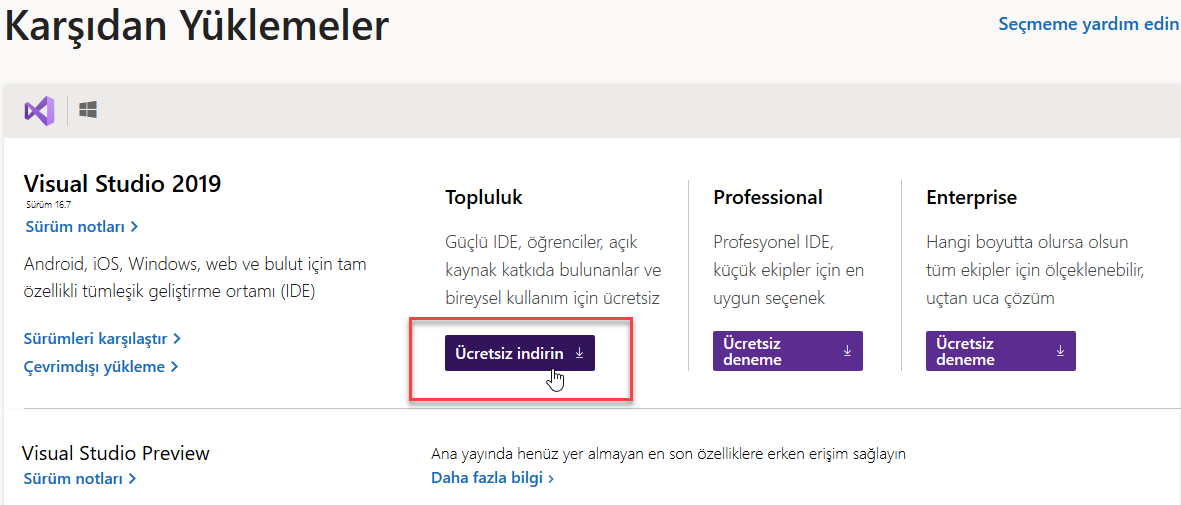
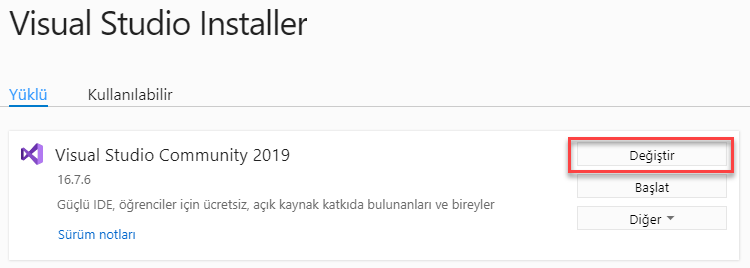
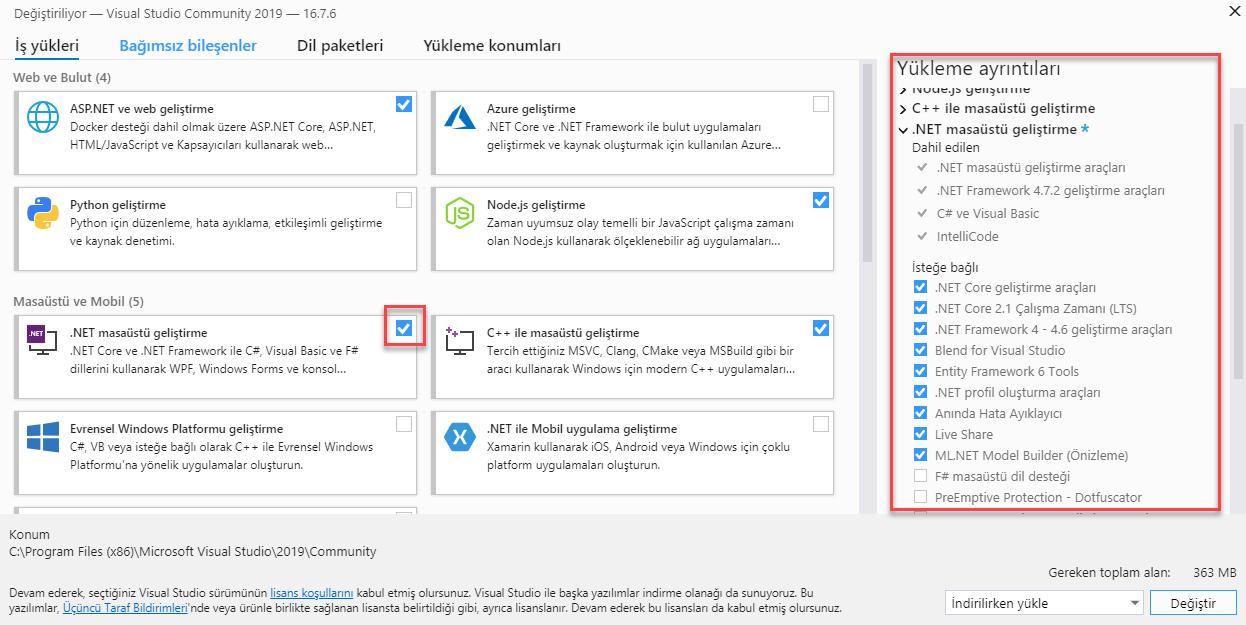
1. **İndirme**

**Adres : https://visualstudio.microsoft.com/tr/downloads/**

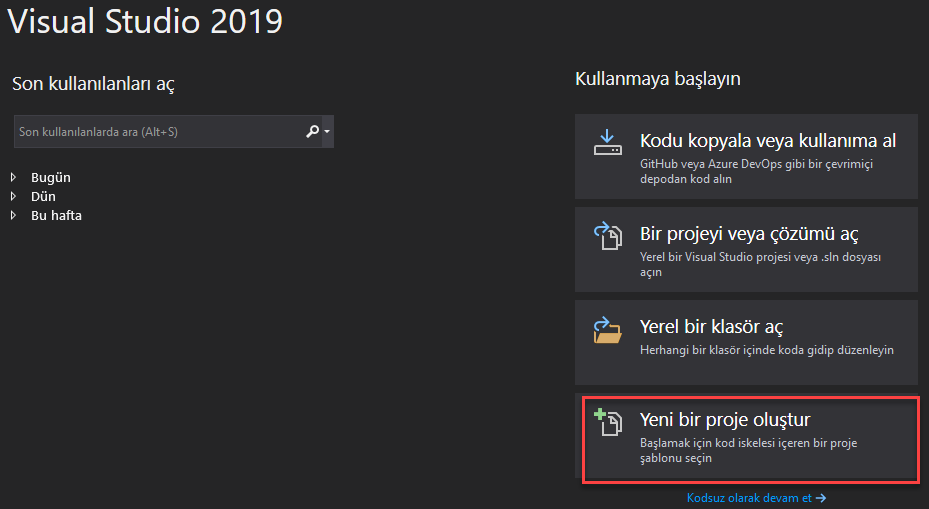


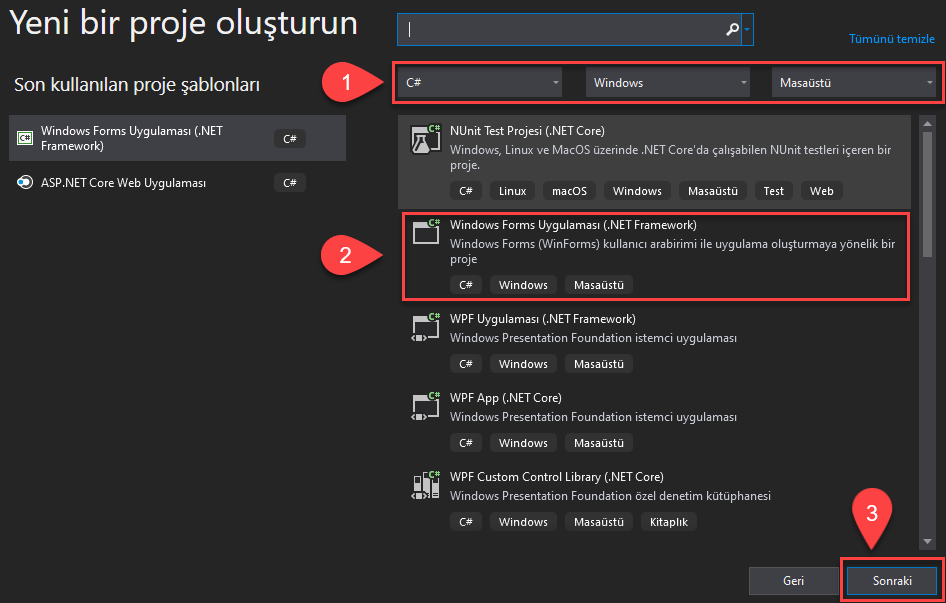
**Kurulum sonrasında:**

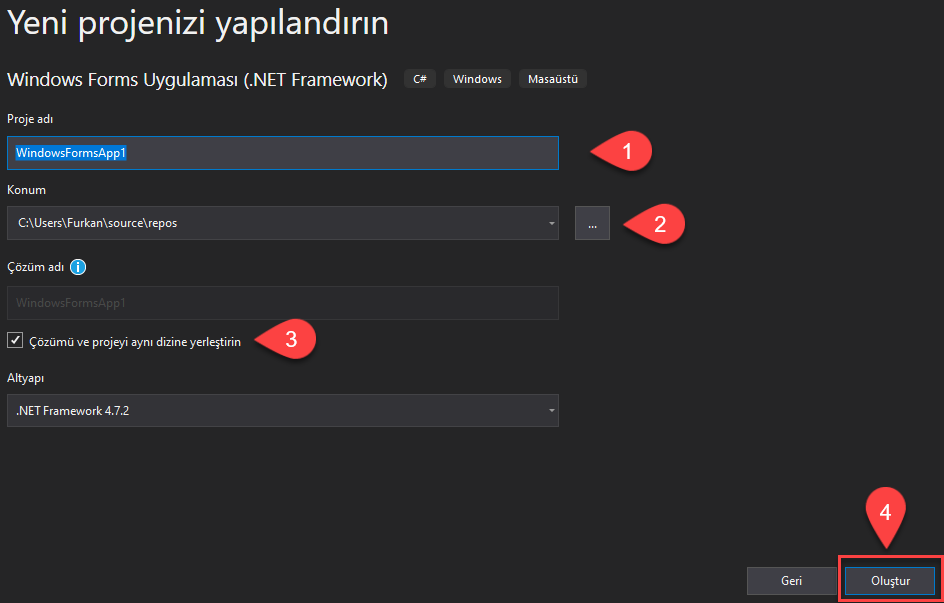


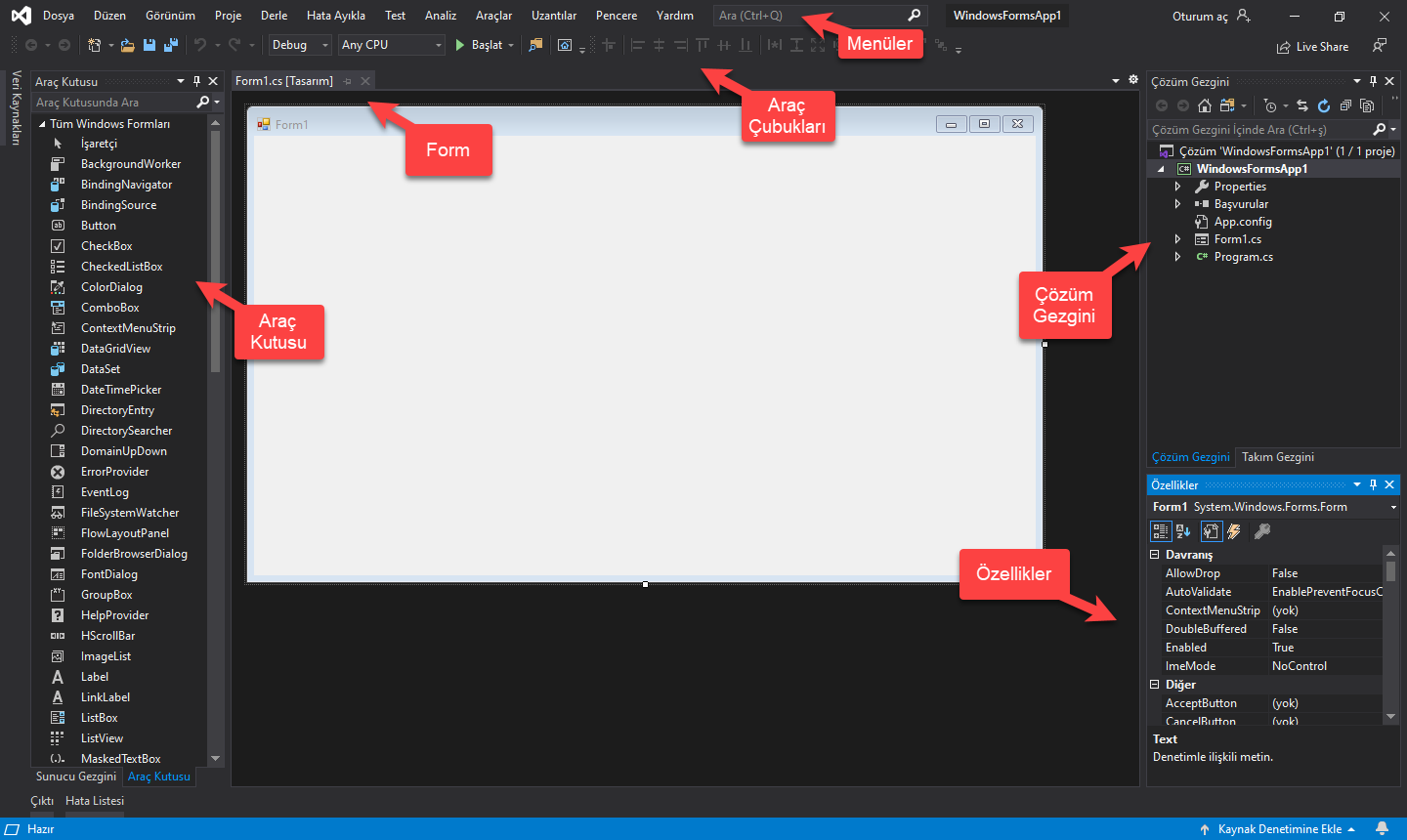


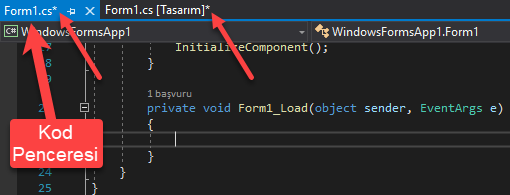
1. **Proje Oluşturma**



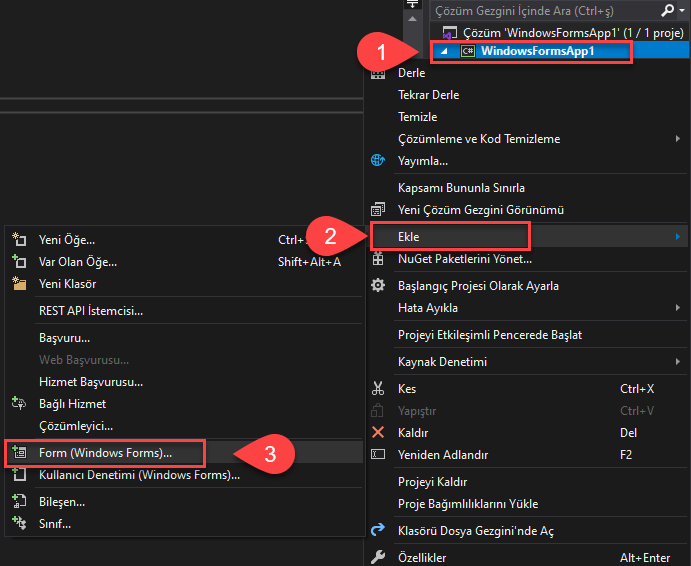


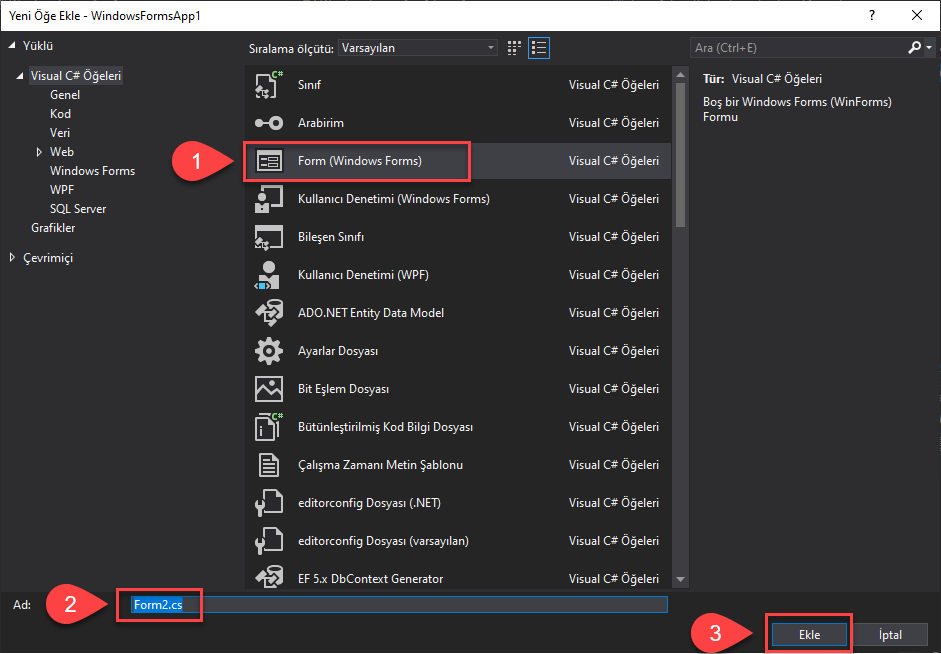






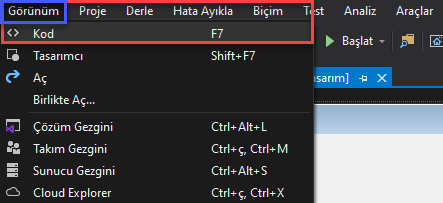
1. **Windows Formlar**





Windows formlar iki başlık altında incelenebilir. SDI (Single Document Interface) ve MDI (Multiple Document Interface) Formlar.

**SDI :** Başka bir forma ihtiyaç duymadan tek başına çalışabilen formlara denir. Projelerin açılışında gelen formlar bu tür formlardır. Görünüm -> Kod yolu ile veya F7 tuşu ile bu forma ait kod sayfasına geçilir.



Bir formdan diğerine geçiş yapabilmek için aşağıdaki örneği inceleyelim. Form1’e bir düğme eklenmiş ve bu düğme çift tıklanarak kod sayfasına geçilmiştir. Bu kısımda aşağıdaki kod yazılmıştır.

private void button1\_Click(object sender, EventArgs e)

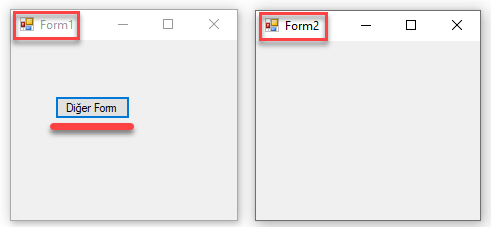
{

Form2 c = new Form2();

form2.Show();

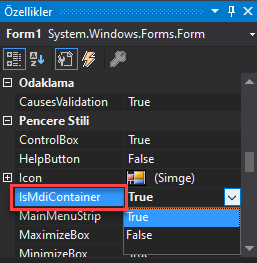
}

Program derlendikten sonra Form1’deki ‘Diğer Form’ düğmesi tıklandığında Form2 açılır. Ekran görüntüsü aşağıdaki şekildedir.

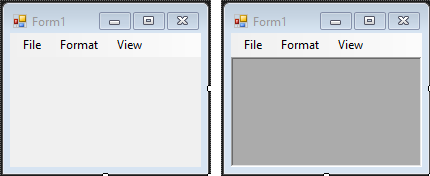


**MDI :** Bir ana pencere altında, bu ana pencereye bağlı olarak çalıştırılabilen çocuk formlardır. Bir veya birden fazla çocuk form olabilir.

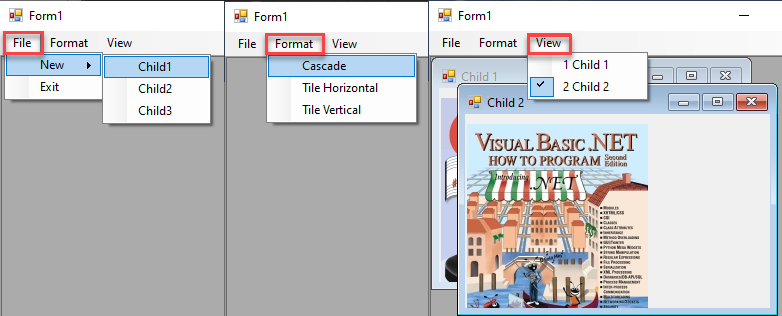
Bir formun MDI olabilmesi için ‘IsMdiContainer’ özelliğinin ‘True’ olarak değiştirilmesi gerekiyor.



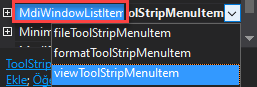
False ve True olma durumları aşağıdaki şekilde görülür.



True yapıldıktan sonra menuStrip1 bileşeni eklenerek aşağıdaki şekilde menüler ayarlanır.



Açık olan çocuk formlar listesinin görüleceği menü menuStrip1 bileşeninin MdiWindowListItem özelliğinden seçilerek belirlenir.



Form1 için yazılması gerek kodlar aşağıda belirtilmiştir.

namespace WindowsFormsSdiMdi

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void exitToolStripMenuItem\_Click(object sender, EventArgs e)

{

Application.Exit();

}

private void child1ToolStripMenuItem\_Click(object sender, EventArgs e)

{

Form2 formChild = new Form2("Child 1", "\\images\\csharphtp1.jpg");

formChild.MdiParent = this;

formChild.Show(); // Çocuk Form Aç

}

private void child2ToolStripMenuItem\_Click(object sender, EventArgs e)

{

Form2 formChild = new Form2("Child 2", "\\images\\vbnethtp2.jpg");

formChild.MdiParent = this;

formChild.Show(); // Çocuk Form Aç

}

private void child3ToolStripMenuItem\_Click(object sender, EventArgs e)

{

Form2 formChild = new Form2("Child 3", "\\images\\pythonhtp1.jpg");

formChild.MdiParent = this;

formChild.Show(); // Çocuk Form Aç

}

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 button2\_Click(object sender, EventArgs e)

{

}

}

}

Form2 üzerine PictureBox bileşeni eklenmiştir. Form2 için yazılması gerek kodlar aşağıda belirtilmiştir.

namespace WindowsFormsSdiMdi

{

public partial class Form2 : Form

{

public Form2(string title, string fileName)

{

InitializeComponent();

Text = title; // Başlık

// PictureBox da gösterilecek

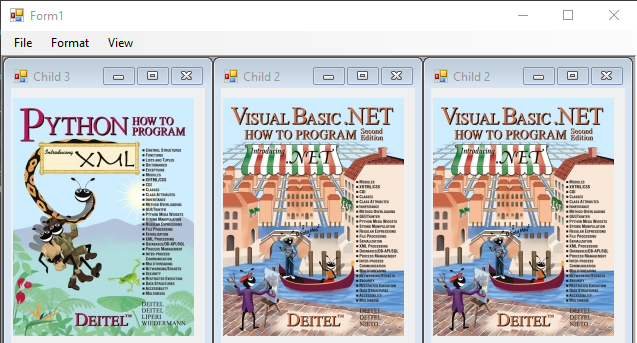
pictureBox1.Image = Image.FromFile(Directory.GetCurrentDirectory() + fileName);

}

}

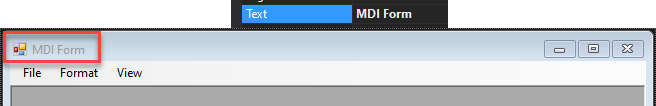
}

File menüsünden New -> Child1 , Child2 ve Child3 tıklanarak 3 çocuk penceresi açılır. Format menüsünden Vertical seçeneği tıklandığındaki ekran görünütüsü aşağıdaki şekildedir.

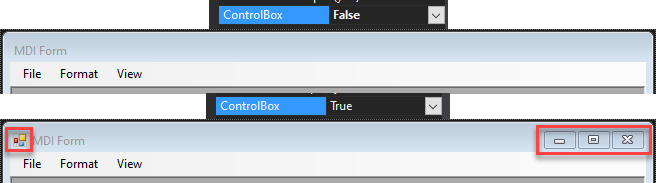


**Form Özellikleri**

**Text :** Formun başlığını değiştirir.



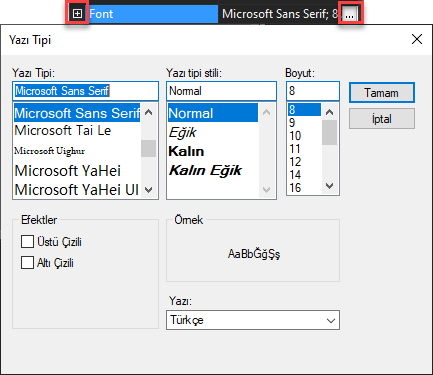
**ControlBox :** Formun sol üst köşesindeki menüyü ve sağ üst köşesindeki düğmeleri kontrol eder.



**Enabled :** Form üzerindeki nesneler ile işlem yapılıp yapılmayacağını belirler. True ise nesnelerde işlem yapılabilir. False ise tüm nesneler pasif durumdadır.



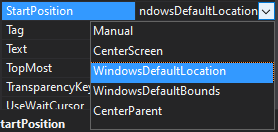
**Font :** Bu özellik ile form üzerindeki nesnelere ait text özellikleri, form üzerindeki yazı stilleri, fontu ve boyutu istenilen ölçüde ayarlanır ve o parametreler doğrultusunda görüntülenmesi sağlanır.



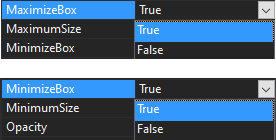
**Visible :** Formun ekranda görünürlülüğünü ayarlar. Özellikler panelinde bulunmamaktadır. Aşağıdaki kod satırı ile özellik değeri değiştirilebilir.

this.Visible = false;

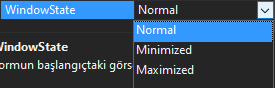
**StartPozition:** Formun ekranın neresinde konumlanacağını ayarlar. **Manuel** seçilirse location özelliği verilerek form konumlandırılır. **CenterScreen** özelliği formu ekranın ortasında konumlandırır. **WindowsDefaultLocation** özelliği formu ekranda Windows’un belirlediği bir konumda gösterir. **WindowsDefaultBounds** özelliği formu Windows uygulamaları için standart bir boyutta varsayılan pozisyonda konumlandırır. **CenterParent** özelliği formu kendini çalıştıracak olan parent formun ortasında konumlandırır. Birbirleriyle ilişkili olan MDI formlar için kullanışlıdır.



**MaximizeBox, MinimizeBox :** Form üzerindeki ekranı kapla ve simge durumuna küçült düğmelerinin görünürlüğünü ayarlar.



**WindowState :** Form ilk açıldığında nasıl açılması gerektiğini ayarlar.



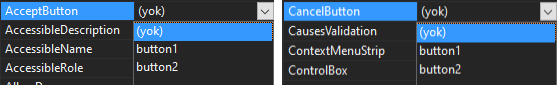
**Opacity :** Formun saydamlığını ayarlar. 0-100 arasında değer alır.



**ShowInTaskbar :** Bu özellik projemizin simgesinin çalışma esnasında görev çubuğu üzerinde bulunup bulunmayacağını ayarlamamıza imkân sağlar.



**AcceptButton ve CancelButton :** Enter tuşunu onay vermek için, Esc tuşunu iptal etmek için kullanılacak butonları atamaya yarar.



namespace WindowsFormsAcceptCancelButton

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

MessageBox.Show("Onay verildi");

}

private void button2\_Click(object sender, EventArgs e)

{

MessageBox.Show("İptal Edildi");

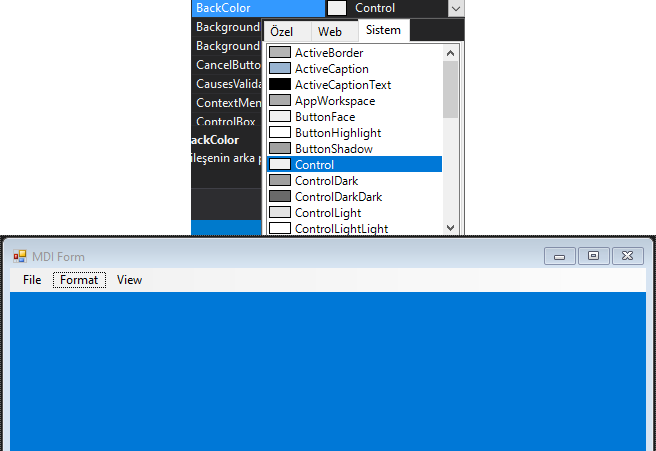
}

}

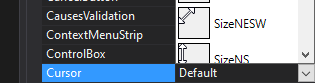
}

Program çalıştırıldıktan sonra Enter tuşuna basıldığında “Onay Verildi” mesajı, Esc tuşuna basıldığında “İptal Edildi” mesajı gösterilir.

**BackColor :** Formun arka plan rengini değiştirir.



**Cursor :** Form üzerinde fare gezinirken istediğimiz şekli almasını sağlar.



1. **Görsel Programlama**

Resim ve grafiksel ögeler ile yapılan bilgisayar programlamasına görsel  
programlama denir. Görsel programlama dilleri program kodunun tamamının ya  
da bir kısmının görsel araçlar ile üretildiği dillerdir.  
Görsel programlama sayesinde bilgisayarlar daha anlaşılır, daha kolay  
kullanımlı, daha göze hitap eden bir hale gelmiştir.  
Windows tabanlı sistemler için Visual Basic, Visual C++, C#, Java gibi diller  
görsel programlama dilleri olarak kullanılabilmektedir.  
Görsel programlama ile bilgisayar işletim sistemleri (Windows, Linux,  
MacOS), aktif web siteleri, cep telefonu uygulamaları, tablet uygulamaları  
geliştirilmektedir.

**.Net Framework Nedir?**

.Net Framework, Microsoft tarafından geliştirilen, açık İnternet  
protokolleri ve standartları üzerine kurulmuş bir "uygulama" geliştirme  
platformudur. Visual Studio.Net ile geliştirilen uygulamaların çalışması için .Net Framework’e ihtiyaç duyulur. Bu yüzden kullanım amacı olarak Java Virtual Machine uygulamasına benzetilebilir. .Net Framework ile Visual Studio.Net platformunda geliştirilen uygulamaların farklı bilgisayarlarda veya aygıtlarda çalışabilmesi sağlanmaktadır. Visual Studio ile geliştirilen uygulamaların ihtiyaç duyduğu tüm kütüphaneler .Net Framework’te tanımlıdır. Kısacası .Net Framework olmadan Visual Studio.Net ile geliştirilen uygulamalar çalışmaz. Bu sebeplerden ötürü .Net Framework birçalıştırma ortamı (runtime environment) olarak adlandırılmaktadır.

**Visual Studio Nedir?**

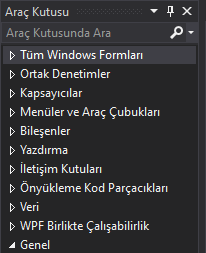
Microsoft tarafından 2000’li yılların başlarında kullanıma sunulan Microsoft  
Visual Studio ürünü, programcıların her türlü yazılım geliştirme ihtiyacını  
karşılamak amacıyla geliştirilen bir yazılım geliştirme platformu olarak  
tanımlanmaktadır. Sağlamış olduğu tümleşik geliştirme ortamı(IDE) sayesinde programcılara yazılım geliştirirken büyük kolaylıklar sağlamaktadır. Visual Studio’da geliştirilen yazılımlar Microsoft Windows, Windows Mobile, Windows CE, .Net Framework, .Net Compact Framework ve Microsoft Silverlight tarafından desteklenen tüm platformlarda çalışır. Bir başka ifadeyle birçok farklı teknoloji ve platform tarafından desteklenmektedir. Visual Studio içinde bulunan Windows Forms uygulamaları, web sayfaları ve  
web servisleri ile konsol ve grafiksel kullanıcı ara yüzüne sahip uygulamalar geliştirilebilmektedir. Bunun yanında aynı projede farklı programlama dilleri ile kod yazabilir, birçok türde veri kaynağına erişim sağlayabilir ve burada sayamadığımız birçok işlemi çok daha hızlı, kolay ve güvenilir bir şekilde gerçekleştirebilmektedir. Bütün bu sebeplerden ötürü Visual Studio günümüzde en çok tercih edilen yazılım geliştirme platformlarından birisidir.

**Kontrol Nesneleri**

Windows işletim sistemindeki pencereler, Visual Studio’da Form olarak  
adlandırılmaktadır. Toolbox’ı kullanarak forma eklediğimiz her kontrol (buton, yazı kutusu), kontrol nesnesi olarak adlandırılmaktadır.

Bütün formlar ve onlara eklediğimiz her kontrol birer nesnedir ve bu nesnelerin özellikleri, olayları ve metotları vardır.

**Formlar ve Kontroller**

Visual Studio’nun içerisinde varsayılan olarak bulunmakta olan  
kontrollerdir. Neredeyse bütün yazılımlarda kullanılan, temel ve en çok ihtiyaçduyulan kontrolleri barındırır.

• Tüm Windows Formları: Bütün sekmelerdeki kontrol nesnelerin topluca bulunduğu sekmedir.

• Ortak Denetimler: En sık kullanılan nesnelerin bulunduğu sekmedir.

• Kapsayıcılar: Kapsayıcılar birer nesne oldukları gibi içerilerine de nesneler eklenebilir. Formlar da kapsayıcılara örnektir ancak buradan eklenmezler.

• Menüler ve Araç Çubukalrı: Arayüze menü ve araç çubuğu nesnelerinin eklendiği sekmedir.

• Bileşenler: Arka planda çalışan önemli bileşenlerini barındırır. Bu sekmedeki nesneler eklendiğinde görsel arayüzde bulunmaz ancak “Design” sekmesinin alt kısmında görülebilir.

• Yazdırma: Yazdırma işlemiyle ilgili nesnelerin bulunduğu sekmedir.

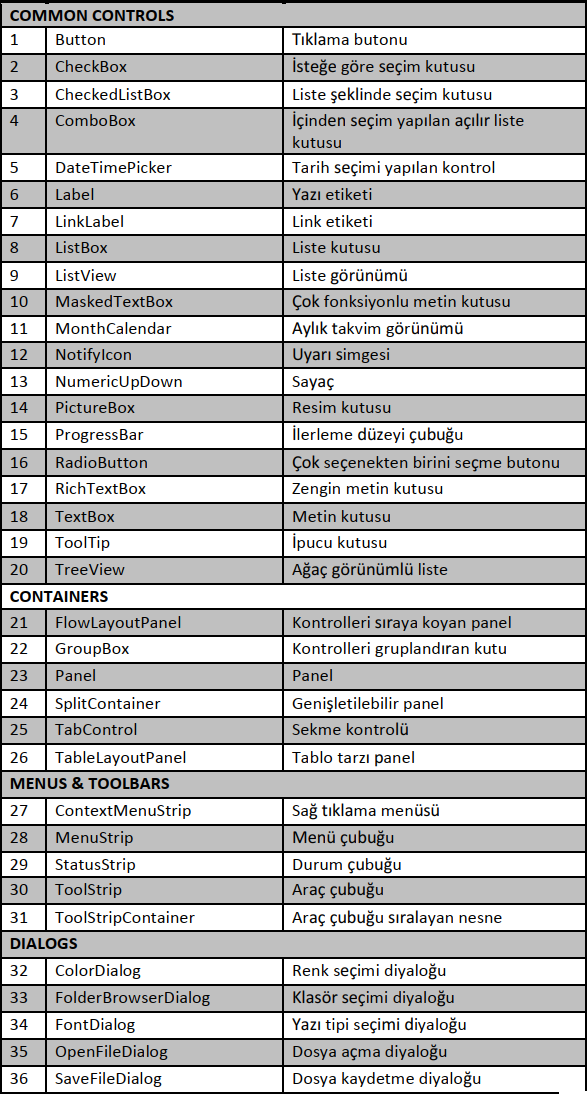
• İletişim Kutuları: Çeşitli iletişim kutularının bulunduğu sekmedir.

• Veri: Veri tabanı işlemlerini gerçekleştirmemizi sağlayan nesnelerin  
bulunduğu sekmedir.

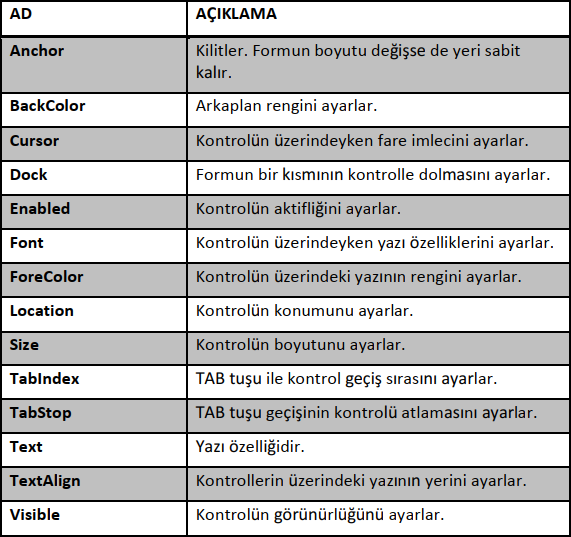
• WPF Birlikte Çalışabilirlik: WPF uygulamaları ile çalışmayı sağlayan nesnelerin bulunduğu sekmedir. İçerisinde fare işaretçisi dışında 1 öge bulunmaktadır.

• Genel: Özelleştirebileceğimiz bir sekmedir. İstediğimiz nesneyi sürükle bırak yöntemiyle bu sekmeye ekleyebiliriz.

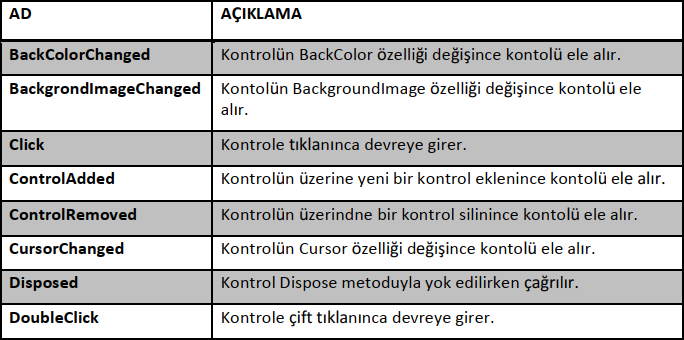
**Sık Kullanılan Kontroller**

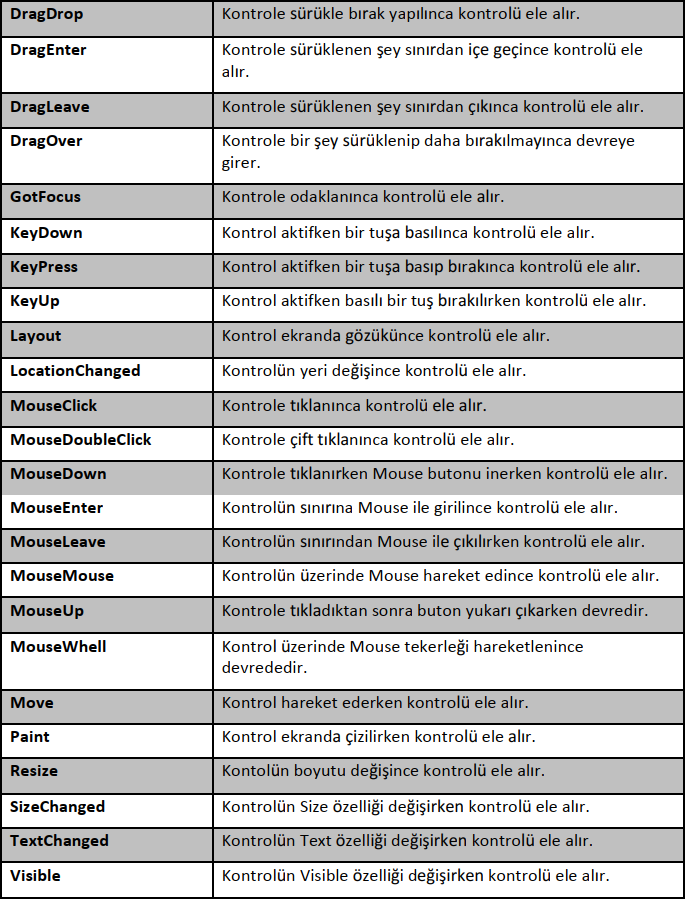


**Kontrollerin Genel Özellikleri**



**Kontrollerin Genel Olayları**





**Standart Kontroller**

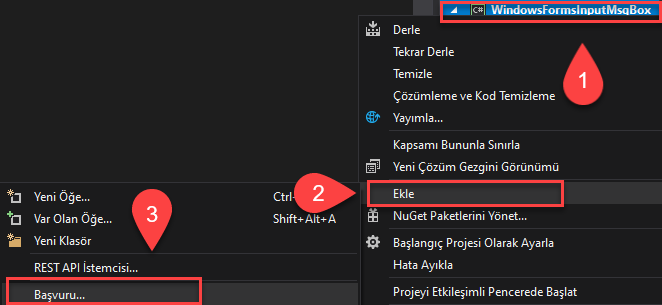


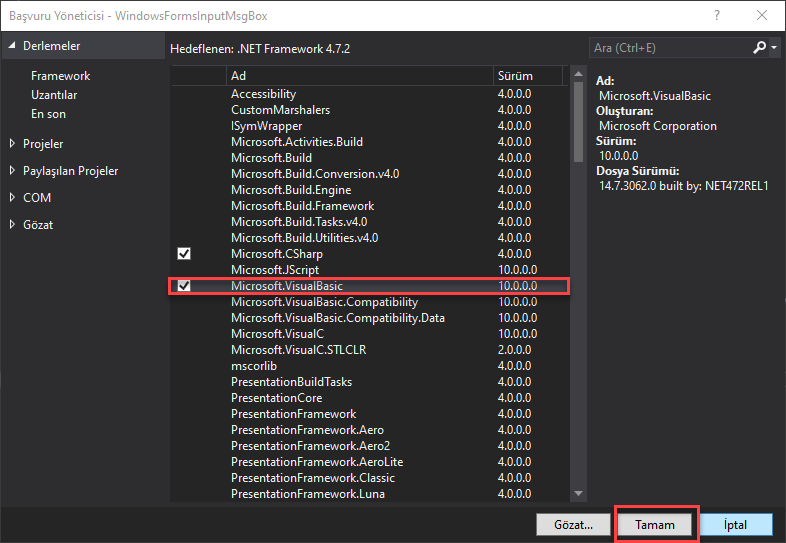
1. **Kod Yazım Kuralları**

* Kod yazarken değişken, sınıf ve metod isimlendirmeleri açıklayıcı ve  
  anlaşılabilir olmalıdır.
* Belirli bir notasyon (yazım tarzı) benimsenmelidir.
* Yazılan kod yorum satırları ile desteklenmeli ve açıklayıcı yorumlar  
  yazılmalıdır.
* Projedeki dosya ve sayfa isimlendirmelerinde özel karakterler kullanılmamalıdır.
* Kodlar {} karakterleri arasına yazılır.
* Nesne adı girildikten sonra . karakteri yazıldığında kullanılabilecek bütün parametreler listelenir.
* Tasarım penceresinden kod penceresine F7, kod penceresinden tasarım penceresine Shift+F7 tuşları ile geçiş yapılır.
* Her komut ; ile bitmelidir.
* Açıklama satırları /\*…\*/ veya // karakterleri ile yazılır.

1. **Bilgi Girişi, Bilgi Çıkışı ve Mesaj**

C#' ta InputBox' ı kullanabilmek için referans olarak Microsoft.VisualBasic dll' i eklenmelidir.





Kod kısmında aşağıdaki satır eklenmelidir.

using Microsoft.VisualBasic;

Aşağıdaki şekilde bilgi girişi yapılabilir.

namespace WindowsFormsInputMsgBox

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

string Isim = Interaction.InputBox("Bilgi Girişi", "Adınızı Giriniz.",

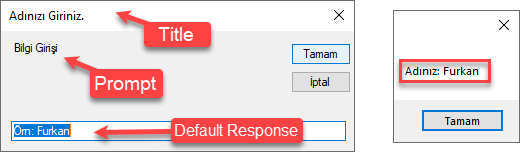
"Örn: Furkan", 0, 0);

MessageBox.Show("Adınız: " + Isim);

}

}

}



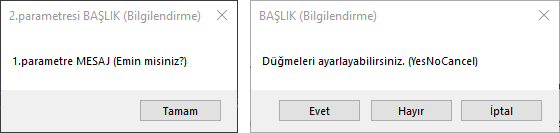
**Prompt:** Açılan pencere üzerinde bulunan etiketi belirler.

**Title:** Açılan pencerenin başlığını belirler.

**DefaultResponse:** Değer hakkında örnek vermek için kullanılır.

**XPos - YPos:** Açılan pencerenin ekranda konumlanacağı pozisyonu belirler.

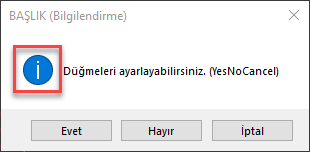
Kullanıcıya bilgi vermek için MessageBox özelliği kullanılır.



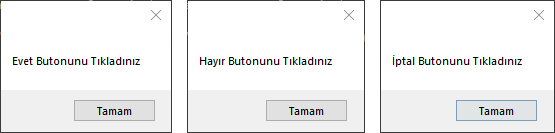
Eklenebilecek düğmeler listesi aşağıdaki şekildedir. Örnek de YesNoCancel değeri kullanılmıştır.

|  |  |  |
| --- | --- | --- |
| **Button No** | **MessageBoxButtons** | **Gösterilecek Seçenekler** |
| 2 | AbortRetryIgnore | İptal-Yeniden Dene-Yoksay |
| 0 | OK | Tamam |
| 1 | OKCancel | Tamam-İptal |
| 5 | RetryCancel | Yeniden Dene-İptal |
| 4 | YesNo | Evet-Hayır |
| 3 | YesNoCancel | Evet-Hayır-İptal |

Açılan Mesaj Kutusuna Icon eklenebilir. Icon isimleri Asterisk, Error, Exclamation, Hand, Information, Question, Stop, Warning dir. Örnek de Information kullanılmıştır.



Mesaj kutusunda basılan düğmeye göre kodlar çalıştırılabilir.



namespace WindowsFormsInputMsgBox

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void button2\_Click(object sender, EventArgs e)

{

MessageBox.Show("1.parametre MESAJ (Emin misiniz?)", "2.parametresi BAŞLIK (Bilgilendirme)");

MessageBox.Show("Düğmeleri ayarlayabilirsiniz. (YesNoCancel)",

"BAŞLIK (Bilgilendirme)", MessageBoxButtons.YesNoCancel);

MessageBox.Show("Düğmeleri ayarlayabilirsiniz. (YesNoCancel)",

"BAŞLIK (Bilgilendirme)", MessageBoxButtons.YesNoCancel, MessageBoxIcon.Information);

DialogResult tiklanan = MessageBox.Show("Çıkmak istiyor musunuz??", "BAŞLIK (Bilgilendirme)",

MessageBoxButtons.YesNoCancel, MessageBoxIcon.Information);

if (tiklanan == DialogResult.Yes)

{

MessageBox.Show("Evet Butonunu Tıkladınız");

}

else if (tiklanan == DialogResult.No)

{

MessageBox.Show("Hayır Butonunu Tıkladınız");

}

else if (tiklanan == DialogResult.Cancel)

{

MessageBox.Show("İptal Butonunu Tıkladınız");

}

}

}

}

1. **Değişkenler**

Değişkenler programın çalışması sırasında girilen bilgileri, belirli bir hesaplama sonucunu elde edilen bilgileri veya çıktıları geçici depolama yeridir. Kodlama yapılırken sözcük, sayı, tarih ya da diğer özellikleri içeren bir ya da birden çok değişken kullanma ihtiyacı oluşabilir.

**Değişken Tanımlama**

degisken\_turu degisken\_adi;

* Değişken adları boşluk, simge içeremez.
* Değişkenler bir numerik karakterle başlayamaz.
* C#'ın diğer bütün komut, metot ve benzerlerinde olduğu gibi değişken adlarında büyük-küçük harf duyarlılığı vardır. Yani “degisken” isimli bir değişkenle “Degisken” isimli bir değişken birbirinden farklıdır.

**Değişkene Değer Atama**

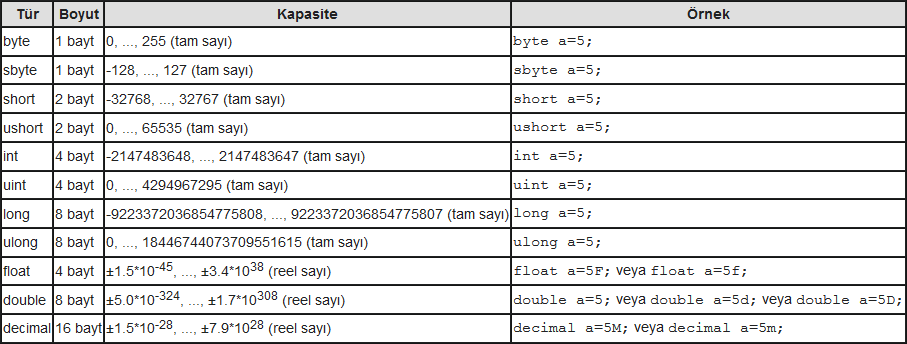
degisken\_adi = deger;

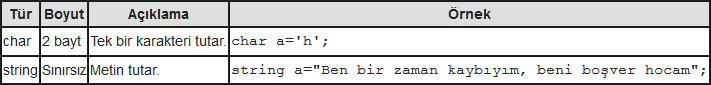
**Örnek :**

int a;

a=5

int a = 5;





**Bool:** Koşullu yapılarda kullanılır. Bool türünden değişkenlere true ve false değerleri atanabilir.

**Object:** Bu değişken türüne her tür veri atanabilir.

**Sabit Değişkenler**

Programımızda bazen değeri hiç değişmeyecek değişkenler tanımlamak isteyebiliriz. Örneğin pi isimli float türünden bir değişken tanımlayıp buna 3.14 değerini verip programımızda pi sayısına ihtiyaç duyulduğunda bu değişkeni kullanabiliriz. Herhangi bir değişkeni sabit olarak belirtmemiz için değişken türünden önce **const** anahtar sözcüğü kullanılır.

const float pi = 3.14;

**Escape Sequence (\) Kullanımı**

Bir string sabitin içinde özel karakterler olması için escape sequence kullanılır.

string yol="Windows\\Program Files"; Çıktı : Windows\Program Files

**Dizi Değişkenler**

**Tek boyutlu dizi:**

degisken\_turu[] dizi\_ismi=new degisken\_turu[dizi\_boyutu];

int[] dizi=new int[25]; **veya**

int[] dizi;

dizi=new int[25]; **veya**

int[] dizi1=new int[10], dizi2=new int[20];

Yukarıdaki iki kodda da int türünden 25 elemanlı dizi adında bir dizi tanımlanır ve dizinin her bir elemanına int türünün varsayılan değeri atanır. Varsayılan değerler, sayısal türler için 0, object türü için NULL (yokluk), string türü için "", char için ' ' (boşluk) ve bool için false değerleridir.

**Değer atama:** dizi[5]=30;

Aşağıdaki şekilde de değer atamaları yapılabilir.

string[] dizi1={"Bir","İki","Üç"};

int[] dizi2={2,-4,6};

float[] dizi3={2f,1.2f,7f};

int[] dizi=new int[]{1, 2, 3};

**Çok boyutlu dizi:**

degisken\_turu[,] dizi\_ismi=new degisken\_turu[dizi\_boyutu, dizi\_boyutu];

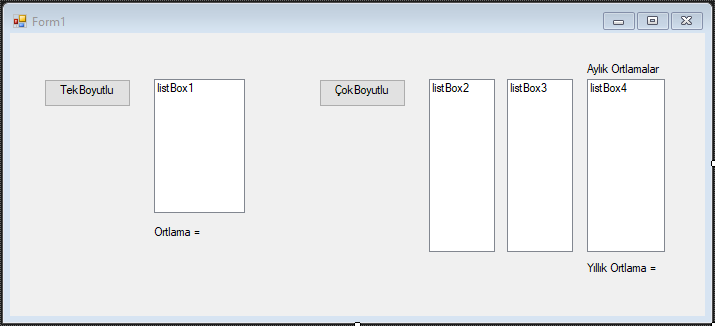
İki boyutlu bir dizinin elemanları indekslerine göre aşağıdaki gibidir.

dizi[0,0] → 1  
dizi[0,1] → 2  
dizi[1,0] → 3  
dizi[1,1] → 4  
dizi[2,0] → 5  
dizi[2,1] → 6

Bu diziyi matris olarak aşağıdaki gibi gösterebiliriz:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | dizi[0,0] | dizi[0,1] | | dizi[1,0] | dizi[1,1] | | dizi[2,0] | dizi[2,1] | | → | |  |  | | --- | --- | | 1 | 2 | | 3 | 4 | | 5 | 6 | |

Aşağıdaki örnekde tek boyutlu ve iki boyutlu dizi elemanları kullanıcıdan alınarak listboxlara yazırılıp ortalamaları hesaplanarak yazılmaktadır.



namespace WindowsFormsList

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

// tip[] dizi\_ismi = new tip[büyüklük];

// InputBox ekleyebilmek için bu satır eklennir.

// Dizi ye sadece rakamlar girin

int[] dizi = new int[10];

double toplam = 0.0;

for (int i = 0; i < 10; i++)

{

dizi[i] = Convert.ToInt32(Interaction.InputBox((i+1) + ". dizi elemanı:", "Dizi Girişi"));

listBox1.Items.Add(dizi[i]);

toplam = toplam + dizi[i];

}

label1.Text = label1.Text + Convert.ToString(toplam / 10);

// Aşağıdaki şekillerde diziler oluşturulabilir ve ilk kullanıma hazırlanabilir

// tip[] dizi\_ismi = new tip[] {deger1, deger2, deger3, ... , degerN}

// tip[] dizi\_ismi = new tip[N] {deger1, deger2, deger3, ... , degerN}

// tip[] dizi = {deger1, deger2, deger3, ... , degerN}

}

private void button2\_Click(object sender, EventArgs e)

{

// 12 aya ait 2 farklı ortalama sıcaklık değeri olduğunu düşünün.

// dizinin boyutu [12,2]

int[,] sicaklik = new int[12,2];

string ay;

double toplam = 0.0, atoplam = 0.0;

for (int i = 0; i < 12; i++)

{

if (i == 0)

ay = "Ocak";

else if (i==1)

ay = "Şubat";

else if (i == 2)

ay = "Mart";

else if (i == 3)

ay = "Nisan";

else if (i == 4)

ay = "Mayıs";

else if (i == 5)

ay = "Haziran";

else if (i == 6)

ay = "Temmuz";

else if (i == 7)

ay = "Ağustos";

else if (i == 8)

ay = "Eylül";

else if (i == 9)

ay = "Ekim";

else if (i == 10)

ay = "Kasım";

else

ay = "Aralık";

listBox2.Items.Add(ay);

for (int j = 0; j < 2; j++)

{

sicaklik[i,j] = Convert.ToInt32(Interaction.InputBox(ay + " ayının " + (j+1) + ". ortalaması :", "Çok Elemanlı Dizi Girişi"));

if ((j % 2) == 1)

{

listBox3.Items.Add(sicaklik[i, j - 1] + "," + sicaklik[i, j]);

atoplam = (sicaklik[i, j - 1] + sicaklik[i, j]) / 2.0;

listBox4.Items.Add(atoplam);

}

}

atoplam = 0.0;

}

for (int i = 0; i < 12; i++)

{

for (int j = 0; j < 2; j++)

{

toplam = toplam + sicaklik[i, j];

}

}

label3.Text = label3.Text + Convert.ToString(toplam / 24.0);

}

}

}

1. **Gelişmiş Nesneler**

**CheckBox:** Onay veya seçim işlemleri için kullanılan kontrol nesnesidir.

if (checkBox1.Checked)

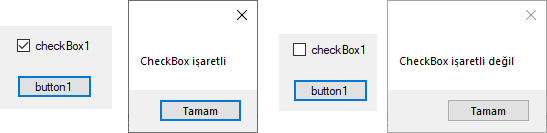
{

MessageBox.Show("CheckBox işaretli");

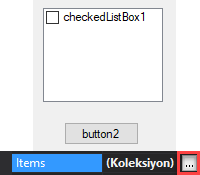
}

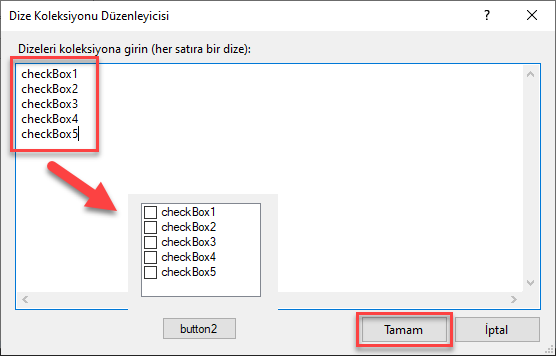
else

MessageBox.Show("CheckBox işaretli değil");

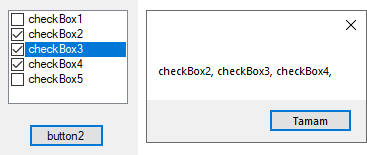


**CheckedListBox:** İçerisindeki değerleri, seçilebilir liste şeklinde görüntüleyen kontrol nesnesidir.





Aşağıdaki örnekde checkedlistbox bileşeninde seçili olan ceheckbox isimleri mesaj kutusunda gösterilmektedir.



string elemanlar = "";

for (int i = 0; i < checkedListBox1.CheckedItems.Count; i++)

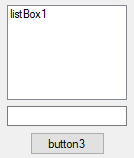
{

elemanlar += checkedListBox1.CheckedItems[i].ToString() + ", ";

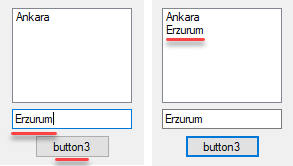
}

MessageBox.Show(elemanlar);

**ListBox**: İçerisindeki değerleri bir liste hâlinde görüntüleyen kontrol nesnesidir.

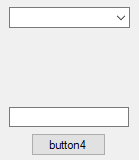


Aşağıdaki örnekde textbox bileşenindeki ifade listbox nesnesine eklenir.

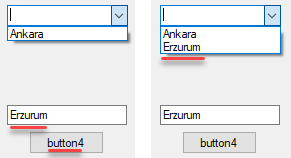


listBox1.Items.Add(textBox1.Text);

**ComboBox**: İçerisindeki değerleri açılabilir bir liste hâlinde görüntüleyen kontrol nesnesidir.

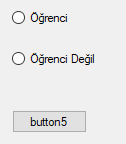


Aşağıdaki örnekde textbox bileşenindeki ifade combobox nesnesine eklenir.

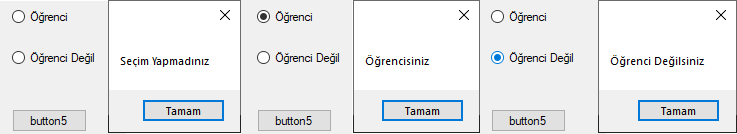


comboBox1.Items.Add(textBox2.Text);

**RadioButton**: Birden fazla seçenekten sadece birini seçmenin zorunlu olduğu  
durumlarda kullanılan kontrol nesnesidir.



Aşağıdaki örnekde radiobutton1 ve radiobutton2 nesnelerinin durumuna göre mesaj kutusu ile bilgi verilmektedir.



if (radioButton1.Checked)

{

MessageBox.Show("Öğrencisiniz");

}

else if (radioButton2.Checked)

{

MessageBox.Show("Öğrenci Değilsiniz");

}

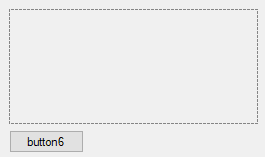
else

{

MessageBox.Show("Seçim Yapmadınız");

}

**PictureBox:** Programa görsellik katmak için bazı yerlere resimler eklenmektedir.



Aşağıdaki örnekde picturebox nesnesinde bazı ayarlar yapılarak resim gösterilmektedir.



pictureBox1.SizeMode = PictureBoxSizeMode.CenterImage;

pictureBox1.BorderStyle = BorderStyle.FixedSingle;

pictureBox1.ImageLocation = "mmyo.jpg";

**SizeMod**: Resim boyutlandırması ile ilgili özellikleri içerir.

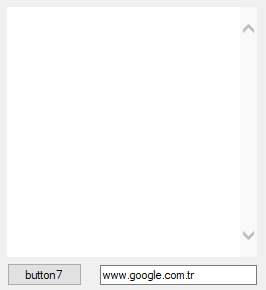
**BorderStyle**: resim etrafındaki border özelliklerini içerir.

**ImageLocation**: Resim yolu veya internet adresi verilerek resim yüklenmesini sağlar.

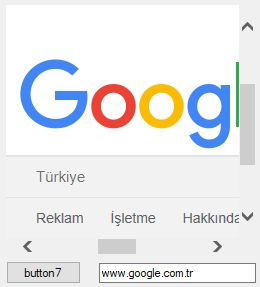
**Image**: PictureBox içerisinde görüntülenecek olan resim dosyası yolunun bilgisini tutar.

**ErrorImage**: Görüntülenecek resim dosyasına erişilemediği zaman gösterilecek alternatif resim dosyasının bilgisini tutar.

**WebBrowser:** web sayfalarını görüntülemek için kullanılan kontrol nesnesidir.

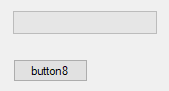


Aşağıdaki örnekde WebBrowser kontrolünün Navigate özelliği sayesinde TextBox içerisindeki adrese gidilmesi sağlanmaktadır.

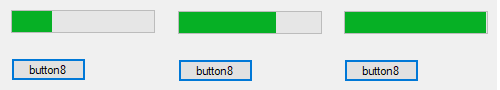


webBrowser1.Navigate(textBox3.Text);

**ProgressBar:** Uygulamalarımızda gerçekleşen bir işlemin ne kadarının tamamlandığını kullanıcıya göstermek amacıyla kullanılır.



Aşağıdaki örnekde progressbar kontrolünün value değeri değiştirilerek prograss’in ilerlemesi gösterilmektedir.



progressBar1.Maximum = 100;

progressBar1.Minimum = 0;

progressBar1.Step = 1;

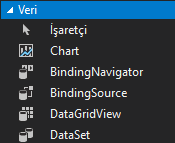
for (int i = 0; i < 100; i++)

{

progressBar1.Value = i;

}

**Veri Erişim Kontrolleri:** Masaüstü, Mobil ya da Web uygulamalarının birçoğu çalışırken veritabanı kullanmaktadır. Araç kutusunda (Toolbox) Data sekmesi altında yer alan veri kontrolleri, ADO.Net adı verilen ve .Net’in veri yönetimi ayağını oluşturan aracın en önemli bileşenleri arasındadır.

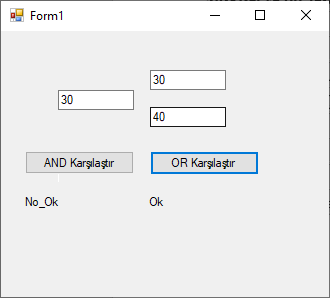


1. **Operatörler**

|  |  |
| --- | --- |
| **Birinci öncelikliler** | **x++, x--** |
| **Tek operand alan operatörler** | **+, -, !, ~, ++x, --x, (Tür)x** |
| **Çarpma ve bölme** | **\*, /, %** |
| **Toplama ve çıkarma** | **+, -** |
| **Kaydırma operatörleri** | **<<, >>** |
| **İlişkisel ve tür testi operatörleri** | **<, >, <=, >=, is, as** |
| **Eşitlik operatörü** | **==, !=** |
| **Bitsel VE (AND)** | **&** |
| **Bitsel Özel VEYA (XOR)** | **^** |
| **Bitsel VEYA (OR)** | **|** |
| **Mantıksal VE** | **&&** |
| **Mantıksal VEYA** | **||** |
| **Koşul operatörü** | **?:** |
| **Atama ve işlemli atama operatörleri** | **=, \*=, /=, %=, +=, -=, <<=, >>=, &=, ^=, |=** |

**Örnek:**

İlk ifade ile ikinci ve üçüncü ifadeleri karşılaştırır. AND ve OR operatorleri ile karşılaştırma yapar.



private void bt\_and\_karsilastir\_Click(object sender, EventArgs e)

{

if ((tb\_ilkifade.Text == tb\_ikinciifade.Text) & (tb\_ilkifade.Text == tb\_ucuncuifade.Text))

{

lb\_and.Text = "Ok";

}

else

lb\_and.Text = "No\_Ok";

}

private void bt\_or\_karsilastir\_Click(object sender, EventArgs e)

{

if ((tb\_ilkifade.Text == tb\_ikinciifade.Text) | (tb\_ilkifade.Text == tb\_ucuncuifade.Text))

{

lb\_or.Text = "Ok";

}

else

lb\_or.Text = "No\_Ok";

}

1. **Kontrol Deyimleri**

**If – Else If – Else:**

if(koşul)

komut1;

else

komut2; **veya**

if(koşul)

{

komutlar1

}

else

{

komutlar2

} **veya**

if(koşul)

{

komutlar1

}

else if

{

komutlar2

}

else if

{

Komutlar3

}

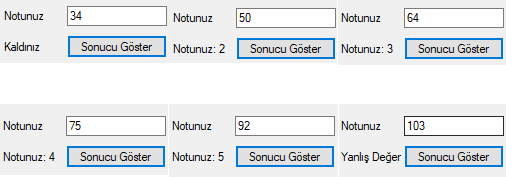
else

{

Komutlar4

}

**Örnek:**



int not = Convert.ToInt32(textBox1.Text);

label2.Text = "Sonuç";

if (not >= 0 & not < 45)

{

label2.Text = "Kaldınız";

}

else if (not >= 45 & not < 60)

{

label2.Text = "Notunuz: 2";

}

else if (not >= 60 & not < 70)

{

label2.Text = "Notunuz: 3";

}

else if (not >= 70 & not < 85)

{

label2.Text = "Notunuz: 4";

}

else if (not >= 85 & not < 101)

{

label2.Text = "Notunuz: 5";

}

else

{

label2.Text = "Yanlış Değer";

}

**İç İçe If:**

if(koşul1)

{

if(koşul2)

komut1;

else

{

komut2;

komut3;

}

}

else

komut4;

**Switch – Case:**

switch(ifade)

{

case sabit1:

komut1;

break;

case sabit2:

komut2;

break;

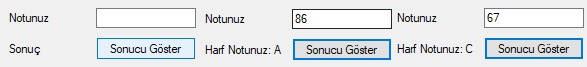
default:

komut3;

break;

}

**Örnek:**



int not = Convert.ToInt32(textBox2.Text);

label2.Text = "Sonuç";

int notdurum = 0;

if (not >= 0 & not < 45)

notdurum = 1;

else if (not >= 45 & not < 60)

notdurum = 2;

else if (not >= 60 & not < 70)

notdurum = 3;

else if (not >= 70 & not < 85)

notdurum = 4;

else if (not >= 85 & not < 101)

notdurum = 5;

else

notdurum = 0;

switch (notdurum)

{

case 1:

label3.Text = "Kaldınız";

break;

case 2:

label3.Text = "Harf Notunuz: D";

break;

case 3:

label3.Text = "Harf Notunuz: C";

break;

case 4:

label3.Text = "Harf Notunuz: B";

break;

case 5:

label3.Text = "Harf Notunuz: A";

break;

case 0:

label3.Text = "Yanlış Değer";

break;

default:

break;

}

1. **Döngüler**

Birçok programlama uygulamasında bazı işlemlerin belirli bir sayıda veya bir koşul sağlandığı müddetçe tekrar ettirilmesi gerekmektedir. Bu amaçla kullanılan programlama yapılarına ise döngü adı verilmektedir.

**Döngü Oluşturma Kuralları**

1 - Döngüyü kontrol edecek bir döngü değişkeni oluşturulur.  
2 - Döngü değişkeninin başlangıç değeri belirlenir.  
3 - Döngü değişkeninin bitiş değeri belirlenir.  
4 - Döngü değişkeninin bitiş değerine ulaşıp ulaşmadığı test edilir.  
5 - Tekrarı istenen işlem gerçekleştirilir.  
6 - Döngü değişkeni bir sonraki iterasyonda kullanılmak üzere adım miktarı  
kadar artırılır ya da azaltılır.

**For Döngüsü:**

for(ifade1;kosul;ifade2)

komut; **veya**

for(ifade1;kosul;ifade2)

{

komut1;

komut2;

.

.

.

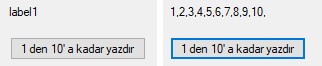
}

**For Döngüsünün Çalışma Prensibi:**

1. ifade1 çalıştırılır.
2. Koşula bakılır. Eğer koşul sağlanıyorsa;
   1. küme parantezleri içindeki ya da -küme parantezleri yoksa- kendinden sonra gelen ilk satırdaki komut çalıştırılır.
   2. ifade2 çalıştırılır.
   3. 2. adıma dönülür.
3. Eğer koşul sağlanmıyorsa küme parantezleri dışına ya da -küme parantezleri yoksa- kendinden sonra gelen ilk satırdaki komuttan hemen sonraki satıra çıkılır. Dolayısıyla döngüden çıkılmış olur.

**Örnek**:

Aşağıdaki örnekde butona bastığında 1 den 10 a kadar sayıları label in textine eklemektedir.



label1.Text = "";

for (int i = 1; i < 11; i++)

{

label1.Text = label1.Text + i + ",";

}

**While Döngüsü:**

while(koşul)

komut; **veya**

while(koşul)

{

komut1;

komut2; …

}

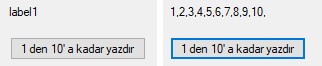
**While Döngüsünün Çalışma Prensibi:**

* + - 1. Koşula bakılır. Eğer koşul sağlanıyorsa;

1. küme parantezleri içindeki ya da -küme parantezleri yoksa- kendinden sonra gelen ilk satırdaki komut çalıştırılır.
2. 1. adıma dönülür.
3. Eğer koşul sağlanmıyorsa küme parantezleri dışına ya da -küme parantezleri yoksa- kendinden sonra gelen ilk satırdaki komuttan hemen sonraki satıra çıkılır. Dolayısıyla döngüden çıkılmış olur.

**Örnek**:

Aşağıdaki örnekde butona bastığında 1 den 10 a kadar sayıları label in textine eklemektedir.



label2.Text = "";

int i = 1;

while (i < 11)

{

label2.Text = label2.Text + i + ",";

i++;

}

**Do-While Döngüsü:**

do

komut;

while(koşul); **veya**

do

{

komut1;

komut2;

.

.

.

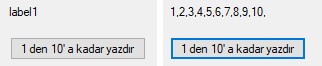
}while(koşul);

**Do-While Döngüsünün Çalışma Prensibi:**

* + - 1. Döngüdeki komutlar bir kez çalıştırılır.
      2. Koşula bakılır.
         1. Eğer koşul sağlanıyorsa 1. adıma dönülür.
         2. Eğer koşul sağlanmıyorsa döngüden çıkılır.

**Örnek**:

Aşağıdaki örnekde butona bastığında 1 den 10 a kadar sayıları label in textine eklemektedir.



label3.Text = "";

int i = 1;

do

{

label3.Text = label3.Text + i + ",";

i++;

} while (i < 11);

**\* Do-While ile While döngüleri arasındaki fark; Şart eğer dogru değilse do-wile 1 kere de olsa çalışır, while hiç çalışmaz.**

**Döngüden Çıkma**

**Continue:**

Mesaj kutusunda 1,2,3,4,6,7,8,9 değerleri görülür.

for (int i = 0; i < 10; i++)

{

if (i == 5)

{

continue;

}

else

MessageBox.Show("i değeri= " + i);

}

**Break:**

Mesaj kutusunda 1,2,3,4 değerleri görülür.

for (int i = 0; i < 10; i++)

{

if (i == 5)

{

continue;

}

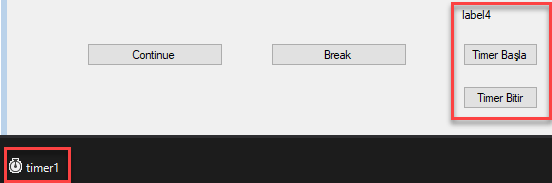
else

MessageBox.Show("i değeri= " + i);

}

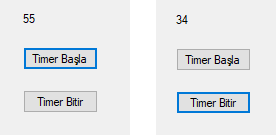
1. **Zamanlayıcılar**

Bazı interaktif uygulamalarda (örneğin bir oyun uygulamasında veya çoktan seçmeli sorulardan oluşan bir test uygulamasında) kullanıcının tepkisinin belirli bir zaman dilimi içerisinde gerçekleşmesi beklenir. Bu amaçla Timer isimli kontrol nesnesinden faydalanılabilir.



**Örnek:**

Aşağıdaki örnekde sayaç 60 dan başlayarak birer azalarak geriye doğru sayar. Enabled özelliğine false değeri erilerek sayaç durdurulur.



int sayi = 60;

int sayac;

private void button6\_Click(object sender, EventArgs e)

{

timer1.Interval = 1000;

timer1.Enabled = true;

}

private void timer1\_Tick\_1(object sender, EventArgs e)

{

if (sayi >= 0)

{

timer1.Interval = 1000;

timer1.Enabled = true;

sayac = sayi--;

label4.Text = sayac.ToString();

}

}

private void button7\_Click(object sender, EventArgs e)

{

timer1.Enabled = false;

}

1. **Diyalog Pencereleri**

Kullanıcıdan bilgi almak için, geliştiricinin kendi tasarımı olan tamamen  
özelleşmiş pencerelerin kullanımı mümkünken, bazı sık kullanılan işlemleri  
yapabilmek için ön tanımlı pencereler de kullanılabilmektedir. Bu pencerelere diyalog pencereleri denilmektedir.

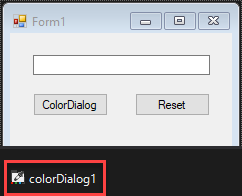
Diyalog pencereleri genellikle ana pencerenin önüne açılan ve giriş işlemi  
tamamlanmadan ana pencereye dönülmesine izin vermeyen modal pencerelerdir.  
Diyalog pencereleri, özelleşmiş pencerelerle karşılaştırıldıklarında aşağıdaki avantajlara sahiptirler:

* Kullanıma hazır bileşenlerdir. Tasarım ve gerçekleştirim gerektirmezler.
* İlişkili metotları aracılığıyla geliştiriciler tarafından kolayca kullanılabilirler.
* Standart bileşenler olduklarından kullanıcılar tarafından bilinirler ve kolayca kullanılabilirler.

**ColorDialog**:

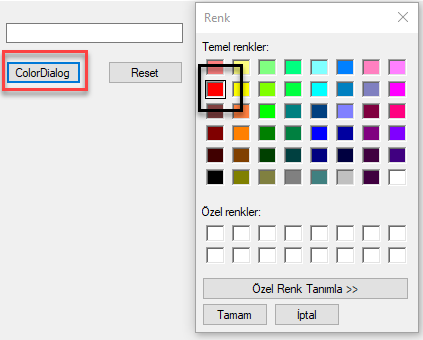
ColorDialog sınıfına ait bazı özelikler aşağıda verilmiştir.

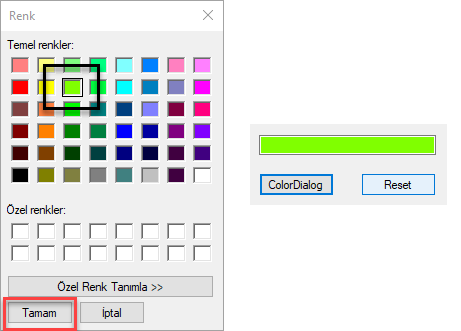
|  |  |
| --- | --- |
| **Özellik** | **Açıklama** |
| AllowFullOpen | Seçim penceresinde özelleşmiş renk tanımlanıp tanımlanamayacağını belirten özelliktir. |
| Color | Kullanıcının seçtiği rengi temsil eder. Bu değer alınabilir ya da değiştirilebilir. |
| CustomColors | Kullanıcıya gösterilen özelleşmiş renk kümesini temsil eder. Bu küme alınabilir veya değiştirilebilir. |
| ShowHelp | Seçim penceresinde “Yardım” (Help) butonunun gösterilip gösterilmeyeceğini belirleyen özelliktir. |

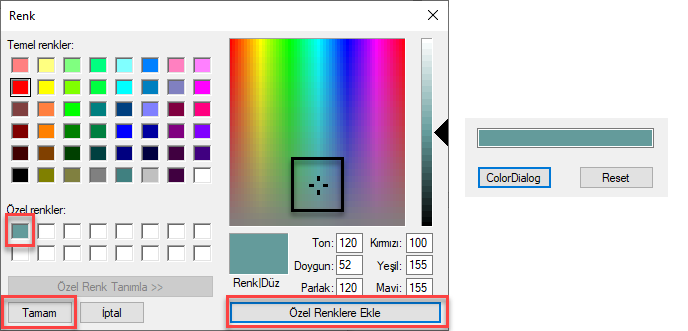


**Örnek:**

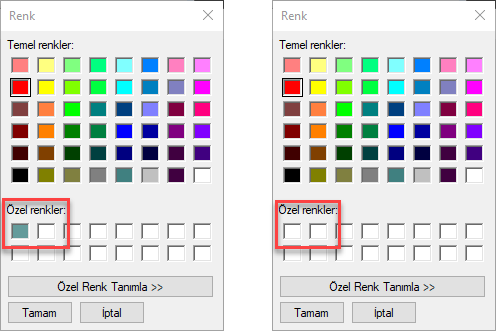
Aşağıdaki örnekde ColorDialog nesnesinden renk seçilerek textbox1 nesnesinin arkaplan rengi değiştirlmektedir. Default renk değeri kırmızı seçilmiştir. Değişiklik yapıldıktan sonra Reset butonu ile seçili renk değeri default değer olan kırmızıya dönüştürülür.







Reset düğmesinden sonra



ColorDialog renk = new ColorDialog();

private void button1\_Click(object sender, EventArgs e)

{

renk.Color = Color.Red;

if (renk.ShowDialog() == DialogResult.OK)

{

textBox1.BackColor = renk.Color;

}

}

private void button2\_Click(object sender, EventArgs e)

{

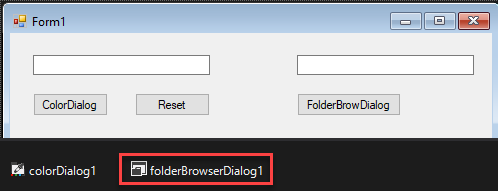
renk.Reset();

}

**FolderBrowserDialog**:

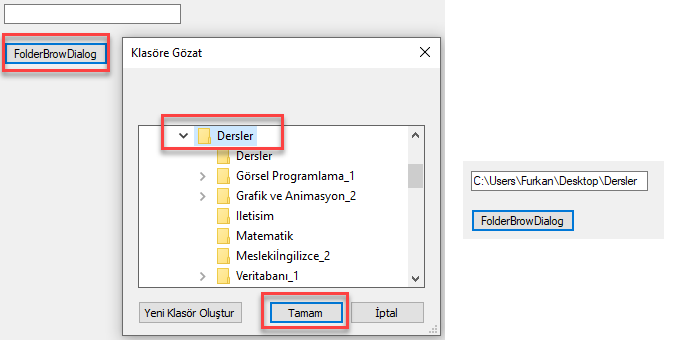
FolderBrowserDialog sınıfına ait bazı özelikler aşağıda verilmiştir.

|  |  |
| --- | --- |
| **Özellik** | **Açıklama** |
| Description | Diyalog penceresinde gösterilen dizin ağacının üstündeki açıklayıcı bilgiyi tutan özelliktir. |
| RootFolder | Dizin taramasının başlayacağı kök dizini (root folder) tutan özelliktir. |
| SelectedPath | Kullanıcı tarafından seçilen yolu (path) tutan özelliktir. |
| ShowNewFolderButton | Kullanıcının yeni dizin oluşturmasını sağlayan “Make New Folder” butonunun diyalog penceresinde gösterilip gösterilmeyeceğini belirleyen özelliktir. |



**Örnek:**

Aşağıdaki örnekde FolderBrowserDialog nesnesinden dizin seçilerek, seçilen dizinin yolu textbox2 nesnesine eklenmektedir.



private void button3\_Click(object sender, EventArgs e)

{

FolderBrowserDialog dizin= new FolderBrowserDialog();

dizin.ShowNewFolderButton = true;

if (dizin.ShowDialog() == DialogResult.OK)

{

textBox2.Text = dizin.SelectedPath;

Environment.SpecialFolder root = dizin.RootFolder;

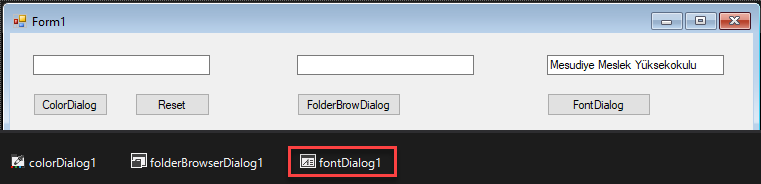
}

}

**FontDialog**:

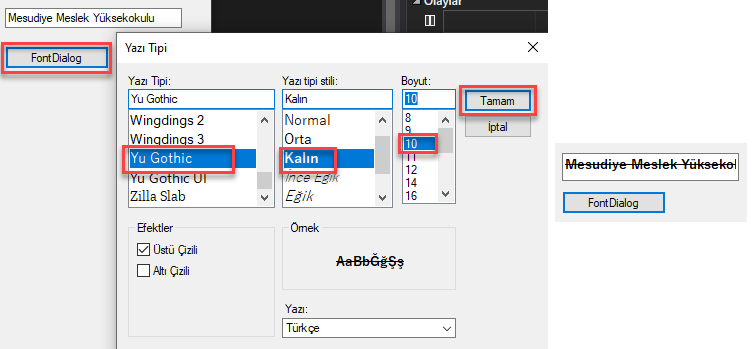
FontDialog sınıfına ait bazı özelikler aşağıda verilmiştir.

|  |  |
| --- | --- |
| **Özellik** | **Açıklama** |
| AllowVectorFonts | Vektör fontların seçilmesine izin verilip verilmeyeceğini belirten özelliktir. |
| AllowVerticalFonts | Yatay yazı tiplerine ek olarak dikey yazı tiplerinin seçilmesine izin verilip verilmeyeceğini belirten özelliktir. |
| Color | Seçilen yazı tipinin rengini tutan özelliktir. |
| Font | Seçilen yazı tipini tutan özelliktir. |
| MaxSize | Kullanıcının seçebileceği en büyük yazı tipi boyutunu belirten özelliktir. |
| MinSize | Kullanıcının seçebileceği en küçük yazı tipi boyutunu belirten özelliktir. |
| ShowColor | Diyalog penceresinin seçilen rengi gösterip göstermeyeceğini belirten özelliktir. |



**Örnek:**

Aşağıdaki örnekde FontDialog nesnesinden font seçimi yapılarak textbox3 nesnesinin font özellikleri değiştirilir.



private void button4\_Click(object sender, EventArgs e)

{

FontDialog fontsec = new FontDialog();

if (fontsec.ShowDialog() == DialogResult.OK)

{

textBox3.Font = fontsec.Font;

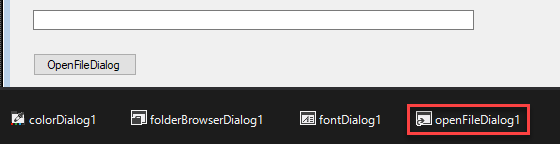
}

}

**OpenfileDialog**:

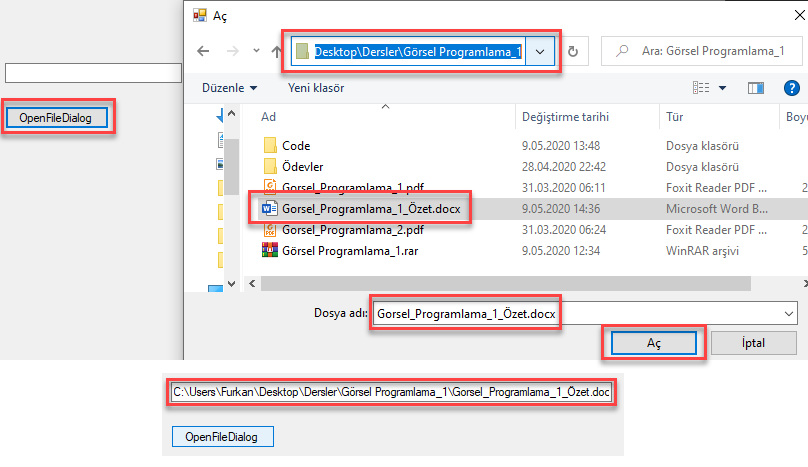
OpenFileDialog sınıfına ait bazı özelikler aşağıda verilmiştir.

|  |  |
| --- | --- |
| **Özellik** | **Açıklama** |
| AddExtension | Kullanıcı dosya uzantısını belirtmemişse dosya uzantısının otomatik eklenip eklenmeyeceğini belirten özelliktir. |
| CheckFileExists | Kullanıcı var olmayan bir dosyayı belirtmişse diyalog penceresinin kullanıcıyı uyarıp uyarmayacağını belirten özelliktir. |
| CheckPathExists | Kullanıcı var olmayan bir dosya yolu belirtmişse diyalog penceresinin kullanıcıyı uyarıp uyarmayacağını belirten özelliktir. |
| DefaultExt | Varsayılan dosya uzantısını belirten özelliktir. |
| FileName | Diyalog penceresinde seçilen dosyayı belirten özelliktir. |
| FileNames | Diyalog penceresinde seçilen dosyaları belirten özelliktir. |
| InitialDirectory | Diyalog penceresi ilk açıldığında içindeki dosyaların gösterildiği başlangıç dizinini belirten özelliktir. |
| Multiselect | Diyalog penceresi ile birden fazla dosya seçilip seçilemeyeceğini belirten özelliktir. |



**Örnek:**

Aşağıdaki örnekde OpenFileDialog nesnesinden dosya seçilerek, seçilen dosyanın yolu textbox4 nesnesine eklenmektedir.



private void button5\_Click(object sender, EventArgs e)

{

OpenFileDialog dosya = new OpenFileDialog();

if (dosya.ShowDialog() == DialogResult.OK)

{

textBox4.Text = dosya.FileName;

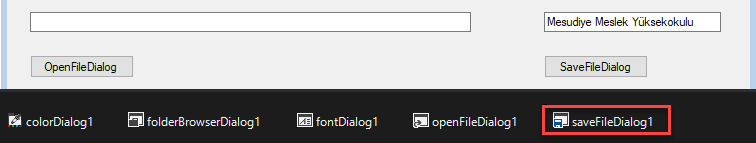
}

}

**SaveFileDialog:**

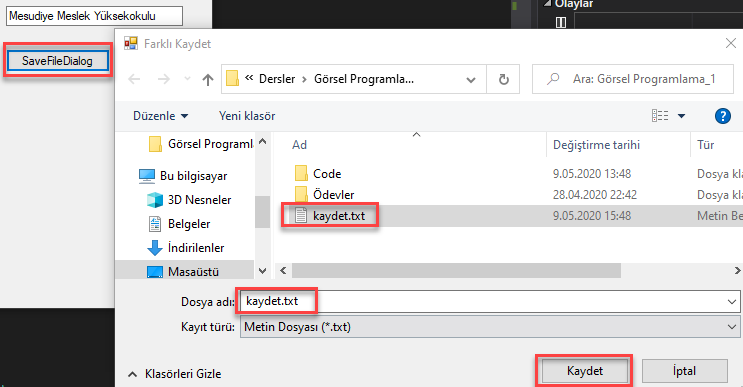
SaveFileDialog sınıfına ait bazı özelikler aşağıda verilmiştir.

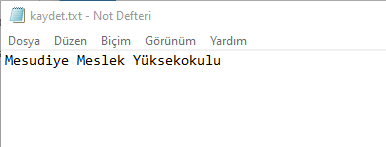
|  |  |
| --- | --- |
| **Özellik** | **Açıklama** |
| CreatePrompt | Kullanıcı var olmayan bir dosyayı seçmişse bu dosyanın oluşturulması için kullanıcıya sorulup sorulmayacağını belirten özelliktir. |
| OverwritePrompt | Kullanıcının var olan bir dosyayı seçmesi durumunda kullanıcının uyarılıp uyarılmayacağını belirten özelliktir. |
| ShowHelp | Diyalog penceresinde “Yardım” (Help) butonunun gösterilip gösterilmeyeceğini belirleyen özelliktir. |
| Title | Dosya kaydetme penceresinin başlığını belirten özelliktir. |



**Örnek:**

Aşağıdaki örnekde SaveFileDialog nesnesinden dosya seçilerek, textbox5 nesnesinindeki metin seçilen dosyaya eklenmektedir.





private void button6\_Click(object sender, EventArgs e)

{

SaveFileDialog kaydet = new SaveFileDialog();

kaydet.Filter = "Metin Dosyası|\*.txt";

kaydet.OverwritePrompt = true;

kaydet.CreatePrompt = true;

if (kaydet.ShowDialog() == DialogResult.OK)

{

System.IO.StreamWriter kayit = new System.IO.StreamWriter(kaydet.FileName);

kayit.WriteLine(textBox5.Text);

kayit.Close();

}

}

1. **Dosya İşlemleri**

.NET Framework'de System.IO namespace'i dosya ve dizin işlemlerini gerçekleştirebilmek için gerekli sınıfları içerir. Bu nedenle, dosya ile ilgili işlemleri yaparken bu namespace'in mutlaka programa eklenmesi gerekir.

using System.IO;

Bir dosya okumak veya yazmak için açıldığında stream adını alır. Stream, tek boyutlu birbirini takip eden veri kümesi olarak düşünülebilir ve bu veri kümesinin bir başlangıcı ve sonu vardır. Cursor, stream içerisindeki o an bulunulan yeri gösteren bir işaretçidir.

Stream içerisindeki veri bellekten, bir dosyadan veya bir TCP/IP soketten  
gelebilir. Bir stream üzerinde aşağıdaki 3 işlem gerçekleştirilebilir.

**Reading:** Bir stream içerisinden veri okunarak, string veya byte dizisi gibi  
bir veri yapısı içerisine aktarılabilir.

**Writing:** Bir veri kaynağından okunan veriler bir stream içerisine yazılabilir.

**Seeking:** Bir stream içerisinde hangi konumda bulunulduğu sorgulanabilir  
veya konum değiştirilebilir.

**Stream Çeşitleri**

.NET framework'te kullanılan stream çeşitleri ve kullanım yerleri aşağıdaki  
gibidir.

**FileStream**: Herhangi bir sürücüdeki dosyayı okumak veya dosyaya yazmak  
için kullanılır.

**IsolatedStorageFileStream**: Geliştirilen uygulamaya özel dosyaları  
oluşturmak, okumak ve yazmak için kullanılır.

**BufferedStream**: Bir stream üzerinde, okuma veya yazma işlemleri için bir  
buffer katmanı ekler. Bu sayede okuma ve yazma performansı artar.

**CryptoStream**: Streamlerin güvenli bir formatta saklanması ve  
gönderilmesi için kullanılır.

**MemoryStream**: Veriyi geçici olarak bellekte saklamayı sağlar.

**NetworkStream**: Ağ üzerinden veri aktarımı için kullanılan temel veri  
yapısıdır.

**.NET I/O Sınıfları**

System.IO namespace'i dosya oluşturma ve silme, dosyadan okuma ve  
dosyaya yazma, dosya açma ve kapatma gibi işlemler için birçok sınıf içermektedir.

Aşağıdaki tabloda System.IO namespace'i içerisindeki çok kullanılan sınıflar ve kullanım yerleri verilmiştir.

|  |  |
| --- | --- |
| **Sınıf** | **Kullanım Yeri** |
| BinaryReader | Stream'den temel veri türlerindeki veriyi okumayı sağlar |
| BinaryWriter | Stream'e temel veri türlerindeki veriyi yazmayı sağlar. |
| BufferedStream | Bir stream üzerinde, okuma veya yazma işlemleri için bir buffer katmanı ekler. |
| Directory | Dizin oluşturma, silme ve taşıma gibi işlemlerin yapılmasını sağlayan statik metodlar sunar. |
| DirectoryInfo | Spesifik bir dizin üzerinde silme ve taşıma gibi işlemlerin yapılmasını sağlar. |
| DriveInfo | Sürücüler hakkında bilgi almayı sağlar. |
| File | Tek bir dosyanın açılması, oluşturulması, taşınması ve silinmesi için statik metotlar sağlar. |
| FileInfo | Dosyalar üzerinde oluşturma, silme ve taşıma gibi işlemlerin yapılmasını sağlar. |
| FileStream | Herhangi bir sürücüdeki dosyayı okumak veya dosyaya yazmak için kullanılır. |
| MemoryStream | Veriyi geçici olarak bellekte saklamayı sağlar. |
| Path | İstenilen sürücü yolu üzerinde işlemler yapmayı sağlar. |
| StreamReader | Bir stream'den karakter veya karakterler okumayı sağlar. |
| StreamWriter | Bir stream'e karakter veya karakterler yazmayı sağlar. |
| NetworkStream | Ağ üzerinden veri aktarımı için kullanılan temel veri yapısıdır. |

**FileStream Sınıfı**

FileStream sınıfı, herhangi bir sürücüdeki dosyayı okumak veya bu dosyaya  
yazmak için kullanılır.

**Kurucu Metot:**

**FileStream**(string path, FileMode mode, FileAccess access, FileShare share);

**Yol:** Üzerinde işlem yapılmak istenen dosyanın yolunun ve isminin verildiği  
parametredir.

**Mod:** Dosyanın hangi modda açılacağının belirlendiği parametredir. Alabildiği değerler ve anlamları aşağıda belirtilmiştir.

• **Append**: Eğer dosya varsa dosyayı bilgi eklemek amacı ile açar ve cursor'ü  
dosyanın sonuna konumlandırır. Eğer dosya yoksa dosyayı oluşturur.

• **Create**: Yeni bir dosya oluşturmak için kullanılır. Eğer oluşturulmak istenen dosya daha önceden mevcutsa var olan dosyanın üzerine yazar.

• **CreateNew**: Yeni bir dosya oluşturmak için kullanılır. Eğer oluşturulmak  
istenen dosya mevcutsa IOException ayrıcalıklı durumu oluşur.

• **Open**: Varolan bir dosyayı açmak için kullanılır. Eğer açılmak istenen dosya belirtilen yolda bulunamamışsa FileNotFoundException ayrıcalıklı durumu oluşur.

• **OpenOrCreate**: Eğer dosya belirtilen dosya mevcutsa açılır aksi takdirde  
belirtilen isimde yeni bir dosya oluşturulur.

• **Truncate**: Varolan dosya açılır ve büyüklüğü 0 byte'a düşürülür.

**Erişim**: Açılan veya oluşturulan dosyaya hangi amaçla ulaşılacağını  
belirlemek için kullanılan parametredir. Kullanılabilecek değerler Read, ReadWrite ve Write olarak belirlenmiştir. Eğer Read kullanılırsa dosyaya okumak için ulaşılacağı, ReadWrite kullanılırsa hem okumak hem de yazmak için ulaşılacağı, Write kullanılırsa yazmak için ulaşılacağı belirlenmiş olur.

**Paylaşım**: Üzerinde işlem yapılan dosyaya diğer processler tarafından  
ulaşılıp ulaşılamayacağını belirlemek için kullanılan parametredir. Bu  
parametrenin alabileceği değerler ve anlamları aşağıda verilmiştir.

• **None**: Üzerinde çalışılan dosyanın diğer processler tarafından kullanımına  
izin vermez.

• **Read**: Üzerinde çalışılan dosyaya diğer processlerin de okuma izni ile  
ulaşmasına izin verir.

• **ReadWrite**: Üzerinde çalışılan dosyaya diğer processlerin de okuma ve  
yazma izni ile ulaşmasına izin verir.

• **Write**: Üzerinde çalışılan dosyaya diğer processlerin de yazma izni ile  
ulaşmasına izin verir.

**FileStream Sınıfının Özellikleri**

|  |  |
| --- | --- |
| **Özellik** | **İşlev** |
| CanRead | Stream'in okumayı destekleyip desteklemediğini öğrenmeyi sağlar. Boolean değer döndürür. |
| CanSeek | Stream'in aramayı destekleyip desteklemediğini öğrenmeyi sağlar. Boolean değer döndürür. |
| CanWrite | Stream'in yazmayı destekleyip desteklemediğini öğrenmeyi sağlar. Boolean değer döndürür. |
| Length | Stream'in byte olarak uzunluğunu öğrenmeyi sağlar. Long değer döndürür. |
| Position | Cursor'ün yerini öğrenmeyi veya değiştirmeyi sağlar. Long değer alır ve Long değer döndürür. |

**FileStream Sınıfının Metodları**

|  |  |
| --- | --- |
| **Metod** | **İşlev** |
| Close() | Stream'i kapatır ve bu stream'e ait olan kaynakları serbest bırakır. |
| CopyTo(hedef AS Stream) | Kaynak stream nesnesi içeriğini hedef stream nesnesine aktarmayı sağlar. |
| Dispose() | Stream'in kullandığı tüm kaynakları serbest bırakmayı sağlar. |
| Flush() | Stream'e ait buffer'ı temizler ve buffer'daki bütün verilerin dosyaya yazılmasını sağlar. |
| Read(buffer AS Byte(),indis AS Integer,sayi AS Integer) AS Integer | Üçüncü parametrede verilen byte kadar bilgiyi birinci parametrede verilen diziye ikinci parametrede verilen indisten itibaren yerleştirir. Toplam okunan byte sayısını geri döndürür. |
| ReadByte() AS Integer | Stream'de cursor'un bulunduğu noktadan  itibaren bir bytelık veri okumayı sağlar. Eğer -1 değeri dönerse, dosya sonuna gelinmiş demektir. |
| Seek(konum AS Long,başlangıç AS SeekOrigin) AS Long | Stream'deki cursor'ün ikinci parametrede verilen başlangıçtan itibaren birinci parametrede verilen pozisyona konumlanmasını sağlar. Stream içerisindeki yeni pozisyonu geri döndürür. |
| Write(buffer AS Byte(),indis AS Integer,sayi AS Integer) | Stream'e birinci parametrede verilen veri dizisindeki bilgilerin ikinci parametrede verilen indisten itibaren üçüncü parametrede verilen sayı kadar yazılmasını sağlar. |
| WriteByte(veri AS Byte) | Stream'e ilk parametrede verilen bir byte'lık veriyi yazmayı sağlar. |

**StreamReader Sınıfı**

StreamReader sınıfı bir dosyadan karakter veya karakterler okumayı sağlar.

**Kurucu Metotlar:**

StreamReader(string path);

**Yol**: Okunmak istenen dosyanın yolunun verildiği parametredir.

StreamReader(string path, Encoding encoding);

**Yol**: Okunmak istenen dosyanın yolunun verildiği parametredir.

**Kodlama**: Dosyanın istenen karakter kodlama sistemi ile okunmasını sağlar.

**StreamReader Sınıfı Özellikleri**

|  |  |
| --- | --- |
| **Özellik** | **İşlev** |
| CurrentEncoding | StreamReader nesnesinin kullandığı karakter kodlama sistemini öğrenmeyi sağlar. |
| EndOfStream | Cursor'ün stream'in sonunda olup olmadığını öğrenmeyi sağlar. Boolean değer döndürür. |

**StreamReader Sınıfı Metodları**

|  |  |
| --- | --- |
| Metod | İşlev |
| Close() | Stream'i kapatır ve bu stream'e ait olan kaynakları serbest bırakır. |
| Dispose() | Stream'in kullandığı tüm kaynakları serbest bırakmayı sağlar. |
| Peek() AS Integer | Sırada okunacak karakter olup olmadığını öğrenmeyi sağlar. Geriye Integer değer döndürür. -1 dönerse sırada okunacak karakter yoktur. |
| Read() AS Integer | Sıradaki karakteri okumayı sağlar. Geriye Integer değer döndürür. -1 dönerse sırada okunacak karakter yoktur. |
| Read(buffer AS Byte(),indis AS Integer,sayi AS Integer) AS Integer | Üçüncü parametrede verilen byte kadar bilgiyi birinci parametrede verilen diziye ikinci parametrede verilen indisten itibaren yerleştirir. Toplam okunan byte sayısını geri döndürür. |
| ReadLine() AS String | Sıradaki satırı okumayı sağlar. |
| ReadToEnd() AS String | Stream'de cursor'un bulunduğu pozisyondan stream'in sonuna kadar tüm karakterleri okumayı sağlar. |

**StreamWriter Sınıfı**

StreamWriter sınıfı bir dosyaya karakter veya karakterler yazmayı sağlar.

**Kurucu Metotlar:**

StreamWriter(string path);

**Yol**: Bilgi yazılmak istenen dosyanın yolunun verildiği parametredir.

StreamWriter(string path, bool append, Encoding encoding);

**Yol**: Okunmak istenen dosyanın yolunun verildiği parametredir.

**Bilgi**: Eğer true verilirse yazılan bilgi dosyaya eklenir, eğer false verilirse dosya üzerine yazılır.  
**Kodlama**: Dosyanın istenen karakter kodlama sistemi ile okunmasını sağlar.  
  
**StreamWriter Sınıfı Özellikleri**

|  |  |
| --- | --- |
| **Özellik** | **İşlev** |
| AutoFlush | Her yazma işleminden sonra buffer'daki bilginin dosyaya yazılması isteniyorsa true, aksi takdirde false yapılmalıdır. |
| Encoding | StreamWriter nesnesinin kullandığı karakter kodlama sistemini öğrenmeyi sağlar. |
| NewLine | Yeni satıra geçme karakterini öğrenmeyi veya değiştirmeyi sağlar. |

**StreamWriter Sınıfı Metotları**

|  |  |
| --- | --- |
| **Metod** | **İşlev** |
| Close() | Stream'i kapatır ve bu stream'e ait olan kaynakları serbest bırakır. |
| Dispose() | Stream'in kullandığı tüm kaynakları serbest bırakmayı sağlar. |
| Flush() | Bütün buffer'ı temizler ve buffer'daki tüm veriyi dosyaya yazar. |
| Write(veri AS veritipi) | Write metodu tüm temel veri tiplerini içeren overload metodlara sahiptir. Stream'e ilgili veri tipindeki veriyi yazmayı sağlar. |
| WriteLine() | Strem'e yeni satıra geçme karakterinin yazılmasını sağlar. |
| WriteLine(veri AS veritipi) | WriteLine metodu tüm temel veri tiplerini içeren overload metodlara sahiptir. Stream'e ilgili veri tipindeki veriyi yazdıktan sonra yeni satıra geçme karakterini de yazmayı sağlar. |

**Directory Sınıfı**

Dizinler üzerinde oluşturma, taşıma, bazı özelliklerini öğrenme ve  
değiştirme gibi işlemler yapmak için statik metotlar sunan bir sınıftır.

**Directory Sınıfı Metotları**

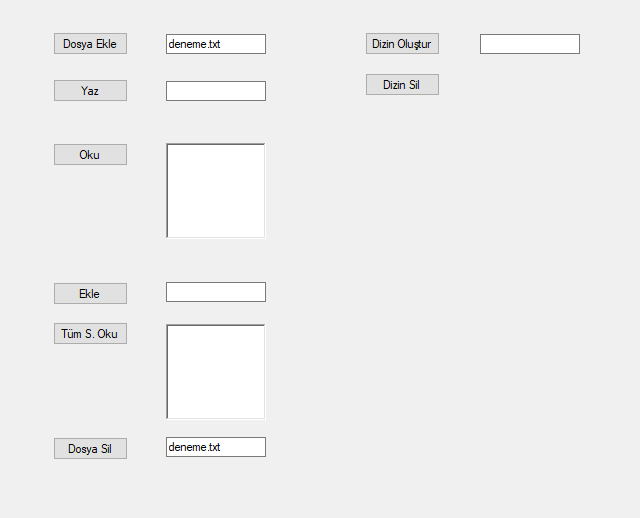
|  |  |
| --- | --- |
| **Metot** | İşlev |
| CreateDirectory(yol AS String) AS DirectoryInfo | Parametre olarak verilen yolda belirtilen klasörü oluşturur. |
| Delete(yol AS String) | Parametre olarak verilen yoldaki boş klasörü siler. |
| Delete(yol AS String, ozyineleme AS Boolean) | Eğer ikinci parametre true verilirse, parametre olarak verilen yolda bulunan dizin içerisindeki tüm alt dizinleri ve dosyaları siler. |
| EnumerateDirectories( yol AS String) AS IEnumerable(Of String) | Parametre olarak gönderilen yolda bulunan dizindeki tüm alt dizin isimlerini (yolları ile beraber) bir koleksiyon olarak geri döndürür. |
| EnumerateFiles(yol AS String) AS IEnumerable(Of String) | Parametre olarak gönderilen yolda bulunan dizindeki tüm dosya isimlerini (yolları ile beraber) bir koleksiyon olarak geri döndürür. |
| Exists(yol AS String) AS Boolean | Parametre olarak gönderilen yolda aynı isimli bir klasör olup olmadığını Boolean olarak geri döndürür. |
| GetCreationTime(yol as String) AS DateTime | Parametre olarak gönderilen yolda bulunan dizinin oluşturulma tarih ve saatini geri döndürür. |
| GetCurrentDirectory() AS String | Uygulamanın çalıştığı yolu verir. |
| GetDirectories(yol AS String) AS String() | Parametre olarak gönderilen yolda bulunan dizindeki tüm alt dizin isimlerini (yolları ile beraber) bir String dizisi olarak geri döndürür. |
| GetFiles(yol AS String) AS String() | Parametre olarak gönderilen yolda bulunan dizindeki tüm dosya isimlerini (yolları ile beraber) bir String dizisi olarak geri döndürür. |
| GetLastAccessTime(yol AS String) AS DateTime | Parametre olarak gönderilen yolda bulunan dizine son ulaşılma tarih ve saatini geri döndürür. |
| GetLogicalDrives() AS String() | Sistemdeki tüm mantıksal sürücüleri bir String dizisi olarak geri döndürür. |
| Move(kaynak AS String, hedef AS String) | Birinci parametre olarak verilen yoldaki kaynak dizini ikinci parametre olarak verilen yoldaki hedef dizine taşır. |
| SetCreationTime(yol AS String, zaman AS DateTime) | Parametre olarak gönderilen yolda bulunan dizinin oluşturulma zamanını parametre olarak gönderilen zaman yapar. |
| SetLastAccessTime(yol AS String, zaman AS DateTime) | Parametre olarak gönderilen yolda bulunan dizine son erişim zamanını parametre olarak gönderilen zaman yapar. |
| SetLastWriteTime(yol AS String, zaman AS DateTime) | Parametre olarak gönderilen yolda bulunan dizine son yazma zamanını parametre olarak gönderilen zaman yapar. |

**File Sınıfı**

Bir dosyayı oluşturma, kopyalama, silme, taşıma ve açma gibi işlemleri  
yapmak için statik metotlar sunan bir sınıftır.

**File Sınıfı Metodları**

|  |  |
| --- | --- |
| **Metod** | **İşlev** |
| Copy(kaynak AS String, hedef AS String) | Birinci parametre olarak verilen yoldaki kaynak dosyayı, ikinci parametre olarak verilen yola kopyalar. |
| Create(yol AS String) AS FileStream | Parametre olarak verilen yola ilgili dosyayı oluşturur. Eğer aynı dosya varsa üzerine yazar. Geriye oluşturulan dosyaya ait FileStream sınıfında bir nesne döndürür. |
| Delete(yol AS String) | Parametre olarak verilen yoldaki dosyayı silmeyi sağlar. |
| Exists(yol AS String) AS Boolean | Parametre olarak gönderilen yolda ilgili dosyanın bulunup bulunmadığı hakkında bilgi verir. Boolean bilgi döndürür. |
| GetAttributes(yol AS String) AS FileAttributes | Parametre olarak verilen yoldaki dosyanın özelliklerini öğrenmeyi sağlar. Bilgi FileAttributes enumeration olarak geri döndürülür. |
| GetCreationTime(yol AS String) AS DateTime | Parametre olarak verilen yoldaki dosyanın oluşturulma zamanını geri döndürür. |
| GetLastAccessTime(yol AS String) AS DateTime | Parametre olarak verilen yoldaki dosyaya son erişim zamanını geri döndürür. |
| GetLastWriteTime(yol AS String) AS DateTime | Parametre olarak verilen yoldaki dosyaya son yazma zamanını geri döndürür. |
| Move(kaynak AS String, hedef AS String) | Birinci parametre olarak verilen yoldaki kaynak dosyayı ikinci parametre olarak verilen yola taşır. |
| Open(yol AS String,mod FileMode ) AS FileStream | Birinci parametrede verilen yolda bulunan dosyayı, ikinci parametrede verilen mod'da açar. Geriye açılan dosyaya ait FileStream sınıfında bir nesne döndürür. |
| OpenRead(yol AS String) AS FileStream | Birinci parametrede verilen yolda bulunan dosyayı okumak için açar. Geriye açılan dosyaya ait FileStream sınıfında bir nesne döndürür. |
| OpenWrite(yol AS String) AS FileStream | Birinci parametrede verilen yolda bulunan dosyayı yazmak için açar veya dosya yoksa dosyayı yazmak için oluşturur. Geriye açılan veya oluşturulan dosyaya ait FileStream sınıfında bir nesne döndürür. |
| ReadAllLines(yol AS String) AS String() | Birinci parametrede verilen yolda bulunan dosyayı açar, bütün satırları okuyarak bir String diziye aktararak geri döndürür ve dosyayı kapatır. |
| SetAttributes(yol AS String, ozellik AS FileAttributes) | Birinci parametrede verilen yolda bulunan dosyaya ikinci parametrede verilen özellikleri atar. |
| SetCreationTime(yol AS String, zaman AS DateTime) | Parametre olarak gönderilen yolda bulunan dosyanın oluşturulma zamanını parametre olarak gönderilen zaman yapar. |
| SetLastAccessTime(yol AS String, zaman AS DateTime) | Parametre olarak gönderilen yolda bulunan dosyaya son erişim zamanını parametre olarak gönderilen zaman yapar. |
| SetLastWriteTime(yol AS String, zaman AS DateTime) | Parametre olarak gönderilen yolda bulunan dosyaya son yazma zamanını parametre olarak gönderilen zaman yapar. |
| WriteAllLines(yol AS String, icerik AS String()) | İlk parametrede verilen dosyayı oluşturur, ikinci parametrede verilen içerik dizisindeki tüm elemanları bu dosyaya yazar ve dosyayı kapatır. |



public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

StreamWriter SW = new StreamWriter("c:\\Temp\\deneme.txt");

SW.WriteLine(textBox1.Text);

SW.Close();

}

private void button2\_Click(object sender, EventArgs e)

{

try {

StreamReader SR = new StreamReader("c:\\Temp\\deneme.txt");

richTextBox1.Text = SR.ReadLine();

SR.Close();

}

catch {

MessageBox.Show("Dosya Bulunamadı...");

}

}

private void button3\_Click(object sender, EventArgs e)

{

StreamWriter SW = File.AppendText("c:\\Temp\\deneme.txt");

SW.WriteLine(textBox2.Text);

SW.Close();

}

private void button4\_Click(object sender, EventArgs e)

{

try {

richTextBox2.Clear();

StreamReader SR = new StreamReader("c:\\Temp\\deneme.txt");

string metin = SR.ReadLine();

while (metin != null)

{

richTextBox2.Text = richTextBox2.Text + "\n" + metin;

metin = SR.ReadLine();

}

SR.Close();

}

catch

{

MessageBox.Show("Dosya Bulunamadı...");

}

}

private void button5\_Click(object sender, EventArgs e)

{

try

{

if (textBox3.Text == "")

{

MessageBox.Show("Oluşturulacak Dizinin Adını Giriniz...");

}

else

{

Directory.CreateDirectory("C:\\Temp\\" + textBox3.Text);

MessageBox.Show("Dizin Başarı ile Oluşturuldu...");

}

}catch (Exception)

{

MessageBox.Show("Dizin Oluşturulamadı...");

}

}

private void button6\_Click(object sender, EventArgs e)

{

try

{

if (textBox3.Text == "")

{

MessageBox.Show("Silinecek Dizinin Adını Giriniz...");

}

else

{

Directory.Delete("C:\\Temp\\" + textBox3.Text);

MessageBox.Show("Dizin Başarı ile Silindi...");

}

}

catch (Exception)

{

MessageBox.Show("Dizin Silinemedi...");

}

}

private void button8\_Click(object sender, EventArgs e)

{

try

{

if (textBox5.Text == "")

{

MessageBox.Show("Oluşturulacak Dosyanın Adını Giriniz...");

}

else

{

File.Create(@"C:\\Temp\\" + textBox5.Text);

MessageBox.Show("Dosya Başarı ile Oluşturuldu...");

}

}

catch (Exception)

{

MessageBox.Show("Dosya Oluşturulamadı...");

}

}

private void button7\_Click(object sender, EventArgs e)

{

try

{

if (textBox4.Text == "")

{

MessageBox.Show("Silinecek Dosyanın Adını Giriniz...");

}

else

{

File.Delete(@"C:\\Temp\\" + textBox4.Text);

MessageBox.Show("Dosya Başarı ile Silindi...");

}

}

catch (Exception)

{

MessageBox.Show("Dosya Silinemedi...");

}

}

}

1. **Yazdırma İşlemleri**

Dokümandan takip edilecek.

1. **Grafik Uygulamaları**

GDI (Graphic Device Interface) Bitmap, Brush, Font, Graphics, Icon, Image ve Pen sınıfları kullanılarak anlatılacaktır.



**pictureBox**’ların özellik kısımlarından **SizeMode** özelliğini(properties) **StretchImage** yaparak resmi **pictureBox** boyutuna sığdırabiliriz. Aksi takdirde resmin sadece bir kısmı görülebilir.



public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

openFileDialog1.Filter = "Resim Dosyaları|" + "\*.bmp;\*.jpg;\*.gif;\*.wmf;\*.tif;\*.png";

if (openFileDialog1.ShowDialog() == DialogResult.OK)

{

pictureBox2.Image = null;

pictureBox1.Image = Image.FromFile(openFileDialog1.FileName);

}

}

private void button2\_Click(object sender, EventArgs e)

{

if (pictureBox1.Image == null)

{

MessageBox.Show("Önce bir resim Seçin");

}

else

{

progressBar1.Visible = true;//progress(işlem) çubuğu koyalım

int i, j;

Color r;

Bitmap bmp = new Bitmap(pictureBox1.Image);

progressBar1.Maximum = bmp.Width \* bmp.Height;

for (i = 0; i <= bmp.Width - 1; i++)

{

for (j = 0; j <= bmp.Height - 1; j++)

{

r = bmp.GetPixel(i, j);//i,j noktasının rengini öğren

//alfa kanalı hariç diğer kanalların tersini al

r = Color.FromArgb(r.A, (byte)~r.R, (byte)~r.G, (byte)~r.B);

bmp.SetPixel(i, j, r);//aynı noktaya tekrar koy

if ((i % 10) == 0)//her on satırda bir göstergeyi güncelle

{

progressBar1.Value = i \* bmp.Height + j;

Application.DoEvents();

}

}

}

pictureBox2.Image = bmp;

progressBar1.Visible = false; //işlem bittiğinde çubuklar görünmez olsun

}//else sonu

}

private void button3\_Click(object sender, EventArgs e)

{

if (pictureBox1.Image == null)

{

MessageBox.Show("Önce bir resim Seçin");

}

else

{

progressBar1.Visible = true;

int i, j;

Color renk1, renk2, renk3;

Bitmap bmp = new Bitmap(pictureBox1.Image);

int r, g, b;

progressBar1.Maximum = bmp.Width \* bmp.Height;

for (i = 0; i <= bmp.Width - 2; i++)

{

for (j = 0; j <= bmp.Height - 2; j++)

{

renk1 = bmp.GetPixel(i, j);//i,j noktasının rengini öğren

renk2 = bmp.GetPixel(i + 1, j + 1);//sonraki noktanın rengini öğren

r = Math.Abs((int)(renk1.R) - renk2.R) + 128;

if (r > 255)

r = 255;

g = Math.Abs((int)(renk1.G) - renk2.G) + 128;

if (g > 255)

g = 255;

b = Math.Abs((int)(renk1.B) - renk2.B) + 128;

if (b > 255)

b = 255;

renk3 = Color.FromArgb(r, g, b);

bmp.SetPixel(i, j, renk3);

if ((i % 10) == 0)//her on satırda bir göstergeyi güncelle

{

progressBar1.Value = i \* bmp.Height + j;

Application.DoEvents();

}

}

}

pictureBox2.Image = bmp;

progressBar1.Visible = false;

}//else sonu

}

private void button4\_Click(object sender, EventArgs e)

{

if (pictureBox1.Image == null)

{

MessageBox.Show("Önce bir resim Seçin");

}

else

{

progressBar1.Visible = true;

int i, j;

Color renk;//Color sınıfından bir renk nesne tanımlıyoruz.

Bitmap bmp = new Bitmap(pictureBox1.Image);

//int r,g,b;

progressBar1.Maximum = bmp.Width \* bmp.Height;//İşlem çubuğunun maksimim olduğu yer for döngüsünün sonundaki piksel değerine erişmemiz durumundadır.

for (i = 0; i <= bmp.Width - 1; i++)//dikey olarak görüntümüzü tarıyoruz.

{

for (j = 0; j <= bmp.Height - 1; j++)//yatay olarak görüntümüzü tarıyoruz.

{

renk = bmp.GetPixel(i, j);

renk = Color.FromArgb((byte)((renk.R + renk.G + renk.B) / 3), (byte)((renk.R + renk.G + renk.B) / 3), (byte)((renk.R + renk.G + renk.B) / 3));

bmp.SetPixel(i, j, renk);

if ((i % 10) == 0)//her on satırda bir göstergeyi güncelle

{

progressBar1.Value = i \* bmp.Height + j;

Application.DoEvents();

}

}

}

pictureBox2.Image = bmp;

progressBar1.Visible = false;

}//else sonu

}

}

**Kaynaklar:**  
  
1. Kocaeli Üniversitesi İşaret ve Görüntü İşleme Laboratuvarı  
  
Hazırlayan: Yılmaz Ürgün   
Kocaeli Üniversitesi Elektronik ve Haberleşme Mühendisliği

Makale:  
C# ile Temel Görüntü İşleme Yöntemleri C#, Visual C# ve .NET Yılmaz Ürgün

1. <http://www.csharpnedir.com/articles/read/?id=657&title=C>



**Graphics Sınıfı**

private void button5\_Click(object sender, EventArgs e)

{

System.Drawing.Graphics grafiknesne;

grafiknesne = this.CreateGraphics();

//Kırmızı

Brush dolgu = new SolidBrush(System.Drawing.Color.Red);

//Sarı

Brush dolgu1 = new SolidBrush(System.Drawing.Color.Yellow);

//Siyah

Brush dolgu2 = new SolidBrush(System.Drawing.Color.Black);

grafiknesne.FillRectangle(dolgu, 20, 320, 200, 100);

grafiknesne.FillRectangle(dolgu1, 20, 425, 50, 50);

grafiknesne.FillEllipse(dolgu2, 20, 480, 100, 50);

grafiknesne.FillEllipse(dolgu, 20, 540, 50, 50);

}

private void button6\_Click(object sender, EventArgs e)

{

System.Drawing.Graphics grafiknesne;

grafiknesne = this.CreateGraphics();

//Dikdörtgen

Pen firca = new Pen(System.Drawing.Color.Red, 5);

grafiknesne.DrawRectangle(firca, 270, 340, 200, 50);

//Kare

Pen firca1 = new Pen(System.Drawing.Color.Blue, 5);

grafiknesne.DrawRectangle(firca1, 270, 400, 100, 100);

}

private void button7\_Click(object sender, EventArgs e)

{

System.Drawing.Graphics grafiknesne;

grafiknesne = this.CreateGraphics();

Pen firca = new Pen(System.Drawing.Color.Red, 5);

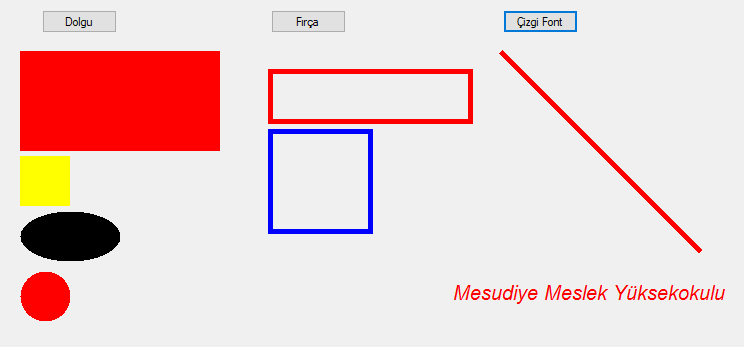
grafiknesne.DrawLine(firca, 500, 320, 700, 520);

Brush firca1 = new SolidBrush(System.Drawing.Color.Red);

Font yazi = new System.Drawing.Font("Helvetica", 15, FontStyle.Italic);

grafiknesne.DrawString("Mesudiye Meslek Yüksekokulu", yazi, firca1, 450, 550);

}



**Kaynak**: https://www.bilisimogretmeni.com/tag/c-grafik-kutuphanesi

1. **Örnek Uygulamalar**

Dokümandan takip edilecek.

1. **Hata Yakalama**

Programlarımızdaki hatalar ikiye ayrılır:

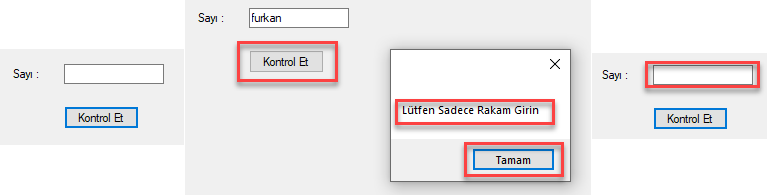
1. Derleme zamanı hataları
2. Çalışma zamanı hataları

Programımızda bir derleme zamanı hatası oluşursa derleme işlemi olmaz. Genellikle derleme zamanı hataları yanlış söz diziminden (sintaks) kaynaklanır ve tespiti kolaydır.

Çalışma zamanı hataları ise daha çok mantıksal yanlışlıklardan kaynaklanır ve tespiti nispeten daha zordur. Örneğin kullanıcının girdiği iki sayıyı toplayan bir program yazdığımızı düşünelim. Kullanıcı sayıların yerine herhangi bir harf girerse programımız çalışma zamanı hatası verir ve kendisini sonlandırır. Böyle bir durumu engellemek için toplama işlemini yapmadan önce kullanıcının girdiği veriyi bir kontrolden geçirebiliriz. İstisnai durum yakalama mekanizması programımızın bir çalışma zamanı hatasıyla karşılaştığında hata verip kendini sonlandırması yerine daha farklı işlemler yapılabilmesini sağlar.

**Örnek:**

Aşağıdaki örnekte textbox1 nesnesine string girildikten sonra buton tıklandığında kontrol yapılır ve string girildiği için textbox1 temizlenir.



private void textBox1\_Leave(object sender, EventArgs e)

{

try

{

int temp = Convert.ToInt32(textBox1.Text);

}

catch (Exception h)

{

MessageBox.Show("Lütfen Sadece Rakam Girin");

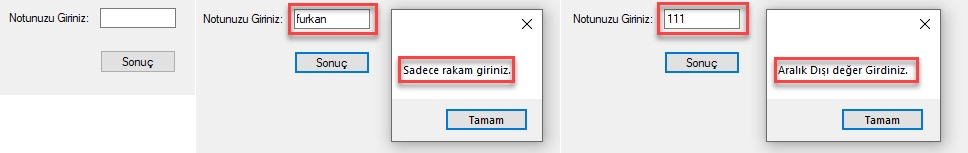
textBox1.Clear();

}

}

**Örnek:**

Aşağıdaki örnekde textbox2 nesnesine girilen değerler için birden fazla kontrol yapılmaktadır.



private void button2\_Click(object sender, EventArgs e)

{

try

{

int not = Convert.ToInt32(textBox2.Text);

if (not < 0 || not > 100)

throw new OverflowException();

//Exception("0-100 aralığı bir değer giriniz.");

if (not >= 50)

MessageBox.Show("Geçtiniz :)");

else

MessageBox.Show("Kaldınız :D");

}

catch (OverflowException)

{

MessageBox.Show("Aralık Dışı değer Girdiniz.");

}

catch (FormatException)

{

MessageBox.Show("Sadece rakam giriniz.");

}

catch (Exception m)

{

MessageBox.Show("Hata:" + m.Message);

}

}

1. **Methodlar**

**İsimsiz Metodlar (Anonymous Methods)**

* Adından da anlaşılacağı gibi, bir anonymous yöntem, isimsiz bir yöntemdir. C # 'daki anonymous yöntemler, temsilci (delegate) anahtar sözcüğü kullanılarak tanımlanabilir ve bir temsilci türü değişkenine atanabilir.

public delegate void Yaz(int value);

Yaz yaz = delegate (int deger) {

label4.Text = Convert.ToString(deger);

};

**yaz(200);**

//Lambda

Topla t1 = new Topla((a, b) => { return a + b; });

//Lambda

Topla t2 = (a, b) => { return a + b; };

//Anonymous Method

Topla t3 = delegate (int Sayi1, int Sayi2)

{

return Sayi1 + Sayi2;

};

label1.Text = Convert.ToString(**t1(1, 3)**);

label2.Text = Convert.ToString(**t2(2, 4)**);

label3.Text = Convert.ToString(**t3(3, 5)**);

**Metodlar**

1. **Method Bildirimi**

donusTipi methodIsmi ( parametreListesi )

{

method gövde ifadeleri - kod bloğu

return deger;

}

**donusTipi**, bir tür adıdır ve yöntemin işlenmesi sonucunda döndürdüğü bilgi türünü belirtir. Bu int veya string gibi herhangi bir tür olabilir. Değer döndürmeyen bir yöntem yazıyorsanız, dönüş türü yerine **void** anahtar sözcüğünü kullanmanız gerekir.

**methodIsmi**, yöntemi çağırmak için kullanılan addır. Yöntem adları, değişken adlarıyla aynı tanımlayıcı kurallara uyar.

**parametreListesi**, isteğe bağlıdır ve işlemesi için yönteme iletebileceğiniz bilgilerin türlerini ve adlarını açıklar. Parametreler, değişkenler bildiriyormuş gibi açılış ve kapanış parantezleri arasına, türün adını ve ardından parametrenin adı şeklinde yazılır. Yazdığınız yöntemin iki veya daha fazla parametresi varsa, virgülle ayırmanız gerekir.

**method\_gövde\_ifadeleri**, method çağrıldığında çalıştırılan kod satırlarıdır. Bunlar açılış ve kapanış ayraçları {} arasındadır.

1. **Method Çağrımı**

sonuc = methodIsmi ( bagimsizdegiskenListesi)

**methodIsmi**, aradığınız yöntemin adıyla tam olarak eşleşmelidir. Unutmayın, C# büyük / küçük harfe duyarlı bir dildir.

‘**sonuc =**’ kısmı isteğe bağlıdır. Belirtilirse, sonuçla tanımlanan değişken yöntem tarafından döndürülen değeri içerir. Yöntem geçersizse (yani, bir değer döndürmezse), ifadenin ‘sonuc =’ ifadesini atlamanız gerekir. ‘sonuc =’ yantümcesini belirtmezseniz ve yöntem bir değer döndürürse, yöntem çalışır ancak döndürülen değer atılır.

**bagimsizdegiskenListesi**, yöntemin kabul ettiği isteğe bağlı bilgileri sağlar. Her parametre için bir bağımsız değişken sağlamalısınız ve her bağımsız değişkenin değeri karşılık gelen parametrenin türüyle uyumlu olmalıdır. Aradığınız yöntemde iki veya daha fazla parametre varsa, bağımsız değişkenleri virgülle ayırmanız gerekir.

int degerleriTopla(int solText, int sagText)  
{

// ...  
}

degerleriTopla; // compile-time error, no parentheses  
degerleriTopla (); // compile-time error, not enough arguments  
degerleriTopla (39); // compile-time error, not enough arguments  
degerleriTopla ("39", "3"); // compile-time error, wrong types

int sonuc = degerleriTopla (39, 3); // on right-hand side of an assignment  
sonucGoster(degerleriTopla (39, 3)); // as argument to another method call

class Example  
{

void firstMethod()  
{

int myVar;  
...

}  
void anotherMethod()  
{

myVar = 42; // error – variable not in scope  
...

}  
}

class Example  
{

void firstMethod()  
{

myField = 42; // ok

...  
}

void anotherMethod()  
{

myField++; // ok

...  
}

int myField = 0;

}

1. **Overloading Methods**

İki tanımlayıcı aynı ada sahipse ve aynı kapsamda bildirildiyse, bunların aşırı yüklendiği söylenir. Genellikle aşırı yüklenmiş bir tanımlayıcı derleme zamanı hatası olarak bir hatadır.

static void Main() {

Console.WriteLineInt/String("The answer is ");

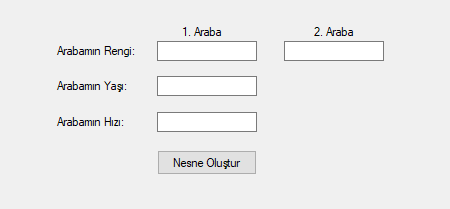
Console.WriteLine(42);

}

**\* Örnek eklenmiştir.**

**Classes**

**Örnek-1**



public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

Car myCar = new Car();

Car yourCar = new Car();

tb\_carRenk.Text = myCar.color;

myCar.color = "blue";

tb\_carRenk.Text = myCar.color;

tb\_carYas.Text = Convert.ToString(myCar.yas);

tb\_carRenk2.Text = yourCar.color;

myCar.maxSpeed = 151;

textBox1.Text = Convert.ToString(myCar.maxSpeed);

if (myCar.maxSpeed>150)

{

myCar.fullThrottle();

}

}

}

**Car.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsClass

{

class Car

{

// Class members

public string color = "red"; // field

public int yas = 25; // field

public int maxSpeed = 200; // field

public void fullThrottle() // method

{

MessageBox.Show("The car is going as fast as it can!");

}

}

}

**Örnek-2**

class Program

{

static void DoWork()

{

Point origin = new Point();

Point bottomRight = new Point(300, 1280);

double distance = origin.DistanceTo(bottomRight);

Console.WriteLine("Distance is: {0}", distance);

Console.WriteLine("No of Point objects: {0}", Point.ObjectCount());

}

static void Main(string[] args)

{

try

{

DoWork();

Console.ReadLine();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

}

}

**Point.cs**

#region Using directives

using System;

using System.Collections.Generic;

using System.Text;

#endregion

namespace Classes

{

class Point

{

private int x, y;

private static int objectCount = 0;

public Point()

{

this.x = -1;

this.y = -1;

objectCount++;

}

public Point(int x, int y)

{

this.x = x;

this.y = y;

objectCount++;

}

public double DistanceTo(Point other)

{

int xDiff = this.x - other.x;

int yDiff = this.y - other.y;

return Math.Sqrt((xDiff \* xDiff) + (yDiff \* yDiff));

}

public static int ObjectCount()

{

return objectCount;

}

}

}

**KAYNAKLAR**

1. Kaynak : Atatürk Üniversitesi Teknik Bilimler Meslek Yüksekokulu <https://yokdersleri.yok.gov.tr/#ders>
2. <https://tr.wikibooks.org/wiki/Kategori:C_Sharp_Programlama_Dili>
3. w3schools.com/
4. Visual C# 2010 Step by Step\_Code, John Shar, Microsoft Press
5. Herkes İçin C#, Herbert Schildth, Alfa Yayıncılık