

**T.C.
MİLLÎ EĞİTİM BAKANLIĞI**

BİLİŞİM TEKNOLOJİLERİ

NESNE TABANLI PROGRAMLAMA 4
482BK0077

Ankara 2011

- Bu modül, mesleki ve teknik eğitim okul/kurumlarında uygulanan Çerçeve Öğretim Programlarında yer alan yeterlikleri kazandırmaya yönelik olarak öğrencilere rehberlik etmek amacıyla hazırlanmış bireysel öğrenme materyalidir.
- Millî Eğitim Bakanlığınca ücretsiz olarak verilmiştir.
- PARA İLE SATILMAZ.

İÇİNDEKİLER

AÇIKLAMALAR	iii
GİRİŞ	1
ÖĞRENME FAALİYETİ-1	3
1. HATA AYIKLAMA ARAÇLARI.....	3
1.1. Hata Türleri	3
1.2. Debug Menüsü Elemanları.....	4
1.2.1. Start	4
1.2.2. Restart	4
1.2.3. Stop Debugging	4
1.2.4. Step Into.....	4
1.2.5. Step Over	4
1.2.6. Step Out	4
1.2.7. Run to Cursor.....	4
1.2.8. Watch.....	5
1.3. TaskList (Görev Listesi Penceresi)	7
1.4. Breakpoint ve Breakpoints Penceresi	8
1.5. Output Penceresi	9
1.6. Command Window (Komut Penceresi)	10
UYGULAMA FAALİYETİ	12
ÖLÇME VE DEĞERLENDİRME	14
ÖĞRENME FAALİYETİ-2.....	15
2. HATA AYIKLAMA VE İZ BULMA SINIFLARI	15
2.1. Trace ve Debug Sınıfları	15
2.2. TraceListener Sınıfları	16
2.2.1. DefaultTraceListener	17
2.2.2. TextWriterTraceListener	17
2.2.3. EventLogTraceListener	19
2.3. App.Config Dosyası.....	21
2.4. Trace ve Debug Sınıfıyla Kullanılan Komutlar	24
2.4.1. Write	24
2.4.2. WriteLine.....	24
2.4.3. WriteIf	24
2.4.4. WriteLineIf	24
2.4.5. Assert	24
2.4.6. Fail	26
UYGULAMA FAALİYETİ	29
ÖLÇME VE DEĞERLENDİRME	32
ÖĞRENME FAALİYETİ-3.....	33
3. İSTİSNAİ OLAYLAR	33
3.1. System.Exception Sınıfı.....	33
3.2. Try-Catch Kullanımı	33
3.3. Finally Kullanımı	38
3.4. Throw Rethrow Kullanımı	42
3.4.1. Throw Kullanımı	42
3.4.2. Rethrow Kullanımı	46
UYGULAMA FAALİYETİ	49

UYGULAMA FAALİYETİ	52
ÖLÇME VE DEĞERLENDİRME	55
CEVAP ANAHTARLARI.....	57
KAYNAKÇA	58

AÇIKLAMALAR

KOD	482BK0077
ALAN	Bilişim Teknolojileri
DAL/MESLEK	Veritabanı Programcılığı
MODÜLÜN ADI	Nesne Tabanlı Programlama 4
MODÜLÜN TANIMI	Nesne tabanlı programlamada hata ayıklama araçlarını kullanabilme, hata ayıklama sınıflarını oluşturabilme ve istisnai olayları kontrol edebilmeyle ilgili öğrenme materyalidir.
SÜRE	40/32
ÖN KOŞUL	Nesne Tabanlı Programlama 3 modülünü bitirmiş olmak
YETERLİK	Nesne tabanlı programdaki hataları gidermek
MODÜLÜN AMACI	Genel Amaç Gerekli ortam sağlandığında, programdaki hataları giderebilecek ve istisnai olayları kontrol altına alabileceksiniz. Amaçlar 1. Hata ayıklama araçlarını kullanabileceksiniz. 2. Hata ayıklama ve iz bulma sınıflarını kullanabileceksiniz. 3. İstisnai olayları kontrol edebileceksiniz.
EĞİTİM ÖĞRETİM ORTAMLARI VE DONANIMLARI	Ortam Atölye, laboratuvar, ev, bilgi teknolojileri ortamı (İnternet) vb, kendi kendinize veya grupta çalışabileceğiniz tüm ortamlar. Donanım Programlama dilini çalıştırabilecek yeterlikte bilgisayar, yedekleme için gerekli donanım (cd yazıcı, flash bellek), raporlama için yazıcı, sayfa için internet bağlantısı, kâğıt ve kalem.
ÖLÇME VE DEĞERLENDİRME	Modülün içinde yer alan her öğrenme faaliyetinden sonra verilen ölçme araçları ile kendinizi değerlendireceksiniz. Modül sonunda ise, bilgi ve beceriyi belirlemek amacıyla, öğretmeniniz tarafından belirlenecek ölçme aracıyla değerlendirileceksiniz.

GİRİŞ

Sevgili Öğrenci,

Okul yaşantınızda öğreneceğiniz her konu, yaptığınız uygulama ve tamamladığınız her modül bilgi dağarcığınızı geliştirecek ve ilerde atılacağınız iş yaşantınızda size başarı olarak geri dönecektir. Eğitim sürecinde daha öz verili çalışır ve çalışma disiplini kazanırsanız; başarılı olmamanız için hiçbir neden yoktur.

Günümüzde Windows tabanlı görsel programlama dillerinin hızla gelişmekte olduğu ve kullanımının oldukça yaygınlaştığı görülmektedir. Bu programlama dilleri ile sizler programlama mantığını ve becerisini çok daha kolay kavrayacaksınız.

Bu modülle, programın çalışması esnasında ortaya çıkan kullanıcı kaynaklı hataları kontrol altına alabilecek, hataların nasıl denetlenebileceğiyle ilgili düzenlemeleri editör menülerinden yapabilecek ve istisnai olaylarla ilgili bilgileri öğreneceksiniz.

Bu modülde anlatılan konuların tümünü öğrendiğinizde, Nesne Tabanlı Programlama dilinde yazmış olduğunuz programların hata oluşumunu en asgari seviyeye çekmiş olacak ve hatasız, kırılmayan programlar yazabileceksiniz.

ÖĞRENME FAALİYETİ-1

AMAÇ

Hata türlerini ve hata ayıklama araçlarını kullanarak program kod satırlarını kontrol altında tutabileceksiniz. Oluşabilecek hatalara karşı önlemler alabileceksiniz.

ARAŞTIRMA

- Bitirmiş olduğunuz modüllerde ne tür hatalarla karşılaştınız?
- Hataları gidermek için ne tür çözümler uyguladınız?

1. HATA AYIKLAMA ARAÇLARI

Hata, programın çalışması sırasında ortaya çıkan bir durumdur. Hatalar çoğunlukla kullanıcı kaynaklıdır. Örneğin, kullanıcı programda sayı girilmesi gereken alanda metin bilgisi girerse hata meydana gelir. Programcı bu nedenle oluşabilecek durumlar için hataları kontrol edebilen program kod satırları yazarak programın kırılmasını önler.

Hata ayıklamanın avantajı, bir hata oluştuğunda otomatik olarak çalıştırılan **ExceptionHandler** (İstisna Durum Yöneticisi) adında bir kod bloğu tanımlamanıza imkan vererek hata yönetimi verimliliğini artırmaktır. Bir diğer avantajı ise, çok sık oluşan program hataları için nesne tabanlı programlamada standart istisnai durumların tanımlanmış olmasıdır.

1.1. Hata Türleri

Hatalar üç çeşittir.

➤ Yazım hataları

Program içinde kullanılan fonksiyon adları, değişken adları, yapı isimleri ve komutların karakterlerinin yanlış yazımından meydana gelen hatalardır.

Örneğin, **MessageBox.Show** yazılması gerekirken **MessageBoxShow** yazılırsa yazım hatası oluşur. Oluşan bu hata görev listesi (**TaskList**) penceresinde görüntülenir.

➤ Çalışma zamanı hataları

Uygulamanın çalışması icra edilirken bir işlem çalışmayabilir. Örneğin, sifıra bölme hatası bu tür hatalara bir örnektir.

➤ Mantıksal hatalar

Bu hatalar uygulama derlenirken ve yürütülürken oluşur. Bu durumda program çalışsa bile yanlış sonuç üretir. Örneğin, yaş bilgisi girilmesi istenen bir alana negatif değer girilemez ya da bir işçinin maaşı hesaplanırken haftada 50 saat yazılması gereken yere 150 saat yazılırsa program çalışır ancak sonuç doğru olmaz. Bu hata bir mantık hatasıdır.

1.2. Debug Menüsü Elemanları

1.2.1. Start

Uygulamayı çalıştırmak için kullanılır. Klavye kısayol tuşu F5'tir.

1.2.2. Restart

Debug modunda iken çalışmakta olan uygulamayı sonlandırır ve uygulamanın yürütülmesini yeniden başlatır. Kısayol tuşu Ctrl+Shift+F5'tir.

1.2.3. Stop Debugging

Hata ayıklama işlemini sonlandırır ve tasarım moduna geri döner. Kısayol tuşu Shift+F5'tir.

1.2.4. Step Into

Program kodlarını satır satır çalıştırmak için kullanılır. Kısayol tuşu F11'dir.

Eğer bir sonraki satır, bir metodu çağırırsa Step Into, o metodun başlangıcında durur. Yani, sadece bulunduğu metod içinde çalışır.

```
private void button1_Click(object sender, System.EventArgs e)
{
    byte i;int toplam=0;
    int[] dizil= new int[5];
    for(i=0;i<5;i++)
```

1.2.5. Step Over

Program satırlarını adım adım çalıştırmak için kullanılır. Bir sonraki satırda çağrılan bir metotsa çağrılan metodun içine girmeden metodu çalıştırır ve bir sonraki satırdan itibaren çalışmaya devam eder. Kısayol tuşu F10'dur.

1.2.6. Step Out

Çalıştırılan metodun içindeki belirtilen bütün komutlar işletilir. Eğer metod içerisinden bir başka metod çağrılıyorsa, çağrılan metod işletilerek metodun işleyişi bir sonraki satırdan devam eder.

1.2.7. Run to Cursor

İmlecin bulunduğu satırdan itibaren programın çalışmasını yürütür.

1.2.8. Watch

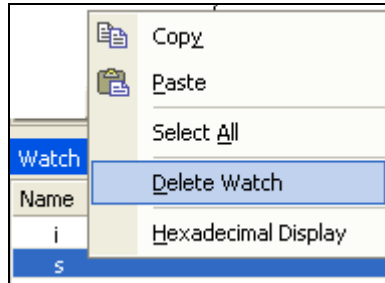
Bir deęişken ya da ifadenin programın icrası esnasında hangi deęerleri aldığını görmenizi sağlar. Görmek istediğiniz deęişkenin üzerinde iken sağ tıkla açılan menüden **Add Watch** komutu tıklanarak deęişken **Watch** penceresine aktarılır ve burada deęerleri izlenir.

Burada dikkat edilmesi gereken husus, program **debug** modunda olmalı ve adım adım çalıştırılmalıdır. Bu şartlarda deęişkenlerin deęerleri izlenebilir.



Resim 1.1: Watch penceresi

Watch penceresinde bulunan deęişkeni çıkarmak için deęişken üzerinde sağ tık yapılır ve **Delete Watch** komutu çalıştırılır.



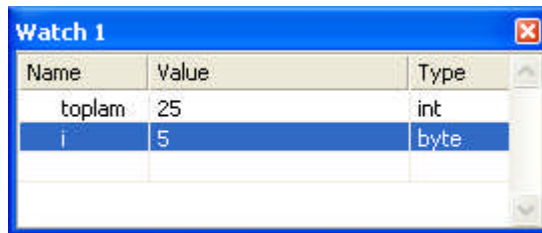
Resim 1.2: Watch penceresinden deęişken silmek

Çalışma zamanında debug menüsünden **QuickWatch** komutu çalıştırılırsa ekrana gelen pencerede public olan tüm değişkenler ve işletilen metot içindeki diğer değişkenler izlenir.



Resim 1.3: QuickWatch iletişim kutusu

Recalculate düğmesiyle seçilen değişkenin değeri tekrar hesaplatılabilir. Seçilen değişken istenirse Add Watch düğmesiyle Watch penceresine de aktarılabilir.

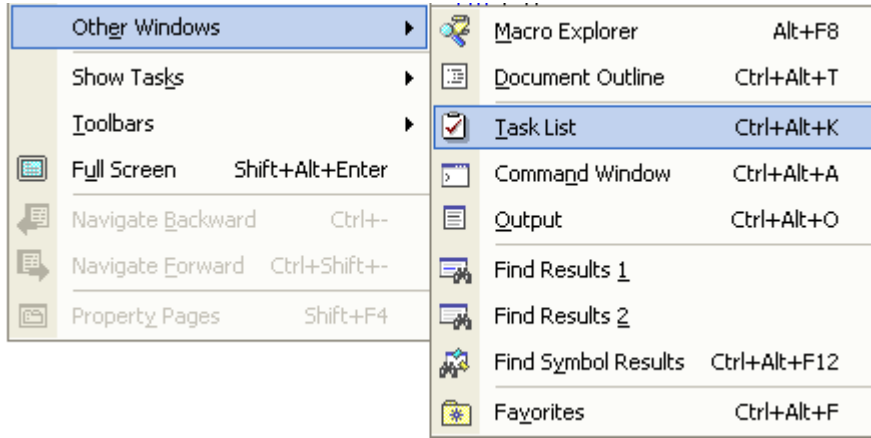


Resim 1.4: Seçilen değişkenin Watch penceresine aktarılması

1.3. TaskList (Görev Listesi Penceresi)

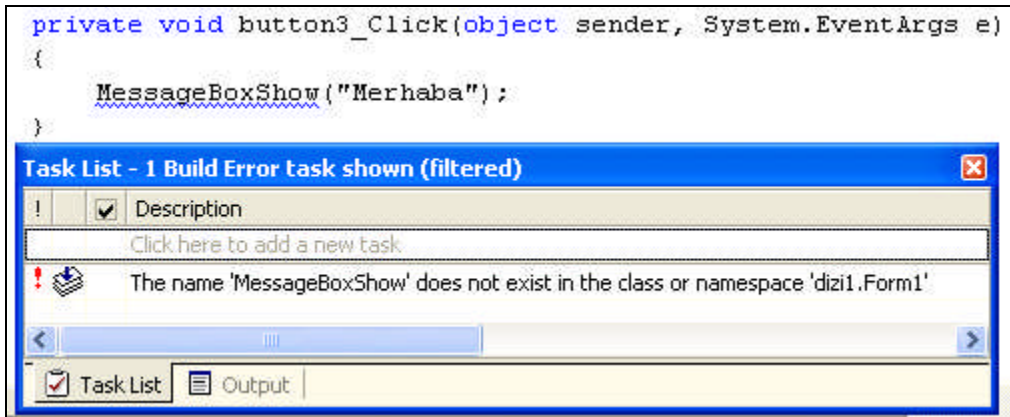
Program içinde tanımlanmayan değişken, metod ya da yazım hatalarını gösteren penceredir.

TaskList penceresini açmak için Project menüsünden Other Windows seçeneği üzerine gelip açılan menüden Task List komutu seçilmelidir.



Resim 1.5: Project menüsünden TaskList pencersini açmak

Diğer bir yöntem olarak ise, klavye kısayol tuşu Ctrl+Alt+K'yı kullanmaktır.

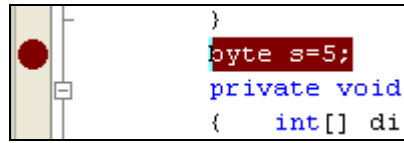


Resim 1.6: Programın derlenmesinde hatanın Tasklist'te gösterilmesi

1.4. Breakpoint ve Breakpoints Penceresi

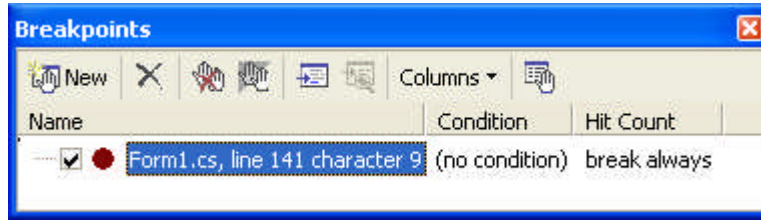
Programın işleyişini istediğiniz bir noktada durdurmak için kod satırına Breakpoint (kesme noktası) ekleyebilirsiniz. Böylece program sizin işaret ettiğiniz kesme noktasında duracaktır.

Bir kod satırına Breakpoint eklemek için F9 kısayol tuşu veya sağ tıklama açılan menüden “**Insert Breakpoint**” komutunu vermeniz gerekir. Böylece kesme noktası eklenen program satırı şekildeki gibidir.



Resim 1.7: Programa Breakpoint eklenmesi

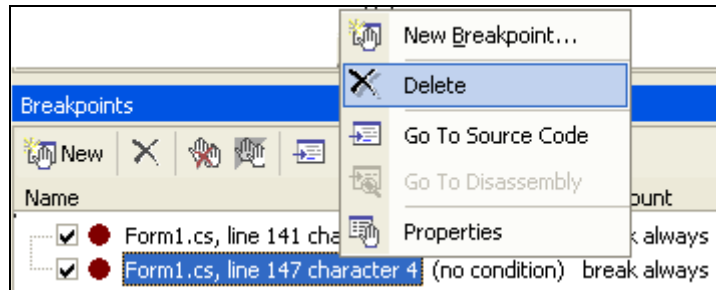
Kesme noktası eklenmiş satır Breakpoints penceresinde gösterilir.



Resim 1.8: Breakpoints penceresi

Program çalıştırılıp kesme noktası eklenen satıra geldiğinde programın çalışması durur. Kırmızı zemin rengi sarıya döndürür.

Breakpoint'i silmek için Breakpoints penceresindeki kesme noktası üzerine sağ tıklanıp açılan menüden **Delete** komutu seçilir.

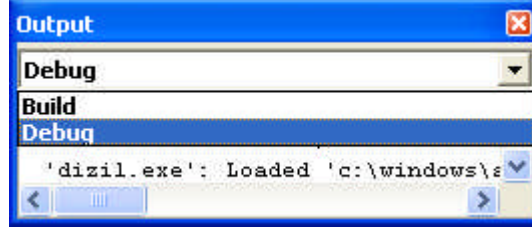


Resim 1.9: Breakpoints penceresinden kesme noktasının silinmesi

1.5. Output Penceresi

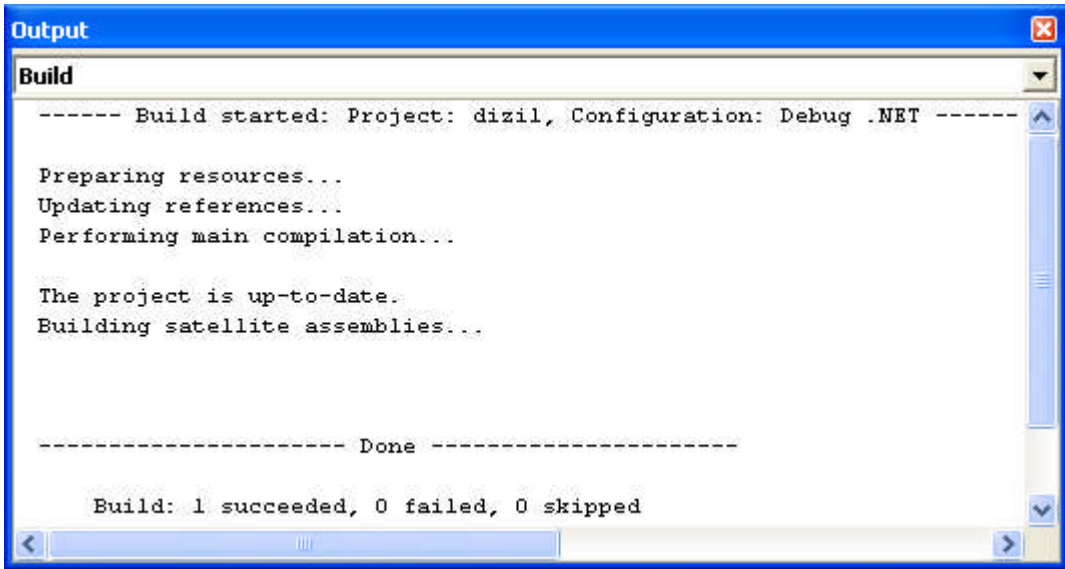
Programın derlenmesi aşamalarının gösterildiği penceredir. Kısayol tuşu Ctrl+Alt+O'dur.

Output penceresinde bulunan açılır liste kutusunda iki seçenek vardır. Bunlar Build ve Debug'dır.



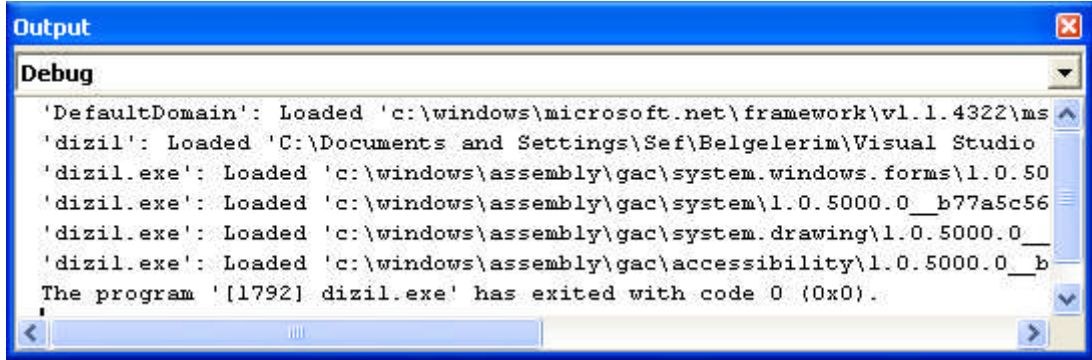
Resim 1.10: Output penceresinde Build ve Debug seçenekleri

Açılır liste kutusundan Build seçildiğinde derlemeyle ilgili bilgiler görüntülenir.



Resim 1.11: Output penceresinde Build seçimi

Debug seçildiğinde ise programı EXE haline dönüştürürken kullanılan dosyalar görüntülenir.

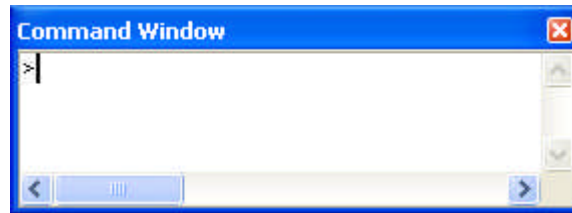


Resim 1.12: Output penceresinde Debug seçimi

1.6. Command Window (Komut Penceresi)

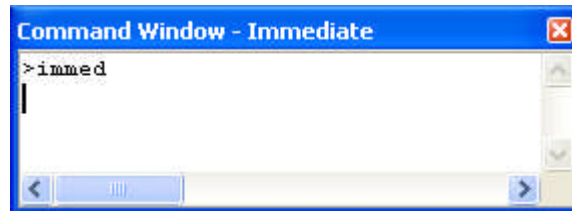
Uygulamanızı çalışma zamanında takip etme işlemine **Trace** denir. Trace anında, kod satırlarında yer alan değişkenlerin o anki değerlerini görmeyi ve bu değerlerle matematiksel işlem yapmayı sağlayan pencere komut penceresidir. Kısayol tuşu Ctrl+Alt+A'dır.

Komut penceresi ilk açıldığında Command Window penceresi ekrana gelir



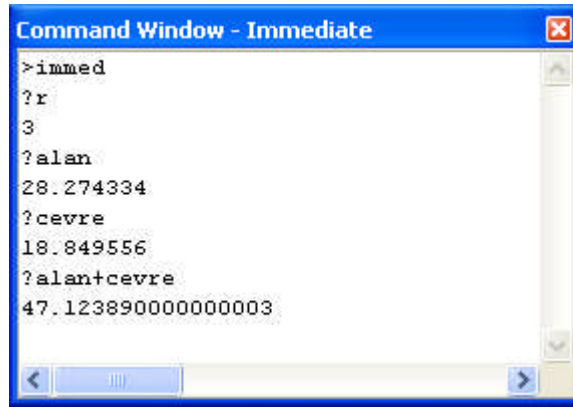
Resim 1.13: Command Window penceresi

Kod satırlarıyla ilgili işlem ve hesaplamalar yapabilmek için Command Window'da iken **immed** yazılarak Command Window'un **Immediate** moduna geçilir.



Resim 1.14: Immediate modu

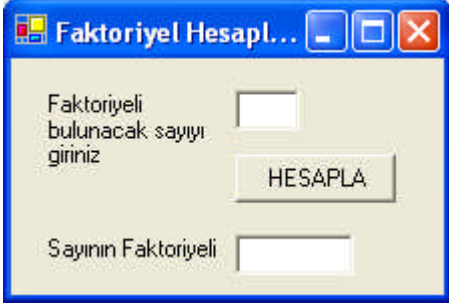
Bu moda değişkenlerle ilgili işlem yapabilmek için ilk önce ? (soru işareti) yazılır.

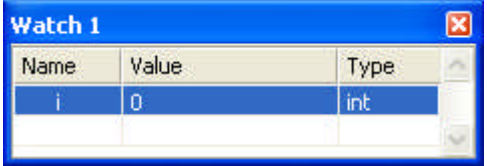
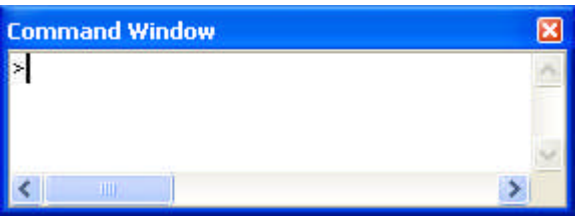
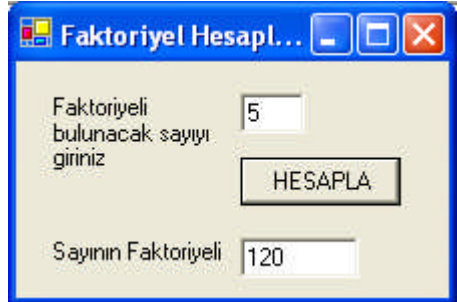


Resim 1.15: Immediate modunda işlem yapılması

Tekrar Command Window'a geçmek istenirse > **cmd** satırı yazılır.

UYGULAMA FAALİYETİ

İşlem Basamakları	Öneriler
<p>➤ Bir Windows Application oluşturarak formu aşağıdaki gibi tasarlayınız.</p> 	<p>➤ 2 metin kutusu, 2 Label ve 1 buton olduğundan formun boyutunu küçük tasarlayınız.</p>
<p>➤ Girilen sayının faktoriyeli “HESAPLA” düğmesine tıklandığında bulunacağı için program kodlarını Click() metoduna yazınız.</p>	<p>➤ “HESAPLA” düğmesinin üzerinde çift tıklayınız.</p>
<p>➤ “int” tipinde bir değişken tanımlayınız.</p>	<p>➤ int sayi;</p>
<p>➤ Metin kutusundan girilen sayıyı değişkene aktarınız.</p>	<p>➤ Metin kutusundan girilen bilgi string tipte olduğu için “Parse” metoduyla int tipine çeviriniz.</p> <p>➤ sayi=int.Parse(textBox1.Text);</p>
<p>➤ Sonucu aktaracağınız değişkeni int tipinde tanımlayarak 1 (bir) değerini atayınız.</p>	<p>➤ int faktor=1;</p> <p>➤ Çarpma işleminde 0 (sıfır) yutan eleman olduğu için faktor değişkeni 1 ile başlamalıdır.</p>
<p>➤ 1’den başlayarak faktoriyeli bulunacak sayıdan küçük ve eşit olacak şekilde bir for döngüsü bloğu oluşturunuz.</p>	<p>➤ for (int i=0;i<=sayi;i++) { }</p>
<p>➤ for bloğu içine faktoriyel hesabını yapacak kodları yazınız.</p>	<p>➤ Faktoriyel hesabı 1’den başlayarak belirtilen sayıya kadar bu sayıların birbiriyle çarpımıdır. Buna girilen sayı da dâhildir. faktor=faktor*i;</p>
<p>➤ Sonucu ikinci metin kutusuna string tipe çevirerek yazdırınız.</p>	<p>➤ textBox2.Text=faktor.ToString(); kod satırını yazınız.</p>
<p>➤ Sayı girişi yapıldığı satırdan itibaren işleyişi test ediniz.</p>	<p>➤ Sayi=int.Parse(textBox1.Text); satırının başında sağ tık yapıp Run to Corsor komutunu veriniz. ve F11 (Step Over) tuşunu kullanınız.</p>
<p>➤ Programı tekrar çalıştırınız. Çalışma</p>	<p>➤ Değişken üzerinde sağ tık yaparak</p>

<p>zamanında bir Watch penceresi açarak “i” değişkeninin aldığı değerleri görünüz.</p> 	<p>Add Watch komutunu kullanınız.</p>
<p>➤ Bu işlem bir kere gerçekleştiğinde faktor değişkenini de Watch penceresine ekleyiniz.</p>	<p>➤ Add Watch komutunu kullanabilirsiniz.</p>
<p>➤ Command Window’u açarak faktor değişkeninin aldığı son değeri ikiye bölünüz.</p> 	<p>➤ Command Window’da immed yazarak işlem yapabilirsiniz.</p>
<p>➤ Uygulamanın sonuç görüntüsünü yandaki resimle karşılaştırınız.</p>	

ÖLÇME VE DEĞERLENDİRME

Aşağıdaki soruları dikkatlice okuyarak doğru/yanlış seçenekli sorularda uygun harfleri yuvarlak içine alınız. Seçenekli sorularda ise uygun şıkkı işaretleyiniz. Boşluk doldurmalı sorularda boşluklara uygun cevapları yazınız.

1. ExceptionHandler (İstisna durum yöneticisi) hata yönetiminin verimliliğini artırır. (D/Y)
2. Mantıksal hata oluştuğunda program çalışmaz. (D/Y)
3. Uygulamaya yeni kod satırları eklendiğinde programedilmelidir.
4. Uygulamayı imlecin bulunduğu satırdan itibaren çalıştırmak için komutu kullanılır.
5. Program Debug modunda iken ve adım adım çalıştırıldığında değişkenler Watch penceresinden izlenebilir. (D/Y)
6. Program yazımında hata oluştuğunda bu hatalar penceresinde görüntülenir.
7. Programda istediğiniz bir satıra kesme noktası (Breakpoint) eklemek için kısa yol tuşu kullanılır.
8. Komut penceresinde işlem yapabilmek için moduna geçilmelidir.
9. Hata ayıklama işlemini sonlandıran komut aşağıdakilerden hangisidir?
A) Step Into
B) Step Out
C) Step Over
D) Stop Debugging
10. Programı adım adım çalıştırırken çağrılan metodun içini işletmeden çalıştıran Step komutu aşağıdakilerden hangisidir?
A) Step Into
B) Step Out
C) Step Over
D) Stop Debugging

DEĞERLENDİRME

Cevaplarınızı cevap anahtarı ile karşılaştırınız. Doğru cevap sayınızı belirleyerek kendinizi değerlendiriniz. Yanlış cevap verdiğiniz ya da cevap verirken tereddüt yaşadığınız sorularla ilgili konulara geri dönerek tekrar inceleyiniz. Tüm sorulara doğru cevap verdiyseniz diğer öğrenme faaliyetine geçiniz.

ÖĞRENME FAALİYETİ-2

AMAÇ

Programda oluşabilecek hataları çeşitli yöntemlerle önceden tespit edebilmeyi ve bu yöntemlerin neler olduğunu kavrayabileceksiniz.

ARAŞTIRMA

- Nesne tabanlı programlamanın derleyicisi hataları nasıl tespit etmektedir? Araştırınız.

2. HATA AYIKLAMA VE İZ BULMA SINIFLARI

2.1. Trace ve Debug Sınıfları


Bu sınıfların kullanımıyla dosyalama mekanizmasının (Log) oluşturulması ve yönetilmesi işlemleri yapılmaktadır. Bu sınıfları kullanabilmek için **using** deyiimiyle birlikte **System.Diagnostics** namespace'inin projenin başına yazılması gerekir.

Bu iki sınıf, proje geliştirme aşamasında hata mesajlarını, projenin uygulanması aşamasında ise gerekli dosyalama bilgilerini gerekli yerlere yazdırmak açısından birçok kolaylık sunar. Trace ve Debug sınıflarının tüm özellikleri aynıdır. Tek fark, Debug sınıfı Release uygulaması içine derlenmemesidir.

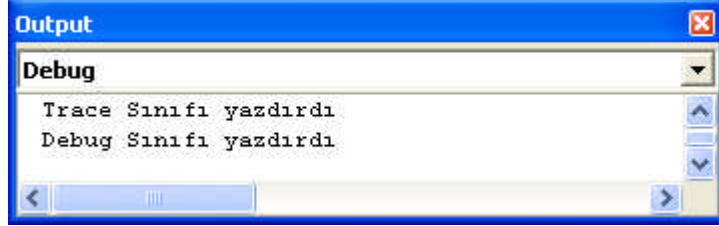
Örnek

Formunuzun üzerinde çift tıklayarak Form'un Load metoduna aşağıdaki kod satırlarını yazınız. Trace ve Debug sınıflarının çalışmasını basit olarak bu şekilde görebilirsiniz.

```
private void Form1_Load(object sender, System.EventArgs e)
{
    Trace.WriteLine("Trace Sınıfı yazdırdı");
    Debug.WriteLine("Debug Sınıfı yazdırdı");
}
```

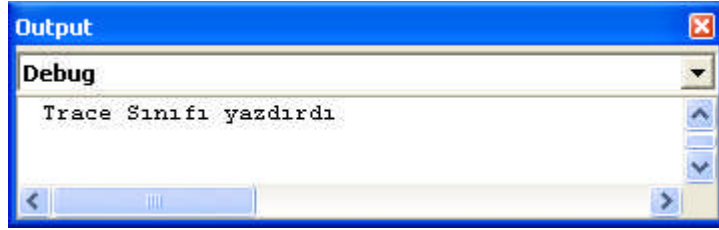
Bu kod satırları yazıldığında projenizin  modunda olmasına dikkat ediniz.

Output penceresinde yürütülen işlem resim 2.1’deki gibi yansıtacaktır.



Resim 2.1: Trace ve Debug sınıflarının işleyişi

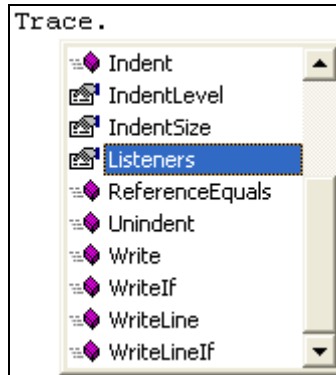
Eğer Release modundayken programı çalıştırırsanız Debug sınıfı işletilmeyecek sadece Trace sınıfı işletilecektir.



Resim 2.2: Release modunda Trace sınıfı işletilir.

2.2. TraceListener Sınıfları

Kendilerine gelen bilgiyi tutarak ilgili kaynaklara gönderebilen sınıflar Trace ve Debug sınıflarıdır. Bu işlemleri yapabilmek için **TraceListener** nesneleri kullanılır. Trace ve Debug sınıflarının **Listeners** (dinleyici) özelliği TraceListener tipinde koleksiyon referansı tutar.

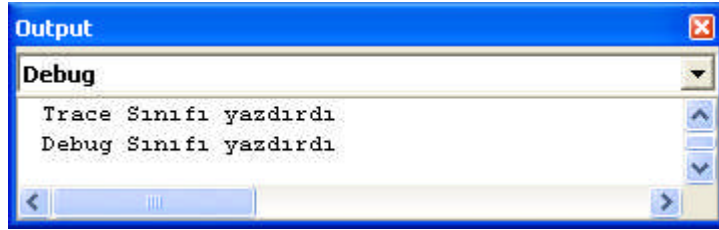


Resim 2.3: Listeners özelliği

Bu koleksiyona mevcut TraceListener nesneleri eklenebildiği gibi siz de bir TraceListener sınıfı oluşturup ekleyebilirsiniz. Hem Trace hem de Debug sınıfı aynı TraceListener koleksiyonunu paylaşır. Bu sınıflara bir Listener eklenirse bu Listener hem Trace hem de Debug sınıfı için geçerli olur.

2.2.1. DefaultTraceListener

Trace ve Debug sınıflarının TraceListener koleksiyonuna varsayılan olarak eklenmiştir. Trace ve Debug'a gönderilen bilgi Output penceresinde gösterilmektedir.



Resim 2.4: Trace ve Debug TraceListener'a default olarak eklenmiştir.

2.2.2. TextWriterTraceListener

Bu listener sınıfı Trace ve Debug sınıflarına iletilen bilgiyi bir dosyaya (Log) yazmak için kullanılır.

Örnek

Bu örnekte metin kutularından girilen iki sayı toplatılıp sonuç yine bir metin kutusuna ve ayrıca, girilen sayılar ve sonuç TextWriterTraceListener ile bir dosyaya yazdırılmaktadır. Bunun için formu resim 2.5'teki gibi tasarlayınız.

The image shows a Windows form titled "Trace ve De...". The form has a light beige background. It contains three text boxes: "Birinci sayı", "İkinci sayı", and "Sonuc". Between the "İkinci sayı" and "Sonuc" text boxes, there is a button labeled "Topla ve dosyaya yazdır". The form has a standard Windows window border with a title bar and control buttons (minimize, maximize, close).

Resim 2.5: Formun tasarlanmış hali

Form tasarımımdan sonra forma eklediğiniz butonun üzerinde çift tıklayarak Click metoduna aşağıdaki kodları yazınız.

```
private void button1_Click(object sender, System.EventArgs e)
{
    int a,b,son;
    a=int.Parse(textBox1.Text);
    b=int.Parse(textBox2.Text);
    son=a+b;
    textBox3.Text=Convert.ToString(son);
    MessageBox.Show("Sonuç dosyaya kaydedildi");

    FileStream dosya = new FileStream("Sonuc.txt",FileMode.OpenOrCreate);

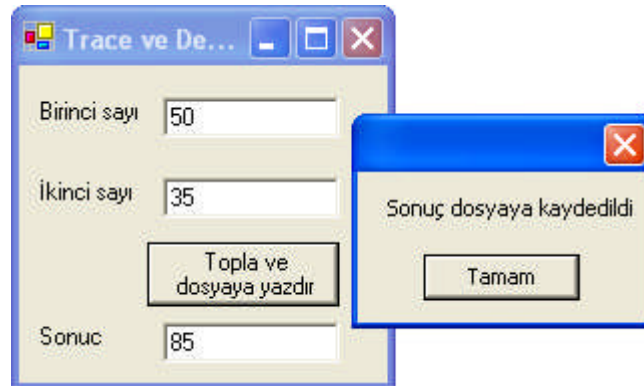
    TextWriterTraceListener dinleyici = new TextWriterTraceListener(dosya);
    Trace.Listeners.Add(dinleyici);

    Trace.WriteLine(Convert.ToString(a),"Birinci sayı");
    Trace.WriteLine(Convert.ToString(b),"İkinci sayı ");
    Trace.WriteLine(Convert.ToString(son),"Sonuç      ");

    Debug.WriteLine(Convert.ToString(a),"Birinci sayı");
    Debug.WriteLine(Convert.ToString(b),"İkinci sayı ");
    Debug.WriteLine(Convert.ToString(son),"Sonuç      ");

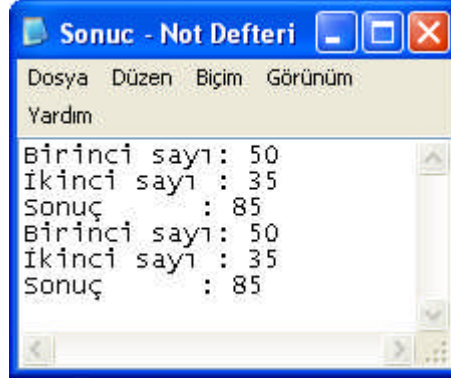
    Trace.Flush();
    dosya.Close();
}
```

Birinci metin kutusundan girilen sayıyı a değişkeni, ikinci metin kutusundan girilen sayıyı b değişkeni tutacaktır. İkisinin toplamı da sonuc değişkenine aktarılacaktır. Girilen sayıları ve sonucu dosyaya yazabilmek için FileStream sınıfından “dosya” adlı bir nesne oluşturulur. TextWriterTraceListener sınıfından da “dinleyici” adında bir nesne oluşturularak sonucun “dosya”ya yönlendirilmesi sağlanır. Oluşturulan dinleyici nesnesi Listener sınıfına eklenir.



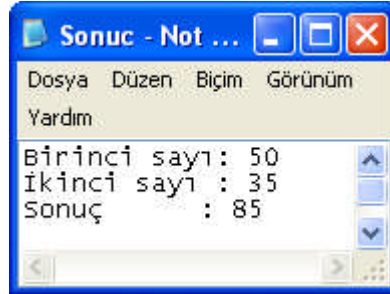
Resim 2.6: Programın çalıştırılıp sonuçların üretilmesi

Program çalıştırılıp sonuçlar üretildikten sonra uygulamanızın varsayılan çalışma klasöründeki “Bin” klasörü altındaki Debug klasörüne program kodunda adını verdiğiniz dosya oluşturulur (Sonuc.txt).



Resim 2.7: Sonuçların dosyaya yazdırılması

Dosya içeriğine bakacak olursanız, veriler iki defa dosyaya yazılmış durumdadır. Bunun sebebi, projenizin Debug modundayken derlenmiş olmasıdır. Eğer Debug modunda değil de Release modunda çalıştırmış olsaydınız resim 2.8’deki gibi bir sonuç alırdınız. Oluşturulan “Sonuc.txt” dosyası da “Bin” klasörü altındaki “Release” klasörü içine oluşturulurdu.



Resim 2.8: Release modunda çalıştırılan projenin ürettiği sonuç

2.2.3. EventLogTraceListener

Trace ve Debug sınıflarına gelen veriyi Windows Eventlog’una (olay dosyasına) gönderir.

Örnek

Bu örnekte TextWriterTraceListener örneğinde olduğu gibi aynı işlemleri yapmaktadır. Fark, sonuçların bir dosyaya değil de sistemin Event Log’una yazdırmasıdır.

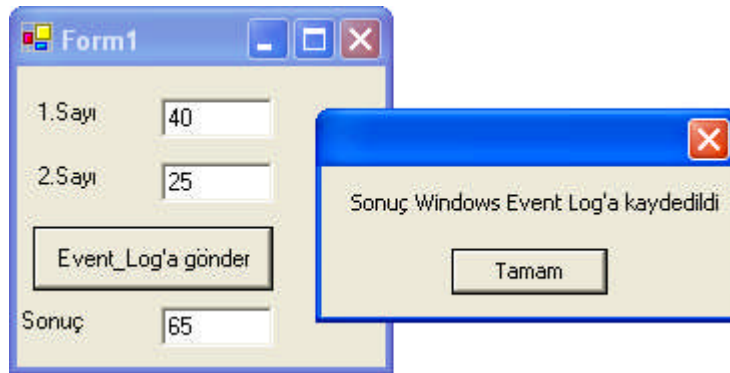
Yani, bir önceki örnekte tasarlamış olduğunuz formu bu örnek için de kullanabilirsiniz. Yalnız, butona tıklandığında yapılacak işlemlerimiz değiştiği için “buton1_Click” yordamını aşağıdaki gibi düzenleyiniz.

```
private void button1_Click(object sender, System.EventArgs e)
{
    int a,b,son;
    a=int.Parse(textBox1.Text);
    b=int.Parse(textBox2.Text);
    son=a+b;
    textBox3.Text=Convert.ToString(son);
    MessageBox.Show("Sonuç Windows Event Log'a kaydedildi");
    if( !EventLog.SourceExists("Test") )
    {
        EventLog.CreateEventSource("Test","dosyam");
    }
    EventLog olayim = new EventLog();
    olayim.Source = "Test";

    EventLogTraceListener olaydinleyici = new EventLogTraceListener(olayim);
    Trace.Listeners.Add(olaydinleyici);
    Trace.WriteLine(Convert.ToString(a),"Birinci sayı");
    Trace.WriteLine(Convert.ToString(b),"İkinci sayı ");
    Trace.WriteLine(Convert.ToString(son),"Sonuç      ");

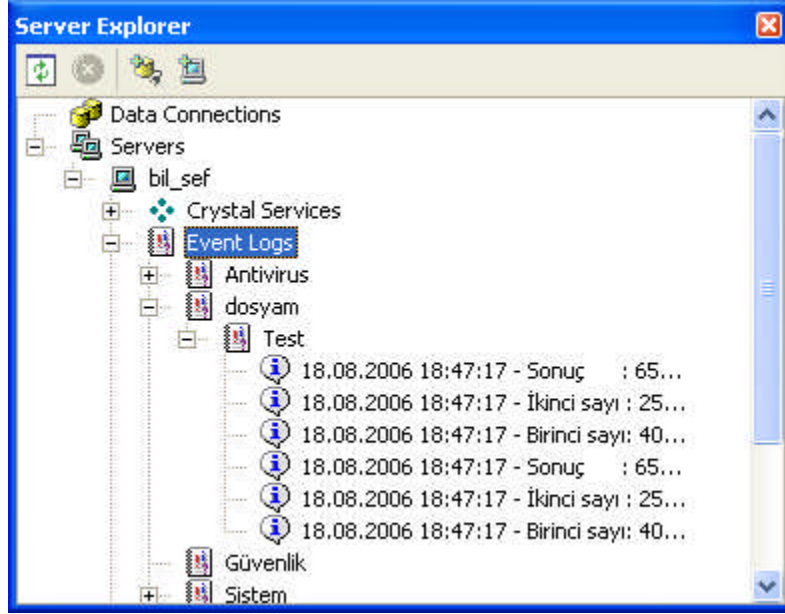
    Debug.WriteLine(Convert.ToString(a),"Birinci sayı");
    Debug.WriteLine(Convert.ToString(b),"İkinci sayı ");
    Debug.WriteLine(Convert.ToString(son),"Sonuç      ");
    Trace.Flush();
}
```

Sayı girişleri ilgili yerlere yapıldıktan sonra EventLog sınıfından “olayim” adlı bir nesne oluşturulmaktadır. EventLogTraceListener sınıfından da “olaydinleyici” adlı bir nesne oluşturularak Listeners sınıfına eklenir. Program çalıştırıldığında elde edilen sonuçlar Trace ve Debug sınıflarıyla EventLog’a yazdırılır.



Resim 2.9: Sonuçlar Event Log’a gönderilir.

Event Logs'a gönderilen verileri Server Explorer penceresinden görülür. Her işlemin ardından Event Logs üzerinde sağ tık yapıp Event Logs'u Refresh etmeniz gerekir.



Resim 2.10: Sonuçların Event Logs'a yazdırılması

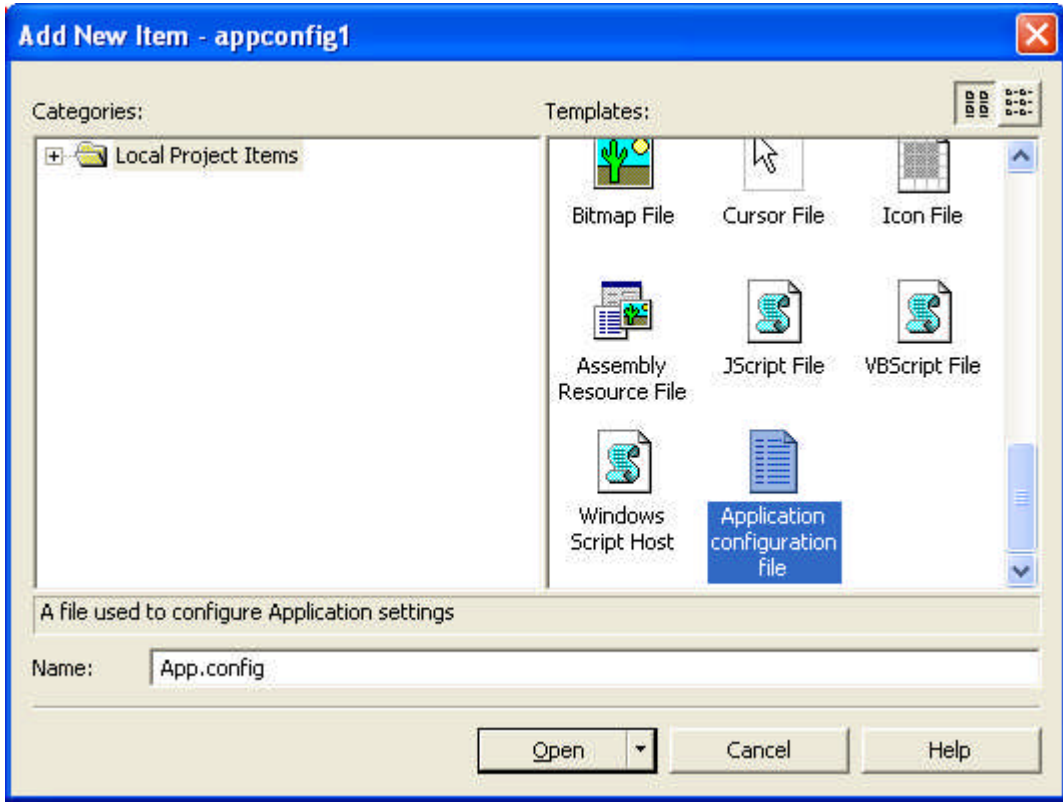
Dikkat edecek olursanız yine sonuçlar iki kez yazılmış durumdadır. Demek ki, program Debug modunda çalıştırılmıştır.

2.3. App.Config Dosyası

Projede kullanılan, örneğin kullanıcı adı ve şifresi, veri tabanının adı ve şifresi ya da bir veri tabanı dosyasına bağlanıyorsanız, veri tabanının konumu gibi bilgiler değiştirilmek istenebilir. Bu değişiklikler için **App.Config** dosyası kullanılır.

App.Config dosyası bir XML dosyasıdır. XML dosyasının içeriği oluşturulurken yazım kurallarına dikkat edilmelidir.

Projenize bir App.Config dosyası eklemek için **Project/Add New Item** seçeneği seçilip açılan Add New Item iletişim penceresinin **Templates** bölümünden **Application Configuration File** seçilip **Open** düğmesine basılmalıdır.



Resim 2.11: Add New Item iletişim penceresi

Projeye eklenen App.Config dosyasının temel yapısı **<Configuration>** etiketiyle başlar. **</Configuration>** etiketiyle biter.

Örnek

Bu örnekte, oluşturulmuş bir App.Config dosyasındaki verilerin formdaki ilgili yerlere getirilmesi yapılacaktır. Verileri App.config dosyasından alabilmek için önce Solution Explorer penceresinde sağ tık yaparak Add seçeneği üzerine gelince açılan menüden Add New Item seçimini yapınız ve projenize bir “Application configuration file” ekleyiniz. Eklediğiniz App.config dosyasının içeriğini aşağıdaki gibi düzenleyiniz.

```
<configuration>
  <appSettings>
    <add key="ad" value="Ahmet"/>
    <add key="soyad" value="ELDEM"/>
  </appSettings>
</configuration>
```

Bu konfigürasyon dosyasına göre kullanacağımız alan anahtarları “ad” ve “soyad”, değerleri ise value ile verilen değerler olacaktır. Şimdi formu resim 2.12’deki gibi tasarlayınız.

A screenshot of a Windows application window titled 'Form1'. The window has a standard Windows XP-style title bar with minimize, maximize, and close buttons. The main area is light beige. It contains two text input fields. The first is labeled 'Adı' and is empty. The second is labeled 'Soyadı' and is also empty. Below these two fields is a rectangular button with the text 'OKU' centered on it.

Resim 2.12: Formun tasarlanması

Form tasarlandıktan sonra “OKU” düğmesinin üzerine çift tıklayıp Click() metoduna aşağıdaki kodları yazınız.

```
private void button1_Click(object sender, System.EventArgs e)
{
    AppSettingsReader oku=new AppSettingsReader();
    textBox1.Text=oku.GetValue("ad",typeof(String)).ToString();
    textBox2.Text=oku.GetValue("soyad",typeof(String)).ToString();
}
```

Burada AppSettingsReader sınıfı ile “oku” adında bir nesne üretilmiş ve AppSettingsReader sınıfının tek metodu olan “GetValue” özelliği kullanılmıştır. Böylece App.config dosyasındaki string veriler alınıp metin kutularına aktarılmış olacaktır.

A screenshot of the same 'Form1' window. The text input fields are now populated. The 'Adı' field contains the text 'Ahmet' and the 'Soyadı' field contains the text 'ELDEM'. The 'OKU' button is still present below the fields.

Resim 2.13: Verilerin App.config dosyasından alınıp metin kutularına aktarılması

2.4. Trace ve Debug Sınıfıyla Kullanılan Komutlar

2.4.1. Write

Bir şarta bağlı olmadan verilen metni Listener koleksiyonuna yazar.

2.4.2. WriteLine

Bir şarta bağlı olmadan verilen metni Listener koleksiyonuna yazar ve bir sonraki satıra geçer.

2.4.3. WriteIf

Mantıksal şart doğru olduğunda yazılması verilen metni Listener koleksiyonuna yazar.

2.4.4. WriteLineIf

Mantıksal şart doğru olduğunda verilen metni Listener koleksiyonuna yazar ve bir alt satıra geçer.

2.4.5. Assert

Koşulun yanlış olması durumunda Listener koleksiyonuna belirtilen mesajı yazdırır.

Örnek

Metin kutularından girilen iki sayı Trace ve Debug sınıfları kullanılarak çeşitli mantıksal işlemlere tabi tutulacaktır. Bu işlem için formunuza iki metin kutusu ve bir buton ekleyiniz. Butonun Click() metoduna aşağıdaki kodları yazınız. Trace ve Debug sınıflarının aynı komutları kullandığını hatırlayınız.

```
private void button1_Click(object sender, System.EventArgs e)
{
    int X,Y;
    X=int.Parse(textBox1.Text);
    Y=int.Parse(textBox2.Text);

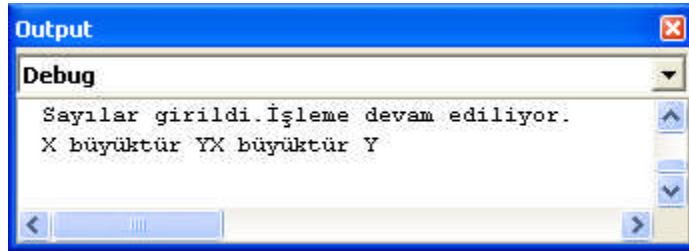
    Trace.Write("Sayılar girildi.");
    Trace.WriteLine("İşleme devam ediliyor.");
    Debug.WriteIf(X>Y, "X büyüktür Y");
    Debug.WriteLineIf(X>Y, "X büyüktür Y");
    Trace.Assert(X>Y, "X Y den küçük","X Her zaman büyük olmalı");
}
```

Projenizi çalıştırıp iki sayı girişı yapınız. Örnekte birinci sayı için 5 rakamı, ikinci sayı için 3 rakamı girilmiştir.



Resim 2.14: Verilerin girilmesi

İşlemin sonuçlarını Output penceresinde göreceğiniz için Output pencereniz açık değilse açınız. Programda ilk önce Trace.Write ve Trace.WriteLine ile verilen mesajlar yazdırılmaktadır. Trace.Write ile imleç bir alt satıra geçmediğinden Trace.WriteLine ile verilen mesaj yazdırıldıktan sonra imleç bir alt satıra geçmiştir. Daha sonra Trace.WriteIf ve Trace.WriteLineIf ile girilen iki sayı mantıksal olarak karşılaştırılmış ve sonuç doğru olduğu için bu iki komutla verilen mesajlar yazdırılmıştır. İmleç, Trace.WriteLineIf ile bir alt satıra geçmiştir. Trace.Assert ile verilen mesaj, şart doğru olduğundan dolayı Output penceresine yazdırılmamıştır.

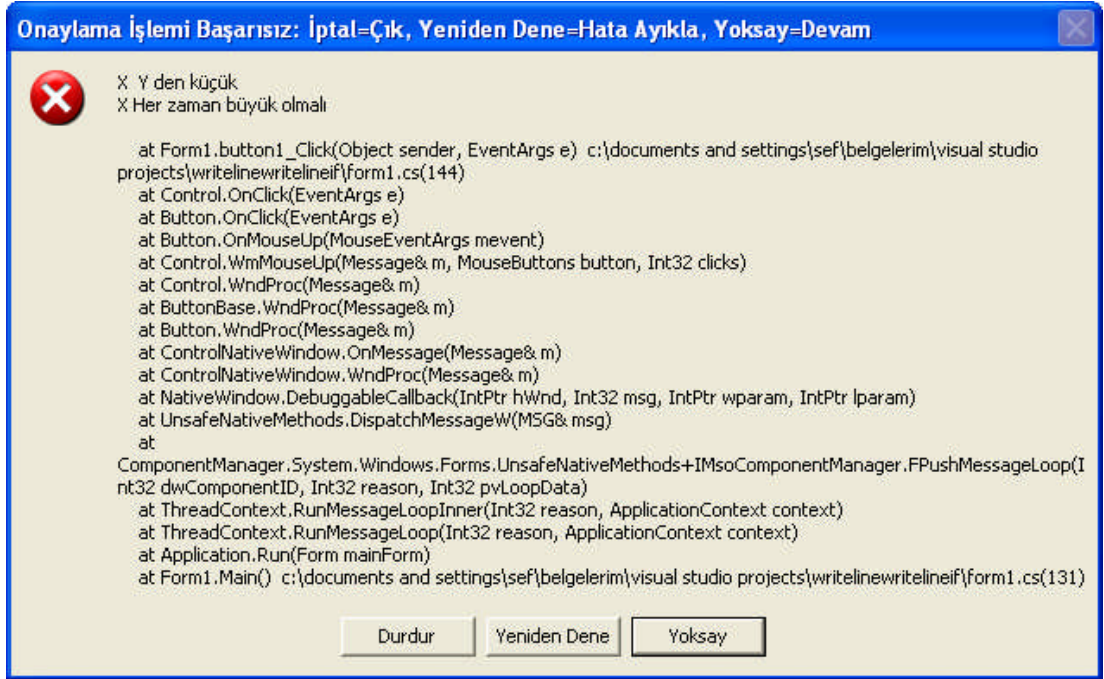


Resim 2.15: Programın çalışması esnasında Output penceresinin görünümü

Programı yeniden çalıştırıp ilk değeri küçük, ikinci değeri büyük girdiğinizde Trace.Assert ile verilen mesaj ekranda gösterilecektir. Şart yanlış olduğundan Trace.Assert komutu devreye girmiştir.



Resim 2.16: Yeniden değer girişi



Resim 2.17: Assert komutuyla verilen mesajın yazdırılması

2.4.6. Fail

Olumsuz bir durum oluştuğunda otomatik olarak bir bildiri oluşturur. Bildiri mesajı listener sınıfına yazdırılır ve bir mesaj kutusuyla ekrana gelir.

Örnek

Bu örnekte de yine iki sayı girişi yapılmakta ve if ile verilen şart kontrol edilmektedir.

Formunuza üç adet metin kutusu ve bir buton ekleyiniz. Butonun Click() metoduna aşağıdaki kod satırlarını yazınız.

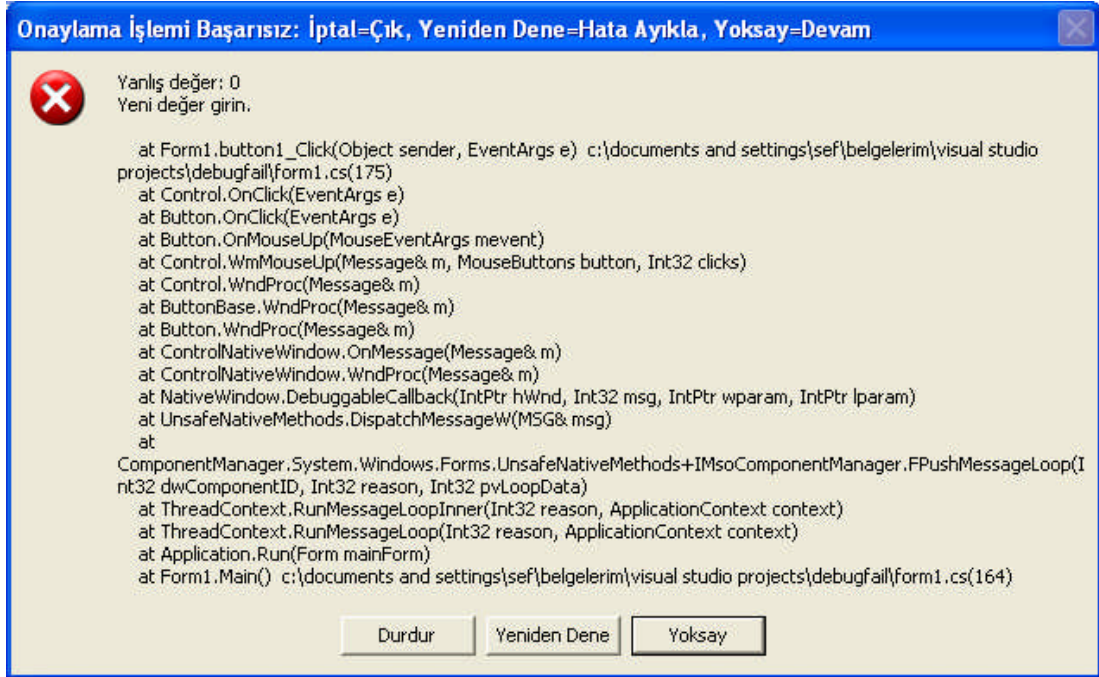
```
private void button1_Click(object sender, System.EventArgs e)
{
    int a,b;
    a=int.Parse(textBox1.Text);
    b=int.Parse(textBox2.Text);

    if (a<=0)
    {
        Debug.Fail("Yanlış değer: " + a.ToString(),"Yeni değer girin.");
        textBox1.Focus();
    }
    else
    {
        textBox3.Text=Convert.ToString(a/b);
    }
}
```

Şartta, birinci metin kutusuna girilen sayının sıfır ve sıfırdan küçük olması durumunda Fail komutu ile verilen mesaj ekrana getirilecektir. Aksi takdirde birinci sayı ikinci sayıya bölünecek ve sonuç üçüncü metin kutusuna yazdırılacaktır.

Resim 2.18: Verilerin girilmesi

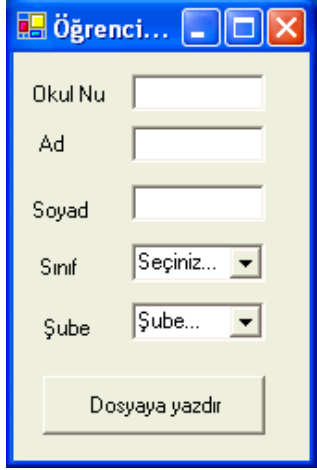
Örnek olarak birinci sayıya sıfır (0) değerini giriniz. Gönder düğmesine bastığınızda Fail komutu ile verilen mesaj devreye girecektir.

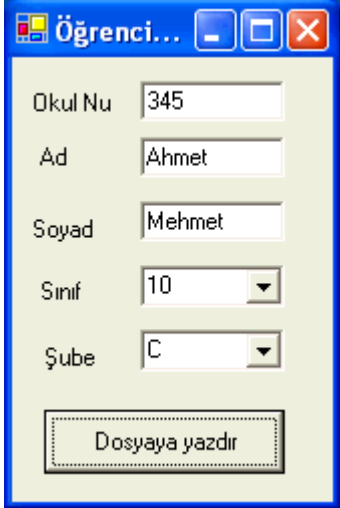
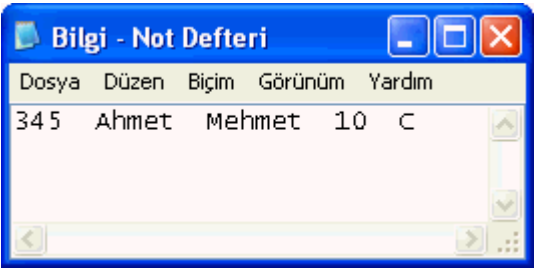


Resim 2.19: Fail mesajının ekranda gösterilmesi

UYGULAMA FAALİYETİ

Bu uygulama, elde edilen bilgilerin dosyaya nasıl yazdırılacağını gösteren bir uygulamadır. Elde edilen veriler bu şekilde dosyaya yazdırılabildiği gibi programda oluşan hataların da dosyaya yazdırılması aynı şekilde olacaktır.

İşlem Basamakları	Öneriler
<ul style="list-style-type: none"> ➤ Yeni bir Windows Application oluşturarak formu resim 2.20'deki gibi tasarlayınız. 	 <p>Resim 2.20: Formun tasarlanması</p>
<ul style="list-style-type: none"> ➤ Formdaki comboBox'ların elemanlarını Properties Window'u kullanarak Item özelliğinden giriniz. 	<ul style="list-style-type: none"> ➤ Combobox1.Items.Add("A"); şeklinde kod satırlarıyla da eleman ekleyebilirsiniz.
<ul style="list-style-type: none"> ➤ Girilen bilgiler butona tıklandığında dosyaya yazdırılacağı için program kodlarını Click() metoduna yazınız. 	<ul style="list-style-type: none"> ➤ "Dosyaya yazdır" butonuna çift tıklayınız.
<ul style="list-style-type: none"> ➤ TextBox ve comboBox'lardan girilen bilgileri aktaracağınız değişkenleri string tipte tanımlayınız. 	<ul style="list-style-type: none"> ➤ string okulnu,ad,soyad,sinif,sube;
<ul style="list-style-type: none"> ➤ Öğrencinin okul numarasını, adını ve soyadını textBox'tan, sınıfını ve şubesini comboBox'lardan ilgili değişkenlere aktarınız. 	<ul style="list-style-type: none"> ➤ sube=comboBox2.SelectedItem.ToString();
<ul style="list-style-type: none"> ➤ FileStream sınıfından "dosya" adında bir dosya değişkeni oluşturarak dosya adını "Bilgi.Log" olarak veriniz. 	<ul style="list-style-type: none"> ➤ FileStream sınıfı kullanılacağından IO namespace'ini projenize ekleyiniz.
<ul style="list-style-type: none"> ➤ Oluşturulan dosya değişkenini işaret ederek StreamWriterTraceListener sınıfından yeni bir dinleyici oluşturunuz. 	<ul style="list-style-type: none"> ➤ Bu sınıfı kullanabilmek için de Diagnostics namespace'ini projenize ekleyiniz.

<p>➤ Oluşturduğunuz dinleyiciyi Listeners sınıfına ekleyiniz.</p>	<p>➤ <code>Trace.Listeners.Add(dinleyici);</code></p>
<p>➤ Girilen bilgileri Trace sınıfını kullanarak dosyaya yazdırınız.</p>  <p>Resim 2.21: Bilgilerin girilmesi</p>	<p>➤ Girilen bilgiler bir bütün olduğundan dosyada yan yana yazdırabilirsiniz.</p>
<p>➤ Belleği boşaltarak dosyayı kapatınız.</p> <p>➤ Projenizin çalıştığı aktif klasörü açarak bilgilerin dosyaya yazılıp yazılmadığını kontrol ediniz.</p>	<p>➤ <code>Flush()</code> ve <code>Close()</code> fonksiyonlarını kullanınız.</p> <p>➤ “Bilgi.Log” dosyasını açınız.</p>  <p>Resim 2.22: Bilgilerin dosyaya yazılmış hali</p>

UYGULAMA FAALİYETİ

Aşağıda uygulamaları yapınız. Sonuçları öğretmeninize sununuz. Eksikliklerinizi öğretmeninizin rehberliğinde tamamlayınız.

- InputBox'tan girilen int tipte 10 adet sayının ortalamasını hesaplatarak Output penceresinde görüntüleyen programı yapınız.
- İki ayrı metin kutusundan girilen sayıların mod'u hesaplatılacaktır. Mod işleminin sonucunda kalan sıfır (0) ise sayıları ve kalanı Log dosyaya yazdıran programı yapınız.
- Aynı örneğin kalanı sıfır (0) olmayanları Windows Event Log'a yazdıran programı yapınız.

ÖLÇME VE DEĞERLENDİRME

Aşağıdaki soruları dikkatlice okuyarak doğru/yanlış seçenekli sorularda uygun harfleri yuvarlak içine alınız. Seçenekli sorularda ise uygun şıkkı işaretleyiniz. Boşlukları uygun şekilde doldurunuz.

1. Trace ve Debug sınıfını kullanabilmek için aşağıdaki namespace'lerden hangisi projeye eklenmelidir?
A) IO
B) Diagnostics
C) Windows.Forms
D) Data
2. Aşağıdakilerden hangisi bir TraceListener sınıfı değildir?
A) EventLog
B) Default
C) Config
D) TextWriter
3. Projenizle ilgili temel bilgiler değiştirilmek istendiğinde dosyası kullanılır.
4. Trace ve Debug sınıfları ayrı TraceListener koleksiyonları kullanır. (D/Y)
5. Bir proje Release modda çalıştırıldığında Debug sınıfıyla üretilen sonuçlar görüntülenmez. (D/Y)

DEĞERLENDİRME

Cevaplarınızı cevap anahtarı ile karşılaştırınız. Doğru cevap sayınızı belirleyerek kendinizi değerlendiriniz. Yanlış cevap verdiğiniz ya da cevap verirken tereddüt yaşadığınız sorularla ilgili konulara geri dönerek tekrar inceleyiniz. Tüm sorulara doğru cevap verdiyseniz diğer öğrenme faaliyetine geçiniz.

ÖĞRENME FAALİYETİ-3

AMAÇ

Hataların oluşumunu kontrol altına alarak oluşabilecek hatalara karşı tedbirler alabileceksiniz.

ARAŞTIRMA

- Programcılar isteğe göre program yazdıklarında, programa özel hataları önceden tespit edip ihtimalleri değerlendirerek ve uyarıcı hata mesajları yazarak kullanıcıları uyarabilirler mi? Araştırınız.

3. İSTİSNAİ OLAYLAR

Bir programın hatasız yazılmış olması o programın hiç hata vermeyeceği anlamına gelmez. Kullanıcıdan ve diğer durumlardan kaynaklanan sebeplerden dolayı doğru yazılmış kodlar da hata verebilir. Bu durumda, hataların oluşumunu her zaman kontrol altında tutamayabilirsiniz. Ancak, oluşabilecek hatalara karşı tedbirler alabilirsiniz. Yapacağınız bu işleme **İstisna Yönetimi** denir.

3.1. System.Exception Sınıfı

Nesne tabanlı programlamada istisnai olaylar sınıflarla simgelenir. Tüm istisnai olay sınıfları **System** namespace'inin **Exception** sınıfından türetilmelidir. Böylece tüm olaylar Exception sınıfının birer alt sınıfıdır.

SystemException ve **ApplicationException**, Exception sınıfından türetilir.

Nesne tabanlı programlamanın çalışma zamanı sistemi (CLR) tarafından üretilen sınıfları **SystemException** ve uygulama programları tarafından üretilen sınıfları da **ApplicationException** sınıfından desteklenen sınıflardır.

SystemException sınıfından türetilen standart istisnai olaylardan bazıları derleyicide tanımlanmıştır. Örneğin, sıfıra bölme hatası için DivideByZeroException sınıfı derleyicinin içerisinde varsayılan olarak vardır.

3.2. Try-Catch Kullanımı

En sık kullanılan hata ayıklama kod bloğudur. Oluşabilecek hatalar **try** bloğunda, hata durumunda işletilecek kodlar ise **catch** bloğunda yazılır. Hata olmaması durumunda catch bloğundaki kodlar çalıştırılmaz. Hata olması durumunda ise, hata oluşan kod satırından catch bloğuna kadar olan kod satırları işletilmez. Catch bloğundaki kod satırları işletilir.

Try-Catch bloğu kullanım şekli şöyledir;

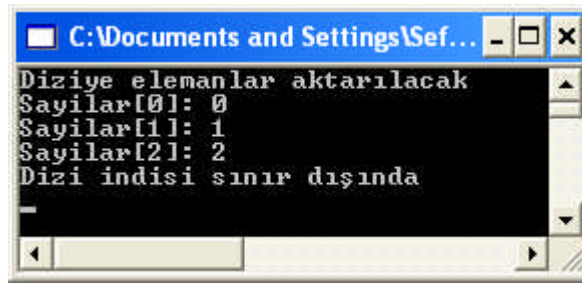
```
Try
{
// Hata oluşturabilecek olan kodlar
}
catch
{
// Hata oluşursa işleyen kodlar
}
```

Örnek

3 elemanlı bir int diziye eleman girişi yapılmaktadır. Eleman sayısından daha fazla eleman girişi yapılmak istendiğinde hata oluşmaktadır. Oluşan hata sonucunda da uyarı mesajı ekrana gelmektedir. Bir console uygulaması başlatarak aşağıdaki kodları yazınız.

```
using System;
namespace trycatch1
{
    //3 elemanlı bir diziye 10 eleman aktarılmak istenmektedir.
    class Class1
    {
        static void Main()
        {
            int[] sayilar=new int[3];
            try
            {
                Console.WriteLine("Diziye elemanlar aktarılacak");
                for(int i=0;i<10;i++)
                {
                    sayilar[i]=i;
                    Console.WriteLine("Sayilar[{0}]: {1}",i,sayilar[i]);
                }
                Console.WriteLine("Hata yakalanınca bu satır çalıştırılmaz");
            }
            catch(IndexOutOfRangeException)
            {
                Console.WriteLine("Dizi indisi sınır dışında");
            }
            Console.ReadLine();
        }
    }
}
```


Başlangıçta tipi `int` olan “sayılar” adında 3 elemanlı bir dizi tanımlanmaktadır. Elemanların girişi `try` bloğunda, hata mesajı da `catch` bloğunda yazılmıştır. Dizi 3 elemanlı olarak tanımlanmasına rağmen `for` döngüsüyle diziye 10 eleman aktarılacak istenmektedir. Dizi indisi sınırına kadar elemanlar diziye girişi yapıldığı için konsolda gösterilecektir. Ancak, dizi indisi sınırı aşıldığı anda `catch` bloğu devreye girecektir. `Catch` bloğu `IndexOutOfRangeException` sınıfıyla kullanılmıştır. Böylece programın konsol görüntüsü resim 3.1’deki gibi olur.

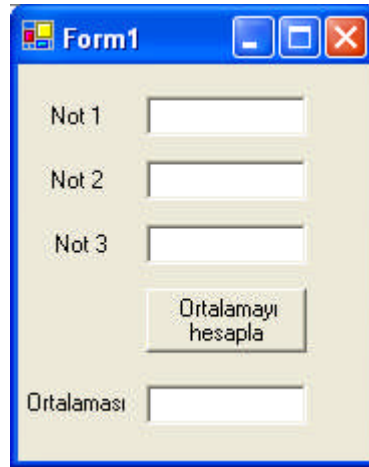


Resim 3.1: Uygulamanın konsol görüntüsü

Oluşabilecek hatalarda sistemin içinde yer alan hata mesajlarını `catch` bloğunda göstermek için **catch** ile birlikte **Exception** parametresi kullanılır.

Örnek

Bir öğrencinin üç notu text kutularından girilerek ortalaması hesaplanıp yine bir text kutusuna yazdırılmak istenmektedir. Ancak kullanıcının not girişlerinde rakam yerine metin girmesi hataya sebep olmaktadır. Bu örnek için önce formunuzu resim 3.2’deki gibi tasarlayınız.



Resim 3.2: Formun tasarlanması

Notlar girildikten sonra hesaplanması için “Ortalamayı hesapla” düğmesine tıklanacaktır. Bunun için, tasarım anında düğme üzerine çift tıklayınız ve Click() metoduna aşağıdaki kodları yazınız.

```
private void button1_Click(object sender, System.EventArgs e)
{
    int nt1,nt2,nt3;
    double ort;
    try
    {
        nt1=int.Parse(textBox1.Text);
        nt2=int.Parse(textBox2.Text);
        nt3=int.Parse(textBox3.Text);
        ort=(nt1+nt2+nt3)/3;
        textBox4.Text=ort.ToString();
    }
    catch(Exception hata)
    {
        MessageBox.Show("Hata meydana geldi\n"+hata.Message,"Dikkat");
    }
}
```

Not girişleri try bloğunda yapılmaktadır. Rakam yerine harf olarak not girildiğinde catch bloğu devreye girecektir. Catch bloğunda Exception sınıfının ürettiği hata mesajı “hata” isimli bir değişkende tutulmaktadır. Oluşan hata MessageBox ile kullanıcıya gösterilecektir.

Veri girişinde notlar düzgün olarak girildiğinde hata meydana gelmez (Resim 3.3).

Resim 3.3: Hata oluşmaz

Ancak rakam yerine yazıyla not girişi yapılırsa hata oluşacaktır. Mesaj kutusundaki ikinci satır sistemin hata mesajıdır (Resim 3.4).



Resim 3.4: Hata oluşur

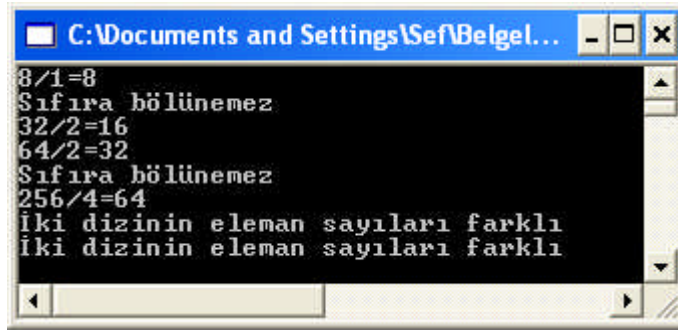
Oluşabilecek birden fazla hata için farklı tipte catch bloğu kullanılabilir. Eğer hata herhangi bir catch bloğunda yakalanırsa diğer catch blokları işletilmez.

Örnek

Aynı tipte iki dizi tanımlı elemanlarının birbirine bölünmesi uygulaması yapılacaktır. Oluşabilecek hatalar göz önüne alınmalıdır. Program kodları aşağıdadır.

```
using System;
namespace trycatch3
{
    class Class1
    {
        static void Main()
        {
            int[] dizi1={8,16,32,64,128,256,512,1024};
            int[] dizi2={1,0,2,2,0,4};
            for(int i=0;i<dizi1.Length;i++)
            {
                try
                {
                    Console.WriteLine(dizi1[i] + "/" + dizi2[i] + "=" + dizi1[i] / dizi2[i]);
                }
                catch(DivideByZeroException)
                {
                    Console.WriteLine("Sıfıra bölünemez");
                }
                catch(IndexOutOfRangeException)
                {
                    Console.WriteLine("İki dizinin eleman sayıları farklı");
                }
            }
            Console.ReadLine();
        }
    }
}
```

Aynı tipte iki farklı dizi tanımlanarak ilk elemanları verilmiştir. Birinci dizinin eleman sayısı 8, ikinci dizinin eleman sayısı 6'dır. Birinci dizi elemanlarının ikinci dizinin elemanlarına bölmek suretiyle sonuçlar konsolda yazdırılacaktır. "for" döngüsü birinci dizinin eleman sayısı kadar çalışacağı için ikinci dizinin eleman sayısını geçince program hata verecektir. Daha öncesinde ikinci dizinin elemanlarında sıfır (0) olduğu için sıfıra bölme hatası meydana gelir. İki tip hata olduğu için iki adet catch bloğu kullanılmıştır. Birinci catch bloğu sıfıra bölme hatasıyla ilgili mesajı göstermekte, ikinci catch bloğu da dizi indis sınırının aşılmasıyla ilgili mesajı göstermektedir. Bunun sonucunda konsol görüntüsü Resim 3.5'teki olur.



Resim 3.5: Uygulamanın konsol görüntüsü

3.3. Finally Kullanımı

Hata oluşması durumunda işletilecek kod satırları yoksa programın kırılmasını engellemek için **finally** bloğu kullanılmalıdır. Finally bloğu hata olsada olmasa da çalışan bir bloktur.

Finally bloğunun kullanım şekli şöyledir;

```
Try
{
// Hata oluşturabilecek olan kodlar
}
finally
{
// Her koşulda çalışacak kodlar
}
```

Örnek

Bir işçinin bir günde çalıştığı toplam saat ve bir saatlik ücreti text kutularından girilerek alacağı parayı hesaplatan program için form tasarımı resim 3.6'daki gibidir.

Resim 3.6:Formun tasarımı

Hesapla düğmesine tıklanınca işçi maaşı hesaplatılacağı için düğmenin Click() metodu aşağıdaki gibidir.

```
private void button1_Click(object sender, System.EventArgs e)
{
    int toplam_saat, saat_ucreti;
    try
    {
        toplam_saat = int.Parse(textBox1.Text);
        saat_ucreti = int.Parse(textBox2.Text);
        textBox3.Text = Convert.ToString(toplam_saat * saat_ucreti);
    }
    finally
    {
        textBox1.Text = "";
        textBox2.Text = "";
        textBox1.Focus();
    }
}
```

Programda işlem yapan satırlar try bloğunda yazılmıştır. “finally” bloğunda yazılan kod satırları her halde çalıştırılacağı için text kutuları boşaltılarak ve birinci text kutusu aktif hale getirilerek yeni veri girişi sağlanmaktadır.

Programın çalıştırılıp veri girişinin yapılması resim 3.7’deki gibi olur.

Resim 3.7: Veri girişinin yapılması

Veri girişi yapıp “Hesapla” düğmesine tıklandığında işçinin maaşı hesaplanıp üçüncü text kutusuna yazdırılır ve finally bloğu işletilir.

Resim 3.8: Hesaplama yapıp finally bloğu işletilir.

Örnek

Bu örnekte sıfıra bölme hatası, dizi indis sınırlarına taşılmasıyla ilgili iki adet catch bloğu ve işlemin bittiğini belirten bir finally bloğu kullanılmıştır. Uygulamada iki adet class bulunmaktadır. “Class1” adlı sınıfta for döngüsüyle ikinci class’a yönlendirme yapılmaktadır. Bu işlem üç defa gerçekleşecektir.

“finallykullan” adlı sınıfta ana sınıftan gelen veri “deger” adlı değişken tarafından karşılanmakta ve switch akış kontrol deyimiyle şartlar kontrol edilmektedir.

```

using System;
class finallykullan
{
    public static void yürüt(int deger)
    {
        int z;
        int[] sayilar=new int[2];
        Console.WriteLine("Alınıyor " + deger);
        try
        {
            switch(deger)
            {
                case 0:
                    z=10/deger;break;
                case 1:
                    sayilar[4]=4;break;
                case 2:
                    return;
            }
        }
        catch(DivideByZeroException)
        {
            Console.WriteLine("Sıfıra bölme hatası");
            return;
        }
        catch(IndexOutOfRangeException)
        {
            Console.WriteLine("Eşleşen eleman bulunamadı");
        }
        finally
        {
            Console.WriteLine("İşlem bitti");
        }
    }
}

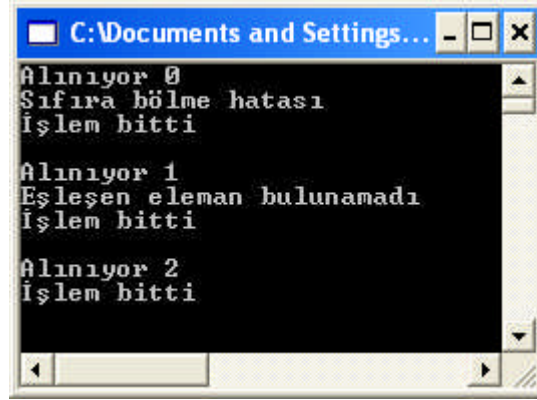
```

```

class Class1
{
    public static void Main()
    {
        for(int i=0;i<3;i++)
        {
            finallykullan.yürüt(i);
            Console.WriteLine();
        }
        Console.ReadLine();
    }
}

```

“Case 0” şartında sayı sıfıra bölünmek istendiği için DivideByZeroException sınıfından bir hata oluşur. Case 1 şartındaysa dizi 3 elemanlı olmasına rağmen dördüncü eleman olarak sayı girişi yapılmak istenmekte ve IndexOutOfRangeException sınıfından bir hata oluşur. İlgili hata mesajları konsola yazdırılır. Son bloğumuz olan finally bloğu ise her halde çalışacaktır. Uygulama çalıştırıldığında resim 3.9’daki sonuçlar elde edilir.



Resim 3.9: Uygulamanın konsol görüntüsü

3.4. Throw Rethrow Kullanımı

3.4.1. Throw Kullanımı

Throw hatayı fırlatma anlamına gelir. Bazı durumlarda hatalar bilerek oluşturulmak istenebilir. Bunun için, “throw” komutu kullanılır. “throw” ifadesinin kullanıldığı noktada program durarak istenilen istisnayı üretir.

Şimdi throw’un kullanımına basit bir örnekle bakalım.

Örnek

Bu örnekte program akışının işleyişi gösterilmektedir.

Try bloğunda önce fırlatmadan önceki mesaj yazdırılır. Sıfıra bölme hatası olan DivideByZeroException sınıfı throw ile fırlatılarak program akışı catch bloğuna yönlendirilmiştir. Try ve catch bloklarından sonra da bir mesaj yazdırılarak işlemin bittiği belirtilmiştir.

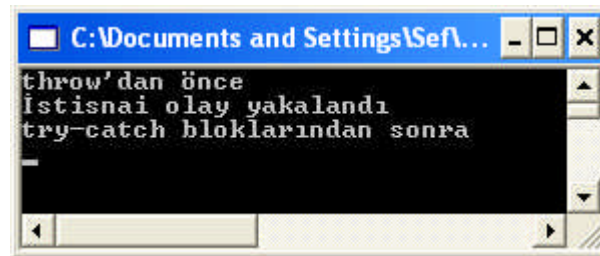

```

using System;
class Class1
{
    static void Main(string[] args)
    {
        try
        {
            Console.WriteLine("throw'dan önce");
            throw new DivideByZeroException();
        }
        catch(DivideByZeroException)
        {
            Console.WriteLine("İstisnai olay yakalandı");
        }
        Console.WriteLine("try-catch bloklarından sonra");
        Console.Read();
    }
}

```

“throw”un kullanımı basit olarak bu şekilde gerçekleşmektedir.

Uygulama çalıştırılırsa sonuç resim 3.10’daki gibidir.



Resim 3.10: Basit bir throw örneği

Örnek

Konsoldan girilen sayının bölünüp bölünemediğini test eden program kodu aşağıdaki gibidir. Temel sınıftan girilen sayı yaz() sınıfına gönderilmektedir. Sayıyı göndermeden önce sayının int tipinde olmaması durumunda da temel sınıfta Exception sınıfı kullanılarak oluşabilecek hata bir mesajla gösterilmektedir. Girilen sayı sıfır ise yaz() metodunda throw ile bir fırlatma gerçekleştirilerek işlemin sonucunda sıfıra bölme hatasının olduğu belirtilmektedir.

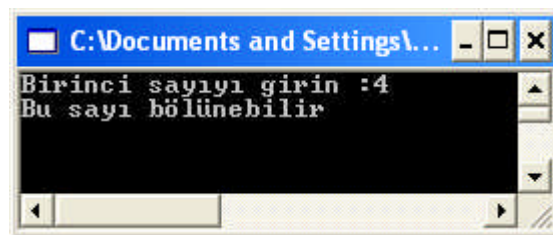
Program kodları aşağıdaki gibidir.

```

using System;
class Class1
{
    public static void Main()
    {
        int s1;
        try
        {
            Console.Write("Birinci sayıyı girin :");
            s1=int.Parse(Console.ReadLine());
            yaz(s1);
        }
        catch(Exception hata)
        {
            Console.WriteLine(hata.ToString());
        }
        Console.Read();
    }
    private static void yaz(int sayi)
    {
        if(sayi==0)
        {
            throw new DivideByZeroException(" Sıfıra bölme hatası");
        }
        else
        {
            Console.WriteLine("Bu sayı bölünebilir");
        }
    }
}

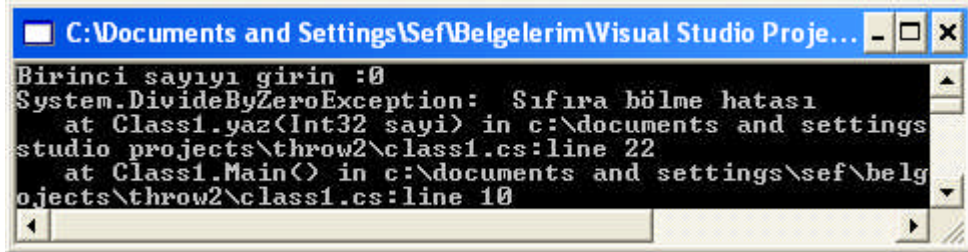
```

Program çalıştırılıp sıfırdan farklı bir sayı girildiğinde hata oluşmayacak ve ekran görüntüsü resim 3.11'deki gibi olacaktır.



Resim 3.11: Sıfırdan farklı sayı girilmesinde hata oluşmaz.

Ancak, sayı girişinde sıfır(0) girilirse hata oluşacaktır.



Resim 3.12: Sıfır girilirse hata oluşur.

Örnek

Tasarlanan formdaki tüm text kutularının zorunlu olarak doldurulması gerektiğiyle ilgili bir uygulamanız olsun. Tamam düğmesine tıklanınca tüm text kutularının doldurulup doldurulmadığı test edilsin. İlgili düğmenin Click() metodu aşağıdaki gibidir.

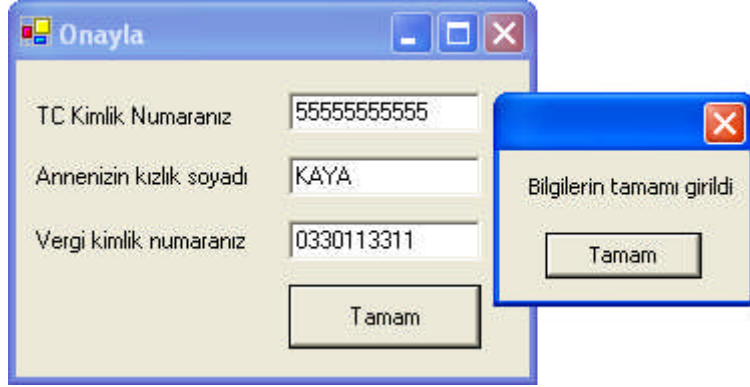
```
private void button1_Click(object sender, System.EventArgs e)
{
    string tckimliknu, annesoyad, verginu;
    tckimliknu = textBox1.Text;
    annesoyad = textBox2.Text;
    verginu = textBox3.Text;

    if ((tckimliknu == "") || (annesoyad == "") || (verginu == ""))
    {
        try
        {
            throw new Exception();
        }
        catch (Exception)
        {
            MessageBox.Show("Bilgilerin tamamını girmelisiniz", "Dikkat");
        }
    }
    else
    {
        MessageBox.Show("Bilgilerin tamamı girildi");
    }
}
```

Eğer, “tckimliknu”, “annesoyad” veya “verginu” alanlarından biri girilmezse throw ile hata fırlatılıp mesaj kutusunda da bu hata gösterilecektir. Bilgilerin tamamı eksiksiz girildiğindeyse else yapısındaki mesaj kullanıcıya gösterilecektir.



Resim 3.13: Bilgilerin eksik girilmesi durumu



Resim 3.14: Bilgilerin tamamının girilmesi durumu

3.4.2. Rethrow Kullanımı

Rethrow hatayı yeniden fırlatma anlamına gelir. Rethrow kullanımı birden fazla catch bloğu kullanıldığı zaman uygulanır. Bir catch bloğunda yakalanan istisnai olay diğer catch bloğunda da yakalanabilsin diye yeniden fırlatılır.

Bir istisnai olayı yeniden fırlatmak için açıkça bir istisnai durum belirtmeden yalnızca **throw;** yazmak yeterli olacaktır.

Örnek

Aynı tipte ancak, eleman sayıları farklı olan iki dizi elemanlarının birbirlerine bölünmesi durumunda oluşabilecek hataları gösteren program kodu aşağıdaki gibidir.

```

using System;
class rethrow_kullan
{
    public static void yürüt()
    {
        int[] dizi1={8,16,32,64,128,256,512,1024};
        int[] dizi2={1,0,2,2,0,4};
        for(int i=0;i<dizi1.Length;i++)
        {
            try
            {
                Console.WriteLine(dizi1[i]+ "/" +dizi2[i]+
                                   "=" +dizi1[i]/dizi2[i]);
            }
            catch(DivideByZeroException)
            {
                Console.WriteLine("Sıfıra bölünemez");
            }
            catch(IndexOutOfRangeException)
            {
                Console.WriteLine("Eşleşen eleman bulunamadı");
                throw;
            }
        }
    }
}

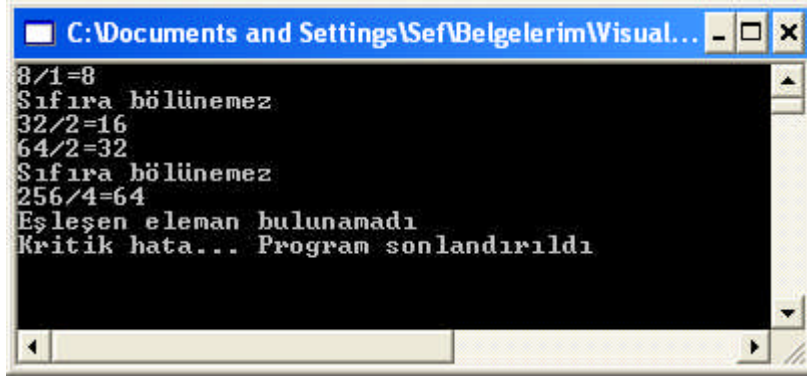
class Class1
{
    static void Main(string[] args)
    {
        try
        {
            rethrow_kullan.yürüt();
        }
        catch(IndexOutOfRangeException)
        {
            Console.WriteLine("Kritik hata..."+" Program sonlandırıldı");
        }
        Console.ReadLine();
    }
}

```

Program birinci dizi elemanlarını ikinci dizi elemanlarına bölmektedir. İşleyiş, temel sınıftaki try bloğundan rethrow_kullan sınıfının yürüt() metoduna yönlendirilmektedir. İkinci dizinin tanımlı elemanlarında sıfır (0) olduğu için sıfıra bölme hatası meydana gelmektedir. Ancak ikinci dizi elemanları birinci dizi elemanlarından az olduğu için birinci dizideki elemanı bölecek sayı olmadığından dizi indisi sınırı aşımı hatası meydana gelir. Birinci dizinin bölünecek başka bir elemanı daha olmasından dolayı tekrar dizi sınırı aşımı

gerçekleşmesin diye yeniden fırlatma gerçekleştirilir. Böylece program akışı temel sınıftaki catch bloğuna yönlendirilerek hata mesajıyla kullanıcı uyarılır.

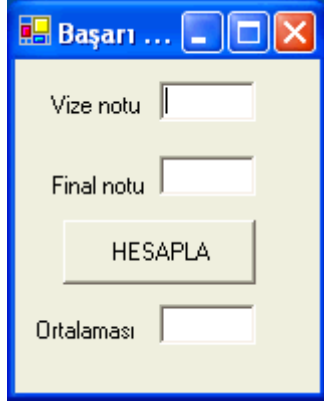
Uygulamanın çalıştırılması sonucunda ekran görüntüsü resim 3.15'teki gibi olmaktadır.

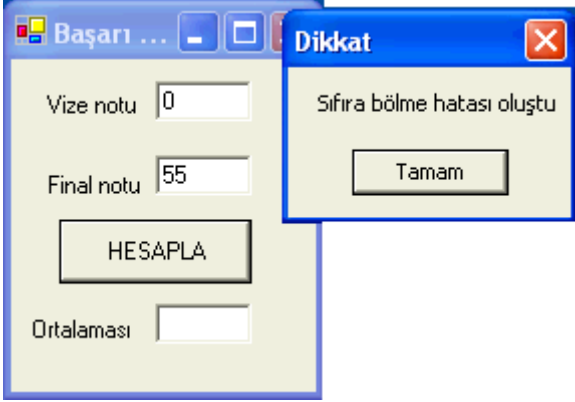
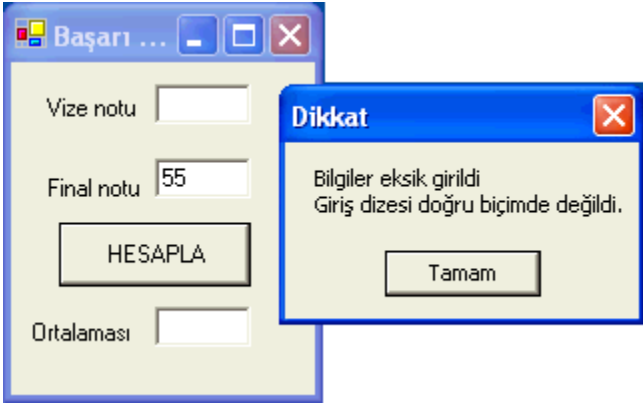



```
C:\Documents and Settings\Sef\Belgelerim\Visual...  
8/1=8  
Sıfıra bölünemez  
32/2=16  
64/2=32  
Sıfıra bölünemez  
256/4=64  
Eşleşen eleman bulunamadı  
Kritik hata... Program sonlandırıldı
```

Resim 3.15: Rethrow kullanımı ekran görüntüsü

UYGULAMA FAALİYETİ

İşlem Basamakları	Öneriler
<ul style="list-style-type: none"> ➤ Yeni bir Windows Application oluşturarak formu Resim 3.16'daki gibi tasarlayınız. 	 <p>Resim 3.16: Formun tasarlanması</p>
<ul style="list-style-type: none"> ➤ Girilen bilgiler “HESAPLA” butonuna tıklandığında işleme tabi tutulacağından program kodlarını Click() metoduna yazınız. 	<ul style="list-style-type: none"> ➤ “HESAPLA” butonunun üzerine fareyle çift tıklayınız.
<ul style="list-style-type: none"> ➤ Metin kutularından girilen bilgileri aktaracağınız değişkenleri int tipinde, ortalamaları için kullanacağınız değişkeni ise double tipte tanımlayınız. 	<ul style="list-style-type: none"> ➤ <code>int vize,final;</code> <code>double ort;</code>
<ul style="list-style-type: none"> ➤ Yapılacak işlemleri try bloğu içine yazınız. 	<ul style="list-style-type: none"> ➤ <code>try</code> <code>{</code> <code>}</code>
<ul style="list-style-type: none"> ➤ Metin kutularından girilen bilgileri int veri tipine çevirerek ilgili değişkenlere aktarınız. 	<ul style="list-style-type: none"> ➤ <code>vize=int.Parse(textBox1.Text);</code> <code>final=int.Parse(textBox2.Text);</code>
<ul style="list-style-type: none"> ➤ Sıfıra bölme hatası için bir catch bloğu tanımlayınız. 	<ul style="list-style-type: none"> ➤ <code>catch(DivideByZeroException)</code> <code>{</code> <code>}</code>

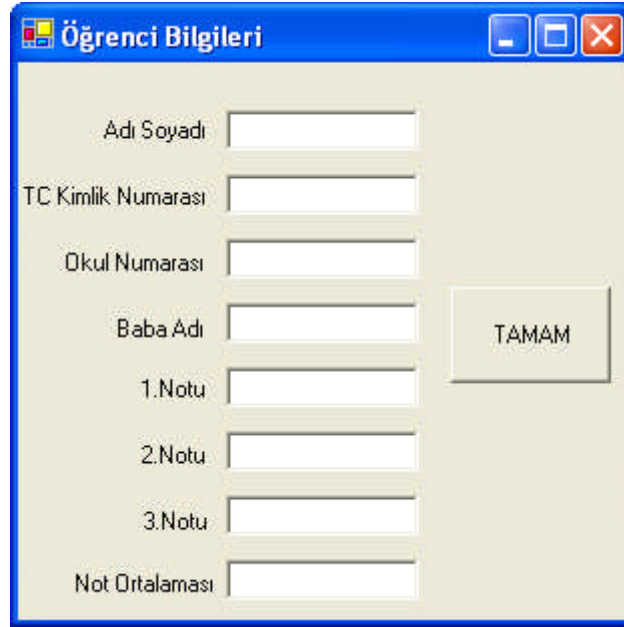
<p>➤ Sıfıra bölme hatası gerçekleştiğinde kullanıcıya verilecek mesajı mesaj kutusunda gösteriniz.</p>	<p>➤ <code>MessageBox.Show("Sıfıra bölme hatası");</code></p>  <p>Resim 3.17: Sıfıra bölme hatası</p>
<p>➤ Bilgilerin eksik girilmesine karşı bir catch bloğu daha tanımlayarak Exception parametresiyle bir hata değişkeni belirtiniz.</p>	<p>➤ <code>catch(Exception hata)</code></p> <pre>{ }</pre>
<p>➤ Eksik bilgi girilmesiyle oluşacak hatayı mesaj kutusuyla kullanıcıya gösteriniz.</p>	<p>➤ <code>MessageBox.Show("Bilgiler eksik girildi\n" + hata.Message, "Dikkat");</code></p>  <p>Resim 3.18: Bilgilerin eksik girilmesi</p>
<p>➤ try bloğunda iken eğer ilk metin kutusuna sıfır girilirse throw ile DivideByZeroException istisna durumunu fırlatınız.</p>	<p>➤ <code>if(vize==0)</code></p> <pre>{ throw new DivideByZeroException; }</pre>

➤ Metin kutularından herhangi birine bilgi girilmemesi durumunda throw ile Exception istisna durumunu fırlatınız.	➤ <pre>if ((vize.ToString()=="") (final.ToString()=="")) { throw new Exception; }</pre>
➤ Daha sonra vize ve final notlarını toplayıp ikiye bölerek üçüncü metin kutusuna yazdırınız.	➤ <pre>ort=(vize+final)/2; textBox3.Text=ort.ToString();</pre>
➤ Her halde çalışacak bir finally bloğu oluşturunuz ve “İşlem bitti” mesajını verdiriniz.	➤ <pre>MessageBox.Show("İşlem bitti");</pre>  Resim 3.19: finally bloğuyla verilen mesaj

UYGULAMA FAALİYETİ

Aşağıdaki maddeleri dikkate alarak uygun projeyi hazırlayınız.

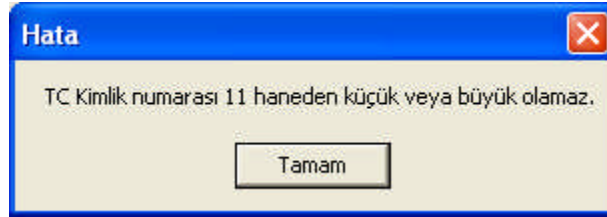
- Bir Windows Application başlatınız.
- Bu uygulamada öğrenciye ait ad soyad, T.C. kimlik numarası, okul numarası, baba adı ve bir dersten aldığı 3 not girilecektir.
- Bu notlara göre ortalaması hesaplatılıp başka bir metin kutusuna yazdırılacaktır.
- Her metin kutusunda Enter tuşuna basıldığında bir sonraki metin kutusuna geçilecektir.
- Oluşabilecek hatalara karşı şu önlemler alınmalıdır.
 - Ad soyad alanı mutlaka doldurulmalıdır.
 - T.C. kimlik numarası alanı 11 haneden küçük ve 11 haneden büyük olmamalıdır.
 - Okul numarası alanı 1 haneden küçük ve 4 haneden büyük olamamalıdır.
 - Baba adı alanı mutlaka doldurulmalıdır.
 - Not 1, Not 2 ve Not 3 alanı mutlaka rakam olmalı ve notlar 0 ile 100 arasında olmalıdır.
- Aşağıdaki ekran görüntülerini dikkate alarak programı yazınız.



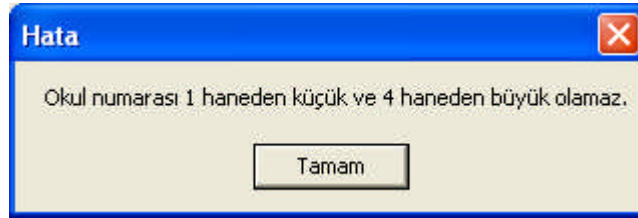
Resim 3.20: Formun tasarlanmış hali



Resim 3.21: Ad Soyad hata mesajı



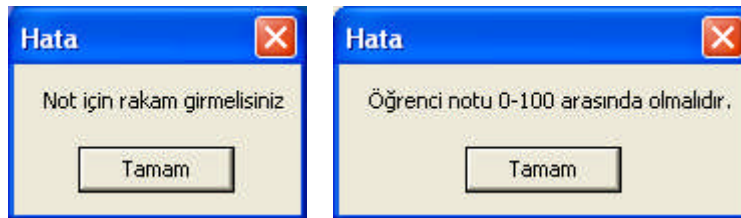
Resim 3.22: T.C. kimlik numarası hata mesajı



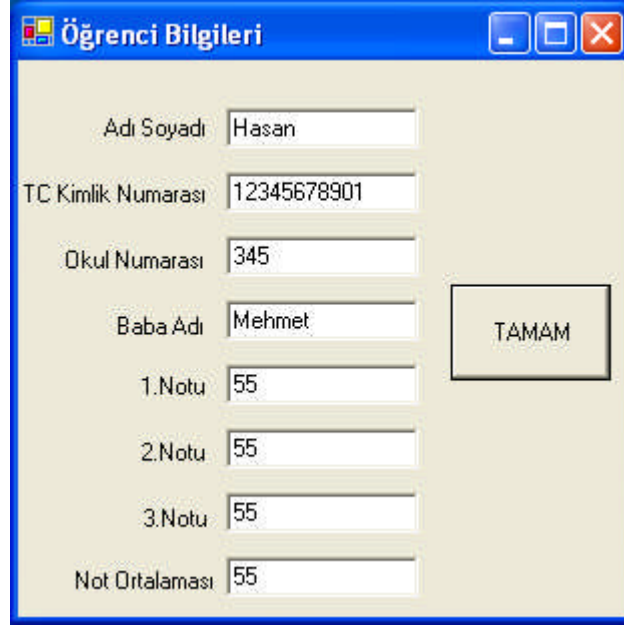
Resim 3.23: Okul numarası hata mesajı



Resim 3.24: Baba adı hata mesajı



Resim 3.25: Notlar için hata mesajı



Öğrenci Bilgileri

Adı Soyadı: Hasan

TC Kimlik Numarası: 12345678901

Okul Numarası: 345

Baba Adı: Mehmet

1.Notu: 55

2.Notu: 55

3.Notu: 55

Not Ortalaması: 55

TAMAM

Resim 3.26: Projenin çalışır hali

ÖLÇME VE DEĞERLENDİRME

Aşağıdaki soruları dikkatlice okuyarak doğru/yanlış seçenekli sorularda uygun harfleri yuvarlak içine alınız. Boşlukları uygun şekilde doldurunuz.

1. Oluşabilecek hataları kontrol altına alma olayına yönetimi denir.
2. Tüm istisnai olay sınıfları System namespace'inin Exception sınıfından türetilmek zorundadır. (D/Y)
3. try-catch bloğu oluşabilecek hatalar ve bu hatalara karşı çalıştırılacak kodların yazıldığı bir bloktur. (D/Y)
4. Her koşulda çalışacak kod satırları bloğunda yazılır.
5. Birden fazla catch bloğu kullanıldığında eğer hata herhangi bir catch bloğunda yakalanırsa diğer catch blokları hata yakalanmasına rağmen yine de çalışır (D/Y)

DEĞERLENDİRME

Cevaplarınızı cevap anahtarı ile karşılaştırınız. Doğru cevap sayınızı belirleyerek kendinizi değerlendiriniz. Yanlış cevap verdiğiniz ya da cevap verirken tereddüt yaşadığınız sorularla ilgili konulara geri dönerek tekrar inceleyiniz. Tüm sorulara doğru cevap verdiyseniz diğer öğrenme faaliyetine geçiniz.

MODÜL DEĞERLENDİRME

PERFORMANS TESTİ (YETERLİK ÖLÇME)

Modül ile kazandığınız yeterliği aşağıdaki kriterlere göre değerlendiriniz.

DEĞERLENDİRME ÖLÇÜTLERİ	Evet	Hayır
➤ Bir Windows application formu tasarladınız mı?		
➤ Sayıların değişken türlerini sınırlarına göre seçtiniz mi?		
➤ Değişkenlerle işlem yaptınız mı?		
➤ Döngü tiplerini kullanabildiniz mi?		
➤ İşlem sonuçlarını metin kutusunda yazdırdınız mı?		
➤ İşleyişi satır satır test ettiniz mi?		
➤ comboBox'a eleman eklediniz mi?		
➤ Bilgileri dosyaya yazdırdınız mı?		
➤ Uygun namespace'leri projeye eklediniz mi?		
➤ Dinleyici oluşturduğunuz mu?		
➤ Dinleyiciyi Listeners sınıfına eklediniz mi?		
➤ Trace sınıfıyla dosyaya bilgi yazdırdınız mı?		
➤ Belleği boşalttınız mı?		
➤ Dosyayı kapattınız mı?		
➤ Projenin çalıştığı klasörü tespit ettiniz mi?		
➤ Hata ayıklama için try-catch bloğu kullandınız mı?		
➤ Her halde çalışacak finally bloğu tanımladınız mı?		
➤ Sıfıra bölme ahatsı için catch bloğu tanımladınız mı?		
➤ Hataya göre uygun hata mesajları yazdırdınız mı?		
➤ Oluşabilecek hatayı fırlattınız mı?		

DEĞERLENDİRME

Yaptığınız değerlendirme sonucunda eksikleriniz varsa öğrenme faaliyetlerini tekrarlayınız.

Modülü tamamladınız, tebrik ederiz. Öğretmeniniz size çeşitli ölçme araçları uygulayacaktır, öğretmeninizle iletişime geçiniz.

CEVAP ANAHTARLARI

ÖĞRENME FAALİYETİ-1 CEVAP ANAHTARI

1	Doğru
2	Yanlış
3	Restart
4	Run to Cursor
5	Doğru
6	TaskList
7	F9
8	immediate
9	D
10	C

ÖĞRENME FAALİYETİ-2 CEVAP ANAHTARI

1	B
2	C
3	app.config
4	Yanlış
5	Doğru

ÖĞRENME FAALİYETİ-3 CEVAP ANAHTARI

1	istisna
2	Doğru
3	Doğru
4	finally
5	Yanlış

KAYNAKÇA

- SCHILDT Herbert, “**Herkes için C#**”, Alfa Basım Yayım Dağıtım Ltd.Şti, İstanbul 2005.
- DEMİRLİ Nihat, İNAN Yüksel, “**Visual C#.NET 2005**”, Palme Yayıncılık, Ankara 2006.
- Karagülle İHSAN, “**Visual C#.Net Başlangıç Rehberi**”, Türkmen Kitabevi, İstanbul 2004.
- YANIK Memik, “Microsoft Visual C#.NET”, Seçkin Yayıncılık, Ankara, 2004.
- ZENGİN Abdullah, “**C# 2005**”, Nirvana Yayınları, Ankara 2006.
- ZEKİ Yasemin, “**Adım Adım C++ Uygulamaları**”, Nirvana Yayınları, Ankara 2006.
- www.csharpnedir.com
- www.msakademik.net