Course 2D_WPF: 2D-Computer Graphics with C# + WPF
Chapter C1: Comments to the Intro Project

German: Alle Links zeigen auf englische Hilfetexte. Viele dieser Texte gibt es aber auch auf Deutsch. Sie müssen nur in allen URLs folgende 5 Buchstaben verändern: `en-us` → `de-de` wie in folgendem Beispiel:
`english:` `http://msdn2.microsoft.com/en-us/library/system.windows.shapes.aspx`
`deutsch:` `http://msdn2.microsoft.com/de-de/library/system.windows.shapes.aspx`

using namespaces

//The .NET Framework 3.5 Class Library FCL contains thousands of classes.
For better orientation it is subdivided into "namespaces" each containing a subset of related classes.
Any class and its members have to be called by writing the full tree of its namespace which forces the programmer to write spaghetti-long identifiers. With the "`using`" directive You can shorten such long identifiers and the compiler will complete the missing namespaces.

`using System;` //Home of the base class of all classes "`System.Object`" and
of all primitive data types such as `Int32, Int16, double, string`. Link: `System`.

`using System.Windows;` //Home of the "`Window`" class (base class of our main window `window1)` and its method `Application().Run`. Link: `System.Windows`.

`using System.Windows.Controls;` //Home of the base class of GUI-Elements
such as `Button, Canvas, TextBox` etc. Link: `System.Windows.Controls`.

`using System.Windows.Shapes;` //Home of the "`Shape`" class = base class of
`Line, Polyline, Rectangle` etc. Link: `System.Windows.Shapes`.

`using System.Windows.Threading;` //Home of the "`DispatcherTimer`" class.
Link: `System.Windows.Threading`.

Entry to start our WPF program: `public class window1 : Window`

//We derive our `window1` from the class `Window`, which the compiler automatically finds in the `System.Windows` namespace.

```
[STAThread] static void Main() { new Application().Run( new window1() ); }
```
//Create a single thread instance of `window1` and
ask the operating system to start it as main window of our program. Link: `Application Class`.

`Canvas myCanvas = new Canvas();` //Create a `Canvas` object to draw `TextBoxes, Lines` on its surface and which covers the client space of `window1`. Link: `Canvas`.

```
Line line1 = new Line();
Line line2 = new Line();
```
//Create two `Line` objects to be drawn on the `Canvas`. Link: `Line`.

```
Rectangle rect = new Rectangle();
Ellipse elli = new Ellipse();
```
//Create a `Rectangle` and an `Ellipse` object to be drawn on the `Canvas`. Link: `Rectangle`.

```
DockPanel myPanel = new DockPanel();
```
//Create a surface where child elements can be docked on either side. Link: `DockPanel`.

```
TextBox left = new TextBox();
TextBox top = new TextBox();
TextBox right = new TextBox();
TextBox bottom = new TextBox();
TextBox central = new TextBox();
```
//Create five `TextBox` objects. Link: `TextBox`.

```
double zoom  = 1.1; //Sets the initial zoom to 10% per step.
double angle = 0  ; //Rotation of the Brush origin
```

```
Random r = new Random(); //Create a Random-value generator object. Link: Random.
```

```
Byte r1, g1, b1, r2, g2, b2; //Create six bytes intended to compose 2 RGB-colors.
```

Constructor `public window1()` inside `public class window1`

```
this.Top = this.Left = 50;
this.Width = this.Height = 500;
this.Title = "intro1"; //Define the position of the upper left corner, the initial width and height and
```
the title text of `window1`. Link: `Window`.

```
myCanvas.Children.Add( line1 );
myCanvas.Children.Add( line2 );
myCanvas.Children.Add( rect );
myCanvas.Children.Add( elli );
myCanvas.Children.Add( myPanel ); //myCanvas adopts five children.
```
`Children` is a property inherited from `Panel`. Link: `Panel.Children Property`.

```
myPanel.Children.Add( top     ); DockPanel.SetDock( top   , Dock.Top    ); top   .Text = "top";
myPanel.Children.Add( bottom  ); DockPanel.SetDock( bottom, Dock.Bottom ); bottom.Text = "bottom";
myPanel.Children.Add( left    ); DockPanel.SetDock( left  , Dock.Left   ); left  .Text = "left";
myPanel.Children.Add( right   ); DockPanel.SetDock( right , Dock.Right  ); right .Text = "right";
myPanel.Children.Add( central );
```
//`myPanel` adopts five children. Four of them are docked to the 4 borders of `myPanel` and
obtain their text strings. The fifth centers itself automatically in the middle of the remaining space.
Link: `DockPanel.SetDock Method`.

```
Background = new LinearGradientBrush( Colors.Red, Colors.Blue, 90 );
```
//The background color of `window1` is changing from red to blue from top to bottom = 90 degrees.
Link: `LinearGradientBrush Class`.

```
Foreground = new SolidColorBrush( Color.FromRgb( 0, 0, 200 ) );
```
//The foreground color of `window1` is blue. Link: `SolidColorBrush Class`.

```
FontFamily = new FontFamily( "Courier New" );
FontSize = 12; //Let us use a simple font. Link: Control.FontFamily Property.
```

```
foreach( TextBox text in myPanel.Children )
{ text.HorizontalAlignment = HorizontalAlignment.Center;
  text.VerticalAlignment   = VerticalAlignment.Center;
} //Align all stings in the centers of their TextBoxes.
```
Link: `FrameworkElement.HorizontalAlignment Property`.

```
foreach( Object obj in myCanvas.Children )
{ if ( obj.GetType() == typeof(DockPanel) ) continue; //Forget about child myPanel.
  ((Shape)obj).Stroke = Brushes.Black;
  ((Shape)obj).StrokeThickness = 5;
}
```
//The shape-children of `myCanvas` are: two lines, the rectangle and the ellipse.
They should be outlined with thick black contours. Link: `Shape.Stroke Property`.

```
rect.Fill = Brushes.White;
```
//Inside the black border the rectangle should be filled with white. Link: `Shape.Fill Property`.

```
r1 = (Byte)r.Next(255); g1 = (Byte)r.Next(255); b1 = (Byte)r.Next(255);
r2 = (Byte)r.Next(255); g2 = (Byte)r.Next(255); b2 = (Byte)r.Next(255);
```
//Initialize the bytes with random values. Link: `Random.Next Method`.

```
DispatcherTimer myTimer = new DispatcherTimer();
myTimer.Interval = TimeSpan.FromMilliseconds( 40 );
myTimer.Tick += TimerOnTick;
myTimer.Start(); //Setup a timer event generator that sends as many timer messages as reasonable.
```
Link: `DispatcherTimer Class`.

Timer event handler `void TimerOnTick( Object sender, EventArgs args )`
inside `public class window1`

`TimerOnTick(...)` is responsible for permanently zooming the window up and down.

```
if ( myCanvas.ActualWidth < 200 ) zoom = 1.1;
```
//After reaching its smallest width of 200 the window will rapidly zoom up in steps of 10%.

```
if ( myCanvas.ActualWidth > 800 ) zoom = 0.99;
```
//After reaching its biggest width of 800 the window will slowly zoom down in steps of 1%.

```
this.Width    *= zoom;
this.Height   *= zoom;
this.FontSize *= zoom;
```
//Width, Height and FontSize should behave the same way.

Resize-Event handler
`protected override void OnRenderSizeChanged( SizeChangedInfo sizeInfo )`
inside `public class window1`

This event handler updates the strings in the `central`-TextBox and adjusts the endpoints of the diagonals and the `Left, Top, Width, Height`-properties of `rect` and `elli`.
It fills `elli` using an animated `RadialGradientBrush` with animated colors.

```
String s1 = "Hello World " + DateTime.Now.ToString() + "\n";
```
//Concatenate two strings and a new-line-escape-character.

```
int width = Convert.ToInt32( this.Width );
int height = Convert.ToInt32( this.Height );
```
//Get the current window size and round it to integer values.

```
String s2 = "Window Size = " + width.ToString() + " x " + height.ToString() + "\n";
```
//Concatenate four strings and a new-line-escape-character.

```
width = Convert.ToInt32( myCanvas.ActualWidth );
height = Convert.ToInt32( myCanvas.ActualHeight );
```
//Get the size of the current client area = canvas size and round it to integer values.

```
String s3 = "Client Size = " + width.ToString() + " x " + height.ToString() + "\n";
```
//Concatenate four strings and a new-line-escape-character.

```
String s4 = String.Format( "Font Size = {0,2:F1}", this.FontSize );
```
//Construct a string from a float variable `this.FontSize` with 2 digits in front and one digit behind the decimal point. Link: `String.Format Method`.

```
central.Text = s1 + s2 + s3 + s4;
```
//Write four lines into the `central`-TextBox.

```
line1.X1 = 0; line1.Y1 = 0; line1.X2 = myCanvas.ActualWidth; line1.Y2 = myCanvas.ActualHeight;
line2.X1 = myCanvas.ActualWidth; line2.Y1 = 0; line2.X2 = 0; line2.Y2 = myCanvas.ActualHeight;
```
//Set line1-start to the upper left, line1.end to the lower right,
line2.start to the upper right and line2.end to the lower left corner of `myCanvas`.

```
Canvas.SetLeft( rect, myCanvas.ActualWidth /5 );
Canvas.SetLeft( elli, myCanvas.ActualWidth /5 );
Canvas.SetTop ( rect, myCanvas.ActualHeight/5 );
Canvas.SetTop ( elli, myCanvas.ActualHeight/5 );
rect .Width  = elli.Width  = 3 * myCanvas.ActualWidth  / 5;
rect .Height = elli.Height = 3 * myCanvas.ActualHeight / 5;
```
//Set the upper left corners of both `rect` and `elli` to 20% left and 20% down.
//Set Width and Height of both `rect` and `elli` to 60%.

```
Color color1 = Color.FromRgb( r1++, g1++, b1++ );
Color color2 = Color.FromRgb( r2++, g2++, b2++ );
```
//Increment all red, green and blue values and create new colors. Values beyond `255` restart at `0`.

```
RadialGradientBrush brush = new RadialGradientBrush( color1, color2 );
brush.SpreadMethod = GradientSpreadMethod.Repeat;
brush.RadiusX = brush.RadiusY = 0.1;
brush.GradientOrigin = new Point( 0.5+0.05*Math.Cos(angle), 0.5+0.05*Math.Sin(angle) );
elli.Fill = brush;
angle += Math.PI / 32;
```

//Create a new radial brush with 5 rings with a slightly rotating origin and apply it to `elli`.
The origin rotates with a velocity of 180/32 = 5.625 degrees/step. Link: `RadialGradientBrush Class`.

```
myPanel.Width = myCanvas.ActualWidth;
myPanel.Height = myCanvas.ActualHeight;
```

//Adjust the size of the DockPanel `myPanel` to the current size of the canvas.

`Content = myCanvas;` //Show everything. Link: `WPF Content Model`.