

**T.C.
MİLLÎ EĞİTİM BAKANLIĞI**

BİLİŞİM TEKNOLOJİLERİ

NESNE TABANLI PROGRAMLAMADA WINDOWS UYGULAMALARI

Ankara, 2013

- 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	ii
GİRİŞ	1
ÖĞRENME FAALİYETİ-1	3
1. WPF Formlar	3
1.1. WPF Uygulaması Oluşturma	5
1.2. WPF Form Özellikleri.....	7
1.3. Nesnelar	14
1.4. Nesne Özellikleri.....	16
1.5. Özellikleri Dinamik Olarak Değiştirme	25
1.6. Olayları İşleme.....	27
UYGULAMA FAALİYETİ	31
ÖLÇME VE DEĞERLENDİRME	33
ÖĞRENME FAALİYETİ-2	34
2. MENÜ VE İLETİŞİM KUTULARI	34
2.1. Menü Oluşturma	35
2.2. Menü Olaylarını İşleme	39
2.3. Kısayol Menüleri	41
2.4. Ortak İletişim Kutuları	43
UYGULAMA FAALİYETİ	46
ÖLÇME VE DEĞERLENDİRME	50
ÖĞRENME FAALİYETİ-3	51
3. DOĞRULAMAYI GERÇEKLEŞTİRME	51
3.1. Veri Doğrulama	51
3.2. Veri Bağlama	59
UYGULAMA FAALİYETİ	65
ÖLÇME VE DEĞERLENDİRME	66
MODÜL DEĞERLENDİRME	67
CEVAP ANAHTARLARI.....	68
KAYNAKÇA	69

AÇIKLAMALAR

ALAN	Bilişim Teknolojileri
DAL/MESLEK	Veri Tabanı Programcılığı
MODÜLÜN ADI	Nesne Tabanlı Programlamada Windows Uygulamaları
MODÜLÜN TANIMI	WPF ile çalışma ve Windows uygulamaları üretme becerilerinin kazandırıldığı bir öğrenme materyalidir.
SÜRE	40/32
ÖN KOŞUL	“Nesne Tabanlı Programlamada Temsilciler ve Olaylar” modülünü tamamlamış olmak
YETERLİK	Windows uygulamaları ile çalışmak
MODÜLÜN AMACI	Genel Amaç Bu modül ile gerekli ortam sağlandığında Windows uygulamaları oluşturabileceksiniz. Amaçlar <ol style="list-style-type: none">1. WPF Formları ile çalışabileceksiniz.2. Menüler ve iletişim kutularıyla çalışabileceksiniz.3. Doğrulamayı gerçekleştirebileceksiniz.
EĞİTİM ORTAMLARI VE DONANIMLARI	Ortam: Bilgisayar laboratuvarı Donanım: Bilgisayar, programlama yazılımı
ÖLÇME DEĞERLENDİRME	Modülün içinde yer alan, her faaliyetten sonra verilen ölçme araçları ile kazandığınız bilgileri ölçerek kendi kendinizi değerlendireceksiniz. Öğretmen, modülün sonunda, size ölçme aracı (test, çoktan seçmeli, doğru-yanlış, vb.) kullanarak modül uygulamaları ile kazandığınız bilgi ve becerileri ölçerek değerlendirecektir.

GİRİŞ

Sevgili Öğrenci,

Geliştirdiğiniz uygulama her ne olursa olsun sahip olduğu çalışma ortamı üzerinde (web, mobil, masaüstü...) mutlaka kullanıcı ile iletişimde bulunacak bir arayüze sahip olmalıdır. Bugün kullandığımız işletim sistemi üzerinde çalıştırdığımız programlar, internet sayfaları, akıllı telefon uygulamalarında ekranınızda gördüğünüz her şey birer arayüzdür. Uygulamanızın gerek kullanıcıdan elde etmek istediği komut veya bilgileri programınıza, gerekse programdan elde edilen bilgilerin kullanıcıya iletilmesinden sorumlu olacağını düşündüğünüzde, tasarlanması ve gerçekleştirilmesi, oldukça önemli bir üretim aşamasıdır. İnsan ve bilgisayarın buluşma noktası olarak da niteleyebileceğiniz arayüzler en uygun etkileşim şeklini yakalamalı ve kullanıcı dostu olmalıdır.

WPF (Windows Presentation Foundation) masaüstü uygulamalarınız için zengin içerikli grafik kullanıcı arayüzü (GUI) oluşturmanıza ve görüntülemenize imkan veren sunum kütüphanesidir.

ÖĞRENME FAALİYETİ-1

AMAÇ

WPF Formları ile çalışabileceksiniz.

ARAŞTIRMA

- İşletim sisteminde çeşitli uygulamalara ait arayüz tasarımlarını inceleyiniz. Herhangi bir programa ait arayüzü kendi görsel bakış açınızla kâğıt üzerinde tasarlayınız.
- Hayali bir projenizi gerçekleştirebilmek için iş ortaklarınıza sunum yapmanız gerekmektedir. Sunumunuzda projenizin hangi özelliklerini öne çıkarırdınız?

1. WPF FORMLAR

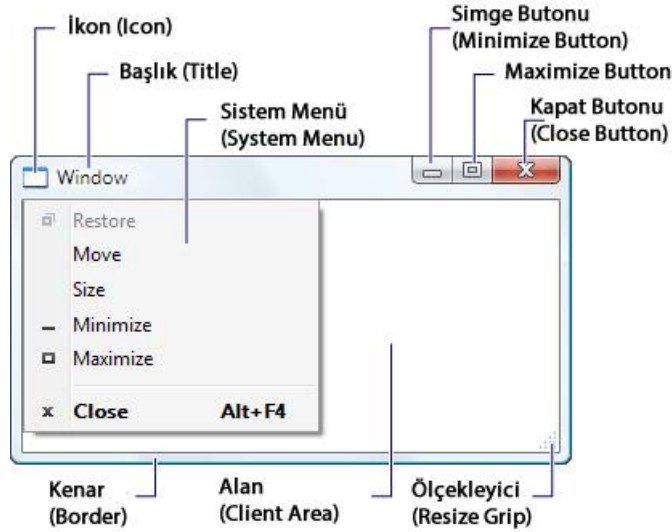
Yazılım projeleri, yoğun olarak katmanlı yapıda inşa edilir. Bütün katmanlarda (veri katmanı, iş katmanı vb.) yapılagelen işlerinizde aldığınız tüm tasarım kararlarının toplamını, sunum katmanında kullanıcı ile iletişime geçmek için kullanırsınız.

WPF (Windows Presentation Foundation)'in temeli, modern grafik donanımının avantajlarından yararlanmak için oluşturulan, çözünürlükten bağımsız ve vektör tabanlı işleme altyapısıdır.

Önceleri kullanılan “WindowsForms” kütüphanelerinden farklı “GDI+” yerine “DirectX” grafik kütüphanesini kullanır. Daha çok bilgisayar oyunlarından hatırladığımız bu kütüphane sayesinde ekran kartları sürücülerini daha etkin kullanılmakta ve daha hızlı görüntü işlemek mümkün olmaktadır.

WPF, Genişletilebilir Uygulama Biçimlendirme Dili (XAML) isimli XML tabanlı, bildirim dayalı işaretleme dili ile denetimler, veri bağlama, düzen, 2-B ve 3-B grafikler, video, animasyonlar, stiller, şablonlar, belgeler, ortam, metin ve tipografi içeren kapsamlı uygulama geliştirme özellikleri kümesi ile söz konusu temeli işler ve genişletir. WPF, “.NET Framework” içinde bulunur. Böylece “.NET Framework” sınıf kitaplığının diğer öğelerini bir araya getiren uygulamalar oluşturabilirsiniz.

Temel anlamda bir form ve onu oluşturan bileşenler resim 1.1’de verildiği gibidir. Formlar WPF kontrolleri arasında bulunur ve kullanıcı ile iletişimde bulunacak denetimleri kapsamakla görevlidir.



Resim 1.1: Windows penceresi (Window)

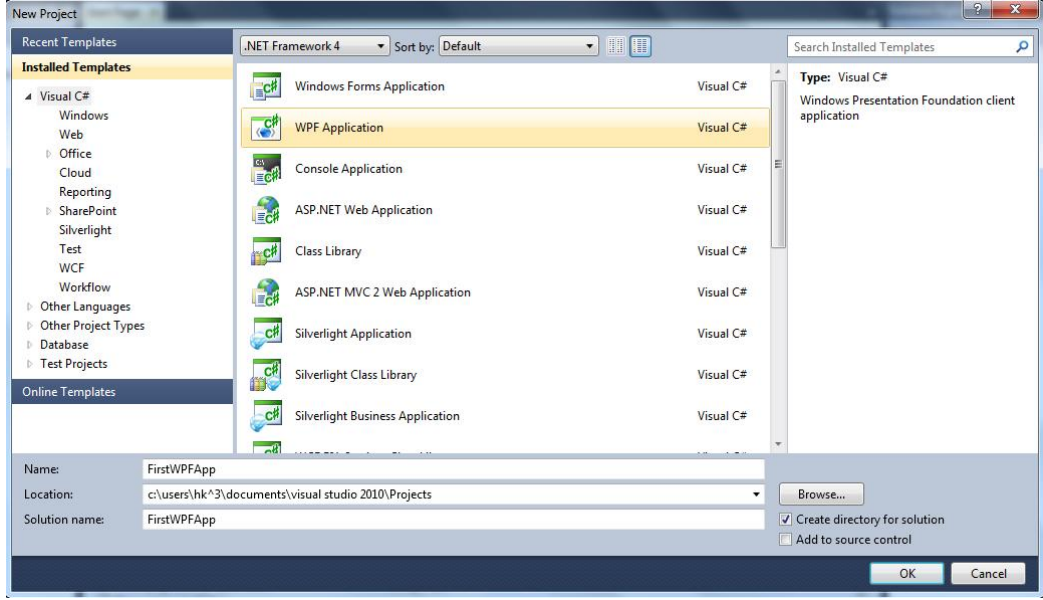
Tasarımda nihai amaç; ürün – kullanıcı entegrasyonunu başarı ile sağlayacak arayüzün işlevsel, kullanılabilir, görsel, açıdan kusursuz olmasını sağlamak olmalıdır. Bu sebepten yazılım üretim aşamasında teknik olarak, uygulama arayüzü programcılar olduğu kadar grafik tasarımcılar da karar sahibidir. Farklı çalışma disiplinlerine sahip bu iki geliştirici (programcılar ve tasarımcılar) çoğu zaman birlikte çalışmak zorunda kalabilirler. WPF ile yapılan arayüz çalışmalarında bu iki takım üyesinin birbirinden bağımsız çalışabilmesine olanak tanımak ve çalışma ortamı verimliliğini arttırmak için tasarım kodu “XAML” ve arayüze işlevselliği sağlayacak program kodu (C# veya VB) birbirinden ayrılmıştır.

Kullanıcı ile iletişime geçme ortamınız çoğu kez farklılık gösterebilir. Örneğin uygulamanız masaüstü bir yazılım olabileceği gibi mobil iletişim cihazları üzerinde çalışan (tablet bilgisayarlar, akıllı telefonlar) veya bir *internet* uygulaması şeklinde sunulabilir. Aralarında çok az farklılık bulunan bu platformlar için “XAML” tasarım odaklı ortak bir dil olma özelliğini geleceğe taşımaktadır. Bu sayede WPF ile geliştirdiğiniz program arayüzleri üzerinde çok fazla değişikliğe gitmeden tasarım kodlarınızı diğer ortamlara rahatlıkla taşıyabilirsiniz.

Bütün tasarımınızı “XAML” kodu yazarak oluşturmak mümkündür ancak bu oldukça zahmetli bir iştir. “Visual Studio” belirli bir ölçekte sürükleyip bırak tekniği ile tasarım ekranında çalışmanıza olanak verir. Ancak bu yeterli değildir. Bunun sebebi “Visual Studio” tasarım aracı değil uygulama geliştirme aracı olarak hizmet etmesidir. Dolayısıyla arayüz tasarımını daha etkin ve hızlı olarak üretebilmeniz adına “Expression Studio” adı verilen ve bir dizi tasarım aracı üretmiştir. Siz uygulama arayüzünüz için görsel tasarımınızı yaparken sizin adınıza gereken “XAML” kodunu uygulamanız için oluşturacaktır.

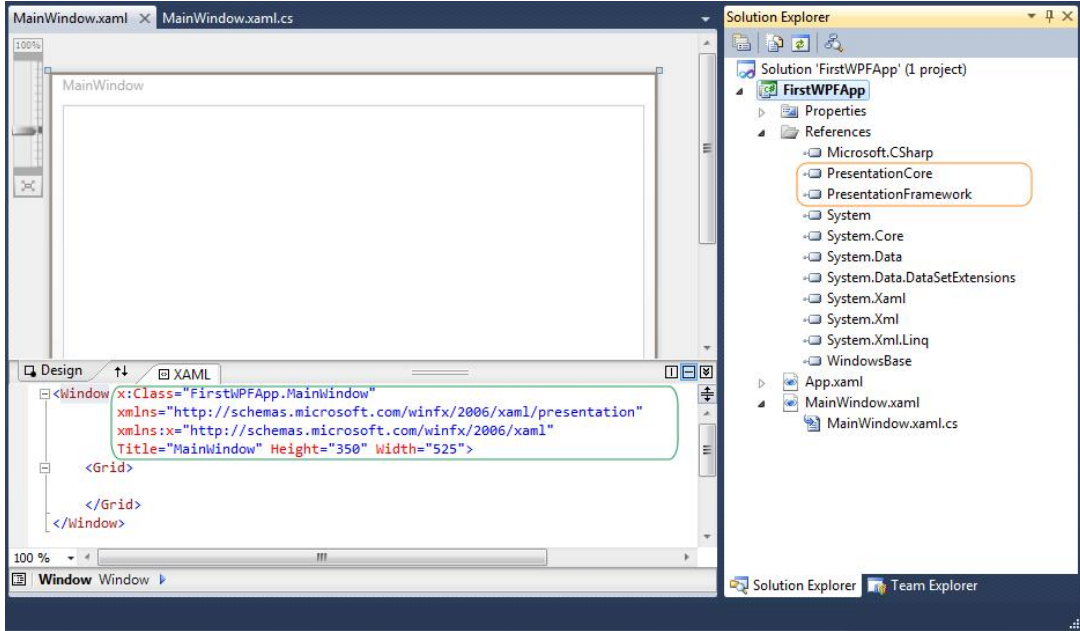
1.1. WPF Uygulaması Oluřturma

Bir WPF uygulaması bařlatmanız oldukça basittir. Öncelikle uygulama geliřtirme aracınız olan “Visual Studio” programını içinde dosya (File) menüsünden yeni bir proje (New Project) emri veriniz. Resim 1.2’de görüldüğü gibi ekrana gelen iletiřim penceresinden ”WPF Application” řablon seçimini yapın, projeniz için bir isim verin ve onaylayın.



Resim 1.2: “New Project” penceresi

Yeni bir WPF uygulaması oluřturduđunuzda Visual Studio sizin adınıza gereken referansları projeye ekleyecek ve tasarım yapmanıza olanak sađlayan arayüzü řablonunu resim 1.3’te görüldüğü gibi oluřturacaktır.



Resim 1.3: WPF tasarım ekranı

Öncelikle “Solution Explorer” penceresini incelediğinizde “App.xaml” ve “MainWindow.xaml” dosyalarının oluşturulduğunu görebilirsiniz. Öncelikle “App.xaml” dosyasına çift tıklayınız ve inceleyiniz.

```
<Application x:Class="FirstWPFApp.App"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  StartupUri="MainWindow.xaml">
  <Application.Resources>

  </Application.Resources>
</Application>
```

Uygulama ile ilgili tanımlamaların yer aldığı “App.xaml” dosyası “XAML” biçimli dosyadır. Uygulamayı çalıştırdığınızda, “App.xaml” içinde yer alan bildirimler derleyici tarafından “Application” nesnesine dönüştürülür. Application nesnesini işletim sistemi ile uygulama kodu arasındaki bir arayüz olarak da düşünebilirsiniz. “App.xaml” içinde tanımlanan “<Application>...</Application>” etiketinin “StartupUri” isimli parametresi ile uygulama nesnesi başlatıldığında başlangıç penceresi tasarımının hangi dosyada yer aldığı bilgisini işaret etmektedir. Bir WPF uygulaması birden fazla pencereden oluşabilir. Farklı bir pencere ile uygulamanın başlatılmasını isterseniz, bu durumda yapmanız gereken “StartupUri” parametresini değiştirmek olacaktır.

“MainWindow1.xaml” dosyasına çift tıkladığınızda tasarım ekranınız resim 1.3’te görüldüğü üzere ekranınıza gelecektir. İlki arayüz tasarımının ön izlemesi olan “design” bölümü diğeri ise tasarımı betimleyen “XAML” kodlarınızın yer aldığı bölümdür.

“Xaml” tasarım kodlarını inceleyiniz.

```
<Window x:Class="FirstWPFApp.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="MainWindow" Height="350" Width="525">
  <Grid>

  </Grid>
</Window>
```

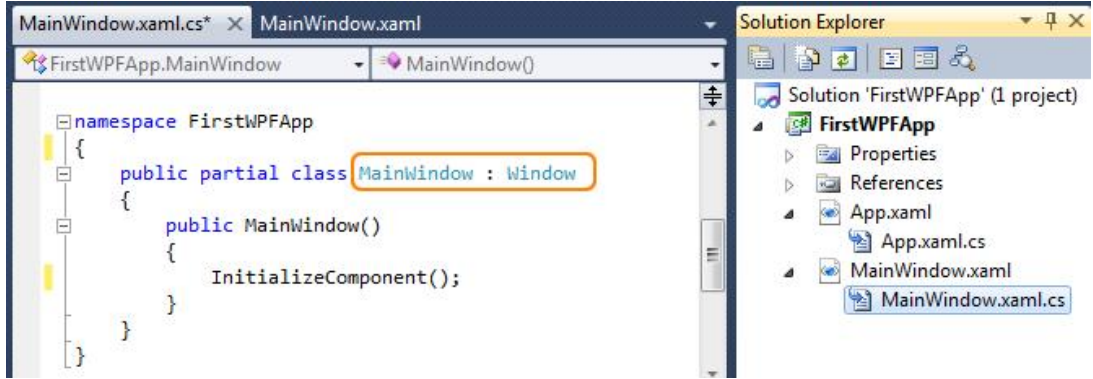
“XAML” dili “XML”in bir türevidir olduğundan XML yazım kuralları burada aynen geçerlidir. Tasarım kodları ile ilgili aşağıda yazılı kurallara dikkat ederseniz, okumanız ve yazmanız daha kolay hale gelecektir.

- Açılan etiketler kapatılmalıdır (Örneğin; <Grid>.....</Grid>).
- Gereken şemalar verilmelidir (xmlns...).
- Büyük-küçük harf ayırımına uyulmalıdır.
- Hiyerarşi korunmalıdır.
- Mutlaka bir tane kök düğüm bulunmalıdır (Örneğin; <Window>.....</Window>).

Uygulama şablonu içinde bulunan “MainWindow.xaml” dosyası “<Window>...</Window>” kök etiketleri başlatılır. Bunun anlamı tıpkı “Application” nesnesinde olduğu gibi bir “Window” nesnesini başlatacak olmasıdır. İlk “Window” etiketi kapatılmadan yapılmış “xmlns” deyimini ile “XAML” dosyası için kullanılacak hizmet sınıflarına ait isim alanı (namespace) bildirimleri yapılmaktadır. “x:” bildirimi, uygulamanın kendi sınıf bildirimini için yazılmış bir önektir. Sahip olduğu sınıfı “Class” bildirimini ile tanımlamıştır. Bu bildirim program kodunuzun tasarım kodu ile birlikte derlenebilmesi için önemlidir.

1.2. WPF Form Özellikleri

WPF şablonu üzerinde bulunan “MainWindow.xaml.cs” dosyasına ait resim 1.4’te gösterilen sınıf tanımını incelediğinizde tasarımında bulunacağınız pencerenin (MainWindow) “Window” sınıfından kalıtılmış olarak elde edildiğini görebilirsiniz.

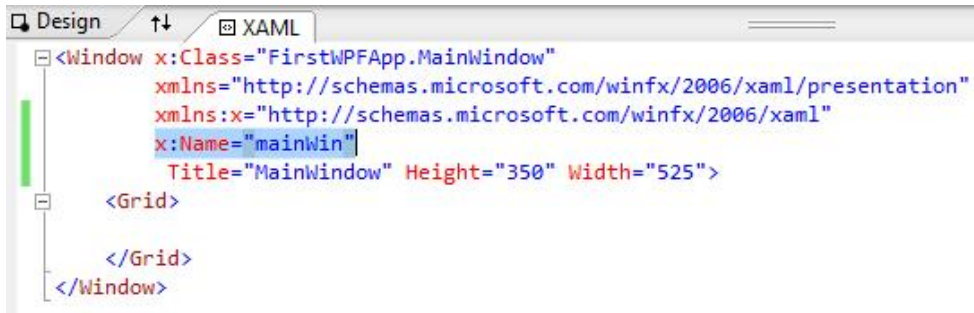


Resim 1.4: “MainWindow.xaml.cs” kod sayfası

“MainWindow” ile oluşturulmuş şablon pencereniz “Window” sınıfının tüm özelliklerini taşıyacaktır. Bu sayede gerek “XAML” tarafında gerekse kod sayfasında pencere ile ilgili işlemek istediğiniz özellikleri belirleyerek pencerenin durumlarını değiştirebilir, metotlarını kullanarak davranışları uygulayabilir, olaylarını işleyerek kullanıcı ile program kodunuzu iletişime geçirebilirsiniz.

Pencerelerin en sık kullanılan özelliklerini örnekleri üzerinden inceleyelim.

- **Name** : Tüm WPF kullanıcı denetimlerinin ortak özellikler arasındadır. Söz konusu nesnenin hafızada sahip olacağı isimdir.



Resim 1.5: “XAML” tasarım kodu ile özellik ataması

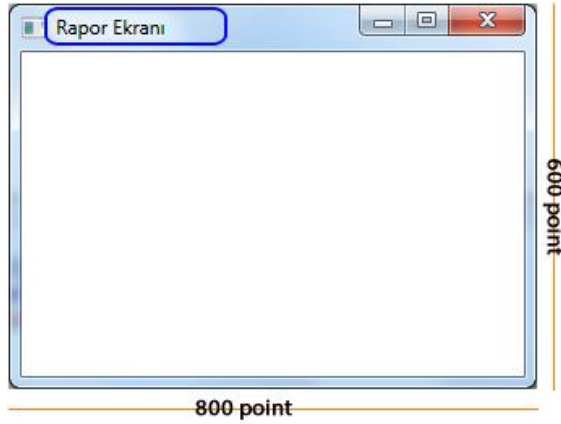
- **Title** : Pencerenin başlık çubuğunda bulunan bilgi metnidir.
- **Width** : Pencerenin sahip olduğu genişlik değeridir.
- **Height** : Pencerenin sahip olduğu yükseklik değeridir.

Örnek 1.1: Uygulama penceresinin başlık bilgisini “Rapor Ekranı”, yüksekliğini 600 point genişliğini 800 point olarak tasarlayınız.

```
<Window x:Class="FirstWPFApp.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  x:Name="mainWin" Title="Rapor Ekranı" Height="600" Width="800">
  <Grid>

  </Grid>
</Window>
```

Uygulamayı çalıştırıp test ediniz.



Resim 1.7: Uygulama test sonucu

- **MinWidth** : Pencerenin sahip olabileceği en az genişlik değeridir.
- **MinHeight** : Pencerenin sahip olabileceği en az yükseklik değeridir.
- **MaxWidth** : Pencerenin sahip olabileceği en fazla genişlik değeridir.
- **MaxHeight** : Pencerenin sahip olabileceği en fazla yükseklik değeridir.
- **ResizeMode** : Pencerenin yeniden boyutlandırılabilir olma özelliğini ayarlar.

Örnek 1.2: Uygulama penceresi, kullanıcı tarafından yeniden boyutlandırılmaya çalışıldığında en az "300 * 500" en fazla "600 * 800" değerlerini geçmemesi tasarlanmak istenmektedir.

```
<Window x:Class="FirstWPFApp.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  x:Name="mainWin" Title="Rapor Ekranı" Height="350" Width="400"
  ResizeMode="CanResize" MinWidth="500" MinHeight="300"
  MaxWidth="800" MaxHeight="600">
  <Grid>

  </Grid>
</Window>
```

Uygulamayı çalıştırdığımız ve ekrana gelen pencerenin sağ alt köşesinden fare yardımıyla yeniden boyutlandırmayı deneyiniz.

- **Background :** Pencerenin zemin rengini belirler.

Örnek 1.3: Uygulama pencere zemin rengini mavi olarak tasarlayınız.

```
<Window x:Class="FirstWPFApp.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  x:Name="mainWin" Title="Rapor Ekranı" Height="350" Width="400"
  Background="Blue">
  <Grid>

  </Grid>
</Window>
```

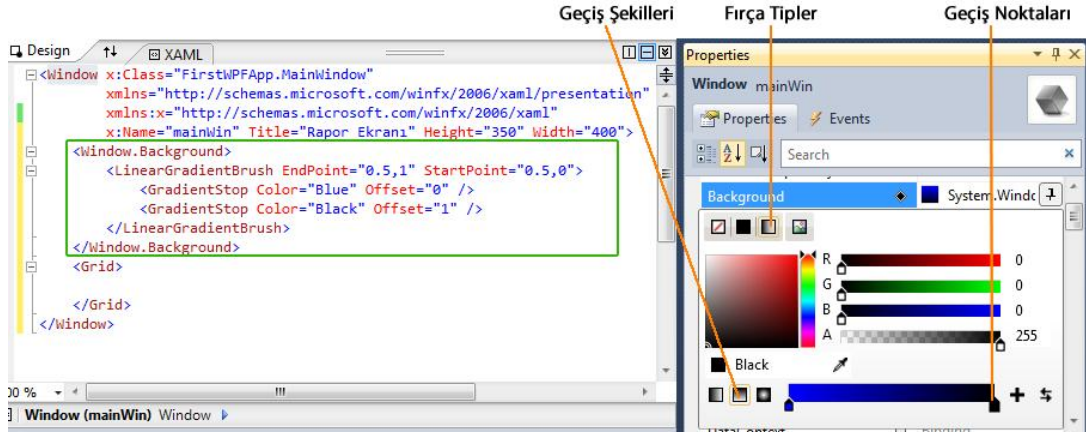
Örnek 1.4: Uygulama pencere zemin rengini maviden siyaha doğru renk geçişi olacak şekilde tasarlayınız.

```
<Window x:Class="FirstWPFApp.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  x:Name="mainWin" Title="Rapor Ekranı" Height="350" Width="400">
  <Window.Background>
  <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
    <GradientStop Color="Blue" Offset="0" />
    <GradientStop Color="Black" Offset="1" />
  </LinearGradientBrush>
  </Window.Background>
  <Grid>
  </Grid>
</Window>
```

Yukarıda bulunan “XAML” kodunu incelediğinizde “Window” sınıfına ait “Background” özelliğinin <Window.Background>...</Window.Background> etiketleri arasında tanımlandığını görebilirsiniz. Nesnelere ait özelliklerin ayrı bir etiket ile düzenlenebiliyor olması “XAML” ile ne kadar esnek tasarımların yapılabileceğinin bir göstergesidir. Elde edilmek istenen geçişli renkler, “LinearGradientBrush” sınıfı altında belirlenir. Bu sınıf WPF için boyama yapabilen bir fırçadır ve renk geçiş noktaları olarak “GradientStop” sınıflarının “Color” özelliğine ihtiyaç duyar.

Tek bir boyama işlemi için bu kadar fazla “XAML” kodu tanımlanıyor olması oldukça detaylı görülebilir. Burada “XAML” kodunun üretimini tasarım araçlarına bırakmak

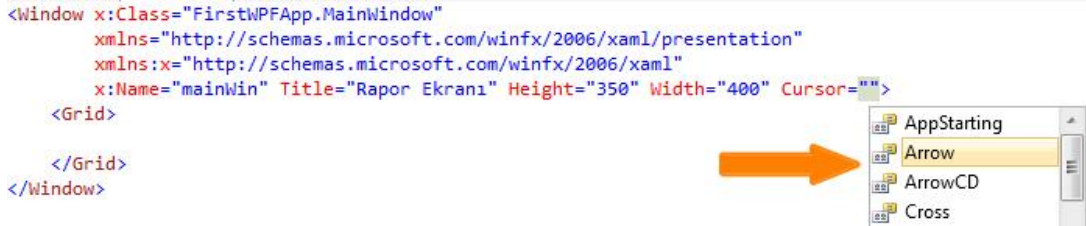
programcı için daha hızlı ve pratik bir yol olacaktır. Bunu gerçekleştirebilmek için tasarım aracınızın “Properties” panelini inceleyiniz.



Resim 1.8: Properties paneli ve arka plan renk özelliği

Seçmiş olduğunuz nesne ile ilgili özellikleri “Properties” panelinde bulabilirsiniz. Burada yapılan tüm değişiklikler tasarım aracınız tarafından “XAML” koduna dönüştürülecektir.

- **Cursor:** Pencere üzerinde fare (mouse) görünümünü belirler. İşletim sistemine ait fare işaretçilerini tanımlama yaparken resim 1.9’da olduğu gibi görebilir ve uygun olanı seçebilirsiniz.

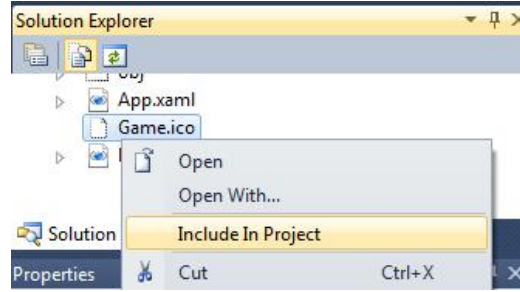


Resim 1.9: Kod yazım yardımcısı (Intellisense) desteği

- **Icon :** Pencerenin başlık çubuğu üzerinde ve işletim sisteminin görev çubuğunda görünen ikon resmini belirler.

Örnek 1.5 : Uygulamanıza ikon ekleyiniz.

Öncelikle uygulamanız için tasarlanan ikon dosyasını projenize dahil etmelisiniz. İkon dosyasını projenizin yer aldığı klasörün içine yerleştirin ardından “Solution Explorer” üzerinde dosyayı seçin ve fare sağ tuşa basınız. Açılan menüden “Include Project” seçimini resim 1.10’da görüldüğü gibi yapınız.



Resim 1.10: Projeye dosya dahil etme işlemi

“Icon” özelliğini “Properties” panelinden bulunuz ve resim seç (Choose Image) butonuna tıklayınız. Ekrana gelen resim seçme iletişim kutusundan projenize dahil etmiş olduğunuz ikon dosyasını seçiniz ve onaylayınız. Pencerenizin “XAML” kodunda “Window” nesnesinin “icon” özelliğine dosya bilgisinin atandığını görebilirsiniz.

```
<Window x:Class="FirstWPFApp.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="MainWindow" Height="350" Width="525"
  Icon="/FirstWPFApp;component/Game.ico">
  <Grid>

  </Grid>
</Window>
```

Uygulamanızı çalıştırınız ve test ediniz.

- **WindowStyle** : Pencere stil tanımlamasıdır.
- **WindowState** : Pencerenin simge durumunda veya ekranı kaplamış olarak boyutlamasını sağlar.
- **WindowStartupLocation** : Pencerenin ekran üzerindeki ilk açılış konumunu belirler.
- **Visibility** : Pencerenin görünürlüğünü belirler.

Örnek 1.6: Uygulama penceresi ekranı kaplayan araç kutusu şeklinde tasarlayınız.

```
<Window x:Class="FirstWPFApp.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="MainWindow" Height="350" Width="525"
  WindowStyle="ToolWindow" WindowState="Maximized">
  <Grid>

  </Grid>
</Window>
```


Örnek 1.7: Uygulama penceresi 300 *400 point büyüklüğünde ve ekranın tam orta noktasına konumlanacak şekilde tasarlayınız.

```
<Window x:Class="FirstWPFApp.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="MainWindow" Height="300" Width="400"
  WindowStartupLocation="CenterScreen">
  <Grid>

  </Grid>
</Window>
```

- **ShowInTaskBar** : Uygulama penceresi simgesinin işletim sistemi görev çubuğu üzerinde görüntülenmesini tayin eder.
- **Topmost** : Uygulama penceresinin diğer uygulama pencerelerine göre daima önünde kalmasını sağlar.

Örnek 1.8: Uygulama penceresi simgesi görev çubuğunda görünmeden diğer uygulamalara göre en önde kalması istenmektedir. Gereken tasarımı yapınız.

```
<Window x:Class="FirstWPFApp.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="MainWindow" Height="300" Width="400"
  ShowInTaskbar="False" Topmost="True">
  <Grid>

  </Grid>
</Window>
```

- **Tooltip**: Pencere üzerinde fare beklemesinin ardından açılan balon gösterimine ait metni belirler.

Örnek 1.9: Uygulama penceresinde açılacak balon bilgisi üzerinde uygulamanın adını göstermesi için gereken tasarımı yapınız.

```
<Window x:Class="FirstWPFApp.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="Benim Uygulamam" Height="300" Width="400"
  ToolTip="Benim Uygulamam">
  <Grid>

  </Grid>
</Window>
```

- **Left:** Pencerenin açıldığı ekran üzerinde soldan belirlenen değer kadar mesafede konumlanmasını sağlar.
- **Top:** Pencerenin açıldığı ekran üzerinde üstten belirlenen değer kadar mesafede konumlanmasını sağlar.
- **BorderThickness:** Pencereye verilen değer kalınlığında kenarlık oluşturur.
- **BorderBrush:** Pencereye çizdirilen kenarlığın rengini tayin eder.
- **AllowTransparency:** Pencerenin şeffaf olmasına izin verir.
- **Opacity:** Pencerenin şeffaflık oranını belirler. Alabileceği değer 0 ile 1 arasındadır.

Örnek 1.10: Uygulama penceresini %50 oranında şeffaflaştırıp, sistem kenarlıklarını kaldırılarak turuncu renk kenarlığa sahip özel bir pencere tasarlayınız.

```
<Window x:Class="FirstWPFApp.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="Benim Uygulamam" Height="300" Width="400"
  BorderThickness="4" BorderBrush="Orange" WindowStyle="None"
  AllowsTransparency="True" Opacity="0.5">
  <Grid>

  </Grid>
</Window>
```

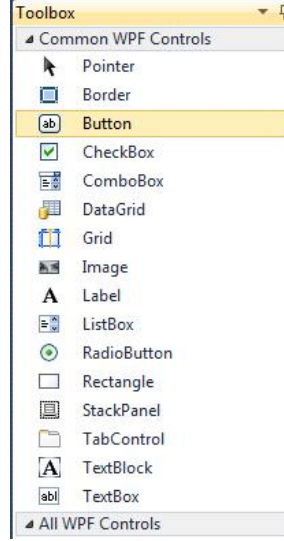
- **Tag:** Pencere üzerinde herhangi bir etkisi yoktur. "Object" tipinde bir özellik olup herhangi bir bilgiyi nesne ile birlikte tutabilme ihtiyacını karşılamak için kullanılır.

1.3. Nesnelere

WPF uygulama arayüzü için pencerelerin (window) yanında pek çok kullanıcı denetimini de kendi çatısı altında sunmaktadır. Pencereler, içeriğinde kullanıcı denetimlerini barındıran konteynerler gibi davranır. Ancak uygulama arayüzünü asıl oluşturan ve anlamlı

hale getiren bu denetimlerdir. Sonuç olarak kullanıcı bu denetimleri kullanarak programınızla etkileşime geçecek ve amacı doğrultusunda kullanabilecektir.

Denetimleri “Visual Studio” ortamında “Toolbox” isimli araç kutusunda bulabilirsiniz. Resim 1.11’de araç kutusu içinde kullanabileceğiniz denetimleri görmekteyiz.



Resim 1.11: Araç kutusu (Toolbox) paneli

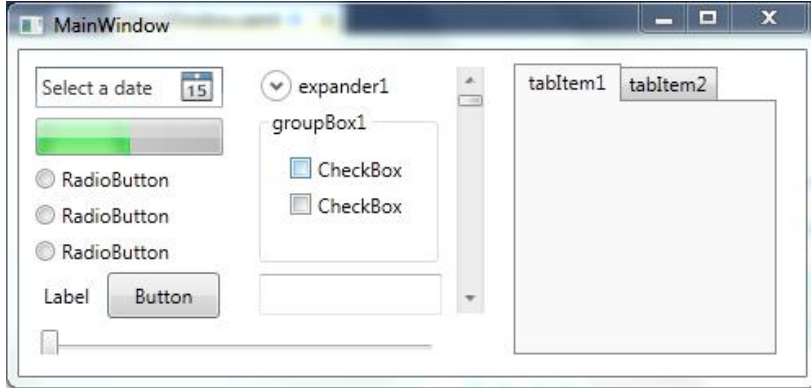
WPF çatısı altında oldukça fazla denetim bulunmaktadır. Bu denetimleri ve genel amaçları aşağıda verildiği şekliyle sıralayabilirsiniz.

- **Düğmeler:** Bir eylemi yerine getirmek için tıklamak suretiyle kullanılan denetimdir. WPF için düğmeler “Button” ve “RepeatButton” nesnelere tarafından kullanılır.
- **İletişim kutuları:** Yazdırma, dosya açma, kaydetme gibi standart kabul edilen pencerelerdir. İletişim kutuları OpenFileDialog, PrintDialog ve SaveFileDialog sınıfları tarafından sunulur.
- **Dijital mürekkep:** InkCanvas ve InkPresenter uygulamalarınız içinde kullanıcının serbest çizim yapmasına olanak veren denetimlerdir.
- **Belgeler:** Raporlama aracı olarak kullanılan belge tipleri ile aynı zamanda yazdırma hizmetini de sunar. WPF için belge nesnelere DocumentViewer, StickyNoteControl, FlowDocumentPageViewer, FlowDocumentReader, FlowDocumentScrollViewer, ve sınıfları tarafından sunulur.
- **Giriş kutuları:** Kullanıcıdan bilgi almak amaçlı kullanılan denetimlerdir. TextBox, RichTextBox ve PasswordBox giriş kutusu sınıflarıdır.
- **Düzen:** Pencere içi yerleşimler düzen denetimleri tarafından yapılır. Border, BulletDecorator, Canvas, DockPanel, Expander, Grid, GridView, GridSplitter, GroupBox, Panel, ResizeGrip, Separator, ScrollBar, ScrollViewer, StackPanel, Thumb, Viewbox, VirtualizingStackPanel, Window ve WrapPanel düzen sağlayan denetim sınıflarıdır.

- **Resim ve medya:** Resim ve video görselleri pencere içinde kullanmanızı sağlar. Image, MediaElement ve SoundPlayerAction sınıfları ile medya denetimlerini kullanabilirsiniz.
- **Menüler:** ContextMenu, Menu ve ToolBar birden fazla komutu bir düzen halinde yapısında sunan denetimlerdir.
- **Gezinme:** Frame, Hyperlink, Page, NavigationWindow, ve TabControl. sayfa içi gezinti ve sayfa geçişleri amaçlı kullanılan denetimlerdir.
- **Seçim:** CheckBox, ComboBox, ListBox, TreeView ve RadioButton, Slider birden fazla seçim imkanı veya liste türü gösterimler için kullanılan denetimlerdir.
- **Kullanıcı bilgileri:** AccessText, Label, Popup, ProgressBar, StatusBar, TextBlock ve ToolTip kullanıcıya bilgi göstermek amaçlı kullanılan denetimlerdir.

“Visual Studio” ortamında tasarım ekranında uygulamanız için kullanmak istediğiniz denetimi araç kutusundan seçip sürükleyip bırak yöntemi ile pencere içine yerleştirebilirsiniz.

Örnek 1.11: Resim 1.12’de verilen ekran tasarımını araç kutusundan sürükleyip bırak yöntemiyle gerçekleştiriniz.



Resim 1.12: Uygulama arayüzü

1.4. Nesne Özellikleri

Kullanıcı denetimleri, tasarım zamanında “XAML” tarafında statik olarak özellikleri uygulanabilir. Bu sayede denetimleri uygulama arayüzü için özelleştirebilirsiniz.

“XAML” ile bir nesnenin özellik ataması sahip olduğu etiketler içinde yapılır.

<Nesne özellik="değer" özellik="değer" ... />

Öncelikle söz konusu denetimleri ve sahip oldukları özellikleri yakından tanıyın.

- **Button:** Komut vermek için kullanılan düğmelerdir. “<Button .../>” etiketi ile tanımlanır. Buton nesnesine ait genel özellikler tablo 1.1’de verilmiştir.

Özellik	Açıklama
Name	Denetimi temsil eden isimdir
Content	Denetime ait içerik metnidir.
Width	Genişlik değeridir.
Height	Yükseklik değeridir.
Left	Denetimin kapsandığı sınırlardan soldan mesafesinin değeridir.
Top	Denetimin kapsandığı sınırlardan üstten mesafesinin değeridir.
VerticalAlignment	Dikey hizalanma durumunu belirler.
HorizontalAlignment	Yatay hizalanma durumunu belirler
BackGround	Arkaplan rengini tayin eder.
ForeGround	Yazı rengini tayin eder.
FontFamily	Yazı tipini belirler.
FontSize	Yazı tipi yüksekliğini belirler.
Cursor	Fare işaretçisi denetimin üzerinde iken işaretçinin şeklini belirler.
IsCancel	Denetimin iptal fonksiyonu olma özelliği sağlar.
IsDefault	Denetimi onay fonksiyonu olma özelliği sunar.
Margin	Denetimin kapsandığı sınırlar içinde sol, üst, sağ, alt noktalardan olan uzaklığını belirler.
Tooltip	Fare işaretçisi denetimin üzerinde açılan balon yardım metnini belirler.
Visibility	Görünürlük durumunu değiştirir.

Tablo 1.1: Denetime ait özellikler

Örnek 1.12: Uygulama penceresine “Tamam” ve “İptal” butonları yerleştiriniz.

Araç kutusundan “buton” nesnesini pencere denetiminin içine sürükleyip bırakınız. İlk butonun “Content” özelliğine “Tamam” diğerine “İptal” metnini yazınız. Mevcut tasarımı oluşturan “XAML” kodu aşağıda verildiği şekildedir.

```
<Window x:Class="UIControls.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="MainWindow" Height="350" Width="525">
  <Grid>
    <Button Content="Tamam" Margin="0,0,12,12" Name="button1"
      Height="54" VerticalAlignment="Bottom"
      HorizontalAlignment="Right" Width="103" />
    <Button Content="İptal" Margin="0,0,121,12" Name="button2"
      HorizontalAlignment="Right" Width="103"
      Height="54" VerticalAlignment="Bottom"/>
  </Grid>
</Window>
```

- **Label:** Bilgi görüntülemek için kullanılan etiket türü denetimlerdir. “<Label .../>” etiketi ile tanımlanır.

Nesne içeriğinde verilmek istenen bilgi “Content” özelliği ile düzenlenir.

```
<Label Content="Birim Fiyat" .../>
```

- **Textbox:** Bilgi girişinin yapılabildiği metin kutusudur. “<TextBox .../>” etiketi ile tanımlanır.

Metin kutuları (TextBox) kullanıcıların programa girdi sağlamak amacıyla en fazla kullandıkları denetimdir. Bu girdi bilgisine denetimin “Text” özelliği ile erişilir.

```
<TextBox Text="Kullanıcı Adını Giriniz"></TextBox>
```

- **PasswordBox:** Bilgi girişinin, gizli olarak yapılabildiği metin kutusudur. “<PasswordBox>” etiketi ile kullanılır.

Örnek 1.13: Kullanıcı adı ve şifre parametrelerinin sorulduğu uygulama giriş ekranı tasarlayınız.

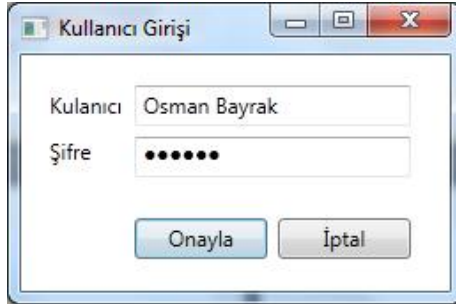
Uygulama arayüzü içinde kullanıcının adını girebileceği “Textbox”, şifresini gizli olarak yazabilmesi için “PasswordBox”, işleminin onaylaması veya iptal edebilmesi için iki adet “Button”, ilgili açıklamalar için “Label” nesnelerini pencere içine yerleştiriniz. Ardından aşağıda tasarım kodu verildiği şekilde nesne özelliklerini ayarlayınız.

```

<Window x:Class="UIControls.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="Kullanıcı Girişi" Height="171" Width="258">
  <Grid>
    <TextBox Height="23" HorizontalAlignment="Left" Margin="64,17,0,0"
      Name="txtAd" VerticalAlignment="Top" Width="156" />
    <PasswordBox Height="23" HorizontalAlignment="Left" Margin="64,46,0,0"
      Name="sifre" VerticalAlignment="Top" Width="156" />
    <Button Content="Onayla" Height="23" HorizontalAlignment="Left"
      Margin="64,92,0,0" Name="onay" VerticalAlignment="Top" Width="75"/>
    <Button Content="İptal" Height="23" HorizontalAlignment="Left"
      Margin="145,92,0,0" Name="iptal" VerticalAlignment="Top" Width="75" />
    <Label Content="Kullanıcı" Height="28" HorizontalAlignment="Left"
      Margin="11,15,0,0" Name="label1" VerticalAlignment="Top" />
    <Label Content="Şifre" Height="28" HorizontalAlignment="Left"
      Margin="11,41,0,0" Name="label2" VerticalAlignment="Top" />
  </Grid>
</Window>

```

Uygulamayı çalıştırmız ve test ediniz.



Resim 1.13: Giriş ekranı tasarımı

- **ListBox:** Liste kutusudur. Liste türü yapıların gösterimini sağlar. Aynı zamanda kullanıcının listeden seçim yapmasına olanak verir. Liste kutuları “<ListBox>...</ListBox>” etiketleri arasında tanımlanır. Sahip olduğu öğeler “<ListBoxItem>” etiketi ile eklenir.

```

<ListBox>
  <ListBoxItem Content="Öğe 1" />
  <ListBoxItem Content="Öğe 2" />
  <ListBoxItem Content="Öğe 3" />
</ListBox>

```

- **ComboBox:** Açılır liste kutusudur. “<ComboBox>...</ComboBox>” etiketleri arasında tanımlanır. Sahip olduğu öğeler “<ComboBoxItem>” etiketi ile eklenir.

```
<ComboBox>
  <ComboBoxItem Content="Seçenek 1"/>
  <ComboBoxItem Content="Seçenek 1"/>
  <ComboBoxItem Content="Seçenek 1"/>
</ComboBox>
```

- **RadioButton:** Seçenek düğmeleri, iki ya da daha fazla seçenek arasından bir seçim yapmanızı sağlar. “<RadioButton .../>” etiketi ile tanımlanır. “IsChecked” özelliği ile onay durumu belirlenebilir.

```
<RadioButton Content="Tercih" IsChecked="True"></RadioButton>
```

- **CheckBox:** Onay kutuları, bir ya da daha fazla bağımsız seçenek belirlemenizi sağlar. “<CheckBox .../>” etiketi ile tanımlanır. “IsChecked” özelliği ile onay durumu belirlenebilir.

```
<CheckBox Content="Seçim Bilgisi" IsChecked="True"></CheckBox>
```

- **Image:** Resim gösterimi için kullanılan denetimdir. “<Image .../>” etiketi ile tanımlanır. Resim kaynağı “Source” özelliği ile belirlenir. Resmin denetim üzerine sığdırmak için “Stretch” özelliği kullanılır.

```
<Image Stretch="Fill" Source="/WpfApp3;component/Images/1.jpg" />
```

- **Media:** Video oynatımı için kullanılan denetimdir. “<Media .../>” etiketi ile tanımlanır. Media kaynağı “Source” özelliği ile belirlenir.

```
<MediaElement Source="video.mp4"...></MediaElement>
```

“MediaElement” nesnesi ile yürütülmek istenen video üzerinde dilediğiniz bir konuma ulaşmak için “Position”, yürütüm hızını belirleyebileceğiniz “SpeedRatio” ve ses seviyesini ayarlayabileceğiniz “Volume” özelliklerini kullanabilirsiniz.

- **DatePicker:** “ComboBox” bileşeni şeklinde açılabilen ve içeriğinde bulunan takvimden tarih seçimi yapmanıza olanak sunan denetimdir. “<DatePicker .../>” etiketi ile tanımlanır. Denetimin “Text” özelliği seçimi yapılan tarih bilgisini vermektedir. “DisplayDate” özelliği ile denetimin işaret ettiği tarihi, “SelectedDateFormat” özelliği ile tarih formatı belirlenmektedir.


```
<DatePicker SelectedDateFormat="Long"... />
```

- **Slider:** Kaydırıcı, bir ayarı bir değer aralığında düzenlemenizi sağlar. “<Slider.../>” etiketi ile tanımlanır. Denetim “Maximum” ve “Minimum” özellikleri aralığında görsel olarak ifade ettiği değeri “Value” özelliği ile elde edilir.

```
<Slider Maximum="100" Minimum="0" Value="50"></Slider>
```

- **ProgressBar:** Uygulama içinde zaman alan bir eylem olduğunda, eylemin tamamlanma sürecini gösteren denetimdir. “<ProgressBar .../>” etiketi ile tanımlanır. Denetim “Maximum” ve “Minimum” özellikleri aralığında görsel olarak ifade ettiği değeri “Value” özelliği ile elde edilir.

```
<ProgressBar Minimum="0" Maximum="100" Value="40" Orientation="Horizontal">
```

WPF uygulamalarında kullanılan ekranlara ait denetimlerin mutlaka bir “Layout” bileşeni içerisinde konuşlandırılmış olmaları gerekmektedir. Layout bileşenleri temelde birer panel olarak düşünülmelidir. Her bir ”Layout” bileşeni içeriğinde taşıdığı denetimleri kendine özgü düzen ve yerleştirme stiline sahiptir. Grid, StackPanel, WrapPanel, Canvas WPF tasarımlarında sıklıkla kullanılan başlıca düzen bileşenleri arasındadır.

- **Grid:** Satır ve sütun bazında yerleşime olanak veren düzen bileşenidir. “Grid” bileşeninde satırlar ve sütunlar geliştirici tarafından daha detaylı bir şekilde ayarlanır. “Grid” bileşeni içerisinde yer alacak elementlerin hangi hücrelere geleceğini belirlemek için eklenmiş özellik (Attached Property) tekniğinden yararlanılmaktadır. Buna göre “Row” ve “Column” özelliklerine atanan değerler ile yerleşim hücresi belirlenir. “Span” özelliği ile denetimin satırda veya sütunda kaç adet hücre içerisine yerleştirileceği bildirilir.

Örnek 1.14: Bir diyalog kutusu tasarımını resim 1.14’te verilen şekliyle oluşturunuz.



Resim 1.14: Uygulama tasarımı

```

<Window x:Class="Grid.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="MainWindow" Height="350" Width="525">
  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition Height="*"></RowDefinition>
      <RowDefinition Height="50"></RowDefinition>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="*"></ColumnDefinition>
      <ColumnDefinition Width="60"></ColumnDefinition>
      <ColumnDefinition Width="60"></ColumnDefinition>
    </Grid.ColumnDefinitions>
    <TextBox Margin="10" Grid.Row="0" Grid.Column="0"
      Grid.ColumnSpan="3" Background="Orange"/>
    <Button Margin="5" Grid.Row="1" Grid.Column="1"
      Padding="3" Content="Tamam"/>
    <Button Margin="5" Grid.Row="1" Grid.Column="2"
      Padding="3" Content="İptal"/>
  </Grid>
</Window>

```

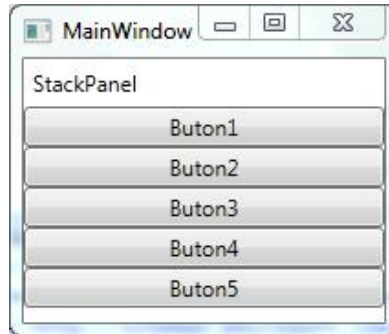
“Grid” içerisindeki satırları belirlemek için “RowDefinitions”, sütunları belirlemek içinse “ColumnDefinitions” kullanılır. Satırlar için yükseklik değeri “Height” özelliği ile sütunlar için genişlik değeri “Width” özelliği ile belirlenir. Dikkat çekici noktalardan birisi de “Auto” ve “*” kullanımınıdır. “Auto” ifadesine göre yükseklik veya genişlik değeri içerideki denetimin boyutuna göre otomatik olarak ayarlanır. Diğer taraftan “*”, kalan tüm mesafeyi kullan anlamında düşünülebilirsiniz.

- **StackPanel** : İçerisine eklenen denetimleri yatayda (vertical) veya dikeyde (horizontal) hizalama yaparak gösterimini sağlar. Yön bilgisi “Orientation” özelliği ile belirlenir. Hizalanması gereken denetimler “<StackPanel>...<StackPanel/>” etiketleri arasına yerleştirilir.

Örnek 1.15: Tasarımda bulunan denetimleri dikey olarak hizalayınız.

```
<Window x:Class="StackPanel.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="MainWindow" Height="350" Width="525">
  <StackPanel>
    <Label Content="StackPanel"></Label>
    <Button Content="Buton1"></Button>
    <Button Content="Buton2"></Button>
    <Button Content="Buton3"></Button>
    <Button Content="Buton4"></Button>
    <Button Content="Buton5"></Button>
  </StackPanel>
</Window>
```

Uygulamayı çalıştırınız ve denetim yerleşimini resim 1.15'te olduğu gibi gözlemleyiniz.



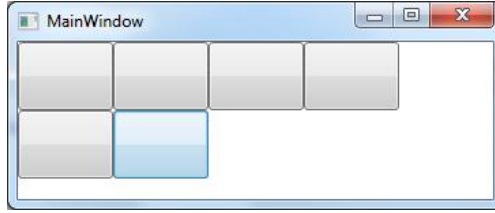
Resim 1.15: “StackPanel” düzeni

- **WrapPanel:** Kapsadığı denetimleri kendi boyutuna göre yatayda veya dikeyde toplar. “StackPanel” bileşeninden farklı olarak bir yönde sığdıramadığı denetimleri diğer yönde hizalamaya devam ettirmesidir.

Örnek 1.16: Basit bir araç kutusu tasarımı yapınız.

```
<Window x:Class="wrapSample.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="MainWindow" Height="350" Width="525">
  <WrapPanel>
    <Button Width="70" Height="50"></Button>
    <Button Width="70" Height="50"></Button>
    <Button Width="70" Height="50"></Button>
    <Button Width="70" Height="50"></Button>
    <Button Width="70" Height="50"></Button>
    <Button Width="70" Height="50"></Button>
  </WrapPanel>
</Window>
```

Uygulamayı çalıştırınız ve pencereyi yeniden boyutlandırarak “WrapPanel” davranışını resim 1.16’da görüldüğü gibi gözlemleyiniz.



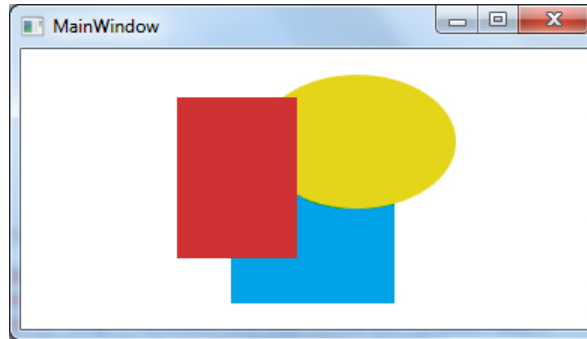
Resim 1.16: Uygulama test sonucu

- **Canvas:** Denetimlerin konumlarını sahip olduğu sınırlardan mesafesi belirtilerek yerleşimini sağlar. Mesafe değerleri üstten (Top), soldan (Left), alttan (Bottom) ve sağdan (Right) özellikleri ile düzenlenir. Varsayılan olarak bu değerler belirtilmediği takdirde bileşen 0,0 noktasına konumlandırılmaktadır. Bir başka deyişle “Canvas” bileşeninin sol üst köşesine yanaştırılmaktadır.

Örnek 1.17: Canvas bileşeni içine farklı konumlara sahip üç geometrik şekil yerleştiriniz.

```
<Window x:Class="CanvasLayout.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="MainWindow" Height="350" Width="525">
  <Grid>
    <Canvas>
      <Rectangle Canvas.Left="104" Canvas.Top="32" Height="107" Width="80"
        Fill="#FFCE3232" Panel.ZIndex="1" />
      <Rectangle Canvas.Left="140" Canvas.Top="80" Height="89" Width="109"
        Fill="#FF003A96" />
      <Ellipse Canvas.Left="158" Canvas.Top="17" Height="89" Width="132"
        Fill="#FFE5D51A" />
    </Canvas>
  </Grid>
</Window>
```

Örnek uygulama için “Canvas” içine yerleştirilen dikdörtgen (Rectangle) ve daire (Ellipse) şekillerinin yerleşimi “Canvas.Top” ve “Canvas.Left” özellikleri ile konumlandırılmıştır. Görünüm açısından örtüşen şekillerin hangisinin en üstte olacağı “Panel.ZIndex” özelliği ile belirlenmiş olmasına dikkat ediniz. Diğer şekillerin “Panel.ZIndex” özelliğine farklı değerler verdiğinizde görünümün nasıl bir değişiklik göstereceğini izleyiniz. Uygulamayı test ettiğinizde resim 1.17’de görülen sonuca ulaşacaksınız.



Resim 1.17: Uygulama sonuç ekranı

1.5. Özellikleri Dinamik Olarak Değiştirme

Şimdiye kadar, özellikleri statik olarak değiştirmek için tasarım görünümünden ve XAML deyimlerinden faydalandınız. Uygulama penceresi üzerinde yer alan denetimlerinizin özelliklerini çalışma anında belirlemek değiştirmek veya değerini kullanmak isteyebilirsiniz. Bunu sağlamak için öncelikle söz konusu denetimlerin mutlaka “Name” parametresi ile

belirlenmiş bir ismi olmalıdır. Kod sayfasında ilgili denetime ismi üzerinden ulaşılarak gerek gördüğünüz değerlere ulaşabilir, değiştirebilirsiniz.

Örnek 1.18: Kullanıcı bilgilerinin girilmesi istenen uygulama ekranında bulunan denetimlerin içeriğini temizleyen metodu oluşturunuz.

Öncelikle uygulama ekranı tasarımını yapınız. Kullandığınız denetimlerin “Name” özelliği ile isimlerini veriniz.



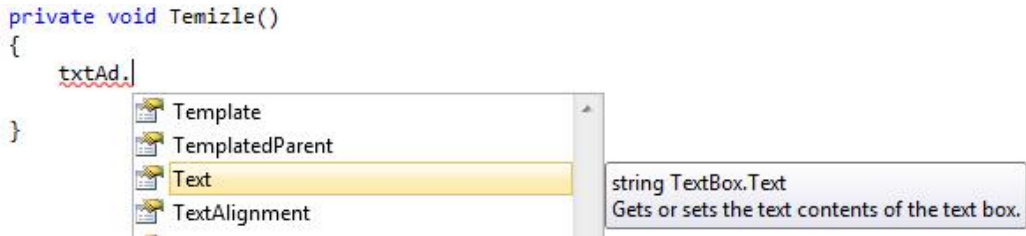
```
<Window x:Class="Dinamik.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="Kayıt İşlemleri" Height="206" Width="310">
<Grid>
<Label Content="Kullanıcı Adı" Margin="40,27,0,0"
HorizontalAlignment="Left" VerticalAlignment="Top"/>
<Label Content="Adres" Margin="40,58,0,0"
HorizontalAlignment="Left" VerticalAlignment="Top"/>
<Label Content="Kayıt Tarihi" Margin="40,89,0,0"
HorizontalAlignment="Left" VerticalAlignment="Top"/>
<CheckBox Name="chBilgi" Content="Kabul Ediyorum" Margin="48,130,0,0"
HorizontalAlignment="Left" VerticalAlignment="Top" />
<TextBox Name="txtAd" Margin="117,27,0,0" HorizontalAlignment="Left"
VerticalAlignment="Top" Height="23" Width="120" />
<TextBox Name="txtAdres" Margin="117,60,0,0" HorizontalAlignment="Left"
VerticalAlignment="Top" Height="23" Width="120" />
<DatePicker Name="dtKayit" Margin="117,90,0,0" HorizontalAlignment="Left"
VerticalAlignment="Top" Height="23" Width="120" />
</Grid>
</Window>
```

Resim 1.18: Uygulama tasarımı

“MainWindow.xaml.cs” dosyasını açınız ve “Temizle” isimli bir metod oluşturunuz. Metod gövdesinde içerik bilgisini değiştireceğiniz denetimleri isimleri üzerinden özelliklerine gereken değer atamalarını yapınız. Bu konuda kod yazım yardımcınız (Intellisense) resim 1.19’da görüldüğü gibi size destek verecektir.

```
private void Temizle()
{
    txtAd.
}

```



Template
TemplatedParent
Text
TextAlignment

string TextBox.Text
Gets or sets the text contents of the text box.

Resim 1.19: “Name” özelliği üzerinden ilgili nesneye kod sayfasından ulaşım

```
MainWindow.xaml.cs* x MainWindow.xaml*
Dinamik.MainWindow Temizle()
using System;
using System.Windows;

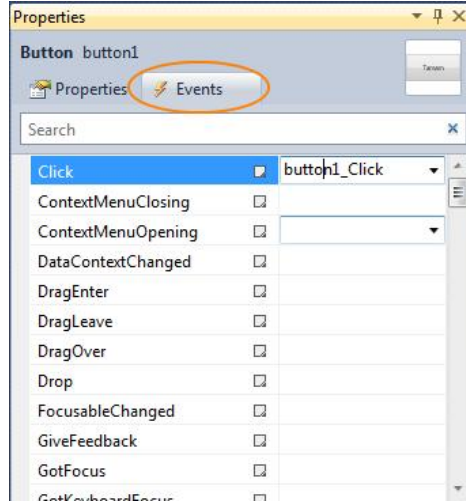
namespace Dinamik
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        private void Temizle()
        {
            txtAd.Text = "";
            txtAdres.Text = "";
            chBilgi.IsChecked = true;
            dtKayit.DisplayDate = DateTime.Now;
        }
    }
}
```

Resim 1.20: Temizle metodu

1.6. Olayları İşleme

Kullanıcı ile programınız arasındaki etkileşimi denetimlere ait olayları yöneterek gerçekleştirebilirsiniz. Uygulama içerisinde ilgilendiğiniz olaylara abone olmanız koşuluyla, olay meydana geldiğinde bir metodu çalıştırabilirsiniz. Denetimle ilgili bir olayı metoda bağlamak oldukça basittir. Tasarım ekranı üzerinden söz konusu nesneyi seçiniz ve ardından “Properties” panelinden “Event” sekmesini açınız. Yönetmek istediğiniz olay tanımı üzerine çift tıklayınız. Denetimle ilgili olay bildirimi ve olay metodu tanımı otomatik olarak oluşturulacaktır.



Resim 1.20: Özellikler (Properties) paneli olay (Events) listesi

Öncelikle WPF denetimlerinin sahip oldukları sıkça kullanılan olayların hangi koşullarda meydana geldiklerini bilmelisiniz.

Olay Adı	Açıklama
Loaded	Denetim hafızaya yüklendiğinde gerçekleşen olaydır.
Closing	Denetim kapatılırken gerçekleşen olaydır.
Closed	Denetim kapatıldıktan sonra gerçekleşen olaydır.
Activated	Denetim aktif olduğunda gerçekleşen olaydır.
DeActivated	Denetimin aktif olmaması durumunda gerçekleşen olaydır.
Click	Fare sol tuşu ile tıklanıldığında tetiklenen olay bildirimidir.
MouseEnter	Fare işaretçisinin denetimin sınırları içerisine girdiğinde gerçekleşen olaydır.
MouseLeave	Fare işaretçisinin denetimin sınırları dışına çıktığında gerçekleşen olaydır.
MouseMove	Fare işaretçisinin denetimin üzerinde dolaştığı her an gerçekleşen olaydır.
SizeChanged	Denetime ait büyüklük (size) değerinin değişiminde meydana gelen olaydır.
TextChanged	Denetimin sahip olduğu "Text" özelliğinin değişiminde meydana gelen olaydır.
ValueChanged	Denetimin sahip olduğu "Value" özelliğinin değişiminde meydana gelen olaydır.

SelectionChanged	Denetim üzerinde bir seçim değişikliği meydana geldiğinde gerçekleşen olaydır.
Checked	Denetimin sahip olduğu "Checked" özelliği değişiminde meydana gelen olaydır.
ImageFailed	Denetim sahip olduğu resim bilgisini kaybetmesinin ardından gerçekleşen olaydır.
MediaOpened	"Media" denetimi kaynağı yüklendiğinde meydana gelen olaydır.
SourceUpdated	Denetim kaynağı güncellendiğinde meydana gelen olaydır.
SelectedDateChanged	Seçilen tarih değiştiğinde gerçekleşen olaydır.
KeyDown	Denetim üzerinde bir tuşa basılırken gerçekleşen olaydır.
KeyUp	Denetim üzerinde bir tuşa basıldıktan sonra gerçekleşen olaydır.

Tablo 1.2: Denetimlere ait olay açıklamaları

WPF denetimlerinin sahip oldukları oldukça fazla olay bildirimi bulunmaktadır. Ancak tablo 1.2'de verilen olay tanımları en sık kullanılanlar arasındadır.

Örnek 1.19: Pencere üzerinde bulunan bir elipsi nesnesini klavye ok tuşları ile hareket ettiriniz.

```
<Window x:Class="WPFApp1.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="MainWindow" Height="350" Width="525"
  KeyDown="Window_KeyDown">
  <Canvas>
    <Ellipse Name="ellipse1" Height="55" Width="57" Stroke="Black"
      HorizontalAlignment="Left" VerticalAlignment="Top"
      Canvas.Left="61" Canvas.Top="23" Fill="Black"/>
  </Canvas>
</Window>
```

Uygulama gereği pencere üzerinde bir nesnenin konum bilgisinin değiştirilmesi için kullanılacak en iyi düzen bileşeni "Canvas" olacaktır. Bu sayede "Window" nesnesi üzerinde iken klavye yön tuşlarına basıldığında (Key Down) elips nesnesinin soldan mesafesi (Canvas.Left) ve yukarıdan mesafesi (Canvas.Top) özellikleri kontrol edilebilir, değiştirilebilir. "Window" denetimine ait "KeyDown" olayına bağlanmış "Window_KeyDown" isimli metodu aşağıda verildiği şekliyle düzenleyiniz.

```

namespace WPFApp1
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        private void Window_KeyDown(object sender, KeyEventArgs e)
        {
            if (e.Key == Key.Left)
            {
                Canvas.SetLeft(ellipse1, Canvas.GetLeft(ellipse1) - 1);
            }
            else if (e.Key == Key.Right)
            {
                Canvas.SetLeft(ellipse1, Canvas.GetLeft(ellipse1) + 1);
            }
            else if (e.Key == Key.Up)
            {
                Canvas.SetTop(ellipse1, Canvas.GetTop(ellipse1) - 1);
            }
            else if (e.Key == Key.Down)
            {
                Canvas.SetTop(ellipse1, Canvas.GetTop(ellipse1) + 1);
            }
        }
    }
}

```

Olay metodu parametrelerini incelediğinizde “KeyEventArgs” sınıfından bir nesneye sahip olduğunu görürsünüz. Olay ile ilgili argümanları metod içine taşımakla görevli bu tip parametreler, “KeyDown” olayı için basılan tuşun bilgisi elde etmek amaçlı kullanılmıştır. Kullanıcının bastığı yön tuşu algılanarak “Ellipse” denetimine ait soldan (Left) ve üstten (Top) mesafe değerlerinin değiştirilmesi ile hareket etkisi sağlanmaktadır. Söz konusu özelliklerin değer değişimi denetimin kendisi yerine kapsayıcı olan “Canvas” bileşeni üzerinden “SetTop()” ve “SetLeft()” metodları ile gerçekleştirilmiştir. Uygulamayı çalıştırınız ve yön tuşlarını kullanarak test ediniz.

UYGULAMA FAALİYETİ

Pencere üzerindeki fare işlecinin konumunu işaretçinin yanında etiket olarak gösterecek bir uygulama gerçekleştiriniz.

- Yeni bir WPF projesi oluşturunuz.
- Pencere içinde bulunan “Grid” bileşenini kaldırınız. Yerine “Canvas” bileşeni yerleştiriniz.

```
<Window x:Class="MousePos.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="MainWindow" Height="350" Width="525"
<Canvas>

</Canvas>
</Window>
```

- Pencere denetimine isim veriniz.
- Canvas içine label yerleştiriniz ve bir isim veriniz.

```
<Window x:Class="MousePos.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="MainWindow" Height="350" Width="525" x:Name="wnd">
<Canvas x:Name="canvas1">
<Label Canvas.Left="208" Canvas.Top="105" Height="25"
Name="label1" Width="86" Background="#FF92F892" />
</Canvas>
</Window>
```

- Pencerenin “MouseMove” olay metodunu oluşturunuz.

```
<Window x:Class="MousePos.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="MainWindow" Height="350" Width="525" x:Name="wnd"
MouseMove="wnd_MouseMove"...
```

- Fare işlecinin pozisyon bilgisini “Label” denetiminin “Content” özelliğine yazdırınız.
- “Label” denetiminin “Canvas” üzerindeki konumunu aynı olay metodu içinde fare işleci konumuna eşitleyiniz.

```
private void wnd_MouseMove(object sender, MouseEventArgs e)
{
    double x=e.GetPosition(wnd).X;
    double y=e.GetPosition(wnd).Y;
    Canvas.SetLeft(label1,x+2);
    Canvas.SetTop(label1,y+2);
    label1.Content = x.ToString() + "*" + y.ToString();
}
```

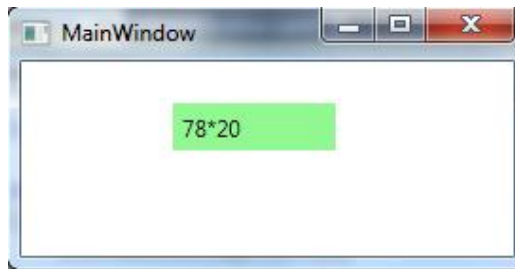
- Fare işleci pencere üzerinden ayrıldığında “MouseLeave” olay metodunda “Label” denetiminin görünürlüğünü gizli yapınız.

```
private void wnd_MouseEnter(object sender, MouseEventArgs e)
{
    label1.Visibility = Visibility.Visible;
}
```

- Fare işleci pencere üzerine girdiğinde “MouseEnter” olay metodunda “Label” denetimini görünür yapınız.

```
private void wnd_MouseLeave(object sender, MouseEventArgs e)
{
    label1.Visibility = Visibility.Hidden;
}
```

- Uygulamayı çalıştırınız ve resim 1.21’de görülen tasarım ile test ediniz.



Resim 1.21: Uygulama sonuç ekranı

ÖLÇME VE DEĞERLENDİRME

Aşağıdaki soruları dikkatlice okuyarak doğru seçeneği işaretleyiniz.

- Aşağıdakilerden hangisi XAML dili yazım kurallarından değildir?
A) Bildirime dayalı bir dildir.
B) Her bir nesne bir bildirim etiketi ile başlar.
C) Açılan her etiket kapatılmalıdır.
D) İsim alanları “xml” ile bildirilir.
- Nesnelere isim vermenizi sağlayan özellik aşağıdakilerden hangisidir?
A) Content
B) Name
C) Text
D) Header
- Aşağıdakilerden hangisi düzen bileşenleri arasında yer almaz?
A) Rectangle
B) Grid
C) StackPanel
D) Canvas
- Aşağıdakilerden hangisi nesnelere “Visibility” özelliğinin alabileceği değerler arasında değildir?
A) Collapsed
B) True
C) Visibility
D) Hidden
- Aşağıdakilerden hangisi denetim üzerinde bir seçim değişikliği meydana geldiğinde gerçekleşen olaydır?
A) SelectionChanged
B) SourceUpdated
C) ValueChanged
D) SizeChanged

DEĞERLENDİRME

Cevaplarınızı cevap anahtarıyla karşılaştırınız. Yanlış cevap verdiğiniz ya da cevap verirken tereddüt ettiğiniz sorularla ilgili konuları faaliyete geri dönerek tekrarlayınız. Cevaplarınızın tümü doğru ise bir sonraki öğrenme faaliyetine geçiniz.

ÖĞRENME FAALİYETİ-2

AMAÇ

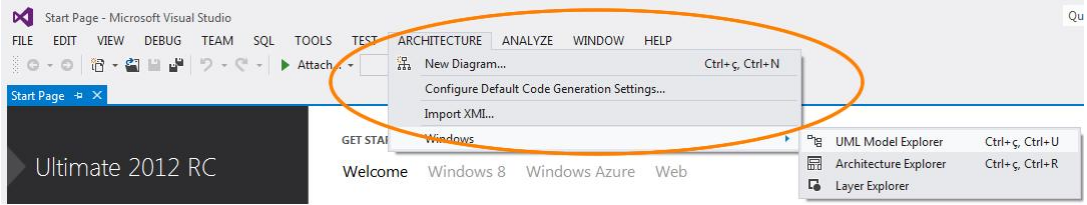
Uygulama penceresi içinde ihtiyacınıza uygun menüler oluşturabilecek ve iletişim kutuları ile çalışabileceksiniz.

ARAŞTIRMA

- Kullandığınız programlarda bulunan menüleri ve sahip olduğu seçeneklerin listesini çıkarınız.
- İşletim sisteminize ait iletişim kutularını inceleyiniz.

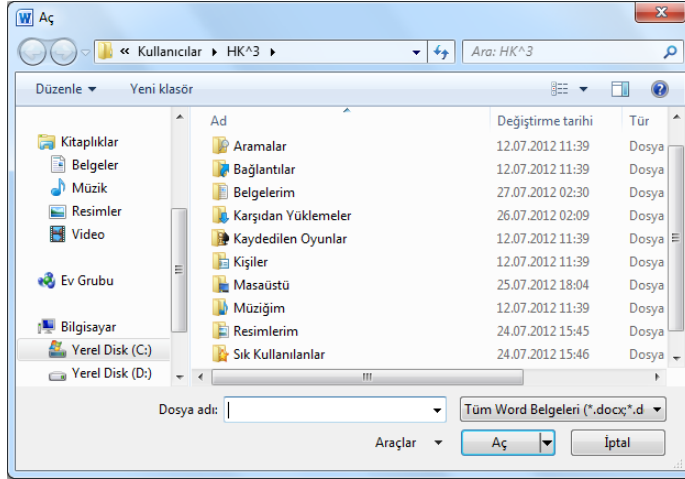
2. MENÜ VE İLETİŞİM KUTULARI

Programla çalışmak için kullandığınız yüzlerce komut (eylem) bulunur. Pencere üzerinde çalışma alanınızın daha verimli kullanımı adına bu komutların çoğu menüler altında düzenlenir. Program menüsü bir restoran menüsüne benzer şekilde size bir seçenekler listesi gösterir. Ekranda karışıklığı önlemek için, başlık çubuğunun hemen altında bulunan menü çubuğundaki başlıklarına tıklatıncaya kadar menüler gizli kalır. İç içe yerleştirilmiş öğeler grubu şeklinde belirli bir düzende bulunur. Bugün pek çok uygulamada resim 2.1'deki gibi menüleri görebilirsiniz.



Resim 2.1: Menü denetimi

İletişim kutuları, kullanıcıya geri besleme sağlamak veya kullanıcıdan giriş almak için kullanılan özelleştirilmiş bir penceredir. İletişim kutuları tek satırlık mesajlar veren basit mesaj kutularından karmaşık kontroller içeren iletişim kutularına kadar değişik biçim ve büyüklükte olabilir.



Resim 2.2: Bir iletişim kutusu

Ayrıca iletişim kutuları, bir program başlatılırken telif (copyright) ve başlangıç bilgilerini görüntülemek veya kullanıcıya uzun süren bir işlemin ilerleyişi hakkında bilgi vermek gibi amaçlarca kullanıcı ile tek taraflı iletişim için de kullanılır.

İletişim kutuları kullanıcıların programlarla etkileşimi için uygun bir yol sunar. Kullanıcılar, bir programla etkileşimlerin çoğunun iletişim kutuları aracılığıyla gerçekleşmesini beklerler. Bütün iletişim kutuları ortak bazı özelliklere sahiptir. Bu ortak özellikler, kullanıcının yaşamını kolaylaştırır. Çünkü kullanıcılar, iletişim kutularının her farklı programda nasıl çalıştığını tekrar tekrar öğrenmek zorunda kalmazlar.

Çeşitli farklı tipte iletişim kutuları vardır ve her biri özel bir amaca sahip, dosya açma, seçme, yazı tipi ve renk seçme ve kelime bulup değiştirme işlemleri için genel iletişim kutuları olduğu gibi kendi özel iletişim kutularımızı da tasarlayabilirsiniz.

2.1. Menü Oluşturma

WPF kütüphanesi içinde programlarımız için kullanabileceğiniz, pencere menüsü (Menu) ve bağlı bulunduğu denetim üzerinde fare sağ tuşuna basıldığında açılan, denetim ile ilgili işlemlerin yer aldığı kısayol menülerine (Context Menu) sahiptir.

Menüler “<Menu>...</Menu>” etiketleri arasında “<MenuItem>...</MenuItem>” menü seçenekleri eklenerek elde edilir. Aşağıdaki örnek tasarım uygulaması bir menü oluşturabilmeniz için gereken “XAML” tanımlamasını ihtiva eder.

```
<Window x:Class="MenuSample.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="MainWindow" Height="350" Width="525">
  <Grid>
    <Menu Height="21" VerticalAlignment="Top">
      <MenuItem Header="Dosya"></MenuItem>
      <MenuItem Header="Düzen"></MenuItem>
      <MenuItem Header="Görünüm"></MenuItem>
    </Menu>
  </Grid>
</Window>
```

Uygulamayı çalıştırıp test ettiğinizde pencere içinde üst kenara hizalanmış bir menüye sahip olduğunuzu göreceksiniz. Uygulamanız bu haliyle sadece menü başlıklarına sahiptir. Sırada menü seçeneklerinin (MenuItem) yerleştirilmesi yapılmalıdır.

```
<Window x:Class="MenuSample.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="MainWindow" Height="350" Width="525">
  <Grid>
    <Menu Height="21" VerticalAlignment="Top">
      <MenuItem Header="Dosya">
        <MenuItem Header="Yeni"></MenuItem>
        <MenuItem Header="Aç"></MenuItem>
        <MenuItem Header="Kapat"></MenuItem>
      </MenuItem>
      <MenuItem Header="Düzen">
        <MenuItem Header="Kes"></MenuItem>
        <MenuItem Header="Kopyala"></MenuItem>
        <MenuItem Header="Yapıştır"></MenuItem>
      </MenuItem>
      <MenuItem Header="Görünüm"></MenuItem>
    </Menu>
  </Grid>
</Window>
```

Uygulamayı test ediniz, resim 2.3'te görülen menü seçenekleri ve onu oluşturan tasarım kodu arasındaki hiyerarşik benzerliği bulmaya çalışınız.



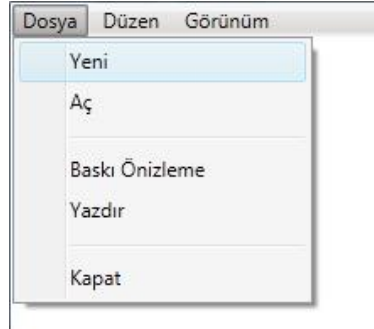
Resim 2.3: Uygulama test sonucu

Örnek 2.1: Yazıcı ve dosyalama işlemlerini barındıran, gruplandırılmış “Dosya” menüsü tasarımını gerçekleştiriniz.

Menu seçeneklerini kendi aralarında gruplandırmak için ayraç (Separator) nesnesi kullanılmalıdır.

```
<Window x:Class="MenuSample.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="MainWindow" Height="350" Width="525">
  <Grid>
    <Menu Height="21" VerticalAlignment="Top">
      <MenuItem Header="Dosya">
        <MenuItem Header="Yeni"></MenuItem>
        <MenuItem Header="Aç"></MenuItem>
        <Separator/>
        <MenuItem Header="Baskı Önizleme"></MenuItem>
        <MenuItem Header="Yazdır"></MenuItem>
        <Separator/>
        <MenuItem Header="Kapat"></MenuItem>
      </MenuItem>
      <MenuItem Header="Düzen">
        <MenuItem Header="Kes"></MenuItem>
        <MenuItem Header="Kopyala"></MenuItem>
        <MenuItem Header="Yapıştır"></MenuItem>
      </MenuItem>
      <MenuItem Header="Görünüm"></MenuItem>
    </Menu>
  </Grid>
</Window>
```

Uygulamayı test ediniz ve ayraçlarının menü seçenekleri arasında yer aldığını gözlemleyiniz.



Resim 2.4: Uygulama test sonucu

Örnek 2.2: Menü seçenekleri için ikon tasarımını gerçekleştiriniz.

Menü seçeneği için ikon bildirimini "MenuItem" nesnesinin "icon" isimli özelliği ile gerçekleştirilir.

```
<Window x:Class="MenuSample.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="MainWindow" Height="350" Width="525">
  <Grid>
    <MenuItem Height="21" VerticalAlignment="Top">
      <MenuItem Header="Dosya">
        <MenuItem Header="Yeni">
          <MenuItem.Icon>
            <Image Source="icon/new.ico" Width="16" Height="16" ></Image>
          </MenuItem.Icon>
        </MenuItem>
        <MenuItem Header="Aç">
          <MenuItem.Icon>
            <Image Source="icon/open.ico" Width="16" Height="16" ></Image>
          </MenuItem.Icon>
        </MenuItem>
        <Separator></Separator>
        <MenuItem Header="Baskı Önizleme"></MenuItem>
        <MenuItem Header="Yazdır">
          <MenuItem.Icon>
            <Image Source="icon/print.ico" Width="16" Height="16" ></Image>
          </MenuItem.Icon>
        </MenuItem>
        <Separator></Separator>
        <MenuItem Header="Kapat"></MenuItem>
      </MenuItem>
    </Grid>
```

```
<MenuItem Header="Düzen">
  <MenuItem Header="Kes"></MenuItem>
  <MenuItem Header="Kopyala"></MenuItem>
  <MenuItem Header="Yapıştır"></MenuItem>
</MenuItem>
<MenuItem Header="Görünüm"></MenuItem>
</Menu>
</Grid>
</Window>
```

Uygulamayı çalıştırınız ve test ediniz.



Resim 2.5: Uygulama test sonucu

2.2. Menü Olaylarını İşleme

Kullanıcı menü üzerinde yapmak istediği işlemi fare sol tuş tıklamasıyla gerçekleştirir. Menü seçeneği (MenuItem) için “fare ile tıkladığında” şeklinde bir kullanıcı etkileşimi WPF denetimi için algılanması gereken bir olay (event) niteliğindedir. Bu olay çoğu WPF denetimlerinde olduğu gibi “Click” adıyla tanımlanır. Olayın meydana gelmesi karşılığında program tarafından bir eylemin gerçekleşmesi gerekir ki bu da ancak bir metodu işaret etmesi ile gerçekleşir.

```
.....
<Menu Height="21" VerticalAlignment="Top">
  <MenuItem Header="Dosya">
    <MenuItem Header="Yeni" Click="YeniIslem"></MenuItem>
    <MenuItem Header="Aç"></MenuItem>
    <MenuItem Header="Kapat"></MenuItem>
  </MenuItem>
  .....

```

Yukarıda bulunan tasarım kodunda “Dosya” menüsünden “Yeni” seçeneği tıkladığında “YeniIslem” metodunun çağırılacağı belirtilmektedir. Böyle bir tanımlamadan

sonra olay metodu kod sayfasında (“MainWindow.xaml.cs” dosyası) mutlaka tanımlanmış olmalıdır. Yapılması istenen işler ise metod gövdesinde kodlanır.

```
namespace MenuSample
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        private void YeniIslem(object sender, RoutedEventArgs e)
        {
            // Yeni menü seçeneği tıklandığında
            // yapılacak işler burada kodlanır.
        }
    }
}
```

Olayları bildirmenin bir başka yolu ise kod sayfasında tanımlamaktır. Yalnız bunun için mutlaka ilgili nesnenin “Name” özelliği ataması yapılmış olmalıdır. Bilindiği üzere söz konusu nesneye kod sayfasında ulaşım bu isim üzerinden gerçekleşmekteydi. Öncelikle “XAML” kodunda menü ile ilgili kısmı resim 2.6’da görüldüğü gibi düzenleyiniz.



Resim 2.6: “Name” özelliğinin ataması

Ardından kod sayfasında (pencereye ait “xaml.cs” uzantılı dosya) bulunan ve class ismini taşıyan kurucu metod içinde “yeni” ismini taşıyan “MenuItem” nesnesinin “Click” olayı “RoutedEventHandler” temsilcisi üzerinden “yeni_Click()” isimli olay metoduna resim 2.7’de görüldüğü gibi bağlayınız. Uygulamanın çalışması ile birlikte “MainWindow” sınıfı hafızada örneklenicek ve ilk olarak kurucu metodu içinde yazmış olduğunuz bu olay aboneliği sağlanacaktır.

```
MainWindow.xaml.cs  MainWindow.xaml
MenuSample.MainWindow
namespace MenuSample
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            yeni.Click += new RoutedEventHandler(yeni_Click);
        }

        void yeni_Click(object sender, RoutedEventArgs e)
        {
            // Yeni menü öğesi tıklandığında
            // çağırımı yapılacak metod
        }
    }
}
```

Resim 2.7: Menü olaylarını işleme

2.3. Kısayol Menüleri

Windows uygulamalarında, fare sağ tuş menüsü, kullanıcıların en fazla kullanma eğilimi gösterdikleri denetimlerin arasındadır. Bu tür menülerin özelliği içerik duyarlı olması ve yalnızca etkin denetime ya da forma özgü komutları listelemesidir. WPF içinde “ContextMenu” olarak tanınan kısayol menü denetimi uygulamalarınızda kullanabilirsiniz.

Kısayol menüleri denetime bağlılığından dolayı söz konusu denetimin “ContextMenu” özelliği altında tanımlanır. “<ContextMenu>...</ContextMenu>” etiketleri arasında “<MenuItem>” öğeleri eklenerek menü seçenekleri oluşturulur.

Örnek 2.3: Uygulama penceresinin şeffaflık değerini %50, %75, %100 olarak belirleyecek kısayol menüsü tasarımı oluşturunuz.

```

<Window x:Class="ContextSample.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="MainWindow" Height="350" Width="525"
  AllowsTransparency="True" WindowStyle="None" Background="Orange">
  <Window.ContextMenu>
    <ContextMenu>
      <MenuItem Header="%50" Click="MenuItem_Click" />
      <MenuItem Header="%75" Click="MenuItem_Click_1" />
      <MenuItem Header="%100" Click="MenuItem_Click_2" />
    </ContextMenu>
  </Window.ContextMenu>
  <Grid>
  </Grid>
</Window>

```

Menü seçenekleri tıklandığında pencerenin şeffaflık oranını belirleyen “Opacity” özelliğinin dinamik olarak değiştirilmesi gerekmektedir. Menü seçeneklerinin “Click” olayı (event) bildirimlerini ve olay metodu gövdelerini aşağıda verildiği gibi düzenleyiniz.

```

namespace ContextSample
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }
        private void MenuItem_Click(object sender, RoutedEventArgs e)
        { // %50 şeffaflık oranı
            this.Opacity=0.5;
        }
        private void MenuItem_Click_1(object sender, RoutedEventArgs e)
        { // %75 şeffaflık oranı
            this.Opacity = 0.75;
        }
        private void MenuItem_Click_2(object sender, RoutedEventArgs e)
        { // %100 şeffaflık oranı
            this.Opacity = 1;
        }
    }
}

```

Uygulamayı çalıştırıp test ediniz.

2.4. Ortak İletişim Kutuları

İletişim kutuları, kullanıcıya geri besleme sağlamak veya kullanıcıdan giriş almak için kullanılan özelleştirilmiş pencerelerdir. Bu pencereler sayesinde uygulamalar kullanılabilirlik adına daha fazla yol gösterici olmaktadır. Örneğin herhangi bir dosyayı açmak veya kaydetmek için farklı programların aynı iletişim kutularını kullandığını görmüşsünüzdür. Bu işlevsellik o kadar yaygındır ki ortak iletişim kutuları, her program tarafından ortak olarak kullanılmaktadır.

Bazı iletişim kutularının kullanılması programlarda o kadar sık gerekli olur ki bunlar işletim sisteminin bir parçası olarak gelmektedir. Genel iletişim kutuları (common dialog boxes) olarak bilinen bu iletişim kutuları bir fonksiyon çağırma yoluyla kullanılabilir ve sizin bir iletişim kutusu kaynağı oluşturmanızı gerektirmez.

Diskte bulunan bir dosyayı açmak için dosya konumunu ve ismini belirleyebileceğiniz iletişim kutusu için "OpenFileDialog", aynı işlevleri kaydetmek üzere kullanacağınız iletişim kutusu için "SaveFileDialog" sınıflarını kullanabilirsiniz.

Ortak iletişim kutularını ekrana getirmek için "ShowDialog()" metodundan faydalanılır.

Örnek 2.4: Basit metin editörü uygulaması gerçekleştiriniz.

```
<Window x:Class="DialogSample.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="TextEditor" Height="350" Width="525">
  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition Height="22*" />
      <RowDefinition Height="289*" />
    </Grid.RowDefinitions>
    <Menu VerticalAlignment="Top" Height="21">
      <MenuItem Header="Dosya">
        <MenuItem Header="Yeni" Click="yeni_Click"></MenuItem>
        <MenuItem Header="Aç" Click="ac_Click"></MenuItem>
        <MenuItem Header="Kaydet" Click="kaydet_Click"></MenuItem>
      </MenuItem>
    </Menu>
    <TextBox Grid.Row="1" Margin="0" Name="txtIcerik" TextWrapping="Wrap" />
  </Grid>
</Window>
```

```

using Microsoft.Win32;
using System.IO;
namespace DialogSample
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        private void ac_Click(object sender, RoutedEventArgs e)
        {
            OpenFileDialog openFileDialog = new OpenFileDialog();
            openFileDialog.Filter = "Metin dosyaları|.txt| + "Bütün dosyalar|*.*";
            openFileDialog.Title = "Dosya Aç";
            // İletişim kutusu ekrana getiriliyor.
            if(openFileDialog.ShowDialog()==true)
            {
                string dosya_adi;
                // Dosya adı iletişim penceresinden alınıyor.
                dosya_adi = openFileDialog.FileName;
                // Dosya text modunda açılıyor.
                TextReader dosya = File.OpenText(dosya_adi);
                string x;
                // Dosya içeriği değişkene alınıyor.
                x = dosya.ReadToEnd();
                dosya.Close();
                // İçerik metin kutusuna yazılıyor.
                txtIcerik.Text = x;
            }
        }

        private void kaydet_Click(object sender, RoutedEventArgs e)
        {
            SaveFileDialog saveDialog = new SaveFileDialog();
            saveDialog.Title = "Dosya Kaydet";
            saveDialog.Filter = "Metin dosyaları|.txt| + "Bütün dosyalar|*.*";
            saveDialog.DefaultExt = ".txt";
            // İletişim penceresi ekrana getiriliyor.
            if (saveDialog.ShowDialog() == true)
            {
                string dosya_adi;
                // Kaydedilecek dosya adı alınıyor.
                dosya_adi = saveDialog.FileName;
            }
        }
    }
}

```



```
// Dosya text modunda oluşturuluyor.  
TextWriter dosya = File.CreateText(dosya_adi);  
dosya.Write(txtIcerik.Text);  
dosya.Close();// Dosya kapatılıyor  
}  
}  
}}
```

Uygulamayı çalıştırmız ve test ediniz.

UYGULAMA FAALİYETİ

Diskte bulunan medya dosyalarınızı çalma listesinden yürütecek uygulamayı gerçekleştiriniz.

- Yeni WPF uygulaması oluşturunuz.
- Uygulama penceresi içerisine “StackPanel” yerleştiriniz.
- “StackPanel” bileşeni içerisine “Menu” ve “MediaElement” nesnelerini araç kutusu (Toolbox) üzerinden sürükleyiniz.
- Menü listesine “Çalma Listesi” ve “Medya” isimli iki seçenek (MenuItem) yerleştiriniz.

```
<Window x:Class="MediaPlay.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="MainWindow" Height="350" Width="525">
  <Grid>
    <StackPanel>
      <Menu Height="23" Name="menu1" Margin="0">
        <MenuItem Header="Çalma Listesi">

          </MenuItem>
          <MenuItem Header="Medya">

          </MenuItem>
          <MenuItem>
          </MenuItem>
        </Menu>
        <MediaElement Name="media" Height="285" />
      </StackPanel>
    </Grid>
  </Window>
```

- Çalma listesine ait menü seçeneğinin içeriğini aşağıda verildiği şekliyle oluşturunuz. “Son İzlenenler” liste elemanına çalışma zamanında öğe ekleneceğinden “Name” özelliği ile isim veriniz.

```
<MenuItem Header="Çalma Listesi">
  <MenuItem Header="Ekle" />
  <MenuItem Header="Liste Temizle" />
  <Separator/>
  <MenuItem Name="liste" Header="Son İzlenenler"/>
</MenuItem>
```

- Yürütümü yapılacak medyanın disk ortamından seçilebilmesi için “Ekle” menü elemanının “Click” olayına “ekle_Click” olay metodunu bağlayınız.

```
<MenuItem Header="Ekle" Click="ekle_Click" />
```

- “ekle_Click” olay metodu içinde “mp4” dosyalarının filtrelenmesi ve seçilen dosyanın medya kaynağına bağlanmasının ardından, “Son İzlenenler” listesine eklendiği kod satırlarını aşağıda verildiği şekliyle kod sayfasında yazınız.

```
Uri medyaYol;  
  
private void ekle_Click(object sender, RoutedEventArgs e)  
{  
    OpenFileDialog dialog = new OpenFileDialog();  
    dialog.Filter = "Medya|*.wmv";  
    dialog.ShowDialog();  
    medyaYol = new Uri(dialog.FileName);  
    string[] dosya = dialog.FileName.Split("\\");  
    liste.Items.Add(dosya[dosya.Length - 1]);  
    media.Source = medyaYol;  
}
```

- “Son İzlenenler” liste içeriğinin temizlenmesini sağlayacak “Liste Temizle” menü elemanının “Click” olayına “temizle_Click” olay metodunu bağlayınız.

```
<MenuItem Header="Liste Temizle" Click="temizle_Click" />
```

- “temizle_Click” olay metodunda ismi “liste” olan menü seçeneği içeriğini silecek olay metodu gövdesini yazınız.

```
private void temizle_Click(object sender, RoutedEventArgs e)  
{  
    liste.Items.Clear();  
}
```

- “Medya” isimli ana menü başlığı içeriğinde bulunacak menü listesini aşağıda verildiği şekliyle oluşturunuz.

```
<MenuItem Header="Medya">  
    <MenuItem Header="Yürüt" />  
    <MenuItem Header="Durdur" />  
    <MenuItem Header="Ses">  
        <Slider Name="ses" Minimum="0" Maximum="1" Width="60" />  
    </MenuItem>
```

Menü listesi tasarımını incelediğinizde “ses” isimli menü öğesinin altında “slider” nesnesinin kullanıldığını görürsünüz. Bu tip bir kullanım sonucu menü öğesi altında bir “slider” nesnesi oluşacaktır.

- “MediaElement” nesnesini yürütme görevi olan “Yürüt” menü seçeneğinin “Click” olayına “yurut_Click” olay metodunu bağlayınız.

```
<MenuItem Header="Yürüt" Click="yurut_Click" />
```

- “yurut_Click” Olay metodunda “media” isimli nesnenin “Play” isimli metodunu çağırınız.

```
private void yurut_Click(object sender, RoutedEventArgs e)
{
    media.Play();
}
```

- “Durdur” fonksiyonuna sahip olacak menü öğesine “dur_Click” olay metodunu bağlayınız.

```
<MenuItem Header="Durdur" Click="dur_Click" />
```

- “MedyaElement” nesnesinin durdurulmasını sağlayacak “Stop” metodunu “dur_Click” olay metodu içinde çağırmasını yapınız.

```
private void dur_Click(object sender, RoutedEventArgs e)
{
    media.Stop();
}
```

- ”MediaElement” nesnesinin “Volume” özelliği 0-1 değer aralığında bulunması sebebiyle “slider” nesnesinin “maximum” ve “minimum” değerleri de bu aralıkta tutulmalıdır. Ses ayarı menü öğe yapısında tutulan “slider” nesnesinin değer değişimi meydana geldiğinde uygulanacak olması nedeniyle “ValueChanged” olayına “ses_ValueChanged” olay metodunu bağlayınız.

```
<MenuItem Header="Ses">
    <Slider Name="ses" Minimum="0" Maximum="1" Value="0.6" Width="60"
        ValueChanged="ses_ValueChanged"/>
</MenuItem>
```

- “ses_ValueChanged” Olay metodu içinde “medya” nesnesinin “Volume” değerini “slider” nesnesinin “Value” özelliğinden almalıdır.

```
private void ses_ValueChanged(object sender,  
RoutedPropertyChangedEventArgs<double> e)  
{  
    media.Volume = ses.Value;  
}
```

- Uygulamayı çalıştırınız ve çalma listesine bir video ekleyip test ediniz



Resim 2.8: Uygulama test sonucu

ÖLÇME VE DEĞERLENDİRME

Aşağıdaki cümlelerde boş bırakılan yerlere doğru sözcükleri yazınız.

1., kullanıcıya geri besleme sağlamak veya kullanıcıdan giriş almak için kullanılan özelleştirilmiş pencerelerdir.
2. Kısayol menüleri denetimin özelliği ile bildirilir.
3. Menü seçenekleri etiketi ile tanımlanır.
4. Menü başlıkları özelliği ile bildirilir.
5. İletişim kutuları metodu ile ekrana getirilir.
6. Menü öğesinin seçimi sonucu olayı tetiklenir.
7. "OpenFileDialog" sınıfı iletişim kutusudur.
8. "SaveFileDialog sınıfı iletişim kutusudur.

DEĞERLENDİRME

Cevaplarınızı cevap anahtarıyla karşılaştırınız. Yanlış cevap verdiğiniz ya da cevap verirken tereddüt ettiğiniz sorularla ilgili konuları faaliyete geri dönerek tekrarlayınız. Cevaplarınızın tümü doğru ise bir sonraki öğrenme faaliyetine geçiniz.

ÖĞRENME FAALİYETİ-3

AMAÇ

Nesneleriniz ile WPF denetimleriniz arasında bağlantı kurabilecek, doğrulama işlemlerini yapabileceksiniz.

ARAŞTIRMA

- Kullanıcının girmiş olduğu verileri işlemeden önce denetlemenin gerekliliklerini araştırınız.
- .NET çatısı altında tanımlanmış arayüzleri (interface) kendi projeleriniz altında kullanılabilir olmasının avantajlarını araştırınız.
- Nesne tabanlı programlamada Observable (gözlemci) tasarım kalıbını araştırınız.

3. DOĞRULAMAYI GERÇEKLEŞTİRME

Doğrulama, kullanıcının girdiği verilerin türleri ve değerlerini denetlemek amacıyla yapılır. Örneğin veri girişinin belirli bir tarih aralığında olması, girilen bir mail adresinin geçerliliğinin denetlenmek istenmesi veya girilen miktarın negatif değer olmaması gerekebilir. Bu gibi senaryolar karşısında doğrulama (validation) işlemlerinin yapılabilmesi, doğrulamadan geçemeyen kullanıcı girişleri karşısında uygulamanızın gereken uyarıları vermesi sağlanabilir.

3.1. Veri Doğrulama

Veri doğrulama konusunda çeşitli yöntemler vardır. Ancak şu an için .NET platformu WPF için varlık sınıflarınızda doğrulama işlemleri yapabilmenizi sağlamak amaçlı “Windows.ComponentModel” isim alanında bulunan “IDataErrorInfo” isimli bir arayüz (interface) sözleşmesi sunar. Doğrulamanın gerçekleştirilebilmesi için varlık sınıflarınız “IDataErrorInfo” arayüzünden bildirim yapılmalıdır.

Örnek 3.1: Stok yönetimini konu alan bir projede programcı olarak görev almaktasınız. Ekranda ürün listesini sunacak ve üzerinde veri işlemleri yapmanıza olanak sağlayacak arayüzü WPF ortamında geliştirmeniz istenmektedir. Tasarımla ilgili olarak ürün ismi ve fiyatı olmadan ürün girişine izin verilmesi istenmektedir.

İlk olarak yeni bir WPF projesi içinde ürün varlık sınıfını oluşturunuz.

```
namespace StokYonetim
{
    class Urun
    {
        private int _id;

        public int Id
        {
            get { return _id; }
            set { _id = value; }
        }
        private string _urunAdi;

        public string UrunAdi
        {
            get { return _urunAdi; }
            set { _urunAdi = value; }
        }
        private float _fiyat;

        public float Fiyat
        {
            get { return _fiyat; }
            set { _fiyat = value; }
        }
        private int miktar;

        public int Miktar
        {
            get { return miktar; }
            set { miktar = value; }
        }
    }
}
```

Ürün ile ilgili talep edilen doğrulama hizmetlerinin yapılabilmesi için varlık sınıfını “IDataErrorInfo” arayüzünden (interface) türetiniz.


```

using System;
using System.ComponentModel;

namespace StokYonetim
{
    class Urun : IDataErrorInfo
    {
        private int _id;
        public int Id
        {
            get { return _id; }
            set { _id = value; }
        }
        private string _urunAdi;
        public string UrunAdi
        {
            get { return _urunAdi; }
            set { _urunAdi = value; }
        }
        private float _fiyat;
        public float Fiyat
        {
            get { return _fiyat; }
            set { _fiyat = value; }
        }
        private int miktar;
        public int Miktar
        {
            get { return miktar; }
            set { miktar = value; }
        }

        public string Error
        {
            get { throw new NotImplementedException(); }
        }

        public string this[string columnName]
        {
            get { throw new NotImplementedException(); }
        }
    }
}

```

“IDataErrorInfo” arayüzü varlık sınıfınız içinde iki özellik (property) daha ekleyecektir. “Error” isimli özellik nesne bazında doğrulama yapmayı sağlar. Şimdilik bu tip bir doğrulamadan faydalanmayacağınızdan “Error” özelliğinin gövdesini aşağıdaki gibi düzenleyebilirsiniz.

```
public string Error
{
    get { return null; }
}
```

“IDataErrorInfo” arayüzün sunduğu diğer özellik ise bir dizinleyicidir (indexer) ve “columnName” parametresi ile alan (field) bazında doğrulama gerçekleştirmek için kullanılacaktır. Uygulamada ihtiyacınız olan, farklı alanlarda farklı hata mesajlarını yönetmek olduğundan, doğrulama işlevi bu dizinleyici içinde aşağıda verildiği şekliyle belirtilir.

```
public string this[string columnName]
{
    get
    {
        string hata=null;
        switch(columnName)
        {
            case "UrunAdi" :
                {
                    if (string.IsNullOrEmpty(this.UrunAdi))
                        hata = "Ürün Adı Boş Geçilemez..";
                    break;
                }
            case "Fiyat" :
                {
                    if (this.Fiyat == 0)
                        hata = "Lütfen Fiyatı Giriniz...";
                    break;
                }
        }
        return hata;
    }
}
```

Alanlar üzerinde doğrulamayı sağlayacak dizinleyici (indexer) üzerinden geriye hata mesajını döndürecektir. İlk olarak “hata” isimli “string” türünden değişken bu amaçlı kullanılmalıdır. Parametrede verilen “columnName” ile doğrulaması yapılacak alan adına göre, hangi kriterde nasıl bir hata mesajının verileceği tanımlamaları basit bir “switch..case”

karar bloğu altında ele alınır ve hata mesajı değişkene aktarılarak “return” deyimi ile geriye döndürülür. Yukarıda yer alan doğrulama kodu “UrunAdı” ve “Fiyat” alanları denetimi için uygulanmıştır. Dilerseniz doğrulaması yapılacak alan sayısını bu esaslarda arttırabilirsiniz.

Varlık sınıfı üzerinde yapılan denetimden sonra program arayüzü tasarımına geçebilirsiniz. “MainWindow.xaml” dosyasını açınız ve pencere üzerinde yer alacak denetimler için panel yerleşimini gerçekleştiriniz. Tablo tipinde bir yerleşim için en uygun düzen denetimi (layout) “grid” bileşeni olacaktır.

```
<Window x:Class="StokYonetim.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:d="clr-namespace:StokYonetim"
  Title="Ürünler" Height="200" Width="325">

  <Grid x:Name="data">
    <Grid.RowDefinitions>
      <RowDefinition Height="40"/>
      <RowDefinition Height="40"/>
      <RowDefinition Height="40"/>
      <RowDefinition Height="0.615*"/>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="0.271*"/>
      <ColumnDefinition Width="0.829*"/>
    </Grid.ColumnDefinitions>
    .....
  </Grid>
</Window>
```

Kullanıcının bilgi girişi yapacağı metin kutuları (textbox) ve bildirecekleri etiketleri (textblock) “grid” içinde yer alan hücelere aşağıda verildiği şekliyle yerleştiriniz.

```
<TextBlock Text="Ürün Adı" Margin="5" Grid.Column="0" Grid.Row="0"/>
<TextBox Margin="5" Grid.Column="1" Grid.Row="0"/>
<TextBlock Text="Ürün Fiyatı" Margin="5" Grid.Column="0" Grid.Row="1"/>
<TextBox Margin="5" Grid.Column="1" Grid.Row="1"/>
<TextBlock Text="Miktar" Margin="5" Grid.Column="0" Grid.Row="2"/>
<TextBox Margin="5" Width="97" HorizontalAlignment="Left"
  Grid.Column="1" Grid.Row="2"/>
```

“Grid” bileşenine veri kaynağı (DataContext) olarak bildireceğiniz nesnenin özelliklerini “textbox” denetiminin “Text” özelliğine bağlamak için tasarım kodunu aşağıda verildiği şekliyle güncelleyiniz.

```

<TextBlock Text="Ürün Adı" Margin="5" Grid.Column="0" Grid.Row="0"/>
<TextBox Margin="5" Grid.Column="1" Grid.Row="0"
    Text="{Binding Path=UrunAdi,
        UpdateSourceTrigger=LostFocus,
        ValidatesOnDataErrors=True}"
    Style="{StaticResource textBoxInError}"/>
<TextBlock Text="Ürün Fiyatı" Margin="5" Grid.Column="0" Grid.Row="1"/>
<TextBox Margin="5" Grid.Column="1" Grid.Row="1"
    Text="{Binding Path=Fiyat,
        UpdateSourceTrigger=LostFocus,
        ValidatesOnDataErrors=True}"
    Style="{StaticResource textBoxInError}" />
<TextBlock Text="Miktar" Margin="5" Grid.Column="0" Grid.Row="2"/>
<TextBox Margin="5" Width="97" HorizontalAlignment="Left"
    Grid.Column="1" Grid.Row="2"
    Text="{Binding Path=Miktar}" />

```

Yukarıda verilen tasarım kodunu incelediğinizde veri girişi olarak kullanılacak metin kutularının (textbox) “Text” özellikleri “Binding” anahtarı ile “Path” değerine verilen alanlara bağlandığını görebilirsiniz. Bu sayede her metin kutusu gösterimini yapacağı nesnenin hangi alanını işaret edeceği belirlenmiş olmaktadır. Modülünüzün bir sonraki konusu olan “Veri Bağlama” ünitesinde bu konuyu detaylı olarak işleyeceksiniz. Veri doğrulamasının yapılacağını bağlama parametrelerinden “ValidatesOnDataErrors” ile bildirilmektedir. Doğrulamanın zamanlama olarak metin kutusunda çıktığı anda gerçekleşeceğini, “UpdateTrigger” özelliğine atanan “Lostfocus” değeri ile bildirilmektedir.

Geçerliliği denetlenen metin kutusunun doğrulamayı yapamaması durumunda ilgili hata mesajını verebilmesi için stil tanımlaması yapılmalıdır. Söz konusu stil metin kutusunun “Style” parametresi ile bildirilir. Stil bildirimlerini “<Window.Resources>...</Window.Resources>” etiketleri arasında aşağıda verildiği şekilde tanımlayınız.

```

<Window.Resources>
  <Style x:Key="textBoxInError" TargetType="TextBox">
    <Style.Triggers>
      <Trigger Property="Validation.HasError" Value="True">
        <Setter Property="ToolTip"
            Value="{Binding RelativeSource={x:Static RelativeSource.Self},
                Path=(Validation.Errors)[0].ErrorContent}"/>
      </Trigger>
    </Style.Triggers>
  </Style>
</Window.Resources>

```

Stil bilgisinin WPF denetimleri tarafından kullanılabilmesi için mutlaka “Key” tanımlaması ile bir isme sahip olmalıdır. Örnek uygulama gereği metin kutusuna girilen bilginin geçerliliği doğrulanamaz ise denetim üzerinde balon yardımı ile hata mesajının gösterilmesi düşünülmektedir. Alınan bu tasarım kararı neticesinde stil içinde “<Trigger>” etiketi ile bir tetikleyici oluşturulmuş ve stile sahip olan metin kutularının “Validation.HasError” değerinin doğru olması durumunda stilin uygulanacağı belirtilir. Bu özellik ancak doğrulamanın yapılması durumunda “true” değerine sahip olduğundan stili uygulayan metin kutusu üzerinde balon yardımı (Tooltip) açılacak ve “value” parametresi ile “IDataErrorInfo” arayüzün (interface) sunduğu hata dizinleyicilerinden (indexer) gelecek uyarı mesajı görüntülenecektir.

Uygulamaya ait tasarım kodunun bütün hali aşağıda verilmiştir.

```
<Window x:Class="StokYonetim.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:d="clr-namespace:StokYonetim"
  Title="Ürünler" Height="200" Width="325">
<Window.Resources>
  <Style x:Key="textBoxInError" TargetType="TextBox">
    <Style.Triggers>
      <Trigger Property="Validation.HasError" Value="True">
        <Setter Property="ToolTip"
          Value="{Binding RelativeSource={x:Static RelativeSource.Self},
            Path=(Validation.Errors)[0].ErrorContent}"/>
      </Trigger>
    </Style.Triggers>
  </Style>
</Window.Resources>

<Grid x:Name="data">
  <Grid.RowDefinitions>
    <RowDefinition Height="40"/>
    <RowDefinition Height="40"/>
    <RowDefinition Height="40"/>
    <RowDefinition Height="0.615*"/>
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="0.271*"/>
    <ColumnDefinition Width="0.829*"/>
  </Grid.ColumnDefinitions>
```

```

<TextBlock Text="Ürün Adı" Margin="5" Grid.Column="0" Grid.Row="0"/>
  <TextBox Margin="5" Grid.Column="1" Grid.Row="0"
    Text="{Binding Path=UrunAdi,
      UpdateSourceTrigger=LostFocus,
      ValidatesOnDataErrors=True}"
    Style="{StaticResource textBoxInError}"/>
  <TextBlock Text="Ürün Fiyatı" Margin="5" Grid.Column="0" Grid.Row="1"/>
  <TextBox Margin="5" Grid.Column="1" Grid.Row="1"
    Text="{Binding Path=Fiyat,
      UpdateSourceTrigger=LostFocus,
      ValidatesOnDataErrors=True}"
    Style="{StaticResource textBoxInError}" />
  <TextBlock Text="Miktar" Margin="5" Grid.Column="0" Grid.Row="2"/>
  <TextBox Margin="5" Width="97" HorizontalAlignment="Left"
    Grid.Column="1" Grid.Row="2"
    Text="{Binding Path=Miktar}" />
</Grid>
</Window>

```

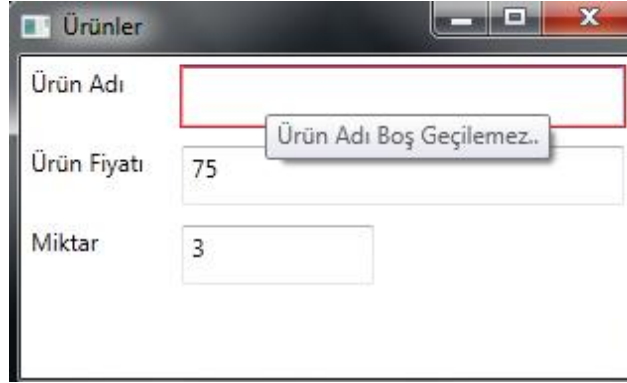
Son olarak çalışma zamanında oluşturulacak “Urun” sınıfından bir test nesnesini, bağlanacağı denetimleri kapsayan “Grid” bileşenine veri kaynağı olarak (DataContext) bildiriniz. “MainWindow.xaml.cs” dosyasını açınız ve aşağıda verilen .NET kodunu buna göre oluşturunuz.

```

namespace StokYonetim
{
  public partial class MainWindow : Window
  {
    public MainWindow()
    {
      InitializeComponent();
      Urun urn=new Urun();
      data.DataContext = urn;
    }
  }
}

```

Uygulamayı çalıştırdınız ve test ediniz. Doğrulamanın geçersiz kaldığı durumlardan biri olan ürün isminin boş geçilmesi durumunda resim 3.1’de görülen uyarıyı almalısınız.



Resim 3.1: Uygulama test ekranı

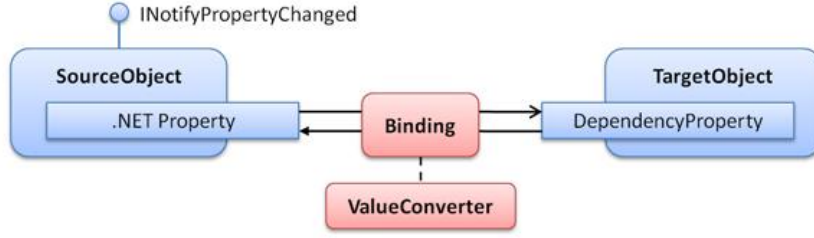
3.2. Veri Bağlama

Uygulamanız iş mantığı üzerinde yer alan varlık sınıflarınız (entity class) ile WPF kullanıcı denetimleriniz arasında bağlantı kurmanızı sağlayan bir mekanizmadır. Bağlantı nesnelerin özellikleri (property) arasında kurulur. Dolayısıyla aralarında bağlantı kurmak istediğiniz hedef nesne özelliği ile kaynak nesne özelliği aynı veri türüne sahip olmalıdır.

Örnek 3.2: Etiket yazısının font büyüklüğünü bir “slider” nesnesi ile ayarlayınız.

```
<Window x:Class="BindingSample.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="MainWindow" Height="350" Width="525">
  <Grid>
    <Label Content="Label" HorizontalAlignment="Left" Margin="21,61,0,0"
      Name="label1" VerticalAlignment="Top"
      FontSize="{Binding ElementName=slider1, Path=Value}" />
    <Slider Height="22" HorizontalAlignment="Left" Margin="21,21,0,0"
      Name="slider1" VerticalAlignment="Top" Width="284" Maximum="100" />
  </Grid>
</Window>
```

Uygulama tasarım kodunu incelediğinizde “label” nesnesinin “FontSize” özelliği, “slider” nesnesinin “Value” özelliğine bağlanmıştır. Bu sayede kullanıcı “slider” ile yapmış olduğu değer değişimi doğrudan “label” font büyüklüğüne uygulanmış olacaktır. Burada bulunan “label” hedef nesne (target object), “slider” ise kaynak nesne (source object) olarak tanımlayabiliriz.



Resim 3.2: Veri bağlama (Data Binding) mekanizması

Bağlama bilgisi “{Binding ...}” yazımı ile sağlanır. Yapılmak istenen başka bir elementin bir özelliğinin değiştirilmesi olduğundan bağlama bilgisine ek olarak söz konusu kaynak nesnenin ismi (ElementName) ve özellik yolu (Path) bilgisinin verilmesi gereklidir.

Kendi varlık sınıflarınız (Entity Class) ile WPF arayüz denetimleri arasında veri bağlama işlemlerinin yapılabilmesi için sınıfınızın “INotifyPropertyChanged” arayüzünden (interface) bildirim yapılması zorunludur. “INotifyPropertyChanged” arayüzü “PropertyChanged” adı verilen bir olay (event) bildirim ekleyecektir. Asıl önemlisi söz konusu olay bildiriminin tüm WPF kullanıcı denetimleri tarafından dinlenebiliyor olmasıdır. Kısaca varlık sınıfınızın özelliklerindeki bir değişim “PropertyChanged” olay bildirimini tetikler ve bağlı olduğu WPF denetimi bu değişiklikten haberdar olur. Veri bağlama işlemi nesnelerin bağlanan özellikleri arasında tek yönlü (one-way) veya çift yönlüde (two-way) şeklinde gerçekleşir.

Örnek 3.3: Ürün sınıfını WPF denetimlerine bağlayınız.

İlk olarak ürün sınıfını “INotifyPropertyChanged” arayüzü uygulanarak aşağıda verildiği şekliyle oluşturunuz.


```

namespace DataBindingSample
{
    class Urun:INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;
        public void OnPropertyChanged(PropertyChangedEventArgs e)
        {
            if (PropertyChanged != null)
                PropertyChanged(this, e);
        }

        private string modelNo;
        public string ModelNo
        {
            get { return modelNo; }
            set
            {
                modelNo = value;
                OnPropertyChanged(new PropertyChangedEventArgs("ModelNo"));
            }
        }

        private string modelAdi;
        public string ModelAdi
        {
            get { return modelAdi; }
            set
            {
                modelAdi = value;
                OnPropertyChanged(new PropertyChangedEventArgs("ModelAdi"));
            }
        }

        private decimal birimFiyat;
        public decimal BirimFiyat
        {
            get { return birimFiyat; }
            set
            {
                birimFiyat = value;
                OnPropertyChanged(new PropertyChangedEventArgs("BirimFiyat"));
            }
        }
    }
}

```

```
private string aciklama;
public string Aciklama
{
    get { return aciklama; }
    set
    {
        aciklama = value;
        OnPropertyChanged(new PropertyChangedEventArgs("Aciklama"));
    }
}

public Urun(string modelNo, string modelAdi, decimal birimFiyat, string aciklama)
{
    ModelNo = modelNo;
    ModelAdi = modelAdi;
    BirimFiyat = birimFiyat;
    Aciklama = aciklama;
}
}
```

Normal sınıf tasarımından farklı olarak her bir özelliğin (property) “set” bloğunda alan (field) bilgisi değiştirildikten sonra “OnPropertyChanged” metodu çağırılmaktadır. Söz konusu metod “PropertyChanged” olayını, nesnenin kendisi ve değişen özelliğin adı olarak tetikleyecektir. Sonuç olarak çalışma zamanında ürün sınıfına ait bir nesnenin herhangi bir özelliği (property) değiştiğinde ortama “PropertyChanged” olayı (event) fırlatılacaktır. Bu olayı dinleyen aboneler (WPF denetimleri) gereken değişimi algırlar.

Örneğimizin uygulama arayüzünü ve kullanıcı denetimlerine varlık sınıfının özelliklerini bağlamak için tasarım kodunu aşağıda verildiği şekliyle düzenleyiniz.

```

<Window x:Class="DataBindingSample.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="MainWindow" Height="350" Width="525">
  <Grid Name="gridProductDetails">
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="Auto"></ColumnDefinition>
      <ColumnDefinition></ColumnDefinition>
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
      <RowDefinition Height="Auto"></RowDefinition>
      <RowDefinition Height="Auto"></RowDefinition>
      <RowDefinition Height="Auto"></RowDefinition>
      <RowDefinition Height="Auto"></RowDefinition>
      <RowDefinition Height="148*"></RowDefinition>
      <RowDefinition Height="44*" />
    </Grid.RowDefinitions>
    <TextBlock Margin="7">Model No:</TextBlock>
    <TextBox Margin="5" Grid.Column="1" Text="{Binding Path=ModelNo}"/>
    <TextBlock Margin="7" Grid.Row="1">Model Adi:</TextBlock>
    <TextBox Margin="5" Grid.Row="1" Grid.Column="1"
      Text="{Binding Path=ModelAdi}"/>
    <TextBlock Margin="7" Grid.Row="2">Birim Fiyat:</TextBlock>
    <TextBox Margin="5" Grid.Row="2" Grid.Column="1"
      Text="{Binding Path=BirimFiyat}"/>
    <TextBlock Margin="7,7,7,0" Grid.Row="3">Açıklama:</TextBlock>
    <TextBox Margin="7,7,7,9" Grid.Row="4" Grid.ColumnSpan="2"
      TextWrapping="Wrap" Text="{Binding Path=Aciklama}"/>
    <Button Content="Ürün Getir" Grid.Column="1" Height="25" Name="button1"
      HorizontalAlignment="Left" Margin="314,7,0,0" Width="80"
      VerticalAlignment="Top" Click="button1_Click" Grid.Row="5"/>
    <Button Content="Ürün Bilgi" Height="25" HorizontalAlignment="Left"
      Margin="228,7,0,0" Name="button2" Grid.Column="1"
      Grid.Row="5"
      VerticalAlignment="Top" Width="80" Click="button2_Click" />
  </Grid>
</Window>

```

Test verisini taşıyan nesnenin oluşturulması ve veri gösterimini yapacak denetimleri kapsayan “grid” bileşenine veri kaynağı bildirimini kod sayfasında aşağıda verildiği şekilde kodlayınız.

```

namespace DataBindingSample
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            LoadData();
        }

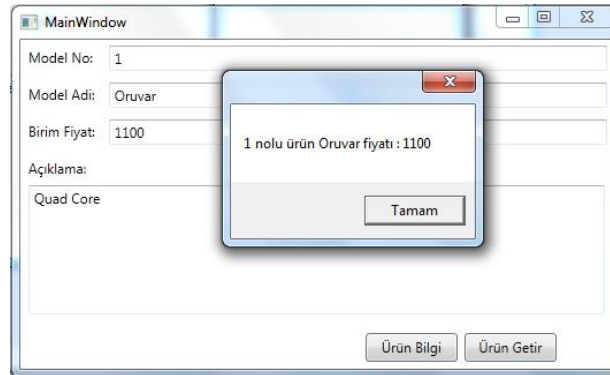
        Urun urun;
        public void LoadData()
        {
            urun = new Urun("1", "Dell", 1200, null);
        }

        private void button1_Click(object sender, RoutedEventArgs e)
        {
            gridProductDetails.DataContext = urun;
        }

        private void button2_Click(object sender, RoutedEventArgs e)
        {
            MessageBox.Show(urun.ModelNo+" nolu ürün "+urun.ModelAdi+" fiyatı : "+urun.BirimFiyat);
        }
    }
}

```

Uygulamayı çalıştırınız. Pencere üzerinden denetimler vasıtasıyla ürün bilgisini değiştiriniz. “Ürün Bilgi” butonuna tıklayınız ve yapmış olduğunuz değişikliklerin uygulandığından emin olunuz.



Resim 3.3: Uygulama test sonucu

UYGULAMA FAALİYETİ

İşlem Basamakları	Öneriler
<p>Üye kaydı yapan bir uygulamada kullanıcının girdiği e-posta bilgisinin geçerliliğini denetleyiniz.</p> <ul style="list-style-type: none">➤ “Üye” varlık sınıfını oluşturunuz. Varlık sınıfını “INotifyPropertyChanged” ve “IDataErrorInfo” arayüzlerinden türetiniz.➤ Pencere tasarımına iki adet metin kutusu yerleştiriniz.➤ Veri bağlama ve doğrulama için “Binding” parametrelerini ve stil tanımlamasını yapınız.➤ Program yüklendiğinde bir üye nesnesini grid bileşeninin kaynağı olarak gösteriniz.	<ul style="list-style-type: none">➤ Üye sınıfı içinde ad ve mail adında iki alan tanımlaması yapınız. E-posta doğrulaması için “System.Text.RegularExpressions” isim alanında bulunan “Regex” sınıfının “IsMatch” özelliğini aşağıda verildiği şekliyle kullanınız. <pre>public string this[string columnName] { get { string hata=null; string desen=@"^([0-9a-zA-Z]([-\.\\w]*[0-9a-zA-Z])*@[([0-9a-zA-Z]([-\\w]*[0-9a-zA-Z])\.)+[a-zA-Z]{2,9})\$"; if (columnName == "Mail") { if (this.Mail!=null && !Regex.IsMatch(this.Mail, desen)) { hata = "Geçersiz mail"; } } return hata; } }</pre> <ul style="list-style-type: none">➤ Veri bağlama ve stil tanımlama işlemleri için örnek 3.3'ten faydalanabilirsiniz.

ÖLÇME VE DEĞERLENDİRME

Aşağıdaki soruları dikkatlice okuyarak doğru seçeneği işaretleyiniz.

1. Aşağıdaki arayüzlerden hangisi veri doğrulama için kullanılır?
A) INotifyPropertyChanged
B) IDataErrorInfo
C) IValidationInfo
D) IComparer
2. Veri bağlı bir denetim için tanımlanan “Path” parametresinin görevi nedir?
A) Doğrulamanın yapılacağı bilgisini verir.
B) Veri kaynağını işaret eder.
C) Verinin varlık sınıfın bir özelliğinden geldiğini gösterir.
D) Denetimin doğrulamayı ne zaman yapacağını gösterir.
3. Doğrulama işleminde hata bilgisinin gösterimini sağlayan tetikleme özelliği hangisidir?
A) ValidationUpdate
B) ValidationTrigger
C) UpdateSourceTrigger
D) Validation.HasError
4. Aşağıdaki arayüzlerden hangisi varlık nesnelерinin WPF denetimlerine bağlanması için bildirim zorunludur?
A) INotifyPropertyChanged
B) IDataErrorInfo
C) IValidationInfo
D) IComparer

DEĞERLENDİRME

Cevaplarınızı cevap anahtarıyla karşılaştırınız. Yanlış cevap verdiğiniz ya da cevap verirken tereddüt ettiğiniz sorularla ilgili konuları faaliyete geri dönerek tekrarlayınız. Cevaplarınızın tümü doğru ise “Modül Değerlendirme”ye geçiniz.

MODÜL DEĞERLENDİRME

Aşağıdaki soruları dikkatlice okuyarak doğru seçeneği işaretleyiniz.

1. Application nesnesinin görevi nedir?
A) İşletim sistemi için uygulamanın yönetilebilir olmasını sağlamak
B) Uygulamanız içindeki pencere yönetimini sağlamak
C) Referans dosyalarının uygulama içindeki yerini belirlemek
D) Tasarım hakkında bilgi vermek
2. Aşağıdakilerden hangisi bir denetimi, kapsayan bileşenin sınırlarına olan uzaklığını belirtmek için kullanılan özelliğidir?
A) Left
B) Top
C) Margin
D) Width
3. Aşağıdakilerden hangisi denetimlere ait “Opacity” özelliğini tanımlar?
A) Zemin rengini belirler
B) Şeffaflığı belirler
C) Yazı rengini belirler
D) Şeklini belirler
4. Aşağıdakilerden hangisi ortak iletişim kutularından değildir?
A) FontDialog
B) SaveFileDialog
C) OpenFileDialog
D) PrintDialog
5. Aşağıdaki özelliklerden hangisi “Canvas” bileşeni içindeki nesnenin görünürlük olarak önde olmasını sağlar?
A) Canvas.Top
B) Canvas.Bottom
C) Panel.ZIndex
D) Panel.BringToFront

DEĞERLENDİRME

Cevaplarınızı cevap anahtarıyla karşılaştırınız. Yanlış cevap verdiğiniz ya da cevap verirken tereddüt ettiğiniz sorularla ilgili konuları faaliyete geri dönerek tekrarlayınız. Cevaplarınızın tümü doğru ise bir sonraki modüle geçmek için öğretmeninize başvurunuz.

CEVAP ANAHTARLARI

ÖĞRENME FAALİYETİ-1'İN CEVAP ANAHTARI

1	D
2	B
3	A
4	B
5	A

ÖĞRENME FAALİYETİ-2'NİN CEVAP ANAHTARI

1	İletişim Kutuları
2	ContextMenu
3	MenuItem
4	Header
5	ShowDialog
6	Click
7	Dosya Aç
8	Dosya Kaydet

ÖĞRENME FAALİYETİ-3'ÜN CEVAP ANAHTARI

1	C
2	B
3	D
4	A

MODÜL DEĞERLENDİRMENİN CEVAP ANAHTARI

1	A
2	C
3	B
4	A
5	C

KAYNAKÇA

- John Sharp, **Microsoft Visual C# 2008**, Microsoft Press / U.S, 2008.
- Chris Sells and Ian Griffiths, **Programming WPF**, O'Reilly / U.S, 2007.
- Matthew MacDonald, **Pro WPF in C# 2010**, Apress/ U.S,2010.
- MSDN, <http://msdn.microsoft.com> (12.07.2012/13.00)
- CodeProject , <http://www.codeproject.com> (10.07.2012/13.00)
- B.Selim ŞENYURT <http://www.buraksenyurt.com> (19.07.2012/13.00)
- Daron YÖNDEM <http://daron.yondem.com/tr/blog/> (20.07.2012/13.00)