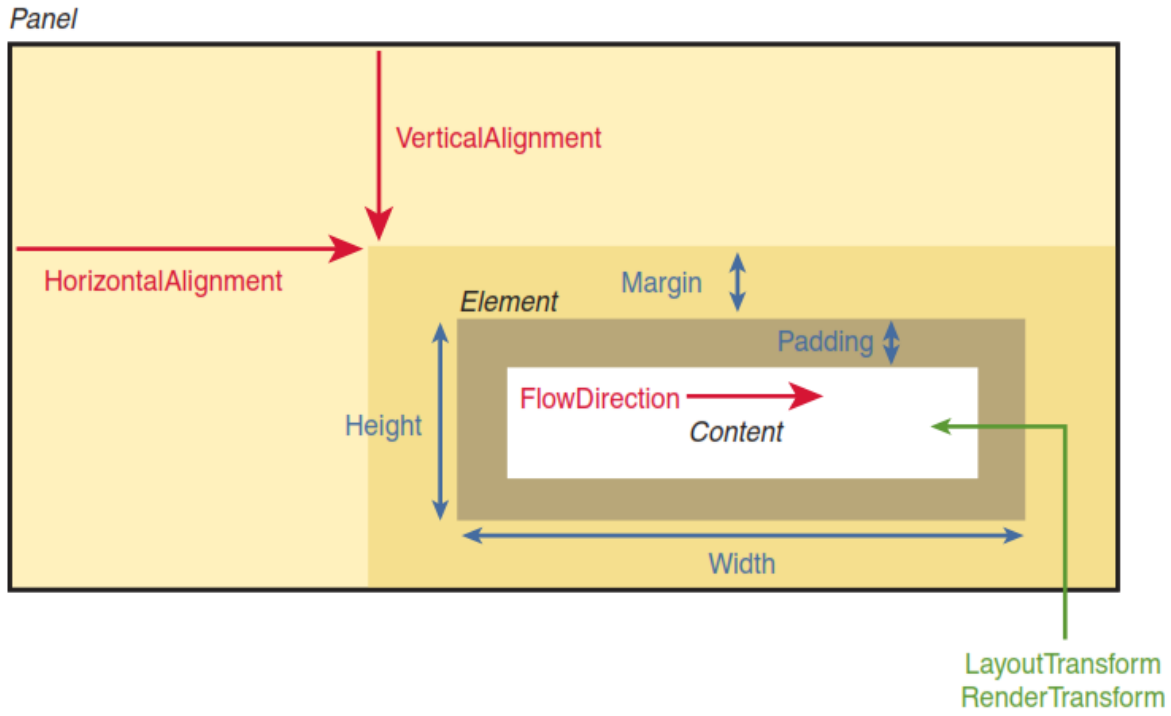
Object-oriented programming - lab in .NET environment

Lecture 06

Arrangement

- We understand the sizing and positioning of controls
- We distinguish between three types of management:
  - Size control
  - Position control
  - Application of transformations



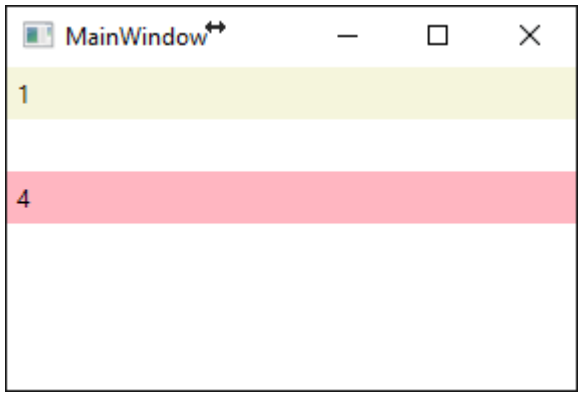
Size control

- WPF elements generally tend to scale to fit their content
- The determination of the size of the element is affected by various properties:
  - **`Width`**, **`Height`** – explicit width and height ("stronger" than `Stretch` settings)
    - It must be between Min… and Max…
  - **`MinWidth`**, **`MinHeight`** – minimum dimensions of the element
  - **`MaxWidth`**, **`MaxHeight`** – maximum dimensions of the element
    - If there is more space in the parent, the element will not expand, even if the setting is Stretch
  - Properties **`Margin`** and **`Padding`** - determine the additional space outside or inside the element (the value is the structure **`Thickness`**)

Size control

- Property **Visibility**:
  - **Visible** – the element is drawn and participates in the layout
  - **Collapsed** – the element is invisible and does not participate in the layout
  - **Hidden** – the element is invisible, but participates in the layout

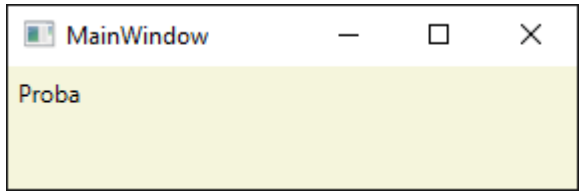
```
<StackPanel>
    <Label Background="Beige"
           Visibility="Visible">1</Label>
    <Label Background="Blue"
           Visibility="Hidden">2</Label>
    <Label Background="Orange"
           Visibility="Collapsed">3</Label>
    <Label Background="LightPink"
           Visibility="Visible">4</Label>
</StackPanel>
```



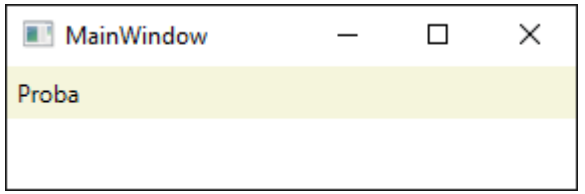
Position control

- Properties **`HorizontalAlignment`** and **`VerticalAlignment`** determine the horizontal and vertical alignment of the control within its container
  - Horizontal: `Left`, `Right`, `Center`, `Stretch`
  - Vertical: `Top`, `Bottom`, `Center`, `Stretch`
  - The default value of both properties is most often `Stretch`

```
<Grid>
    <Label Background="Beige">
        Proba
    </Label>
</Grid>
```




```
<StackPanel>
    <Label Background="Beige">
        Proba
    </Label>
</StackPanel>
```



Position control

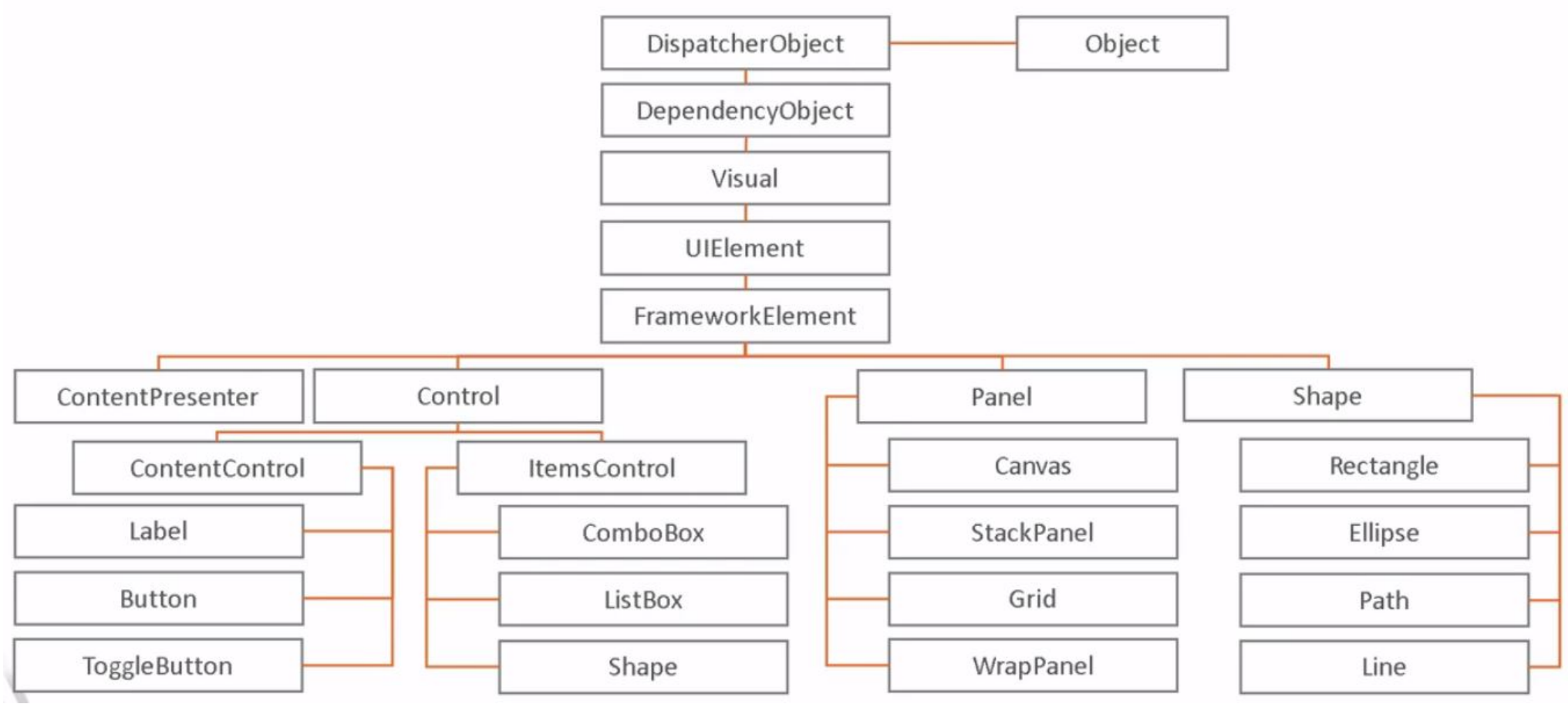
- Properties **`HorizontalContentAlignment`** and **`VerticalContentAlignment`** – determine the alignment of content within the control
  - By default they are **`Left`** and **`Top`**

```
<Grid>
    <Label Background="Blue"
           Foreground="White"
           Content="Proba"
           Width="100"
           Height="75"
           HorizontalAlignment="Right"
           VerticalAlignment="Bottom"
           HorizontalContentAlignment="Right"
           VerticalContentAlignment="Bottom"/>
</Grid>
```

WPF principles

- WPF recommends the following:
  - Elements should not be explicitly sized
    - They should grow with their content
    - Acceptable limits are determined with the Min… and Max… settings
  - Elements do not need to define their position with coordinates
    - They should be sorted by their parent according to their size, order, and other information specific to the parent
    - For additional space Margin should be used
  - Panels divide the available space for children
    - They try to give everyone the desired size
  - Panels are nested: usually first Grid, StackPanel and then others...

Hierarchy of classes



Layout using panels

- The most important panels in WPF are:
  - `Grid`
  - `StackPanel`
  - `WrapPanel`
  - `DockPanel`
  - `Canvas`

Grid

- The most useful and frequently used panel that allows you to arrange your elements in multiple rows and columns
  - It is used to divide windows into multiple regions, and StackPanels are often used in certain regions
  - **RowDefinitions** property contains **RowDefinition** elements that define rows
  - **ColumnDefinitions** property contains **ColumnDefinition** elements that define columns
  - Attached properties **Grid.RowSpan** and **Grid.ColumnSpan** allow content to span across multiple cells

Grid

- The size of rows and columns is expressed by the type value

  `System.Windows.GridLength` which can be:

  - **`Absolute`** – a numerical value is set as height and width
  - **`Auto`** – the size is set automatically according to the content
  - **`Proportional`** **(* - asterisk)** – available space is divided proportionally (default value)

- Controls are placed in the corresponding cell `Grid` using attached properties

  `Grid.Column` and `Grid.Row`

StackPanel

- It is used to arrange a small number of controls and is often placed in a cell `Grid`
- Depending on the property **`Orientation`** arranges elements vertically (by default) or horizontally

WrapPanel

- Arranges the elements as **`StackPanel`**, except that if there is no space, it moves them to additional rows or columns depending on the orientation
- The most important features:
  - **`Orientation`** – like using `StackPanel`, except that horizontal orientation is implied

DockPanel

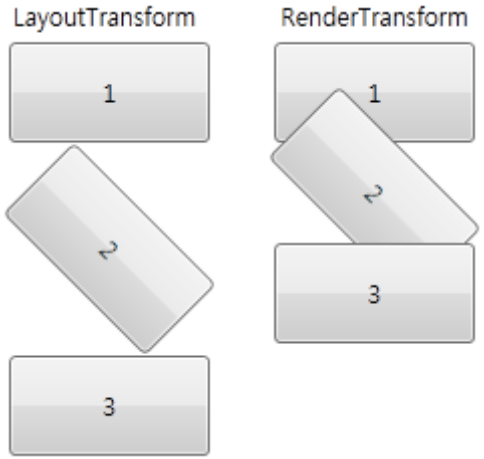
- It allows elements to be attached along the entire side of the panel, stretching them to fill its entire width, or height
- Attached property **`Dock`** can take four values:
  - **`Left`**
  - **`Top`**
  - **`Right`**
  - **`Bottom`**
- The lack of Fill value can be compensated for by configuring the property **`LastChildFill`**

Canvas

- It supports the placement of elements in the classic sense - positioning using explicit coordinates
  - Coordinates are device-independent units
  - Except relative to the upper left corner, coordinates can be specified relative to any other angle
  - Features: `Left, Top, Right, Bottom`
- The predetermined Z-order is determined by the order in which the elements are added
  - Attached property **`Panel.ZIndex`** can change the order
- It is rarely used

Application of transformations

- All elements have two type properties **Transform** which are used for element transformations:
  - **LayoutTransform** – is applied in the arrangement phase, before plotting
  - **RenderTransform** – is applied after plotting
- **RenderTransformOrigin** represents the starting point
- Transformation methods:
  - **RotateTransform** – rotation
  - **ScaleTransform** – increasing or decreasing the element
  - **SkewTransform** – distortion of the element
  - **TranslateTransform** – moving the element



Events

- **Routed events** are events designed to work with the element tree
  - Almost all WPF events are routed
- Each directed event uses one of three strategies:
  - **Tunneling** – the event is first raised at the root and then at each element down the tree until it reaches the element that is the actual source of the event (usually prefixed with `Preview`)
    - for example `PreviewMouseDown()`
  - **Bubbling** – the event is first raised on the original element and then in turn on each element above in the hierarchy until it reaches the root
    - for example `MouseDown()`
  - **Direct** – the event is raised only on the original element
    - for example `MouseEnter()`

Type RoutedEventArgs

- All processing methods have the first argument type of **`object`**, and the second argument type of **`RoutedEventArgs`** (extends **`EventArgs`**)
- Class **`RoutedEventArgs`** has the following useful properties:
  - **`Source`** – an element of the logical tree that raised the event
  - **`OriginalSource`** – an element of the visual tree that raised the event
- If we click on the edge of a `Label` control:
  - **`Source`**: `System.Windows.Controls.`**`Label`**
  - **`OriginalSource`**: `System.Windows.Controls.`**`Border`**

Example of routed events

- The transmission of routed events up and down the tree occurs even if some elements do not support a particular event
  - This is why routed events are also called **attached events**
- For example, `StackPanel` does not support the event `Click`, but it can be processed by:

```
<StackPanel Button.Click="StackPanel_Click">
    <Label Content="Ja sam labela" />
</StackPanel>
```