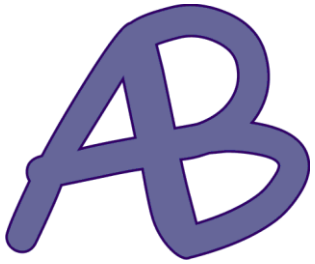


Wykład
Canvas

Canvas

Canvas to kontrolka w WPF (Windows Presentation Foundation) służąca do ręcznego pozycjonowania elementów w układzie współrzędnych X-Y.

W przeciwieństwie do innych paneli (layoutów), takich jak StackPanel czy Grid, Canvas nie narzuca automatycznego ułożenia swoich elementów. Zamiast tego możemy określać położenie kontrolek i kształtów, korzystając z właściwości takich jak Canvas.Left, Canvas.Top, czy Canvas.ZIndex.

- ✓ Elementy wewnątrz Canvas są rysowane na podstawie współrzędnych (X, Y).
- ✓ Możliwość definiowania nakładania się elementów (głębia, kolejność rysowania) przy pomocy właściwości Canvas.ZIndex.
- ✓ Canvas jest wykorzystywany m.in. do tworzenia diagramów, rysunków wektorowych czy własnych kontrolerów do gier 2D.

Deklaracja Canvas w XAML

Deklaracja Canvas w XAML

```
<Canvas x:Name="canvas"
        Background="White"
        MouseMove="canvas_MouseMove">

    <Border x:Name="opis_kursora"
            BorderBrush="Black"
            BorderThickness="1">
        <Label Name="pozycja" Content="0 ; 0"/>
    </Border>

</Canvas>
```

Uwaga: tło obiektu canvas jest istotne, aby mógł on przechwytywać zdarzenia – przezroczysty canvas stanie się „przezroczysty” także dla zdarzeń myszy

Przykładowa zawartość – obiekt leżący na canvasie

Właściwości pozycjonujące na canwasie:

- Canvas.Left="…", Canvas.Top="…": służą do ustalenia położenia w pionie i poziomie.
- Canvas.ZIndex="…": opcjonalnie można ustawić kolejność rysowania elementów, jeśli się nakładają.

Deklaracja Canvas w XAML

WPF udostępnia różne prymitywy graficzne które można rysować na Canvasie:

- Line – linia prostokątna określona punktami początkowymi i końcowymi (X1, Y1, X2, Y2).
- Rectangle – prostokąt z opcjonalnie zaokrąglonymi rogami (RadiusX, RadiusY).
- Ellipse – elipsa (z reguły służy także do rysowania kół).
- Polygon – wielokąt wypełniony, którego wierzchołki definiowane są przez kolekcję punktów (Points).
- Polyline – linia łamana (przedstawiona ciągiem segmentów), podobnie jak w Polygon, z tą różnicą, że końce kształtu nie są domykane wypełnieniem.
- Path – najbardziej uniwersalny kształt, przyjmujący różne geometrie wektorowe (np. łuki, krzywe Beziera). Właściwość Data (typu Geometry) pozwala definiować nawet złożone kształty.

Deklaracja Canvas w XAML

Każdy z tych prymitywów dziedziczy po klasie Shape, przez co posiada szereg wspólnych właściwości związanych ze sposobem rysowania, m.in.:

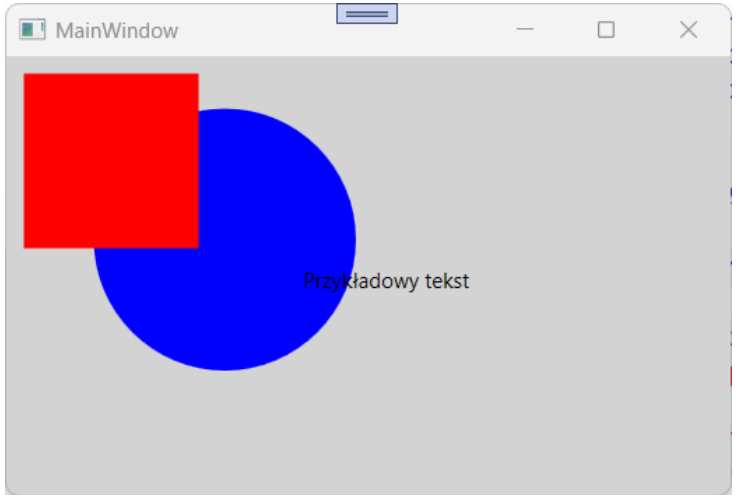
- Fill: pozwala ustawić pędzel (np. kolor, gradient) wypełnienia;
- Stroke: określa kolor i styl (np. przerywana, ciągła) obrysu;
- StrokeThickness: reguluje grubość obrysu;
- StrokeDashArray, StrokeDashOffset itp.: zaawansowane właściwości związane z rysowaniem linii przerywanych.

Deklaracja Canvas w XAML

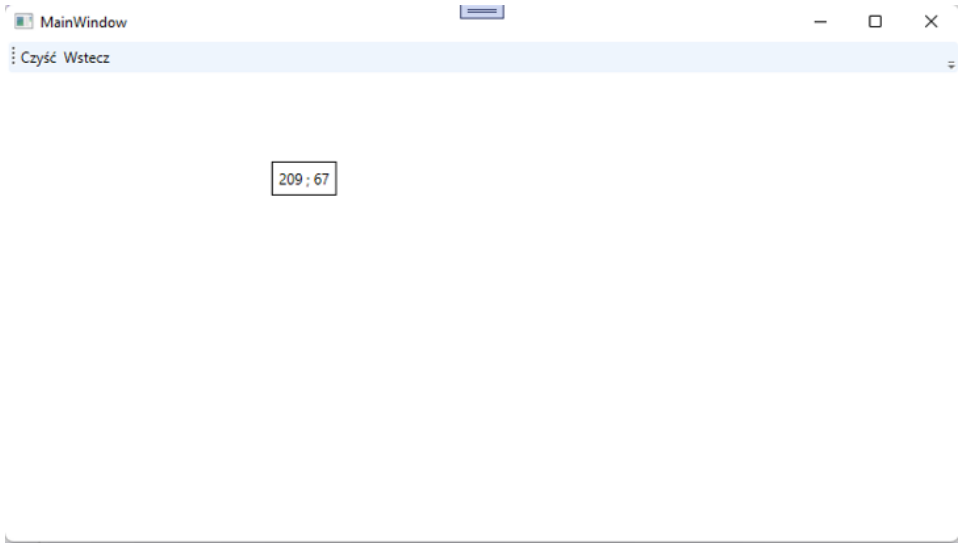
Właściwości pozycjonujące obiekty na Canvasie:

- Canvas.Left="…", Canvas.Top="…": służą do ustalenia położenia w pionie i poziomie.
- Panel.ZIndex="…": opcjonalnie można ustawić kolejność rysowania elementów, jeśli się nakładają.

```
<Canvas Name="MyCanvas" Background="LightGray">
        <!-- Elementy podrzędne Canvas -->
        <Rectangle Width="100" Height="100" Fill="Red"
                   Canvas.Left="10" Canvas.Top="10"
                   Panel.ZIndex="5"/>
        <Ellipse Width="150" Height="150" Fill="Blue"
                 Canvas.Left="50" Canvas.Top="30"/>
        <TextBlock Text="Przykładowy tekst"
                   Canvas.Left="170" Canvas.Top="120"
                   Foreground="Black"/>
    </Canvas>
```



Przykład 1 – pozycja kursora



Canvas – reakcja na zdarzenia

```
<Canvas x:Name="canvas"
        Background="White"
        MouseMove="canvas_MouseMove">

      <Border x:Name="opis_kursora"
                 BorderBrush="Black"
              BorderThickness="1">
          <Label Name="pozycja" Content="0 ; 0"/>
      </Border>

  </Canvas>
```

```
private void canvas_MouseMove(object sender, MouseEventArgs e)
        {
            Point p = e.GetPosition(canvas);
            //Pobieramy pozycję kursora względem elementu canvas
            Canvas.SetLeft(opis_kursora, p.X+10);
            // aktualizujemy położenie kontrolki
            // "opis_kursora" na canvas
            Canvas.SetTop(opis_kursora, p.Y+5);
            pozycja.Content = Math.Round(p.X) + " ; "+ Math.Round(p.Y);
            // wypisujemy aktualna pozycję kursora (po zaokrągleniu)
        }
```

- Obiekt Canvas może reagować na zdarzenia tak jak każda kontrolka.
- W tym przykładzie oprogramowujemy zdarzenie MauseMove.
- Obsługę zdarzenia można przypisać zarówno do całego Canvasu jak i do poszczególnych elementów, które na nim leżą.

Canvas – Obsługa w kodzie C#

```
// Tworzymy obiekt typu Ellipse
  Ellipse dynamicEllipse = new Ellipse
    {
       Width = 30,
       Height = 30,
       Fill = Brushes.Green
     };

// Ustawiamy położenia obiektu względem Canvas
     Canvas.SetLeft(dynamicEllipse, 200);
     Canvas.SetTop(dynamicEllipse, 50);

// Dodajemy obiekt do hierarchii wizualnej
     MyCanvas.Children.Add(dynamicEllipse);
```

Pierwszym krokiem jest utworzenie obiektu i nadanie mu podstawowych własności wysokość szerokość wypełnienia

Canvas.SetLeft(UIElement, double) i Canvas.SetTop(UIElement, double) - służą do ustawiania współrzędnych danego obiektu w obrębie Canvas

.Children.Add(UIElement) – metoda umożliwiająca dodanie nowego obiektu do kolekcji dzieci (Children).

Dopiero w momencie obiekt zostanie wyświetlony.

Canvas – Obsługa w kodzie C#

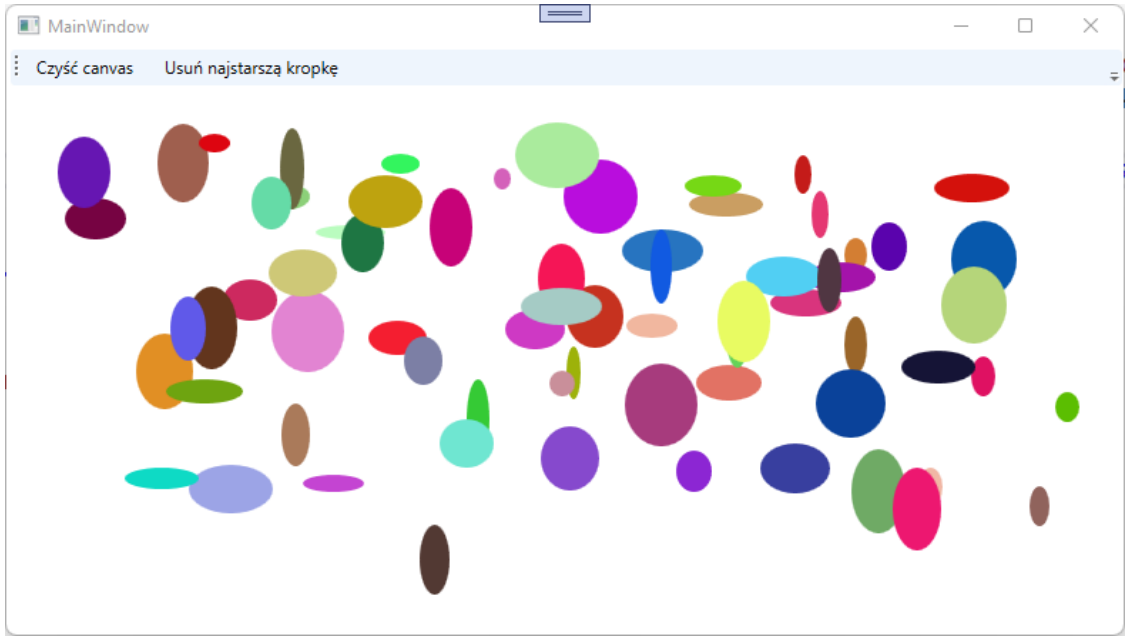
```
// Rysujemy linię z punktu (10, 10) do
punktu (100, 100)
Line line = new Line
{
    X1 = 10,
    Y1 = 10,
    X2 = 100,
    Y2 = 100,
    StrokeThickness = 2,
    Stroke = Brushes.Black
};

MyCanvas.Children.Add(line);
```

W przypadku prostokątów i elips definiujemy ich rozmiar oraz ustalamy pozycję punktu wiodącego lewego górnego rogu względem Canvasu

W przypadku prymitywów takich jak linie ich pozycja względem Canvasu definiuje punkt początkowy i punkt końcowy będący własnością samego obiektu.

Przykład 2 – „Kropki”



Canvas

Layout aplikacji:

```
<DockPanel>
        <ToolBar DockPanel.Dock="Top">
            <Button Padding="10,2" Click="Button_Click">
                 Czyść canvas
             </Button>
            <Button Padding="10,2" Click="Button_Click_1">
                  Usuń najstarszą kropkę
            </Button>
        </ToolBar>
        <Canvas x:Name="canvas" Background="White"
                                MouseDown="canvas_MouseDown" />

</DockPanel>
```

Uwaga: tło obiektu canvas jest istotne, aby mógł on przechwytywać zdarzenia

Canvas

Przygotowujemy listę przechowująca obiekty klasy Ellipse oraz generator liczb pseudolowowych

```
List<Ellipse> ellipseList = new List<Ellipse>();

Random random = new Random();
```

Canvas

Dodawanie elips po kliknięciu myszą

```
private void canvas_MouseDown(object sender, MouseButtonEventArgs e)
        {
            Point p = e.GetPosition(canvas);
            //Pobieramy pozycję kursora względem elementu canvas
            Ellipse e1 = new Ellipse();
            //tworzymy obiekt klasy Ellipse
            e1.Width = random.Next(50) + 10;
            e1.Height = random.Next(50) + 10;
            //losujemy wymiary elipsy
            e1.Fill = new SolidColorBrush(Color.FromRgb((Byte)random.Next(255),
                                                        (Byte)random.Next(255),
                                                        (Byte)random.Next(255)));
            //losujemy kolor wypełnienia
            ellipseList.Add(e1);
            //dodajemy elipsę do listy
            Canvas.SetLeft(e1, p.X);
            Canvas.SetTop(e1, p.Y);
            //ustawiamy połorzenie elipsy na podstawie pozycji kursora
            canvas.Children.Add(e1);
            //dodajemy elipsę do obiektu canvas
        }
```

Canvas

Czyszczenie obiektu canvas (oraz listy elips)

```
private void Button_Click(object sender, RoutedEventArgs e)
        {
            canvas.Children.Clear();
            ellipseList.Clear();
        }
```

Canvas

Usunięcie najstarszej elipsy (pierwszej na liscie)

```
private void Button_Click_1(object sender, RoutedEventArgs e)
        {
            canvas.Children.Remove(ellipseList[0]);
            ellipseList.RemoveAt(0);
        }
```

Canvas – Transformacje obiektów

Każdy obiekt w Canvas może być poddany transformacjom 2D.

WPF umożliwia nakładanie różnego rodzaju macierzy transformacji bezpośrednio na obiekty:

- TranslateTransform – translacja, czyli przesunięcie obiektu w osi X i/lub Y (właściwości X i Y).
- ScaleTransform – skalowanie obiektu względem zdefiniowanego punktu (CenterX, CenterY). Pozwala zmienić wymiary obiektu (proporcjonalnie lub nie) we współrzędnych X oraz Y.
- RotateTransform – obrót obiektu o określony kąt wokół punktu obrotu (CenterX, CenterY).
- SkewTransform – pochylanie (wychylenie) w osi X lub Y, co tworzy efekt „przesunięcia” wzdłuż krawędzi.
- MatrixTransform – najbardziej elastyczna transformacja, pozwalająca ręcznie ustawić macierz przekształcenia 3x3, co umożliwia łączenie w jednej operacji skalowania, przesunięcia, obrotu czy złożonych manipulacji.

Canvas – Transformacje obiektów

```
RotateTransform rotateTransform = new RotateTransform
  {
     Angle = 45,
     CenterX = 25,  // środek obrotu (lokalny)
     CenterY = 25
  };
rotatingEllipse.RenderTransform = rotateTransform;
```

Jeżeli transformację definiujemy w kodzie c# należy najpierw przygotować matryce a następnie zastosować dla obiektu.

Transformację można także zdefiniować bezpośrednio w pliku XAML

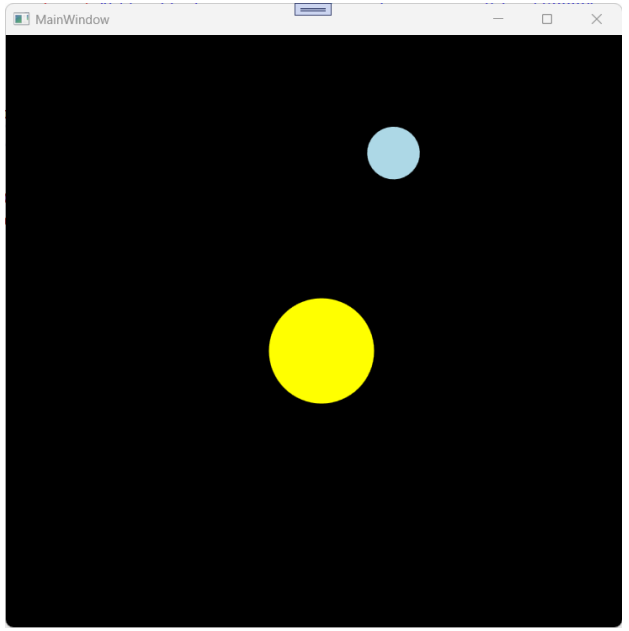
```
<Canvas Name="MyCanvas" Background="Green">
        <Rectangle Width="50" Height="100" Fill="Orange"
                   Canvas.Top="100" Canvas.Left="100">
            <Rectangle.RenderTransform>
                <RotateTransform Angle="45" CenterX="25" CenterY="25" />
            </Rectangle.RenderTransform>
        </Rectangle>
    </Canvas>
```

Canvas – Transformacje obiektów

Jeśli chcemy zastosować kilka transformacji jednocześnie (np. obrócić i przeskalować obiekt), to można połączyć je w jedną grupę, korzystając z klasy TransformGroup, np.:

```
<Ellipse Width="50" Height="50" Fill="Orange">
       <Ellipse.RenderTransform>
           <TransformGroup>
               <TranslateTransform X="10" Y="20" />
               <RotateTransform Angle="45" CenterX="25" CenterY="25" />
               <ScaleTransform ScaleX="1.2" ScaleY="0.8" CenterX="25" CenterY="25" />
           </TransformGroup>
       </Ellipse.RenderTransform>
 </Ellipse>
```

Przykład 3 – „Układ słoneczny – transformacje obiektów”



Canvas

```
public partial class MainWindow : Window
{
    int ang = 0;
    Ellipse slonce, ziemia;
    public MainWindow()
    {
        InitializeComponent();

        slonce = new Ellipse
        {
            Width = 100,
            Height =100,
            Fill = Brushes.Yellow
        };
        Canvas.SetLeft(slonce, 250);
        Canvas.SetTop(slonce, 250);
        MyCanvas.Children.Add(slonce);

        ziemia = new Ellipse
        {
            Width = 50,
            Height = 50,
            Fill = Brushes.LightBlue
        };
        Canvas.SetLeft(ziemia, 475);
        Canvas.SetTop(ziemia, 275);
        MyCanvas.Children.Add(ziemia);

        DispatcherTimer timer = new DispatcherTimer();
        timer.Interval = TimeSpan.FromMilliseconds(50);
        timer.Tick += Timer_Tick;
        timer.Start();
    }

    private void Timer_Tick(object? sender, EventArgs e)
    {

        RotateTransform rotateTransform = new RotateTransform
        {
            Angle = ang,
            CenterX = -175,  // środek obrotu (lokalny)
            CenterY = 25
        };
        ziemia.RenderTransform = rotateTransform;
        ang += 5;
    }
}
```

żeby było jasne – to nie jest animacja ruchu po ścieżce, tylko sztuczka, która sprawia takie wrażenie, bo oś obrotu jest wysunięta poza obiekt

Canvas - Animacje

WPF udostępnia rozbudowany system animacji właściwości (Property Animation System), dzięki któremu można animować dowolne atrybuty elementów, w tym także ich położenie na Canvas, wypełnienie (Fill), obrys (Stroke) czy transformacje (obrót, skalowanie itd.).

WPF korzysta z tzw. timelines (linii czasu) i storyboardów, aby animować różne właściwości obiektów:

- Storyboard – kontener animacji i centralny element, który „zarządza” czasem trwania, kolejnością i sposobem uruchamiania animacji.
- AnimationTimeline – klasa bazowa dla poszczególnych typów animacji, np.:
  - DoubleAnimation (animuje właściwości typu double, np. Canvas.Left, Opacity, Width itp.),
  - ColorAnimation (animuje właściwości typu Color, np. Fill, Stroke),
  - PointAnimation (animuje właściwości typu Point, przydatne np. w PathGeometry),
  - ThicknessAnimation, ObjectAnimationUsingKeyFrames, StringAnimationUsingKeyFrames i inne

Canvas - Animacje

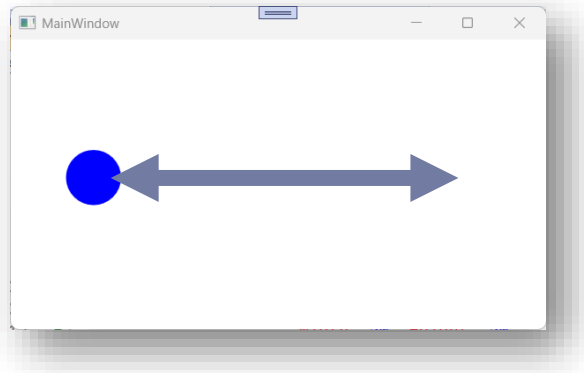
Z punktu widzenia Canvas najczęściej animowane są właściwości:

- Canvas.Left / Canvas.Top: przesunięcie elementu w osiach X i Y,
- Width / Height: rozmiar elementu,
- RenderTransform.(ScaleTransform/RotateTransform/...): stosowanie i animowanie transformacji,
- Opacity: płynne pojawianie się/zanikanie,
- (Shape.Fill).(SolidColorBrush.Color) lub (Shape.Stroke).(SolidColorBrush.Color): zmiana koloru wypełnienia lub obrysu.

Canvas - Animacje

```
<Ellipse x:Name="AnimatedEllipse"
         Width="50" Height="50"
         Fill="Blue"
         Canvas.Left="50"
         Canvas.Top="100">
    <Ellipse.Triggers>
        <EventTrigger RoutedEvent="Ellipse.MouseEnter">
            <BeginStoryboard>
                <Storyboard>
                    <!-- Animacja przesunięcia w poziomie (właściwość Canvas.Left) -->
                    <DoubleAnimation
                        Storyboard.TargetProperty="(Canvas.Left)"
                        From="50" To="400"
                        Duration="0:0:5"
                        AutoReverse="True"
                        RepeatBehavior="1x" />
                </Storyboard>
            </BeginStoryboard>
        </EventTrigger>
    </Ellipse.Triggers>
</Ellipse>
```

W ten sposób określamy jakie zdarzenie ma być „Wyzwalaczem” animacji



W ten sposób określamy którą własność chcemy animować

Definiujemy zakres zmian od - do

Definiujemy czastrwania animacji

Canvas - Animacje

```
<Ellipse.Triggers>
        <EventTrigger RoutedEvent="Ellipse.MouseEnter">
            <BeginStoryboard>
                <Storyboard>
                       <DoubleAnimation
                        Storyboard.TargetProperty="(Canvas.Left)"
                        From="50" To="400"
                        Duration="0:0:5"
                        AutoReverse="True"
                        RepeatBehavior="1x" />
                </Storyboard>
            </BeginStoryboard>
        </EventTrigger>
    </Ellipse.Triggers>
```

- Storyboard.TargetProperty="(Canvas.Left)" – wskazuje, że chcemy animować właściwość Canvas.
- Left.From="50" To="200" – przesuwamy elipsę od położenia X = 50 do X = 200.
- AutoReverse="True" – po zakończeniu animacji w jedną stronę obiekt wraca (cofa animację) do położenia początkowego.
- RepeatBehavior="1x" – animacja wykona się tylko raz (można ustawić np. Forever, aby animacja była ciągła).
- EventTrigger RoutedEvent="Ellipse.MouseEnter" – animacja uruchomi się w momencie najechania myszą na elipsę.

Canvas - Animacje


```
<DockPanel>
        <Button
            Content="START"
            DockPanel.Dock="Bottom"
            Click="Button_Click"/>
        <Canvas x:Name="MyCanvas"
Background="White">
        </Canvas>
    </DockPanel>
```

```
public partial class MainWindow : Window
{

    private Rectangle animatedRectangle;
    public MainWindow()
    {
        InitializeComponent();

        // Dodajemy prostokąt do Canvas
        animatedRectangle = new Rectangle
        {
            Width = 50,
            Height = 50,
            Fill = Brushes.Red
        };
        Canvas.SetLeft(animatedRectangle, 20);
        Canvas.SetTop(animatedRectangle, 50);
        MyCanvas.Children.Add(animatedRectangle);
    }
```

Canvas - Animacje

```
private void Button_Click(object sender, RoutedEventArgs e)
    {
        // Tworzymy animację typu DoubleAnimation
        DoubleAnimation moveAnimation = new DoubleAnimation
        {
            From = 20,
            To = MyCanvas.ActualWidth - 20 - animatedRectangle.ActualWidth,
            Duration = TimeSpan.FromSeconds(2),
            AutoReverse = false
        };

        // Zdefiniowanie Storyboard i powiązanie z właściwością (Canvas.Left)
        Storyboard storyboard = new Storyboard();
        storyboard.Children.Add(moveAnimation);

        Storyboard.SetTarget(moveAnimation, animatedRectangle);
        Storyboard.SetTargetProperty(moveAnimation, new PropertyPath("(Canvas.Left)"));

        // Uruchomienie animacji
        storyboard.Begin();
    }
}
```

1. Tworzymy obiekt DoubleAnimation i określamy parametry (od–do, czas trwania).
2. Zakładamy Storyboard, który będzie zarządzał animacją.
3. Ustawiamy Storyboard.SetTarget na animatedRectangle i Storyboard.SetTargetProperty na (Canvas.Left).
4. Na koniec wywołujemy storyboard.Begin(), aby uruchomić animację.

Canvas – Animacje transformacji

```
<Canvas x:Name="MyCanvas" Background="White">
        <Rectangle Width="60" Height="40" Fill="Green" Canvas.Left="100"
Canvas.Top="100">
            <Rectangle.RenderTransform>
                <RotateTransform x:Name="MyRotate" CenterX="30" CenterY="20" Angle="0" />
            </Rectangle.RenderTransform>
            <Rectangle.Triggers>
                <EventTrigger RoutedEvent="Rectangle.MouseLeftButtonDown">
                    <BeginStoryboard>
                        <Storyboard>
                            <!-- Animacja obrotu od 0 do 360 stopni w ciągu 2 sekund -->
                            <DoubleAnimation
                               Storyboard.TargetName="MyRotate"
                               Storyboard.TargetProperty="Angle"
                               From="0" To="360"
                               Duration="0:0:2"
                               RepeatBehavior="Forever" />
                        </Storyboard>
                    </BeginStoryboard>
                </EventTrigger>
            </Rectangle.Triggers>
        </Rectangle>
    </Canvas>
```

Transformacji którą nakładamy na obiekt nadajemy własną nazwę (MyRotate), aby móc się do niej odnieść i nałożyć na nią animację

Start animacji po kliknięciu na kontrolkę

Animacja właściwości Angle przy pomocy DoubleAnimation.

RepeatBehavior="Forever" sprawia, że obrót będzie wykonywał się w nieskończoność.

Canvas – Animacja z klatkami kluczowymi (KeyFrames)

Aby sterować przebiegiem animacji można skorzystać z DoubleAnimationUsingKeyFrames

W tym przykładzie widzimy tylko definicję storyboardu.

Aby jej użyć należy umieścić ją w kodzie kontrolki – patrz poprzedni slajd

```
<Storyboard>

<DoubleAnimationUsingKeyFrames Storyboard.TargetProperty="(Canvas.Left)"
Duration="0:0:5"
AutoReverse="False"
RepeatBehavior="1x">

     <LinearDoubleKeyFrame Value="50" KeyTime="0:0:0" />
     <LinearDoubleKeyFrame Value="200" KeyTime="0:0:2" />
     <LinearDoubleKeyFrame Value="300" KeyTime="0:0:4" />
     <LinearDoubleKeyFrame Value="50" KeyTime="0:0:5" />
 </DoubleAnimationUsingKeyFrames>

</Storyboard>
```

Tutaj w ciągu 5 sekund obiekt znajdzie się w kluczowych położeniach zdefiniowanych jako Value

Canvas – Synchronizacja i sekwencjonowanie animacji

W rozbudowanych scenariuszach możemy użyć między innymi:

- BeginTime – opóźnienie startu,
- SpeedRatio – manipulacja szybkością animacji,
- Completed (event w C#) – wykrywanie zakończenia animacji i wywoływanie kolejnych w odpowiednim momencie,
- ParallelTimeline, SequenceTimeline – pozwalają łączyć animacje równolegle lub sekwencyjnie.

Canvas – Synchronizacja i sekwencjonowanie animacji

Aby sterować przebiegiem animacji można skorzystać z DoubleAnimationUsingKeyFrames

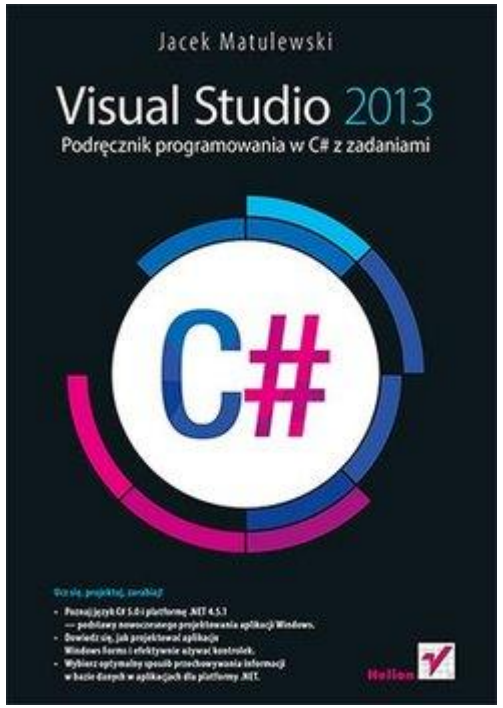
W tym przykładzie widzimy tylko definicję storyboardu.

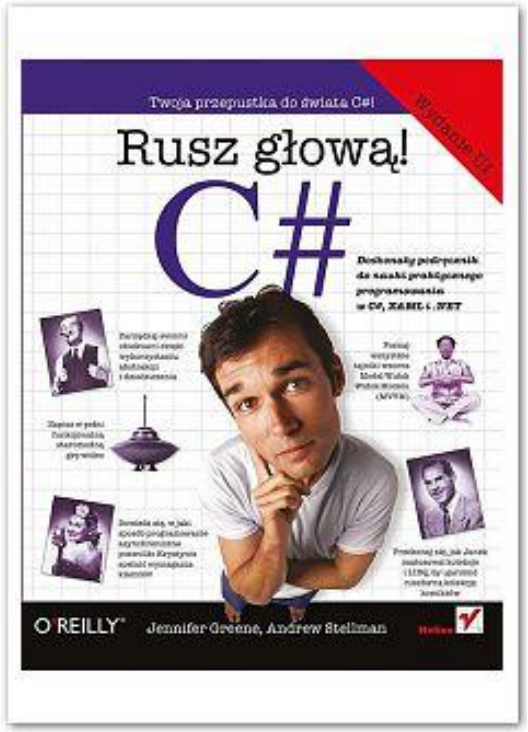
Aby jej użyć należy umieścić ją w kodzie kontrolki – patrz poprzednie przykłady

```
<Storyboard>
   <!-- Animacja 1 -->
   <DoubleAnimation Storyboard.TargetProperty="(Canvas.Left)"
                    From="50" To="200" Duration="0:0:2" />
   <!-- Animacja 2 -->
   <DoubleAnimation Storyboard.TargetProperty="(Canvas.Top)"
                    From="50" To="150" Duration="0:0:2"
                    BeginTime="0:0:2" />
</Storyboard>
```

Dzięki BeginTime="0:0:2" druga animacja rozpocznie się po 2 sekundach, czyli dopiero po zakończeniu pierwszej (zakładając ten sam czas trwania).

Literatura:





Użyte w tej prezentacji tabelki pochodzą z książki: Visual Studio 2013. Podręcznik programowania w C# z zadaniami
Autor: Matulewski Jacek, Helion