PEF – 3528 – Ferramentas Computacionais na Mecânica das Estruturas: Criação e Concepção

Prof. Dr. Rodrigo Provasi

e-mail: provasi@usp.br

Sala 09 – LEM – Prédio de Engenharia Civil

Interfaces em C#

Oxyplot

Oxyplot

- Como dito na aula de bibliotecas, o *Oxyplot* é uma biblioteca que permite a exibição de gráficos.
- A melhor maneira de trabalhar é fazer um *binding* na propriedade *PlotModel*.

Interfaces em C#

Shapes

Shapes

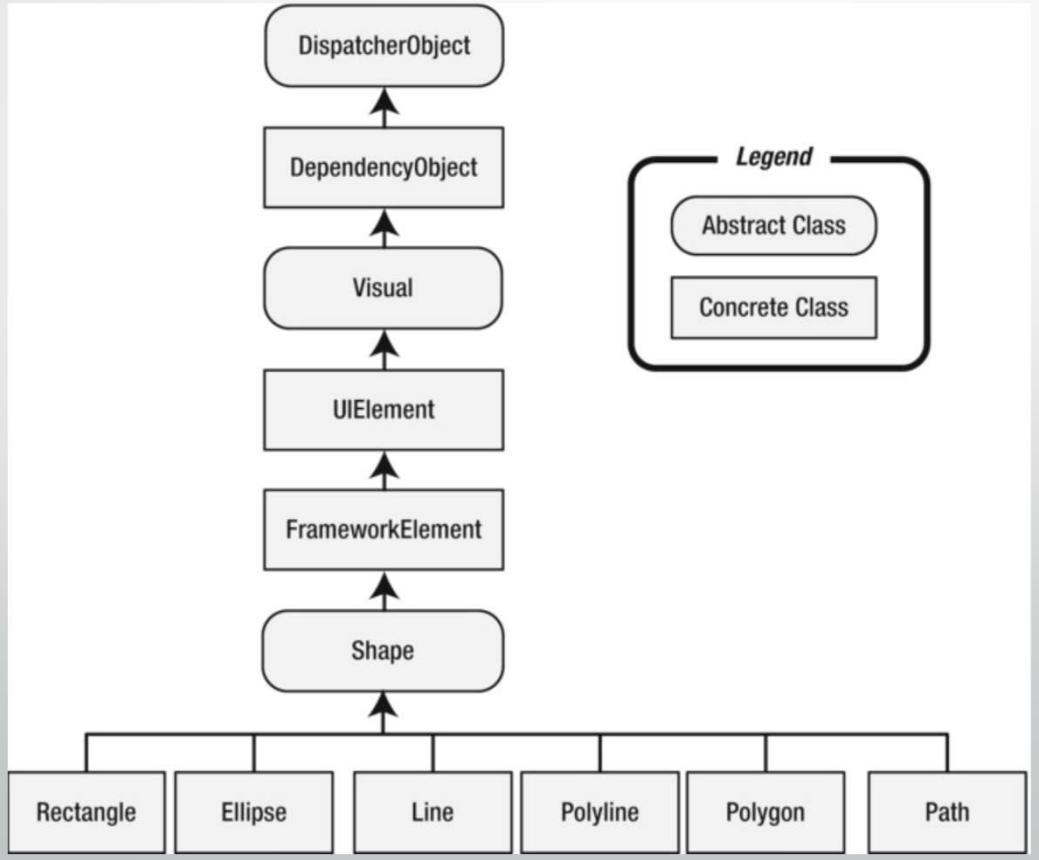
- *Shapes* são a forma mais básica de fazer elementos de desenho em WPF.

- Características importantes:
  - *Shapes draw themselves*: You don't need to manage the invalidation and painting process. For example, you don't need to manually repaint a shape when content moves, the window is resized, or the shape's properties change.

Shapes

- *Shapes are organized in the same way as other elements*. In other words, you can place a shape in any of the layout containers you learned about in Chapter 3. (Although the Canvas is obviously the most useful container, because it allows you to place shapes at specific coordinates, which is important when you're building a complex drawing out of multiple pieces.)
- *Shapes support the same events as other elements*. That means you don't need to go to any extra work to deal with focus, key presses, mouse movements, and mouse clicks. You can use the same set of events you would use with any element, and you have the same support for tooltips, context menus, and drag-and-drop operations.

Shapes


DispatcherObject
DependencyObject
Visual
UIElement
FrameworkElement
Shape
Rectangle
Ellipse
Line
Polyline
Polygon
Path
Legend
Abstract Class
Concrete Class

Propriedades

- **Fill:** Sets the brush object that paints the surface of the shape (everything inside its borders).
- **Stroke:** Sets the brush object that paints the edge of the shape (its border).
- **StrokeThickness:** Sets the thickness of the border, in device-independent units. When drawing a line, WPF splits the width on each side. So a line that's 10 units wide gets 5 units of space on each side of where a single-unit line would be drawn. If you give a line an odd-number thickness, the line will have a fractional width on each side. For example, an 11-unit line has 5.5 units of space on each side. This pretty much guarantees that the line won't line up evenly with the display pixels of your monitor, even if it's running at 96 dpi resolution, so you'll end up with a slightly fuzzy anti-aliased edge. You can use the SnapsToDevicePixels property to clean this up if it bothers you (as described in the section "Pixel Snapping" later in this chapter).

Propriedades

- **StrokeStartLineCap and StrokeEndLineCap:** Determine the contour of the edge of the beginning and end of the line. These properties have an effect only for the Line, the Polyline, and (sometimes) the Path shapes. All other shapes are closed, and so have no starting and ending point.
- **StrokeDashArray, StrokeDashOffset, and StrokeDashCap:** Allow you to create a dashed border around a shape. You can control the size and frequency of the dashes, and the contour of the edge where each dash line begins and ends.
- **StrokeLineJoin and StrokeMiterLimit:** Determine the contour of the shape's corners. Technically, these properties affect the *vertices* where different lines meet, such as the corners of a Rectangle. These properties have no effect for shapes without corners, such as Line and Ellipse.

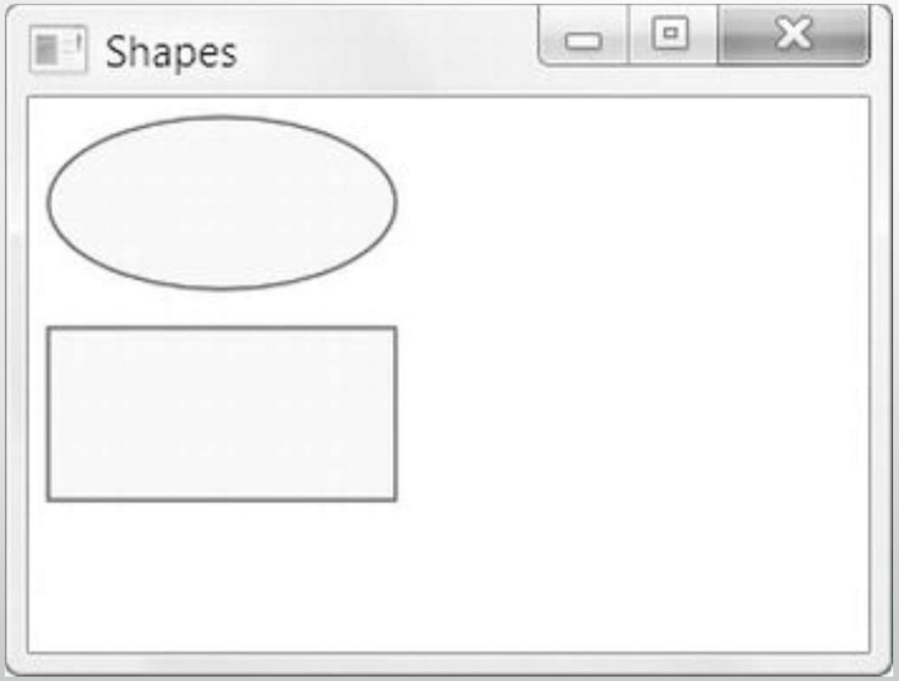
Propriedades

- **Stretch:** Determines how a shape fills its available space. You can use this property to create a shape that expands to fit its container. You can also force a shape to expand in one direction by using a Stretch value for the *HorizontalAlignment* or *VerticalAlignment* properties (which are inherited from the *FrameworkElement* class).
- **DefiningGeometry:** Provides a Geometry object for the shape. A Geometry object describes the coordinates and size of a shape without including the *UIElement* plumbing, such as the support for keyboard and mouse events.
- **GeometryTransform:** Allows you to apply a Transform object that changes the coordinate system that's used to draw a shape. This allows you to skew, rotate, or displace a shape. Transforms are particularly useful when animating graphics.
- **RenderedGeometry:** Provides a Geometry object that describes the final, rendered shape.

Shapes

```
<StackPanel>
    <Ellipse Fill="Yellow" Stroke="Blue‟ Height="50" Width="100" Margin="5" HorizontalAlignment="Left"></Ellipse>
    <Rectangle Fill="Yellow" Stroke="Blue‟ Height="50" Width="100" Margin="5" HorizontalAlignment="Left"></Rectangle>
</StackPanel>
```

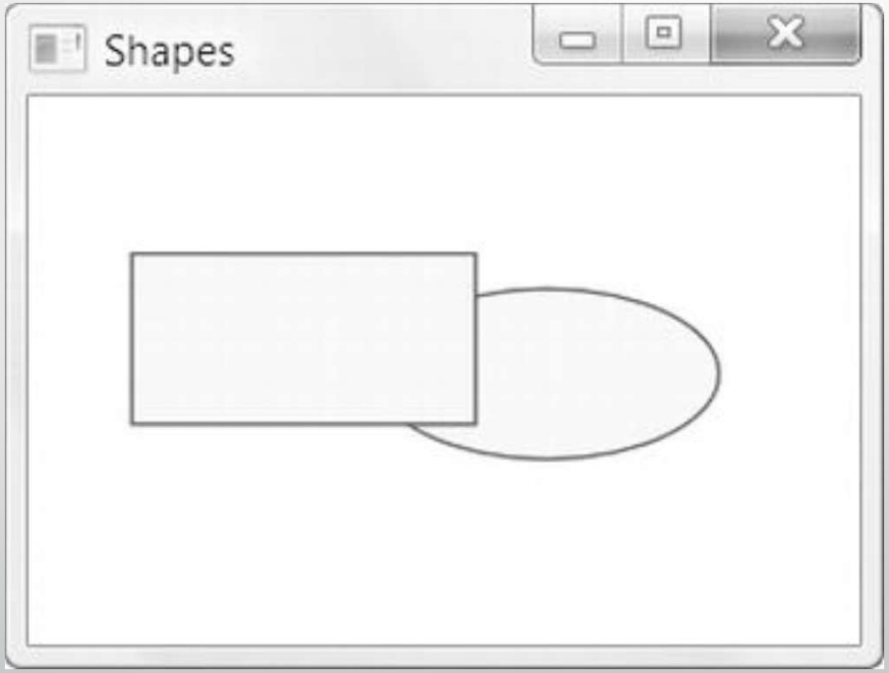
Shapes



Shapes

```
<Canvas>
    <Ellipse Fill="Yellow" Stroke="Blue" Canvas.Left="100" Canvas.Top="50“
Width="100" Height="50"></Ellipse>
    <Rectangle Fill="Yellow" Stroke="Blue" Canvas.Left="30" Canvas.Top="40“
Width="100" Height="50"></Rectangle>
</Canvas>
```

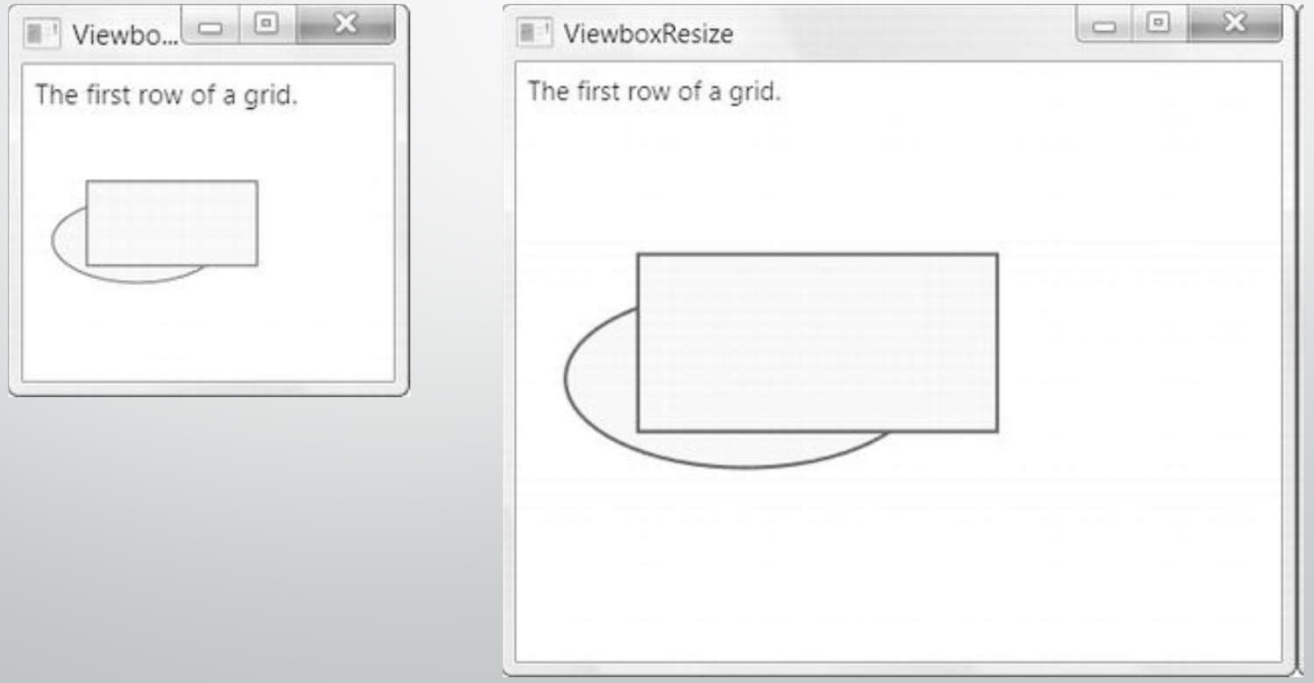
Shapes



Shapes

```
<Grid Margin="5">
        <Grid.RowDefinitions>
                <RowDefinition Height="Auto"></RowDefinition>
                <RowDefinition Height="*"></RowDefinition>
        </Grid.RowDefinitions>
        <TextBlock>The first row of a Grid.</TextBlock>
        <Viewbox Grid.Row="1" HorizontalAlignment="Left" >
                <Canvas Width="200" Height="150">
                        <Ellipse Fill="Yellow" Stroke="Blue" Canvas.Left="10" Canvas.Top="50‟ Width="100" Height="50" HorizontalAlignment="Left"></Ellipse>
                        <Rectangle Fill="Yellow" Stroke="Blue" Canvas.Left="30" Canvas.Top="40‟ Width="100" Height="50" HorizontalAlignment="Left"></Rectangle>
                </Canvas>
        </Viewbox>
</Grid>
```

Shapes


Viewbo...
The first row of a grid.
ViewboxResize
The first row of a grid.

Line

- Linhas permitem desenhar linhas em um objeto no WPF.
- Exemplo:

```
<Line Stroke="Blue" X1="0" Y1="0" X2="10" Y2="100"></Line>
```

Line

<Line Stroke="Blue" X1="0" Y1="0" X2="10" Y2="100“ Canvas.Left="5“ Canvas.Top="100"></Line>

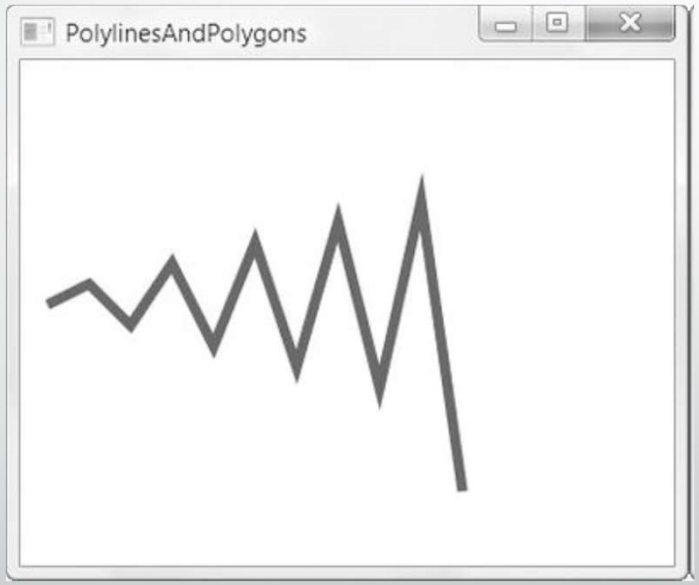
- A linha está em um canvas e trata o (0,0) da linha como sendo o (5,100) do canvas.

Polyline

- Cria uma sequencia de linhas:

```
<Canvas>
    <Polyline Stroke="Blue" StrokeThickness="5" Points="10,150 30,140 50,160 70,130 90,170 110,120 130,180 150,110 170,190 190,100 210,240" >
    </Polyline>
</Canvas>
```

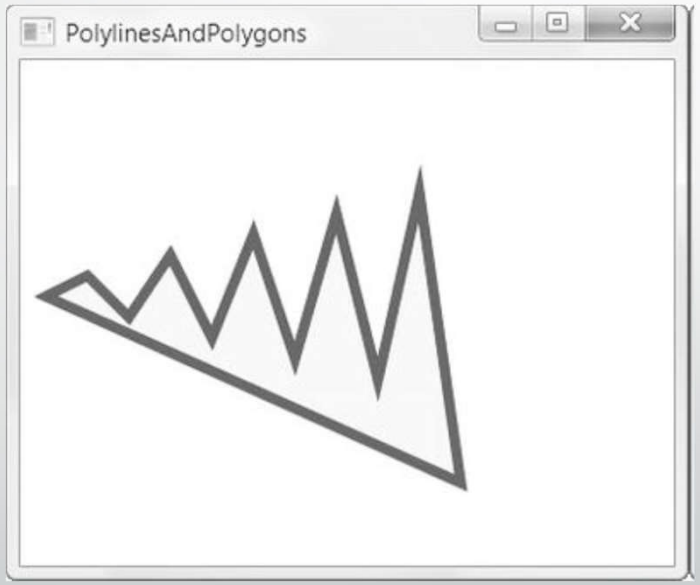
Polyline



Polygon

- O polígono é similar a *polyline*, porém, a forma é fechada ligando-se o último ponto ao primeiro:

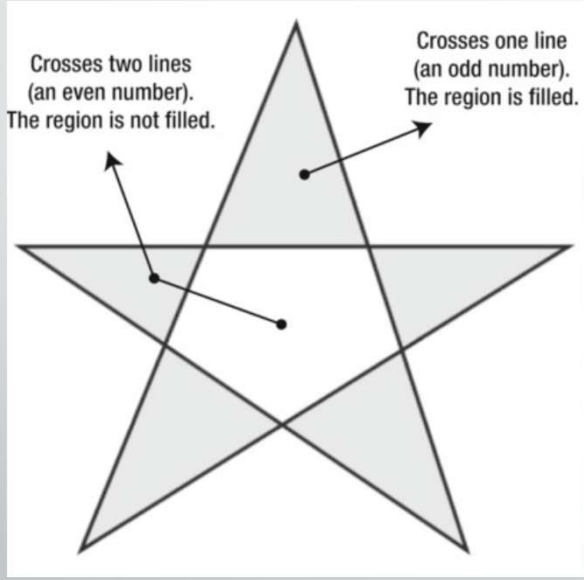
Polygon



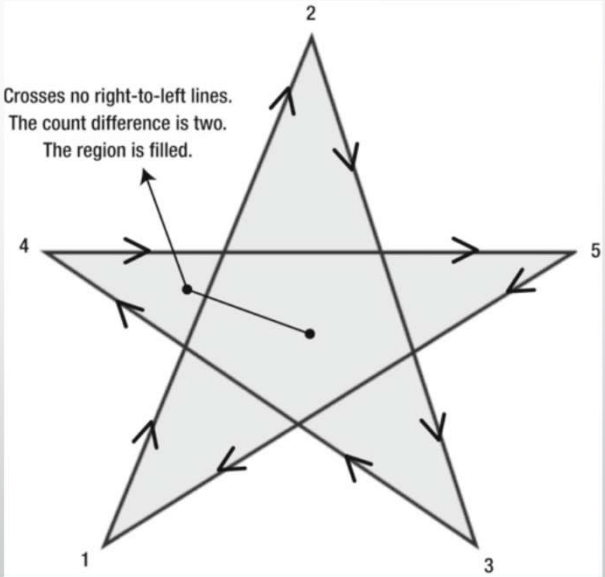
Polygon

- Para um polígono em que as linhas se cruzam, a regra de preenchimento é importante (*FillRule)*.

Polygon



EvenOdd



Nonzero

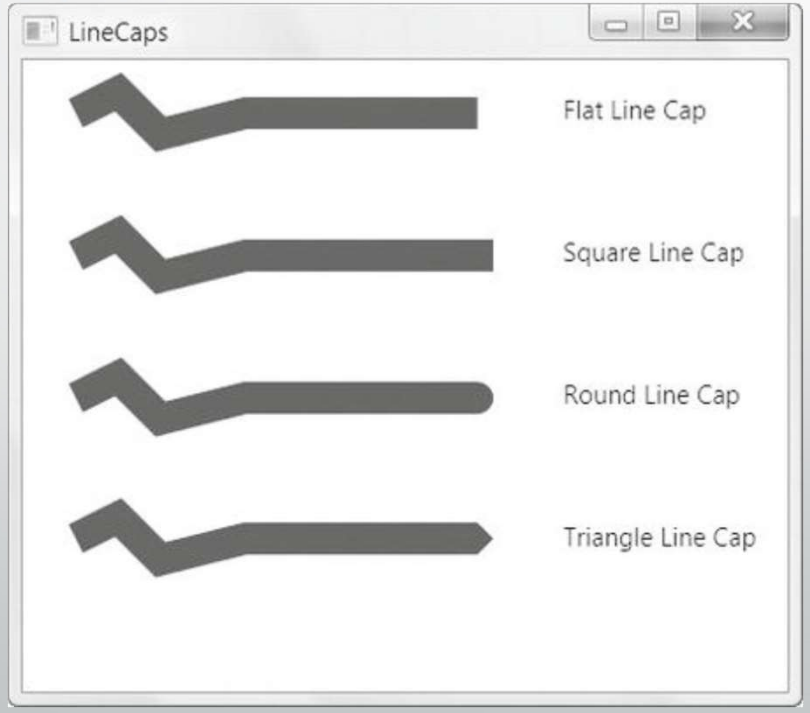
Polygon

<Polygon Stroke="Blue" StrokeThickness="1" Fill="Yellow‟ Canvas.Left="10" Canvas.Top="175" FillRule="Nonzero‟ Points="15,200 68,70 110,200 0,125 135,125">

</Polygon>

Line Caps

- É possível controlar o acabamento das linhas:

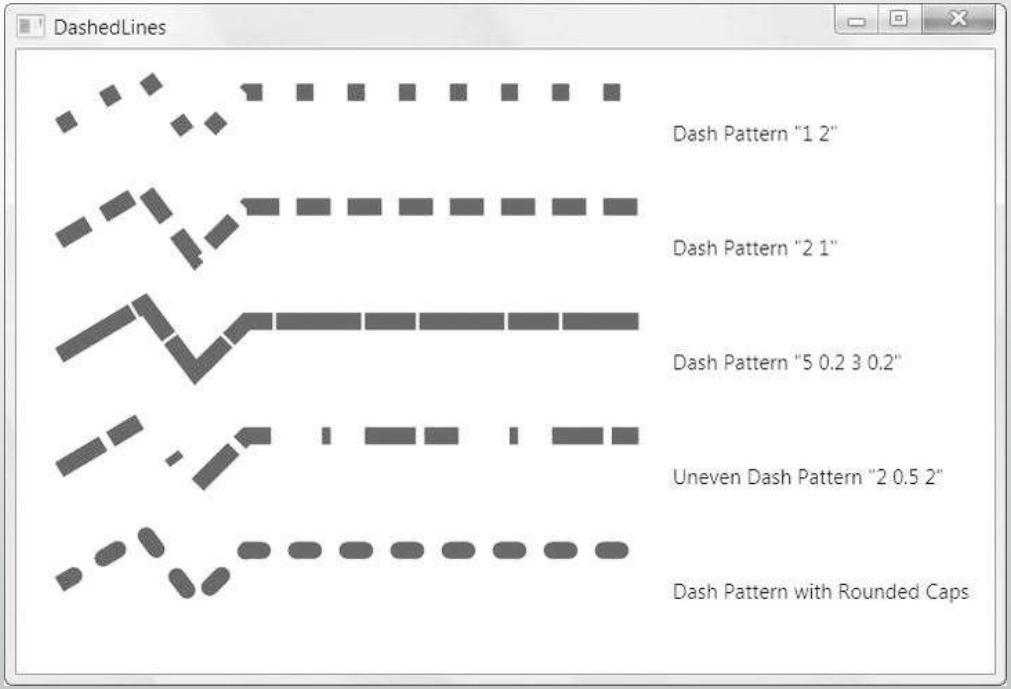


Line Joints

- Também é possível controlar as transições das linhas:

Dashes

```
<Polyline Stroke="Blue" StrokeThickness="14“
StrokeDashArray="1 2“ Points="10,30 60,0 90,40
120,10 350,10">
</Polyline>
```



Brushes

SolidColorBrush	Paints an area using a single continuous color.
LinearGradientBrush	Paints an area using a gradient fill, a gradually shaded fill that changes from one color to another (and, optionally, to another and then another, and so on).
RadialGradientBrush	Paints an area using a radial gradient fill, which is similar to a linear gradient, except that it radiates out in a circular pattern starting from a center point.
ImageBrush	Paints an area using an image that can be stretched, scaled, or tiled.
DrawingBrush	Paints an area using a Drawing object. This object can include shapes you've defined and bitmaps.
VisualBrush	Paints an area using a Visual object. Because all WPF elements derive from the Visual class, you can use this brush to copy part of your user interface (such as the face of a button) to another area. This is useful when creating fancy effects, such as partial reflections.
BitmapCacheBrush	Paints an area using the cached content from a Visual object. This makes it similar to VisualBrush, but more efficient if the graphical content needs to be reused in multiple places or repainted frequently.

Gradients

```
<Rectangle Width="150" Height="100">
        <Rectangle.Fill>
                <LinearGradientBrush StartPoint="0,0" EndPoint="1,1">
                        <GradientStop Color="Yellow" Offset="0.0" />
                        <GradientStop Color="Red" Offset="0.25" />
                        <GradientStop Color="Blue" Offset="0.75" />
                        <GradientStop Color="LimeGreen" Offset="1.0" />
                </LinearGradientBrush>
        </Rectangle.Fill>
</Rectangle>
```

Transforms

- É possível aplicar transformações às formas:

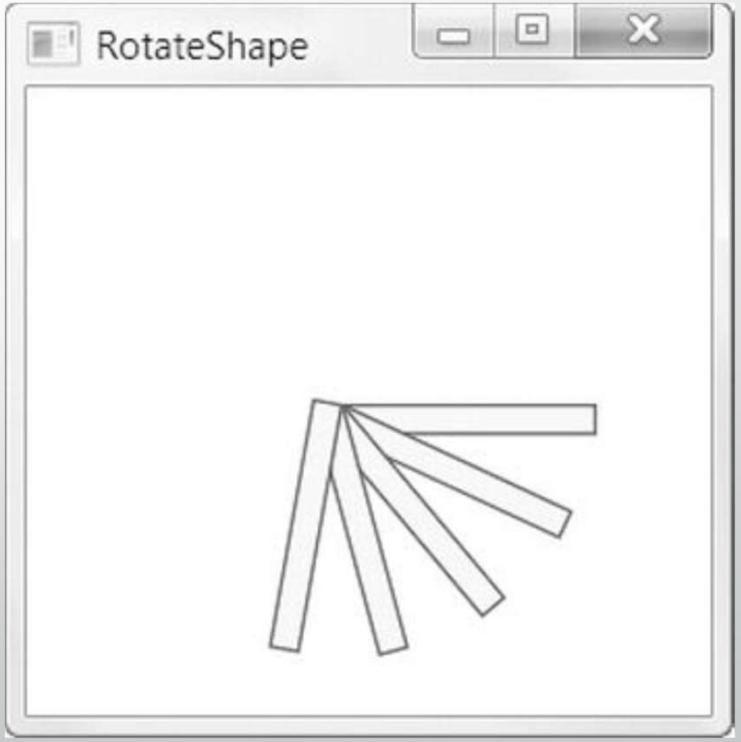
TranslateTransform	Displaces your coordinate system by some amount. This transform is useful if you want to draw the same shape in different places.	X, Y
RotateTransform	Rotates your coordinate system. The shapes you draw normally are turned around a center point you choose.	Angle, CenterX, CenterY
ScaleTransform	Scales your coordinate system up or down, so that your shapes are drawn smaller or larger. You can apply different degrees of scaling in the X and Y dimensions, thereby stretching or compressing your shape.	ScaleX, ScaleY, CenterX, CenterY

Transforms

SkewTransform	Warps your coordinate system by slanting it a number of degrees. For example, if you draw a square, it becomes a parallelogram.	AngleX, AngleY, CenterX, CenterX
MatrixTransform	Modifies your coordinate system by using matrix multiplication with the matrix you supply. This is the most complex option; it requires some mathematical skill.	Matrix
TransformGroup	Combines multiple transforms so they can all be applied at once. The order in which you apply transformations is important because it affects the final result. For example, rotating a shape (with RotateTransform) and then moving it (with TranslateTransform) sends the shape off in a different direction than if you move it and *then* rotate it.	N/A

Transforms

```
<Rectangle Width="80" Height="10"
Stroke="Blue" Fill="Yellow"
Canvas.Left="100" Canvas.Top="100">
    <Rectangle.RenderTransform>
        <RotateTransform Angle="25" />
    </Rectangle.RenderTransform>
</Rectangle>
```



Transforms

```
<Rectangle Width="80" Height="10" Stroke="Blue"
Fill="Yellow" Canvas.Left="100" Canvas.Top="100">
    <Rectangle.RenderTransform>
        <RotateTransform Angle="25" CenterX="45"
CenterY="5" />
    </Rectangle.RenderTransform>
</Rectangle>
```