












Hedefler

Bu üniteyi çalıştıktan sonra;

-  Nesneye Yönelik Programlamayı öğrenmek,
-  Sınıfları (Classes) öğrenmek,
-  Sınıf Elemanlarının Erişim Kapsamını öğrenmek,
-  Sınıf Özelliklerinin Oluşturulmasını öğrenmek,
-  Sınıf Metotlarının Oluşturulmasını öğrenmek,
-  Sınıfa Olay Eklemeyi öğrenmek,
-  Sınıflardan Nesneler Oluşturmayı öğrenmek,
-  Sınıflardaki Olayları Kullanmayı öğrenmek,
-  Miras Almayı (Inheritance) öğrenmek,

İçindekiler

- Nesneye Yönelik Programlama,
- Sınıflar (Classes),
- Projeye Yeni Bir Sınıf Ekleme,
- Sınıf Elemanlarının Erişim Kapsamı,
- Sınıf Değişkenlerinin Tanımlanması,
- Sınıf Özelliklerinin Oluşturulması,
- Sınıf Metotlarının Oluşturulması.
- Sınıfa Olay Ekleme,
- Sınıflardan Nesneler Oluşturmak,
- Sınıflardaki Olayları Kullanmak,
- Miras Alma (Inheritance),
- Miras Alma (Inheritance) Özellikleri,
- Miras Alınan Metotların Kullanılma Özellikleri,
- Çok Biçimlilik (Polymorphism) ve Aşırı Yükleme (Overloading)

Nesneye Yönelik 'Programlama

Visual Basic için yapılan en önemli eleştiriler bu dilin nesneye yönelik bir dil olmaması üzerineydi. Özellikle C ve C++ programcıları, bu sebeple dili çok küçümsemekteydi. Programcılığa yeni soyunanlar, bu özellik sebebiyle dili çok basit görüp uzaklaştılar. Ama artık Visual Basic.NET'le birlikte bu dil tümüyle nesneye yönelik programa dili oldu.

Yazılım sektöründe program geliştirme konusunda günümüze kadar birçok yaklaşım denenmiştir. Bunların ilki programın baştan aşağıya sırası ile yazılıp çalıştırılmasıdır. Bu yaklaşımla BASIC dili kullanılarak birçok program yazıldığını biliyoruz. Burada sorun programın akışı sırasında değişik kısımlara **goto** deyimi ile atlanmasıdır. Program kodu bir kaç bin satır olunca, kodu okumak ve yönetmek gerçekten çok büyük sorun oluyordu. GWBasic, QBasic, Quick Basic gibi derleyiciler buna iyi örnekti.

İkinci yaklaşım ise yapısal yaklaşımdır. Programlarda birçok işin tekrar tekrar farklı değerleri kullanılarak yapıldığı fark edildi. Mesela herhangi bir programda iki tarih arasında ne kadar gün olduğunu bulmak birçok kez gerek olabilir. Bu durumda başlangıç ve bitiş tarihlerini alıp aradaki gün sayısını veren bir fonksiyon yazılabilir ve bu fonksiyon ihtiyaç duyulduğu yerde uygun parametrelerle çağrılıp istenen sonuç elde edilebilir. Yapısal yaklaşım Pascal ve C dillerinde uzun yıllar başarı ile kullanılmıştır.

Ama her geçen gün programların daha karmaşık bir hal alması, program kodunun kurumsal uygulama projelerinde on binlerce satırı bulması ve yazılım geliştirme maliyetinin çok arttığını gören bilim adamları, programcılara yeni bir yaklaşımın kullanılabilineceğini öğretiler. Bu yaklaşımın ismi Nesneye Yönelik Programlama(Object Oriented Programming) dir.

Nesne yönelimli programlama tekniği, diğer yaklaşımlara nazaran, yazılım geliştiren insanlara büyük avantajlar sağlamaktadır. Birincisi karmaşık yazılım projelerinin üretilmesini ve bakımını kolaylaştırıyor olmasıdır. Diğeri ise program kodunun tekrar kullanılabilmesine olanak sağlamasıdır. Bu noktada program kodunun tekrar kullanılabilmesi profesyonel yazılım şirketlerinin maliyetlerini azaltmıştır. Dolayısı ile programların lisans ücretleri düşmüş ve sektörün sürekli olarak canlı kalmasına ve rekabet içinde gelişmesine yardımcı olmuştur.

Programlarımız içerisinde prosedür, fonksiyon yada değişkenler, bir çatı altında tanımlanarak, tekrar tekrar yazmaya gerek kalmadan kullanılabilir. Bu tip tanımlamalar, **Class** yapıları içerisinde yapılabilir. Class yapıları, projeler içerisinde kullanılabileceği gibi, derlenip **dll** dosyası haline getirilerek, başka projelerde de referans olarak kullanılabilir.

Nesne yönelimli programlamada esas olan, gerçek hayatta varolan nesnelerin programlamaya aktarılmasındaki yeni yaklaşımdır. Nesneye yönelik programlama kodların yeniden kullanılabilmesini ve verilere işlevler yüklenebilmesini sağlar.

Bir nesne onu tanımlayan bilgileri ve bunlar üzerinde işlem yapmaya yarayan yöntemleri içerir. Otomobil, köpek, sandalye gerçek dünyada tanımlayabileceğimiz nesnelerin sadece birkaçıdır. Gerçek dünyadaki bu nesneleri 2 şekilde tanımlarız. Birincisi yaşı, rengi, cinsi gibi özellikleriyle. İkincisi ise havlaması, koşması ve ısırması gibi olaylarıyla. Yazılım nesneleri de bu 2 özelliği taşır.

Nesneye yönelik programlamanın bir diğer özelliği ise programların sınıflardan oluşuyor olmasıdır. Sınıflar (Class) benzer özellikleri taşıyan nesnelere ait değişkenleri ve yöntemleri içeren prototiplerdir. Başka bir ifadeyle, sınıfları kağıtlar üzerine baskı yapan bir kalıp şeklinde düşünülebilir. Ürünler (nesneler) için üzerine baskı yapılan kağıdı yada renk gibi özellikleri değiştirilerek yeni özellikler kazandırılabilirken kalıp (sınıf) asla değişmez.

Sınıf yani "Class" kavramının dışında başka bir açıklanması gereken kavramda "Namespace" kavramıdır. Bu da Class (Sınıf) kütüphanesi olarak düşünülebilir.

Nesneye yönelik programlamanın 4 temel özelliği vardır.

1. Paketleme (Encapsulation)
2. Soyutlama (Abstraction)
3. Çok biçimlilik (Polymorphism)
4. Kalıtım (inheritance)

1. Paketleme:

Paketleme (Encapsulation) kodu ve kodu meydana getiren verileri bir araya getiren bir mekanizmadır. Bu mekanizma nesneyi dış etkilerden korumak için kullanılır. Veriler ve metotlar nesneyi oluşturur. Yani sınıfı kullanan kişi metotların nasıl yazıldığını görmez. Buna paketleme denir.

2. Soyutlama:

Soyutlama (Abstraction) nesneyi karakteristikleri olan ve eylemleri olan veri tipi olarak anlaşılmalıdır.

3. Çok biçimlilik:

Çok biçimlilik (Polymorphism) 'deyince aklımıza gelen bir fonksiyon birden çok biçimde kullanılmalıdır. Bu durumda fonksiyonların aşırı yüklenmesi gündeme gelir. Yani bir fonksiyon birden fazla biçimde deklare edilir. Kullanılırken hangi türde veri içeriyorsa o tür kullanım çağırılır.

4. Kalıtım

Kalıtım (inheritance) varolan sınıfları temel alarak yeni sınıflar oluşturabilme anlamına gelir. Varolan sınıfa temel (ebeveyn) sınıf, yeni oluşturulan sınıfa da türetilmiş (çocuk) sınıf denir. Türetilen sınıf, temel sınıfın tüm özelliklerine sahiptir.

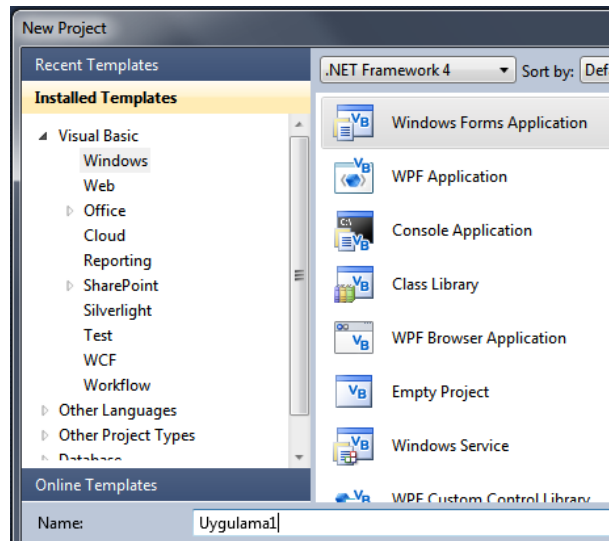
Sınıflar (Classes)

Nesneye yönelik programlamada her şey; sınıflardan (**classes**) ve sınıfları referans olarak oluşturulan nesnelerden (**object**) oluşur. Visual Basic.NET tam anlamıyla nesneye dayalı olduğundan sınıf yapısını destekler. Nesneye yönelik programlamada nesneler değil, sınıflar yazılır. Sınıf belirli bir türün tüm nesneleri için kodu ve veri bildirimlerini sağlar. Sınıfların değişkenleri, özellikleri, metotları ve olayları vardır. Bir uygulama ile paralel olarak, **Class** kavramlarını açıklamaya çalışalım.

Uygulama : Dairenin Alanını Bulan Program

Dairenin yarıçapı verildiğinde, alanını bulan programı Daire Class 'ı (Sınıf) yardımıyla bulmaya çalışacağız ..

Adım 1 : Visual Basic .NET 2003'ü çalıştırmak için, **Start** (Başlat) butonu tıklanır. **All Programs** (Programlar) seçeneğinin altındaki **Microsoft Visual Studio .NET 2010** seçeneği bulunur ve burada açılan seçeneklerden **Microsoft Visual Studio .NET 2010** seçeneği tıklanır.

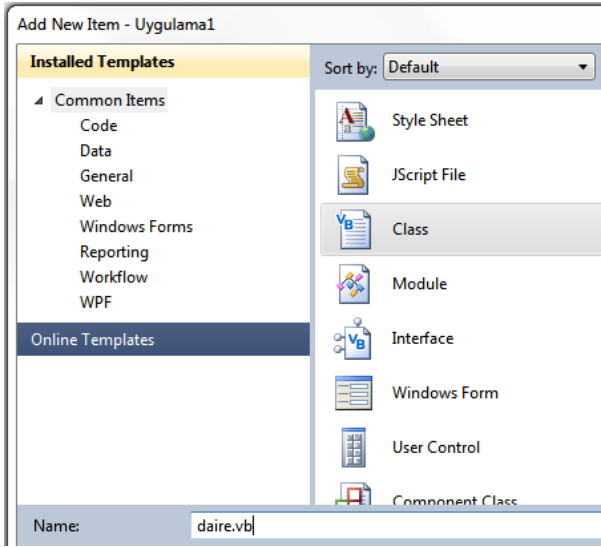


Adım 2 : Program çalıştığında ekranda **Start Page** sayfası görüntülenecektir. **New Project** tıklanır. **New Projects** Penceresinde, **Visual Basic**'de **Windows** ve sağ taraftan **Windows Forms Application** seçilir. **Name** kutusuna **Uygulama1** yazıyoruz. Ve **OK** butonuna basarak yeni projemize başlıyoruz

Projeye Yeni Bir Sınıf Ekleme: Daire alanını hesaplayan bir sınıf ekleyelim.

Visual Basic .NET'de Sınıf, **Class** deyimi ile deklare edilir. Sınıfı tanımlayan ifadeler, **Class** ve **End Class** deyimlerinin arasına yazılır.

```
Class daire
...
End Class
```

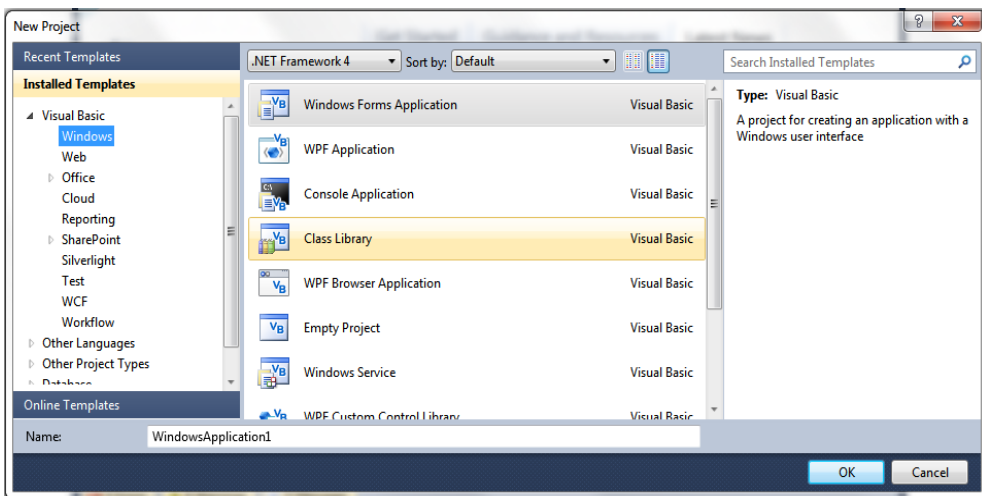


Visual Basic .NET projesi içinde **Class** (Sınıf) oluşturmak için, **Project** menüsünden **Add Class...** seçeneği yada **Project** menüsünden **Add New Item** seçeneği tıklanarak, gelen pencereden **Class** seçeneğine tıklanır. **Name** kutusuna, sınıf ismi (örneğin daire.vb) yazılarak **Add** butonuna basılır.

Code Editor içinde **daire.vb** isimli boş bir sınıf modülü açılacak ve aynı zamanda **Solution Explorer** penceresinde **daire.vb** sınıfı listelenecektir.



Eğer Class projesi ayrı bir **dll** dosyası şeklinde derlenecekse; File → New Project seçilirdikten sonra gelen proje şablonlarından Class Library şablonu seçilmelidir.



Sınıf Elemanlarının Erişim Kapsamı

Sınıf ve elemanlarının erişim kapsamı, erişebildiği kod bölgesi anlamına gelir.

Public sözcüğü ile bildirilen sınıf elemanlarına her yerden erişilebilir.

Private sözcüğü ile bildirilen sınıf elemanlarına sınıf içinden erişilebilir.

Friend sözcüğü ile bildirilen sınıf elemanlarına sınıf tanımını içeren program içinden erişilebilir.

Protected sözcüğü ile bildirilen elemanlara sınıf içinde ya da o sınıfı kalıtımla alan bir sınıf içinde erişilebilir.

Protected Friend, Protected ve Friend erişimlerinin birleşimidir.

Sınıf Değişkenlerinin Tanımlanması

Daire sınıfını yazıyoruz. Sınıf değişkeni olarak yalnızca yarıçap (**R**) değişkenini kullanacağız. **Public Class Daire** ifadesinin hemen altına aşağıdaki değişken tanımlamasını yapıyoruz:

```
Public Class daire
    Public R As Single

End Class
```

Sınıf Özelliklerinin Oluşturulması

Sınıf özelliklerinin oluşturulması için, aşağıdaki **Property** ifadesi kullanılır. **Property** ifadesi kullanılarak, sınıflar içerisinde değer gönderip, değerleri okuyabiliriz. **Get** bloğu, programcıların özelliğe baktıklarında ne göreceklarini

belirler. **Set** bloğu ise programcıların özelliği ayarladıklarında ya da değiştirdiklerinde ne olacağını belirler. (**Set** ile değer gönderirken, **Get** ile değer alırız.)

```
Public Özellik_ismi as Değişken
    Get
        ...
    End Get

    Set
        ...
    End Set

End Property
```

Şimdi Daire sınıfı için yarıçap (Yaricap) özelliğini yazalım:

```
Public Class daire
    Public R As Single
    Public Property Yaricap() As Single
        Get
            Return R
        End Get
        Set(ByVal value As Single)
            If value <= 0 Then
                MessageBox.Show("Yarıçap 0 veya negatif olamaz.")
            Else
                R = value
            End If
        End Set
    End Property
End Class
```

Sınıf Metotlarının Oluşturulması

Sınıf içinde bir eylemi yapan bir metod oluşturmak için **Sub** veya **Function** alt programları eklenir. Örneğimizde Dairenin alanını hesaplayan bir metod oluşturacak. **Class** (Sınıf) tanımımızda, **End Property** satırından sonra aşağıdaki **Function**'ı ekliyoruz:

```
Public Function Alan()  
    Return Math.PI * Yarıcap * Yarıcap  
End Function
```

Sınıfa Olay Ekleme

Olay programları, belli bir durum meydana geldiğinde çalışan alt programlardır. **Button1_click**, **text1_changed** gibi. Bir sınıfa olay eklemek için **Event** ifadesi kullanılır. **Class** içinde aşağıdaki ifade deklare edilmelidir.

```
Public Event event_ismi(Parametreler)  
  
Olayı başlatmak için ayrıca kod içine ;  
  
RaiseEvent event_ismi(Parametreler)  
  
ifadesi eklenmelidir.
```

Örneğimize dönersek, **Yarıcap değiştiğinde** devreye girecek bir olay ekleyelim. **Class daire** sınıfına aşağıda altı çizili 2 satırı ekleyeceğiz.

```
Public Class daire
    Public Event YaricapDegisikligi()
    Public R As Single
    Public Property Yaricap() As Single
        Get
            Return R
        End Get
        Set(ByVal value As Single)
            If value <= 0 Then
                MessageBox.Show("Yarıçap 0 veya negatif olamaz.")
            Else
                R = value
                RaiseEvent YaricapDegisikligi()
            End If
        End Set
    End Property
    Public Function Alan()
        Return Math.PI * Yaricap * Yaricap
    End Function
End Class
```

Sınıflardan Nesneler Oluşturmak

Sınıfı kullanabilmek için, o sınıfın bir nesnesi oluşturulmalıdır. Nesne oluşturmanın 3 yolu vardır ve bu tanımlamalarda **New** ifadesinden yararlanılır:

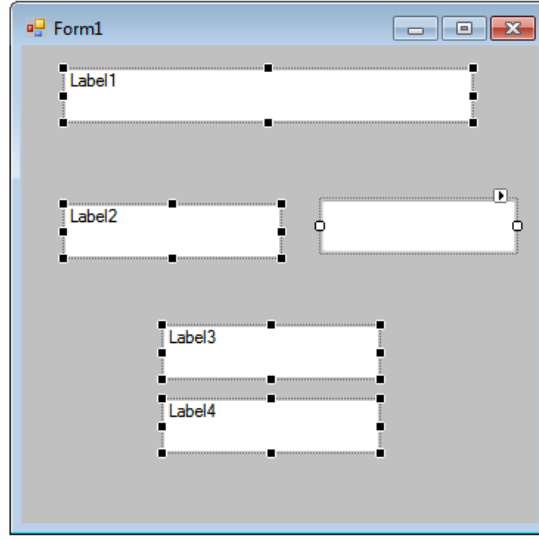
1. `Dim Nesne_adi As Sinif_adi`
 `Nesne_adi = New Sinif_adi()`
2. `Dim Nesne_adi As New Sinif_adi`
3. `Dim Nesne_adi As Sinif_adi = New Sinif_adi`

Eğer oluşturulan sınıfta Olay veya Olaylar deklare edilmişse, nesne oluşturulurken, nesne isminden önce **WithEvents** ifadesi kullanılmalıdır.

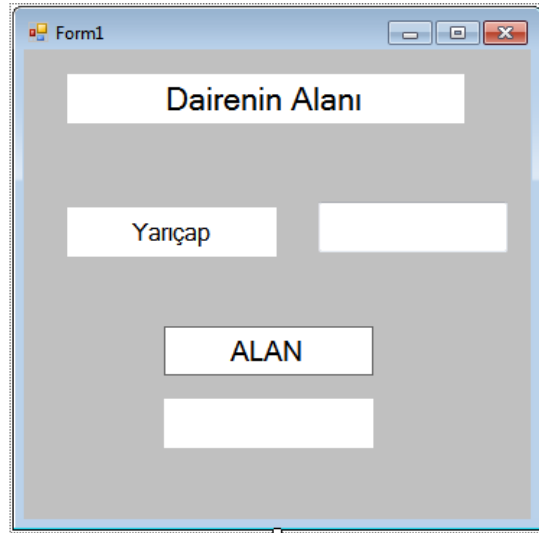
Örneğin ;

```
Dim WithEvents Daire1 As New Daire()
```

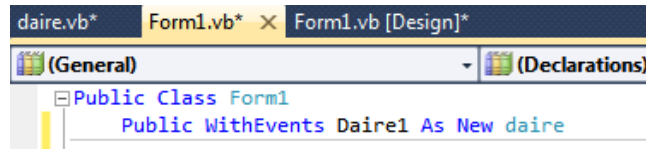
Uygulamamızda bunu kullanalım. İlk önce **Form** tasarımı yapalım. Sağ taraftaki **Solution Explorer**'da **Form1** üzerine çift tıklayalım. **Form1** üzerine aşağıdaki gibi, 4 tane **Label** ve 1 tane **TextBox** kontrolü ekleyelim.



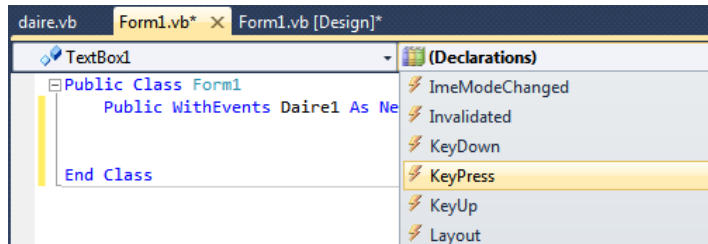
Daha sonra **Label**'ları ve **TextBox**'ı aşağıdaki gibi düzenleyelim.



Form1 [Design] sayfasındayken, **Solution Explorer**'daki **View Code** ikonunu tıklayalım ve kod sayfasına geçelim. Kod sayfasının baş tarafında **Daire1** nesnesini (Daire sınıfından yararlanarak) tanımlayalım :



İlk önce **TextBox1** bilgi kutusuna girişi kontrol için bir alt program yazalım. Sayı girişi sırasında, yalnızca 0-9 arası rakamlara, ondalık noktaya (.) , düzeltme yapmak için Back Space tuşuna ve de ENTER tuşuna (sayı girişini bitirmek için) izin vermek istiyoruz. Bunun için **KeyPress** olayını kullanacağız. Kod sayfasında sağ tarafta açılan pencereden **KeyPress** olayını bulup tıklıyoruz.



Bu tıklamanın ardından kod sayfasında **KeyPress** olayı ile ilgili alt programın satırları gözükecektir. Bu satırlar arasına ilgili program kodunu yazıyoruz:

```
Private Sub TextBox1_KeyPress(ByVal sender As Object, ByVal e As
System.Windows.Forms.KeyPressEventArgs) Handles TextBox1.KeyPress

    Dim Tus As Integer

    Tus = Asc(e.KeyChar)

    Select Case Tus

        Case Asc("0") To Asc("9"), Asc(ControlChars.Back), Asc(".")

            e.Handled = False

        Case Asc(ControlChars.Cr)

            Daire1.Yaricap = Val(TextBox1.Text)

            e.Handled = False

        Case Else

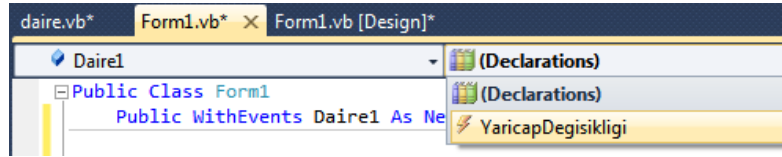
            e.Handled = True

    End Select

End Sub
```

Sınıflardaki Olayları Kullanmak

Daire sınıfında, **YaricapDegisikligi** isminde bir olay (Event) tanımlamıştık. Şimdi bunu geliştirdiğimiz program içinde kullanalım. Aynı diğer olay programlarında yaptığımız gibi, kod sayfasında, sol taraftaki açılan listeden nesneyi (yani **Daire1**) seçiyoruz. Sağ taraftaki açılan listeden de olayı (yani **YaricapDegisikligi**) seçeceğiz.



Çıkan satırlar arasına aşağıdaki kodu yazıyoruz: Kodu yazarken dikkatinizi çekmiştir. Örneğin **Daire1**. dedikten sonra aynı Visual Basic' in diğer nesnelerinde olduğu gibi açılan listede özellikler ve metotlar listelenmektedir. Buradan istediğimiz özelliği veya metodu seçebiliriz.

```
Private Sub Daire1_YaricapDegisikligi() Handles Daire1.YaricapDegisikligi
    Label14.Text = Format(Daire1.Alan, "####.##")
End Sub
```

F5 tuşuna basılır, Program çalıştırılır. Çeşitli denemeler yapılarak, programın doğru çalışıp çalışmadığı test edilir.

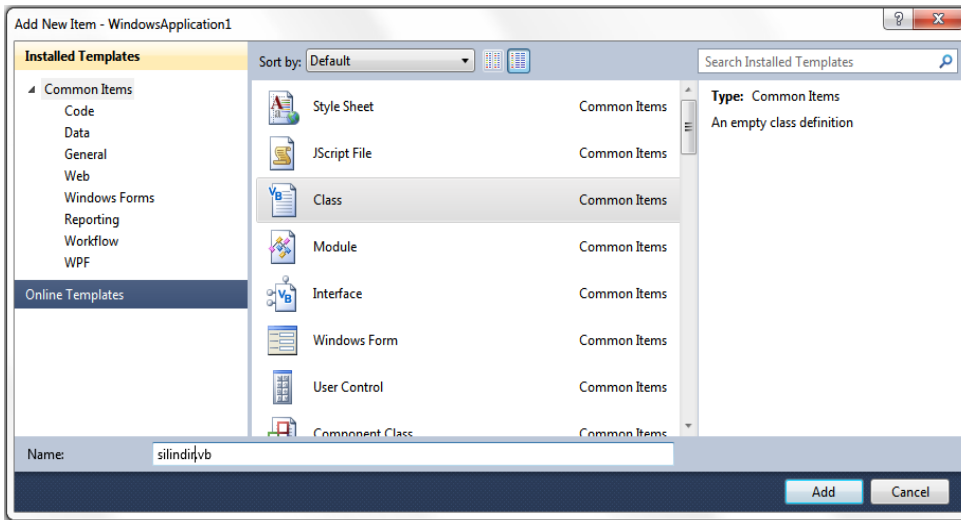
YaricapDegisikligi olayı, **Yarıçap** değiştiğinde otomatik olarak **ALAN** bilgisini değiştirecektir.

Miras Alma (Inheritance)

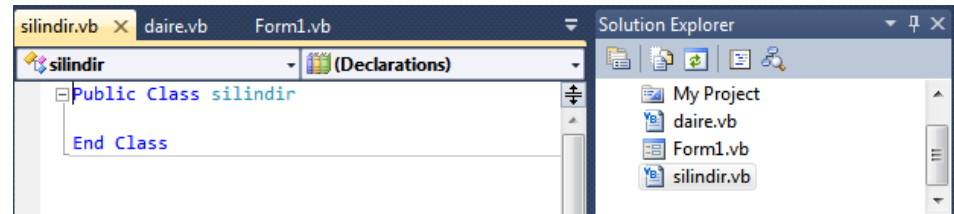
Inheritance (Miras alma) nesneye yönelik programlamada kodun yeniden kullanılabilirliğini sağlayan güçlü bir yöntemdir. **Inheritance** bir sınıfın özelliklerini ve metotlarının diğer sınıf tarafından miras olarak alınıp, yeni özellikler eklenmesine denir. Temel olan ilk sınıfa Temel Sınıf (**Base Class**), buradan türetilen sınıfa Türetilmiş Sınıf (**Derived Class**) denir. Bu yöntem ile temel sınıftaki özellik ve metotlar tekrar yazılmadan sadece **Inherits Sınıf_adi** ifadesi ile türetilmiş sınıfa aktarılır. Türetilmiş sınıfa yeni özellikler ve metotlar ekleyebiliriz.

Mevcut projemizde, **daire** sınıfı oluşturduk. **Daire** sınıfının **Yaricap** özelliği, **Alan** metodu ve **Yaricapdegisikligi** olayı vardı. Biz bu proje üzerinde Silindirin taban alanını (aslında dairenin alanı) ve hacmini hesaplayacak bir Silindir sınıfı oluşturacağız. Gerek **Yaricap** özelliği gerekse **Alan** metodu için **Daire** sınıfından miras alacağız.

Şimdi yeni bir sınıf ekleyelim. **Project** menüsünden **Add Class** seçeneği tıklanır. **Name** kutusuna, sınıf ismi (silindir.vb) yazılarak **Add** butonuna basılır.



Code Editor içinde **silindir.vb** isimli boş bir sınıf modülü açılacak ve aynı zamanda **Solution Explorer** penceresinde **silindir.vb** sınıfı listelenecektir.



İlk olarak Daire sınıfını miras olarak alacağız. **Inherits daire** yazıyoruz:

```
Public Class silindir
    Inherits daire
End Class
```

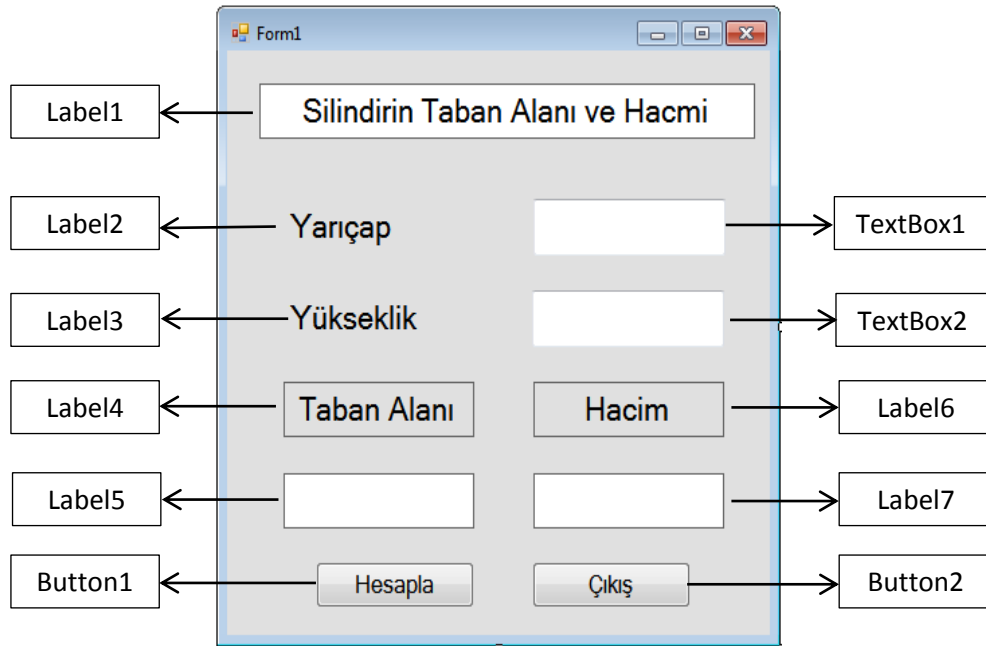
Inherits daire satırını eklemekle, dairenin **yaricap** özelliği ve alan metodu silindir sınıfına miras olarak aktarılmış oldu. **Hacim** hesabı için ayrıca **Yukseklik** özelliğine ihtiyaç vardır. Ve ardından **Hacim** metodunu tanımlamak için **Function Hacim** satırlarını ekliyoruz:

```
Public Class silindir
    Inherits daire

    Public H As Single
    Public Property Yukseklik() As Single
        Get
            Return H
        End Get
        Set(ByVal value As Single)
            If value <= 0 Then
                MessageBox.Show("Yükseklik 0 veya negatif olamaz")
            Else
                H = value
            End If
        End Set
    End Property

    Public Function Hacim()
        Return Alan() * Yukseklik
    End Function
End Class
```


Uygulamamızda **Silindir** sınıfını kullanalım. İlk önce **Form** tasarımı yapalım. Sağ taraftaki **Solution Explorer**'da **Form1** üzerine çift tıklayalım. **Form1** üzerinde değişiklikler yapıp aşağıdaki şekle getirelim:



Form1 [Design] sayfasındayken, **Solution Explorer**'daki **View Code** ikonunu tıklayalım ve kod sayfasına geçelim. Kod sayfasının baş tarafında **Silindir1** nesnesini (Silindir sınıfından yararlanarak) tanımlayalım:

```
Public Class Form1
    Public Silindir1 As New silindir
```

Şimdi **Button1 (Hesapla)** butonu için kodu yazalım. **Silindir1** nesnesinin **Yaricap** özelliği ile **Silindir1** nesnesinin **Alan** özelliği **Daire** sınıfından miras alınmıştır. Bunlar da kullanılırken, aynı **Silindir** sınıfında tanımlanmış gibi kullanılmaktadır.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

    Silindir1.Yaricap = Val(TextBox1.Text)
    Silindir1.Yukseklık = Val(TextBox2.Text)

    Label5.Text = Format(Silindir1.Alan, "####.##")
    Label7.Text = Format(Silindir1.Hacim, "####.##")

End Sub
```

F5 tuşuna basılır, Program çalıştırılır. Çeşitli denemeler yapılarak, programın doğru çalışıp çalışmadığı test edilir.

Miras Alma (Inheritance) Özellikleri

NotInheritable : Bir Class (Sınıf) **NotInheritable** ile tanımlanmışsa, miras olarak alınamaz. Örneğin **Daire** sınıfını ;

Public NotInheritable Class Daire

şeklinde kullansaydık, **Silindir** sınıfı bunu miras olarak alamayacaktı.

MustInherit : Bir **Class** (Sınıf) **MustInherit** ile tanımlanmışsa, bu sınıfa ait nesneler direkt olarak oluşturulamaz. Sadece bu sınıftan türetilmiş sınıftan nesneler direkt olarak üretilebilir.

Miras Alınan Metotların Kullanılma Özellikleri

Overridable : Temel Sınıfa (**Base Class**) ait bir elemanın (özellik veya metot) yeniden tanımlanabilmesini sağlar.

Örneğin: Public Overridable Function Alan()

NotOverridable : Temel Sınıfa (**Base Class**) ait bir elemanın (özellik veya metot) yeniden tanımlanabilmesine izin vermez.

MustOverride : Sınıf elemanları (özellik veya metot) yeniden tanımlanmak zorundadır.

Örneğin : Public MustOverride Function Alan ()

Overrides : Türetilmiş Sınıfta (**Derived Class**), temel sınıfta yer alan metodu tanımlamak için kullanılır.

Örneğin : Overrides Function Alan()

Çok Biçimlilik (Polymorphism) ve Aşırı Yükleme (Overloading)

Çok biçimlilik (**Polymorphism**) farklı sınıflar tarafından farklı yollarla kullanılan özellikler yada metotlar iki yada daha fazla sınıf tarafından aynı adla sağlandığında oluşur. Örneğin geometrik cisimlerin alanını hesaplayan sınıflar oluşturalım. Kare sınıf, **Alan** metodunda karenin alanını, üçgen sınıfı, Alan metodunda, üçgen in alanını, dikdörtgen sınıfı, **Alan** metodunda, dikdörtgenin alanını hesaplasın. Her sınıf için metot isimi aynı (**ALAN**) olmasına rağmen, her sınıfa ait nesne kendi alanını hesaplayacaktır.

Aşırı yükleme (**Overloading**), bir sınıf aynı adlı fakat farklı türde değer gönderen birkaç yöntem içerdiğinde oluşur.

Örnek : Deneme sınıfı oluşturalım. Burada 4 farklı toplama işlemini yapan 4 metod oluşturacağız. Metodların hepsinin ismi, **TOPLAMA** olacak.

1. metod 2 tamsayıyı,
2. metod 2 kesirli sayıyı,
3. metod 2 tamsayıyı,
4. metod 2 alfabetik ifadeyi toplayacak.

```
Public Class deneme
    Overloads Function Toplama(ByVal A As Integer, ByVal B As Integer) As Integer
        Return A + B
    End Function

    Overloads Function Toplama(ByVal A As Single, ByVal B As Single) As Single
        Return A + B
    End Function

    Overloads Function Toplama(ByVal A As Single, ByVal B As Single, ByVal C As Single)
As Single
        Return A + B + C
    End Function

End Class

    Overloads Function Toplama(ByVal A As String, ByVal B As String) As String
        Return A + B
    End Function
```

Program içinde **Deneme1** nesnesi tanımlayalım.

Dim Deneme1 As New Deneme

Bu tanımlamayı yaptıktan sonra,

Deneme1.toplama (3.2, 5.4) şeklinde yazarsak, parametrelerin her ikisi de kesirli sayı (**Single**) olduğu için, 2. Toplama metodu devreye girecek ve sonucu 8.6 olarak belirleyecektir.

Deneme1.toplama (3, 5) şeklinde yazarsak, parametrelerin her ikisi de tamsayı (**Integer**) olduğu için, 1. Toplama metodu devreye girecek ve sonucu 8 olarak belirleyecektir.

Deneme1.toplama ("Sakarya" , "Üniversitesi") şeklinde yazarsak, parametrelerin her ikisi de alfabetik (***String***) olduğu için, 4. Toplama metodu devreye girecek ve sonucu "Sakarya Üniversitesi" olarak belirleyecektir.

Deneme1.toplama (3, 5, 6) şeklinde yazarsak, parametreler 3 tane. ve hepsi tamsayı (***Integer***) olduğu için, 3. Toplama metodu devreye girecek ve sonucu 14 olarak belirleyecektir.

Deneme1.toplama (3.2, 7) şeklinde yazarsak, parametrelerin birincisi kesirli (***Single***), ikincisi tamsayı (***Ineteger***) 'dır. Buna uyan fonksiyon olmadığı için hata mesajı verecektir.