.Format("({0},{1})", i, k);
                    button.Size = new Size(50, 50);
                    button.Location = new Point(x, y);
                    this.Controls.Add(button);
                    x += 60;
                }
                y += 60;
            }
        }
    }
}
```



Control sınıfının Rectangle türünden Bounds isimli read/write property elemanı tek hamlede pencerenin hem konumunu hem de boyutunu ayarlamakta kullanılabilir. Örneğin:

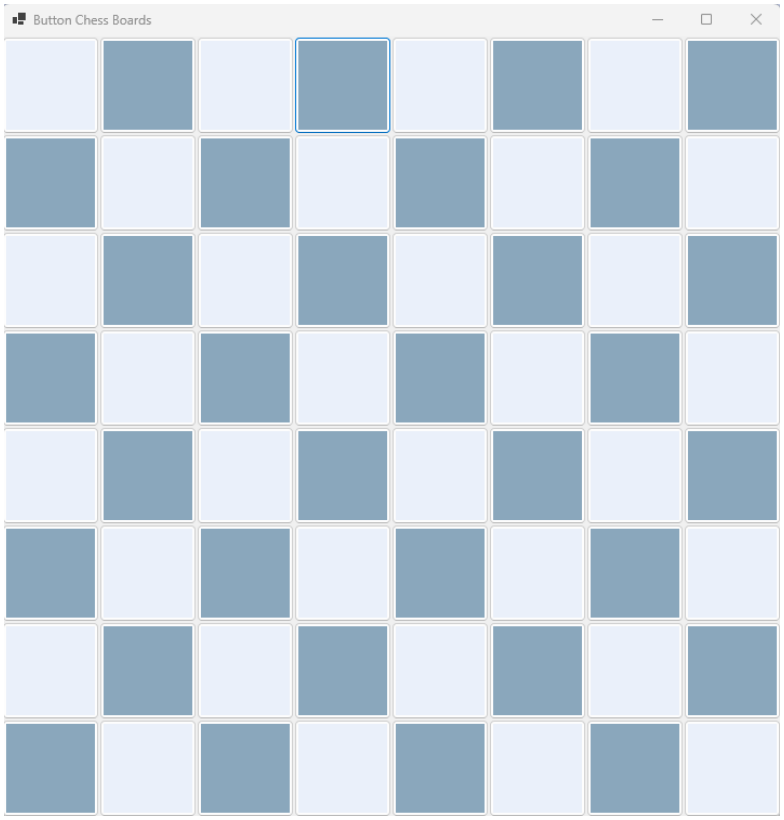
```
ctrl.Bounds = new Rectangle(30, 30, 100, 50);
```

Control sınıfının Size türünden ClientSize isimli read/write property elemanı pencerenin çalışma alanının genişlik ve yüksekliğini set edip almakta kullanılır. Bu property set edilirse pencerenin çalışma alanı belirtilen değerde olacak biçimde pencere boyutlandırılır.

Control sınıfının Rectangle türünden read-only ClientRectangle isimli property elemanı çalışma alanının konum ve boyutunu bize Rectangle yapısı olarak verir. Ancak property'nin verdiği Rectangle yine çalışma alanı orijinlidir. Yani bize verilen Rectangle'ın sol-üst köşesi her zaman sıfırdır.

Control sınıfının Color türünden BackColor isimli property elemanı pencerenin zemin rengini alıp set etmekte kullanılır.

Sınıf Çalışması: Ana pencerede bir satranç tahtasını düğmelerle oluşturunuz. Satranç tahtası 8x8 kareden oluşur. Kareler arasında boşluk yoktur ve sağ tarafta beyaz kare bulunur. Düğmelerin üzerinde bir yazı olmamalıdır. Beyaz kare rengi için R: 234, G: 240, B: 250 siyah kare rengi için de R:138, G:167, B:188 renklerini kullanınız.



Çözüm:

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

namespace BUUBM
{
public partial class Form1 : Form
    {
        private const int SquareSize = 90;
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            this.Text = "Button Chess Boards";
            this.ClientSize = new Size(SquareSize * 8, SquareSize * 8);

            int x;
            int y = 0;
            Color whiteColor = Color.FromArgb(234, 240, 250);
            Color blackColor = Color.FromArgb(138, 167, 188);
```

```
            for (int i = 0; i < 8; ++i)
            {
                x = 0;
                for (int k = 0; k < 8; ++k)
                {
                    Button button = new Button();
                    button.Bounds = new Rectangle(x, y, SquareSize, SquareSize);
                    button.BackColor = (i + k) % 2 == 0 ? whiteColor : blackColor;
                    this.Controls.Add(button);

                    x += SquareSize;
                }
                y += SquareSize;
            }

        }
    }}
```

Ana Pencerelerin İlk Boyutu ve Konumu

Ana pencerelere özgü olmak üzere bu pencereler açıldığında bunların konumu ve boyutu için Form sınıfının StartPosition isimli property'sine bakılır. Bu property FormStartPosition isimli bir enum türündendir. Bu enum'un elemanları ve anlamları şöledir:

Manual: Bu durumda ana pencerenin konumu da, boyutu da programcı tarafından ayarlanır.

CenterScreen: Burada ana pencerenin boyutunu programcı belirler ancak konumunu belirleyemez. Ana pencere ekranın tam ortasında olacak biçimde görüntülenir.

CenterParent: Burada ana pencere onun üst penceresinin ortasında görüntülenir. Fakat boyutunu programcı ayarlayabilir.

WindowsDefaultLocation: Burada ana pencerenin yerini Windows kendisi taktir eder. Fakat boyutunu biz belirleyebiliriz. Bu seçenek zaten property'nin default değeridir.

WindowsDefaultBounds: Bu durumda pencerenin boyutu ve konumu Windows'un kendisi tarafından belirlenir.

Ana Pencerenin Boyutlandırılması

Ana pencerenin boyutlandırılması için Form sınıfının FormBorderStyle isimli property elemanı kullanılmaktadır. Bu property ana pencerenin hem sınır çizgilerinin hem de boyutlandırılabilmesinin belirlenmesi için kullanılır. FromBorderStyle isimli property FormBorderStyle isimli bir enum türündendir. Bu enum türünün FixedXXX biçimindeki elemanları ana pencereyi boyutlandırılamaz hale yapmaktadır. Örneğin:

```
this.FormBorderStyle = FormBorderStyle.Fixed3D;
```

Tabii boyutlandırılamayan pencerelerde genellikle Maximize düğmesinin de kaldırılması istenir. İşte Form sınıfının bool türden MaximizeBox isimli property elemanı bu düğmeyi pasif hale getirir. Ayrıca Form sınıfının MinimizeBox isimli property elemanı da vardır.

.NET'te Mesajların İşlenmesi

Daha önceden de söz edildiği gibi, GUI programlama modelinde birtakım girdi olaylarına mesaj denir ve bunun takibini işletim sisteminin kendisi yapmaktadır. İşletim sistemi bir programı ilgilendiren bir olayı tespit ettiğinde onu mesaj kuyruğuna bir mesaj olarak kodlar. Application.Run metodu bir döngü içerisinde sıradaki mesajı alıp işlemektedir. Mesajların çok büyük çoğunluğu (neredeyse hepsi) bir pencereye yöneliktir.

Windows işletim sistemi ister ana pencere olsun isterse alt pencere olsun tüm pencereler için sistem genelinde tek olan bir HANDLE değeri atar. Pencereler üzerinde API düzeyinde işlemlerde bu HANDLE değeri kullanılmaktadır.

.NET ortamında ne zaman Control sınıfı türünden ya da ondan türetilmiş bir sınıf türünden bir nesne yaratılsa bir pencere yaratılmış olur. Yaratılan pencerenin sistem genelindeki HANDLE değeri Control sınıfının içerisinde de saklanmaktadır.

Windows işletim sisteminde kuyruğa bırakılan mesajın içeriğinde şunlar vardır:

- Mesaj hangi pencerede bir olay olmuş da kuyruğa bırakılmıştır? (Yani mesaj hangi pencereye yollanmıştır.) Windows işletim sistemi mesajın hangi pencereye gönderilmiş olduğunu mesajın içerisinde bulundurduğu handle değeri ile ifade eder.

- Mesaj ne amaçla gönderilmiştir? Yani ne olmuş da bu mesaj gönderilmiştir? Aşağı seviyede Microsoft her mesaja bir numara vermiştir. Fakat API programcıları bu numaralar yerine bunları temsil eden sözcükleri kullanırlar (WM_PAINT, WM_SIZE gibi).

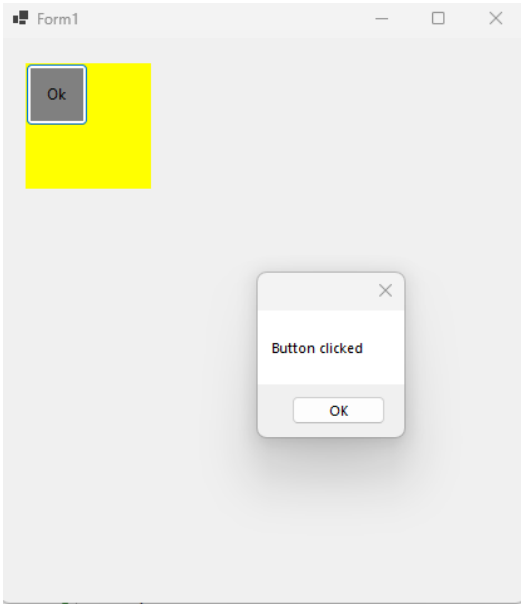
- Mesaja ilişkin ek birtakım başka bilgiler. (örneğin fare ile pencerenin neresine tıklanmıştır ya da hangi tuşa basılmıştır gibi.)

.NET’te mesaj işleminin iki yolu vardır:

1) Control sınıfından gelen protected OnXXX sanal metotlarını override etmek.
2) Control sınıfındaki XXX event elemanlarını kullanmak.

Biz .NET'te ne zaman bir pencere yaratsak .NET ortamı bizim pencereyi yaratmakta kullandığımız Control nesnesinin referansını bir collection'da saklar. Application.Run kuyuktan mesajı aldığında onun HANDLE değerine bakarak o pencerenin ilişkin olduğu Control nesnesinin referansını bulur. Sonra o kontrol referansıyla Control sınıfının mesajla ilgili olan protected virtual OnXXX metodunu çağırır. Her mesaj için Control sınıfında bir tane OnXXX metodu vardır. Örneğin fare ile pencereye

tıkladığımızda ve elimizi çektiğimizde bu oluşan mesaj için Application.Run OnClick isimli sanal metodu çağırmaktadır. O halde .NET'te mesaj işlemenin birinci yolu Control sınıfının bu OnXXX metotlarını override etmektir. Örneğin:




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

namespace CSD
{
    public partial class Form1 : Form
    {
        private MyControl m_mc;
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            m_mc = new MyControl();
            m_mc.Bounds = new Rectangle(20, 20, 100, 100);
            m_mc.BackColor = Color.Yellow;

            this.Controls.Add(m_mc);
        }

        protected override void OnClick(EventArgs e)
        {
            MessageBox.Show("Form clicked");
        }
```

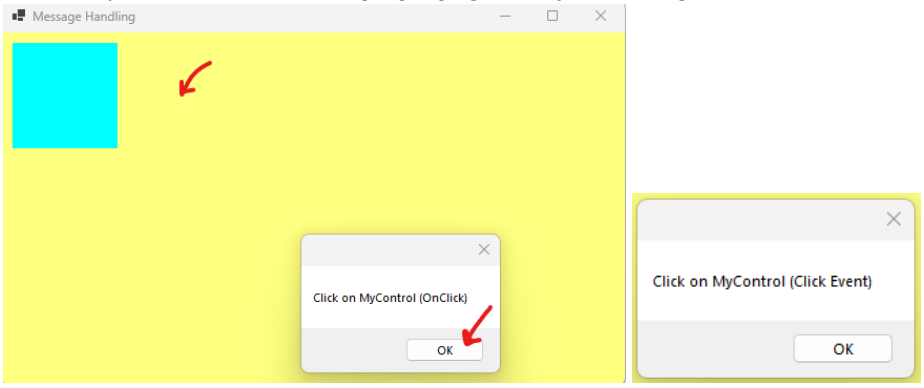
```
        //...
    }

    class MyControl : Control
    {
        private MyButton m_buttonOk;
        public MyControl()
        {
            m_buttonOk = new MyButton();
            m_buttonOk.Text = "Ok";
            m_buttonOk.Bounds = new Rectangle(0, 0, 50, 50);
            m_buttonOk.BackColor = Color.Gray;
            this.Controls.Add(m_buttonOk);
        }
        protected override void OnClick(EventArgs e)
        {
            MessageBox.Show("MyControl clicked");
        }
    }

    class MyButton: Button
    {
        protected override void OnClick(EventArgs e)
        {
            MessageBox.Show("Button clicked");
        }
    }
}
```

Pekiyi biz Control sınıfındaki OnXXX sanal metodunu override etmezsek ne olur? Bu durumda Control sınıfındaki OnXXX metodu çağrılır. O da XXX isimli event elemanın metotlarını çağırmaktadır. Contorl cınısında neredeyse her OnXXX metodu için bir tane XXX isimli event delege eleman da vardır. Biz doğrudan bu event elemana += operatörüyle metot ekleyebiliriz. Örneğin:


```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
```

```
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WinFormsApp1
{
    public partial class Form1 : Form
    {
        private MyControl m_myControl;
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            this.Text = "Message Handling";
            this.BackColor = Color.FromArgb(178, 255, 128);
            this.Click += new EventHandler(clickHandler);

            m_myControl = new MyControl();
            m_myControl.Bounds = new Rectangle(10, 10, 100, 100);
            this.Controls.Add(m_myControl);

        }

        private void clickHandler(object sender, EventArgs e)
        {
            MessageBox.Show("Clicked on Main Form");
        }
    }

    class MyControl: Control
    {
        public MyControl()
        {
            this.BackColor = Color.Cyan;
            this.Click += new EventHandler(clickHandler);
        }

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

Pekiyi her iki yöntemi de birlikte kullanabilir miyiz? Dikkatli olmak koşuluyla evet. Biz OnXXX sanal metodunu override ettiğimizde artık Control sınıfındaki OnXXX çağrılmamaktadır. Halbuki XXX event elemanını Control sınıfının OnXXX metodu tetikler. Eğer her iki yöntemi de uygulamak istiyorsak override edilen OnXXX içerisinde base anahtar sözcüğü ile taban sınıfın (yani Control sınıfının) OnClick metodu açıkça çağırmalıyız. Örneğin:




```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WinFormsApp1
{
    public partial class Form1 : Form
    {
        private MyControl m_myControl;
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            this.Text = "Message Handling";
            this.BackColor = Color.FromArgb(255, 255, 128);
            this.Click += new EventHandler(clickHandler);

            m_myControl = new MyControl();
            m_myControl.Bounds = new Rectangle(10, 10, 100, 100);
            this.Controls.Add(m_myControl);


        }
```

```
        protected override void OnClick(EventArgs e)
        {
            MessageBox.Show("Clicked on Main Form (OnClick)");
            base.OnClick(e);
        }

        private void clickHandler(object sender, EventArgs e)
        {
            MessageBox.Show("Clicked on Main Form (Click Event)");
        }

    }

    class MyControl: Control
    {
        public MyControl()
        {
            this.BackColor = Color.Cyan;
            this.Click += new EventHandler(clickHandler);
        }

        protected override void OnClick(EventArgs e)
        {
            MessageBox.Show("Click on MyControl (OnClick)");
            base.OnClick(e);
        }

        private void clickHandler(object sender, EventArgs e)
        {
            MessageBox.Show("Click on MyControl (Click Event)");
        }

    }
}
```

Control sınıfında ilgili kısım şöyle yazılmış olabilir (temsili kod (pseudo code))

```
    class Control
    {
        public event EventHandler Click;
        //...

        public virtual void OnClick()
        {
            if (Click != null)
                Click();
        }
        //...
```

```
    }
```

Tabii bizim mesaj işlemek için override tekniğini kullanabilmemiz için ilgili sınıfı bizim yazmamız ya da bir türetme uygulamamız gerekir. Örneğin bir düğme için Click mesajını işlemek istediğimizde Button sınıfını biz yazmadığımız için Onclick metodunu ancak ondan türetme yaparak override edebiliriz. Halbuki biz her zaman ilgili event elemana metot girip event yöntemini kullanabiliriz. Uygulamada mesaj işlemleri ağırlıklı olarak event yöntemiyle yapılmaktadır. Ancak bazı durumlarda override işlemi gerekebilmektedir.

Mesaj Parametre Sınıfları

Bazı mesajların ek birtakım bilgileri de vardır ve Windows işletim sistemi o bilgileri mesajın içerisinde kodlamaktadır. **Application.Run** metodu kuyruktan mesajı aldığında bu ek bilgileri bir sınıf nesnesinin içerisine yerleştirip OnXXX metodunu bu parametreyle çağırır. Biz de o bilgileri parametre ile verilen nesnenin içerisinden ilgili sınıfın property'leri yoluyla alırız.

.NET'teki mesaj parametre sınıflarının hepsi **EventArgs** isimli bir sınıftan türetilmiştir. **EventArgs** sınıfının içerisinde parametrik bir bilgi yoktur. Yani bu sınıf mesaj parametre sınıflarına taban sınıflık yapmaktadır ancak kendisinde bizim faydalanacağımız değerli bilgiler yoktur. Yani bir mesajın mesaj parametre sınıfı **EventArgs** ise biz o mesajın bize değerli bir parametrik bilgi vermediğini anlamalıyız.

Mesaj oluştuğunda çağrılan Control sınıfının OnXXX metotlarının tek bir parametresi vardır. O da mesaj parametre sınıfı türündendir. Örneğin OnClick metodunun parametrik yapısı şöyledir:

```
protected virtual void OnClick(EventArgs e)
```

OnMouseMove isimli metodun parametrik yapısı ise şöyledir:

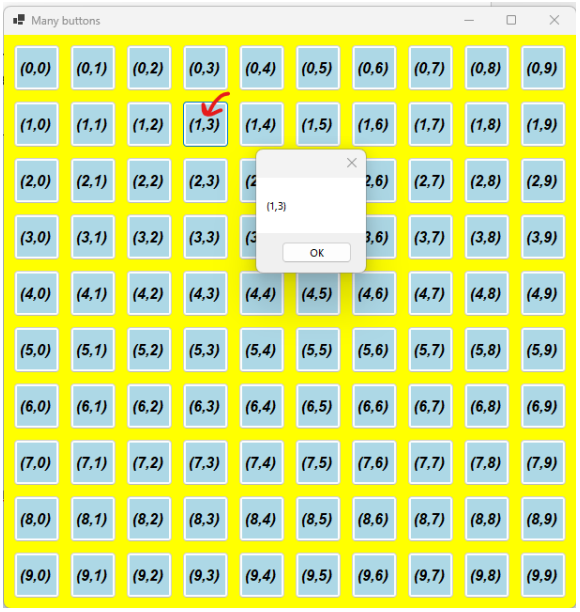
```
protected virtual void OnMouseMove(MouseEventArgs e)
```

Control Sınıfının OnXXX Metotları Tarafından Tetiklenen Event Elemanları

Anımsanacağı gibi Control sınıfında her **OnXXX** metodu için onun tetiklediği bir XXX event elemanı vardır. Örneğin OnClick metodu Click isimli event elemanını tetikler. **OnMouseMove** metodu ise MouseMove isimli event elemanını tetikler. Pekiyi bu event elemanlar hangi delege türündendir? İşte genel olarak bu event elemanlar YYYEventHandler biçiminde isimlendirilmiş bir delege türündendir. Tüm YYYEventHandler isimli delege türlerinin geri dönüş değeri void biçimdedir. Ve bunların hepsinin iki parametresi vardır. Bu iki parametrenin birincisi object türündedir. İkincisi ise OnXXX metodununa geçirilen mesaj parametre sınıfı türündendir. Örneğin OnMouseMove metodunun tetiklediği MouseMove event elemanı MouseEventHandler isimli delege türündendir. Bunun birinci parametresi object türünden ikinci parametresi MouseEventArgs türündendir. MouseEventArgs OnMouseMove metoduna geçirilen mesaj parametre sınıfıdır.

EventHandler isimli deleginin mesaj parametre sınıfı EventArgs sınıfıdır. Yani EventHandler delegesine ilişkin mesajlar bize başka bir bilgi vermezler.

Delegelerin birinci object parametreleri mesaj hangi kontrol nedeniyle gönderilmişse yani hangi kontrole ilişkinse onun referansını içerir. Böylece çeşitli kontrollere aynı delege metodunu verebiliriz. Bu parametre sayesinde olayın hangi kontrole ilişkin olduğunu anlayabiliriz. Örneğin:




```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WinFormsApp1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            this.Text = "Many buttons";
            this.ClientSize = new Size(610, 610);
            this.BackColor = Color.Yellow;

            int x;
```

```
            int y = 10;
            for (int i = 0; i < 10; ++i)
            {
                x = 10;
                for (int k = 0; k < 10; ++k)
                {
                    Button button = new Button();
                    button.Font = new Font("Arial", 12, FontStyle.Bold |
FontStyle.Italic);
                    button.Text = string.Format("({0},{1})", i, k);
                    button.Bounds = new Rectangle(x, y, 50, 50);
                    button.BackColor = Color.LightBlue;
                    button.Click += new EventHandler(clickHandler);
                    this.Controls.Add(button);

                    x += 60;
                }
                y += 60;
            }
        }
        private void clickHandler(object sender, EventArgs e)
        {
            Button button = (Button)sender;

            MessageBox.Show(button.Text);
        }
    }
}
```

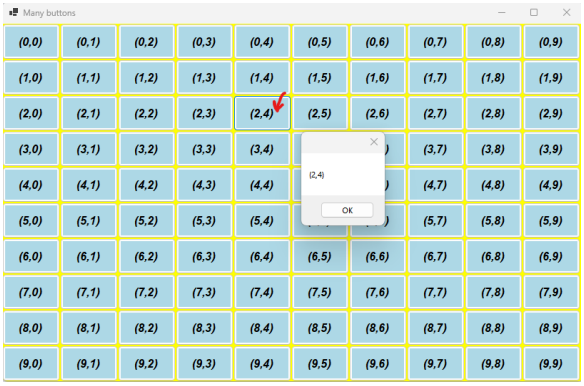
Resize Mesajı

Bir pencerenein boyutu değiştirildiğinde Windows kuyruğa Resize mesajını (orijinali WM_SIZE) bırakır. Bunun için **Application**.Run Control sınıfının OnResize sanal metodunu çağırır. Bu metot da Resize isimli event'i tetikler. Resize event'i EventHandler isimli delege türündendir. Mesajın parametre sınıfı EventArgs sınıfıdır. Yani bu mesaj bize ilave bilgi vermez. Ancak mesaj oluştuğunda artık pencerenin genişlik ve yükseklik değerleri zaten güncellenmiş durumdadır.

Resize mesajı pencere boyutunu değiştirirken sürekli gelmektedir. Aslında bu ayar Windows 10'da "Denetim Masası/Sistem/Gelişmiş Sistem Ayarları/Performans/Göresel Effektler/Sürüklerken Pencere İçeriğini Göster" seçenek kutusu ile değiştirilebilmektedir. Bu seçenek kutusunun çarpısı kaldırılırsa pencere boyutlandırıldığında yalnızca bir kez Resize mesajı oluşmaktadır.

Ana pencere ya da diğer pencereler ilk kez açıldığında Resize mesajı oluşmamaktadır (Halbuki Windows aslında pencereler ilk göründüğünde WM_RESIZE mesajını pencereye göndermektedir.)

Sınıf Çalışması: Bir tane düğme yaratınız. Öyle ki düğme ana pencerenin çalışma alanını kaplamasın her köşeden 10 pixel boşluk kalsın. Pencere Resize edildiğinde yine her köşeden 10 pixel boşluk devam etsin.




```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WinFormsApp1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            this.Text = "Many buttons";
            this.ClientSize = new Size(500, 500);
            this.BackColor = Color.Yellow;
            this.Resize += new EventHandler(resizeHandler);

            int x;
            int y = 0;
            for (int i = 0; i < 10; ++i)
            {
                x = 0;
                for (int k = 0; k < 10; ++k)
                {
                    Button button = new Button();
```

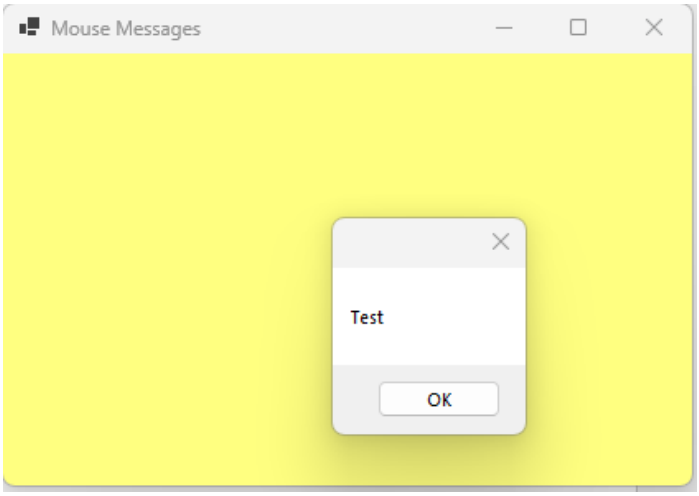
```
                    button.Font = new Font("Arial", 12, FontStyle.Bold |
FontStyle.Italic);
                    button.Text = string.Format("({0},{1})", i, k);
                    button.Bounds = new Rectangle(x, y, 50, 50);
                    button.BackColor = Color.LightBlue;
                    button.Click += new EventHandler(clickHandler);
                    this.Controls.Add(button);

                    x += 50;
                }
                y += 50;
            }
        }

        private void clickHandler(object sender, EventArgs e)
        {
            Button button = (Button)sender;

            MessageBox.Show(button.Text);
        }
        private void resizeHandler(object sender, EventArgs e)
        {
            int newWidth = this.ClientSize.Width / 10;
            int newHeight = this.ClientSize.Height / 10;
            //int rowleftSpace = this.ClientSize.Height % 10;
            //int colleftSpace = this.ClientSize.Width % 10;

            int x;
            int y = 0;
            for (int i = 0; i < 10; ++i)
            {
                x = 0;
                for (int k = 0; k < 10; ++k)
                {
                    this.Controls[i * 10 + k].Location = new Point(x, y);
                    this.Controls[i * 10 + k].Size = new Size(newWidth, newHeight);

                    x += newWidth;
                }
                y += newHeight;
            }
        }
    }
}
```

Temel Fare Mesajları

Fare ile ilgili tipik mesajlar aşağıda açıklanmaktadır.

Click Mesajı

Bu mesaj pencerenin içerisinde farenin herhangi tuşuna basılıp yine pencerenin içerisinde çekildiğinde tetiklenir. Bu mesaj için Application.Run OnClick sanal metodunu çağırır. Bu metot da Click event'ini tetikler. Click event'i EventHandler isimli delege türündendir. Bu mesaja ilşkin mesaj parametre sınıfı EventArgs isimli sınıftır. Yani mesaj bize ilave hiçbir bilgi vermez. (Örneğin farenin nerede tıklandığı, hangi tuşa basıldığı gibi). Örneğin:




```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WinFormsApp1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            this.Text = "Mouse Messages";
            this.BackColor = Color.FromArgb(255, 255, 128);
            this.Click += new EventHandler(mouseClickHandler);
        }
        private void mouseClickHandler(object sender, EventArgs e)
        {
            MessageBox.Show("Test");
```

```
        }

    }
}
```

MouseClick Mesajı

Bu mesaj farenin bir düğmesine basılıp çekildiğinde tetiklenir. Bunun için Application.Run Control sınıfının OnMouseClick sanal metodunu çağırır. Bu metot da MouseClick isimli event'i tetikler. MouseClick event'i MouseEventHandler isimli bir delege türündendir. Mesajın parametre sınıfı da MouseEventArgs sınıfıdır. Mesajın Click mesajından farkı farenin çekim koordinatlarını ve basılan tuşu bize vermesidir. MouseEventArgs sınıfının Location isimli property elemanı Point yapısı türündendir. X ve Y isimli property elemanları da int türdendir. Bu property elemanlar bize farenin hangi pozisyonda tıklandığını (elin fareden çekildiği pozisyonu) çalışma alanı orijinli olarak verir. MouseEventArgs sınıfının Button isimli property elemanı ise MouseButtons isimli enum türündendir. Bu property bize hangi tuşa bastığımız bilgisini verir.

Sınıf Çalışması: Bir ana pencere oluşturunuz. Sonra MouseClick mesajını işleyerek pencere içerisine tıklandığında tam tıklanılan yer ortada kalacak biçimde 50X50'lik bir düğme yaratınız. Düğmelerin üzerine sırasıyla 1, 2, 3, gibi sayıları yazdırınız.

Alt pencerelerin sınırları kesiştiğinde hangisi daha yukarıda görünür? Buna Windows'ta Z sırası (Z order) denilmektedir. Üst pencerenin Controls collection elemanında hangi alt pencere daha gerideyse o daha yukarıda görüntülenmektedir. Bu işi yapan Control sınıfının BringToFront isimli metodu da vardır. Bu metod ilgiliyi pencere nesnesini Controls collection elemanının önüne taşır. Benzer biçimde Control sınıfının SendToBack metodu da nesneyi Controls collection elemanının en gerisine alır.

Sınıf Çalışması: Ana pencere içerisinde 100x100 lük 10 tane düğmeyi aynı koordinatta ve büyüklükte üst üste gelecek biçimde oluşturunuz. Düğmelerin üzerinde sırasıyla 1'den 10'a kadar sayılar bulunmalıdır. (Başlangıçta en üstte 10 numaralı düğme olsun). Hep en üsttekine tıklandığında bu en üstteki düğme en alta geçsin.

MouseDown Mesajı

Bu mesaj fareyle çalışma alanın içerisinde tıklandığında oluşturulur. Bu mesaj için Application.Run metodu OnMouseDown sanal metodunu çağırır. Bu metot da MouseDown isimli event'i tetikler. MouseDown isimli event MouseEventHandler isimli delege türündendir. Mesajın parametre sınıfı yine MouseEventArgs sınıfıdır.

Sınıf Çalışması: Bir ana pencere oluşturunuz. Sonra MouseClick mesajını işleyerek pencere içerisine tıklandığında tam tıklanılan yer ortada kalacak biçimde 50X50'lik bir düğme yaratınız. Düğmelerin üzerine sırasıyla 1, 2, 3, gibi sayılar bulunsun. Düğmelerin üzerine tıklandığında (düğmelerin Click event'ini işleyeceksiniz) tıklanan düğme yok olsun. Kontrolü yok etmek için Control sınıfının Dispose metodu kullanılmaktadır.

Sınıf Çalışması: Yukarıdaki çalışmayı öyle bir hale getiriniz ki yeni bir düğme yaratılacağı zaman bu yeni yaratılacak düğme silinen en düşük numaralı düğmenin numarasını alsın.

MouseUp Mesajı

Bu mesaj farenin tuşundan el çekilince oluşturulur. Mesaj için Application.Run metodu Control sınıfının OnMouseUp sanal metodunu çağırır. Bu metot da MouseUp event'ini tetikler. MouseUp event'i yine MouseDown event'inde olduğuğu gibi MouseEventHandler isimli delege türündendir. Dolayısıyla mesajın parametre sınıfı yine MouseEventArgs sınıfıdır.

Sınıf Çalışması: Fare ile tıkladığınız zaman MouseDown mesajında farenin konumunu kaydediniz. Sonra fareyi hareket ettirip elinizi fareden çekince MouseUp mesajında yeni yeri elde ediniz. Sol üst köşesi eski yerde sağ alt köşesi yeni yerde olacak biçimde düğme yaratınız.

Fare ile farenin düğmesine basıp elimizi çektiğimizde mesajlar şu sırada oluşur:

(Elimizle farenin tuşuna bastık)
MouseDown
(Elimizi farenin tuşundan çektik)
Click
MouseClick
MouseUp

DoubleClick ve MouseDoubleClick Mesajları

Bu mesajlar pencere içerisinde çift tıklandığı zaman gönderilir. Bunlar için sırasıyla Control sınıfının OnDoubleClick ve OnMouseDoubleClick sanal metotları çağrılmaktadır. Bu sanal metotlar da DoubleClick ve MouseDoubleClick isimli event'leri tetikler. Mesajın parametre sınıfları DoubleClick için EventArgs, MouseDoubleClick için MouseEventArgs sınıfıdır.

Bir çift tıklama eyleminde yalnızca DoubleClick mesajları oluşmaz. Çünkü eyleme başlandığında sistem bizim çift tıklamaya niyetlendiğimizi anlayamaz. Çift tıklama eyleminde mesajlar şu sırada oluşmaktadır:

MouseDown
Click
MouseClick
MouseUp
MouseDown
DoubleClick
MouseDoubleClick
MouseUp

Görüldüğü gibi çift tıklama sürecinin ikinci tıklamasında artık Click mesajı oluşmamakta bunun yerine DoubleClick mesajları oluşmaktadır.

MouseMove Mesajı

Fareyi çalışma alanı içerisinde harek ettirdiğimizde MouseMove mesajları oluşur. Mesajın hangi yoğunlukta oluşturulacağı sistemin o anda içinde bulunduğu duruma göre ve fare hareketinin hızına göre değişebilir. Tabi fare durunca böyle bir mesaj oluşmaz. Mesaj için Application.Run Control sınıfının OnMouseMove sanal metodunu çağırır. Bu sanal metot MouseMove isimli event'i tetikler. MouseMove event'i yine MouseEventHandler isimli delege türündendir ve mesajın parametre sınıfı MouseEventArgs sınıfıdır.

Sınıf Çalışması: Ana pnecereye farenin sol tuşuyla tıkladığımızda bir düğme yaratınız. Sonra fareyi sürklediğimizde o düğme büyüyüp küçülsün. El fareden çekilince düğme o boyutta kalsın. Düğmenin üzerinde de yine düğmenin yaratılma numarası basılsın.

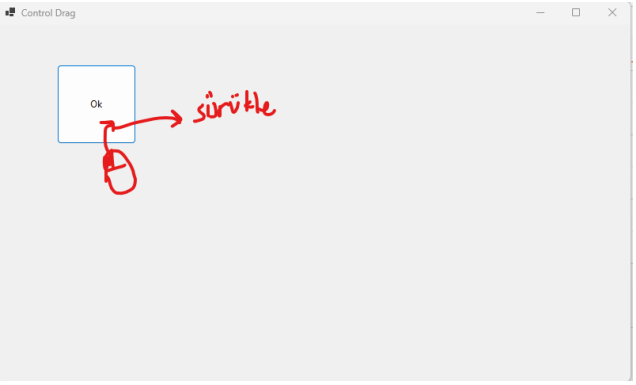
İp Ucu: MouseDown, MouseMove ve Mouse mesajları işlenmelidir. Fare hareket ettirildikçe düğmenin konumu değişmeyecektir. Yalnızca boyutu değişecektir.

Pencerelerin Sürüklenmesi

Penecereleri fare taşımak oldukça kolaydır. Bunun için kontrolün MouseDown, MouseUp ve MouseMove mesajları işlenir. Her MouseMove mesajında farenin ne kadar kaydırıldığı deltaX, deltaY biçiminde hesaplanır.

Sonra kontrol o kadar ötelinir. Kontrol ötelenince artık göreli olarak basım noktası yine aynı yere gelmiş olur.

Anahtar Notlar: Bir değişkenin değerini farklı mesaj metotlarında kullanmaya devam etmek istiyorsak bu değişkeni sınıfın veri elemanı olarak bildirmemiz gerekir.



```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WinFormsApp1
{
    public partial class Form1 : Form
    {
        private bool m_flag;
        private Button m_buttonOk;
        private Point m_firstPoint;

        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            this.Text = "Control Drag";

            m_buttonOk = new Button();
            m_buttonOk.Bounds = new Rectangle(50, 50, 100, 100);
            m_buttonOk.Text = "Ok";
            m_buttonOk.Parent = this;

            m_buttonOk.MouseDown += new MouseEventHandler(mouseDownHandler);
            m_buttonOk.MouseUp += new MouseEventHandler(mouseUpHandler);
            m_buttonOk.MouseMove += new MouseEventHandler(mousemoveHandler);

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

        private void mouseUpHandler(object sender, MouseEventArgs e)
        {
            m_flag = false;
        }

        private void mousemoveHandler(object sender, MouseEventArgs e)
```

```
        {
            if (m_flag)
            {
                int dx = e.X - m_firstPoint.X;
                int dy = e.Y - m_firstPoint.Y;

                m_buttonOk.Top += dy;
                m_buttonOk.Left += dx;
            }
        }
    }
}
```