



Curso de Extensão
Tecnologias Microsoft



INF-0996

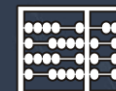
Desenvolvimento de Interface de Usuário

AULA 1

Prof. Dr. Rodrigo Bonacin

RBonacin@unicamp.br

05 de Novembro de 2022



INSTITUTO DE
COMPUTAÇÃO



1. Apresentação da Disciplina
2. Introdução ao *Windows Presentation Foundation* (WPF)
 - Conceitos e Características
 - XAML + C#, Exemplos
 - Arquitetura do WPF
3. Atividades práticas em WPF
 - Funcionamento Básico
 - Alguns elementos de interface
 - Eventos, Code-behind, Data binding ...

The background features a network of gray lines connecting various colored circles (orange, yellow, light blue, green) and a dark blue horizontal band with a white circuit-like pattern.

Apresentação da Disciplina



Monitor:

- Luã Muriana 1163144@dac.unicamp.br

Aulas:

- Aula: 05/11/2022 – 13:30 às 17:30
 - Sala 351 (até o intervalo) e Labs 303 e 304 (após o intervalo)
- Aula: 12/11/2022 – 13:30 às 17:30
 - Sala 351 (até o intervalo) e Labs 303 e 304 (após o intervalo)

Apresentação da Disciplina



- Atendimento Online

- 07/11/2022 – 19:00 às 20:00 – Luã Muriana
- 08/11/2022 – 19:00 às 20:00 – Rodrigo Bonacin - meet.google.com/rsy-sjqc-obu
- 09/11/2022 – 19:00 às 20:00 – Luã Muriana
- 14/11/2022 – 19:00 às 20:00 – Luã Muriana
- 16/11/2022 – 19:00 às 20:00 – Luã Muriana
- 17/11/2022 – 19:00 às 20:00 – Rodrigo Bonacin - meet.google.com/rmf-wspa-crj

Apresentação da Disciplina



Ementa:

- Introdução ao Windows Presentation Foundation (WPF)
 - Conceitos e Características
 - XAML + C#, Exemplos
 - Arquitetura do WPF
- Atividades práticas em WPF
 - Funcionamento Básico
 - Alguns elementos de interface
 - Eventos, Code-behind, Data binding ...
- Extensões para WPF no VS Code
- Guidelines de design Microsoft (em WinUI 3)
- Padrão MVVM (Model–View–ViewModel)
- MVVM Toolkit



Avaliação:

Grupos de no máximo quatro pessoas deverão desenvolver o Trabalho abaixo.

- *Descrição:* Este trabalho tem como objetivo desenvolver habilidades de programação utilizando o Windows Presentation Foundation (WPF) com XAML e C#. Para tanto cada grupo deve desenvolver tocador de mídias (áudio e vídeo) utilizando o WPF com XAML e C#. O design da solução é aberto, sendo parte integrante do trabalho a construção de uma interface com boa usabilidade e um conjunto de recursos relevantes aos usuários.

Os grupos são livres para criar seus próprios aplicativos, contanto que possuam recursos básicos que permitam aos usuários selecionar arquivos de mídias (áudio e vídeo) e manipular controles (*Tocar, Pausar, Parar, Pular, ...*). Recursos avançados como criar, armazenar e compartilhar *playlists* (por exemplo), efeitos visuais (transparência, animação, etc), perfil de usuário, para citar algumas alternativas, são de livre escolha dos grupos de acordo com a concepção do aplicativo. Esses recursos adicionarão nota ao trabalho.

Boas práticas de design de interfaces (como as dos Guidelines do Windows e WPF) e boas práticas arquiteturais (como o MVVM) devem ser adotadas, bem como boas práticas de programação .net/C#.

A descrição completa do trabalho estará disponível no sistema moodle na sua data de início.



Avaliação:

- *Datas de Início e Entrega:* Início: 05/novembro/2022 e Entrega: 18/novembro/2022

- *Calculo da média final e critérios de aproveitamento:*

A nota será atribuída de acordo com critérios de avaliação do trabalho, que incluem: (1) qualidade da interface, (2) funcionalidades e recursos, (3) qualidade do código - arquitetura, projeto, boas práticas e (4) qualidade do relatório e apresentação.

Serão considerados aprovados os alunos com frequência mínima de 80% e média final mínima de 7,0.

Apresentação da Disciplina



- Recursos e Referências

- Documentação do Windows Presentation Foundation (.net 6): <https://learn.microsoft.com/pt-br/dotnet/desktop/wpf/?view=netdesktop-6.0>
- Avançado (Windows Presentation Foundation): <https://learn.microsoft.com/pt-br/dotnet/desktop/wpf/advanced/?view=netframeworkdesktop-4.8&viewFallbackFrom=netdesktop-6.0>
- Design e código de aplicativos do Windows: <https://learn.microsoft.com/pt-br/windows/apps/design/>
- Uno Platform documentation: <https://platform.uno/docs/articles/intro.html>
- Patterns - WPF Apps With The Model-View-ViewModel Design Pattern: <https://learn.microsoft.com/en-us/archive/msdn-magazine/2009/february/patterns-wpf-apps-with-the-model-view-viewmodel-de>
- Introdução ao Kit de Ferramentas do MVVM: <https://learn.microsoft.com/pt-br/dotnet/communitytoolkit/mvvm/>
- Documentação Visual Studio Code: <https://code.visualstudio.com/docs/>
- Slides das aulas, incluindo links para páginas e ferramentas utilizadas.

The background features a network of gray lines connecting various colored circles (orange, yellow, light blue, green) in a non-linear fashion. A solid blue horizontal band spans the width of the slide, serving as a backdrop for the title text.

Introdução ao *Windows Presentation Foundation (WPF)*



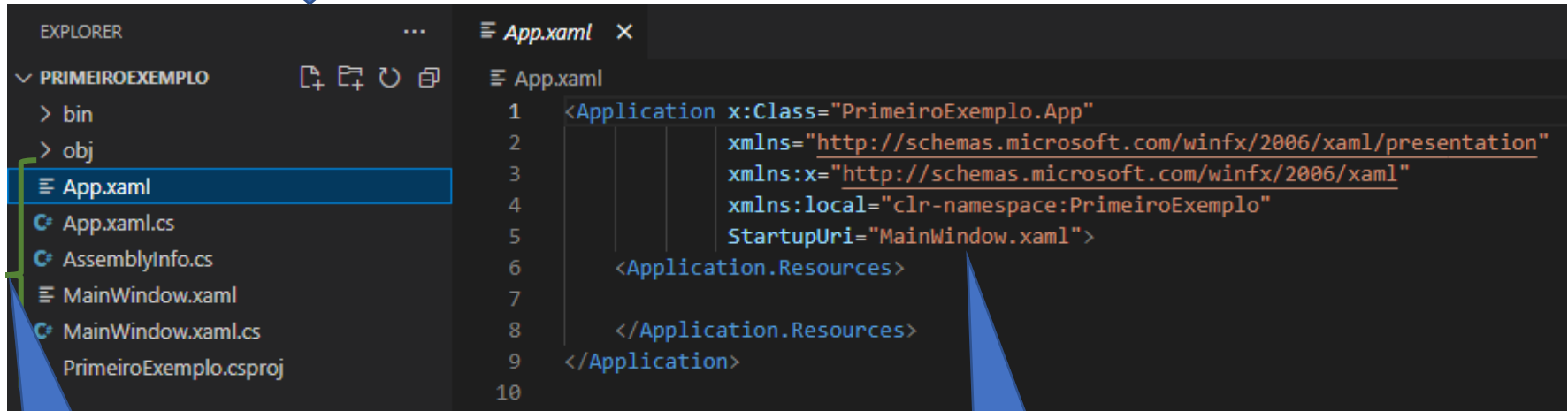
- *Windows Presentation Foundation* (WPF) é uma estrutura de interface do usuário que cria aplicativos cliente da área de trabalho
 - Suporta vários recursos de desenvolvimento de aplicativos, incluindo um modelo de aplicativo, recursos, controles, gráficos, layouts, associação de dados, documentos e segurança
 - Faz parte do .NET (nesta disciplina versão superior a 5)
 - Usa a linguagem XAML para fornecer um modelo declarativo para programação de aplicativos

Conceitos Básicos - WPF



- Criando um projeto WPF no VS Code

```
1\PrimeiroExemplo> dotnet new wpf
```



Arquivos Gerados

Janela Inicial

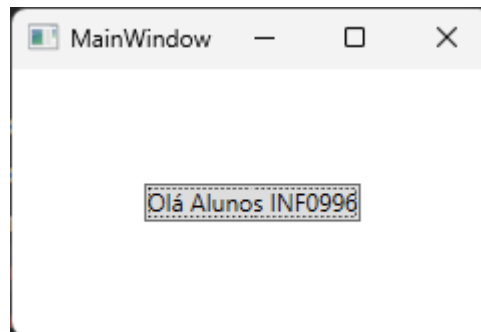
Conceitos Básicos - WPF



- Criando um projeto WPF no VS Code

The screenshot shows the Visual Studio Code interface. On the left, the Explorer pane displays the project structure for 'PRIMEIROEXEMPLO', including folders 'bin' and 'obj', and files 'App.xaml', 'App.xaml.cs', 'AssemblyInfo.cs', 'MainWindow.xaml' (selected), 'MainWindow.xaml.cs', and 'PrimeiroExemplo.csproj'. The main editor area shows the content of 'MainWindow.xaml' with the following XML code:

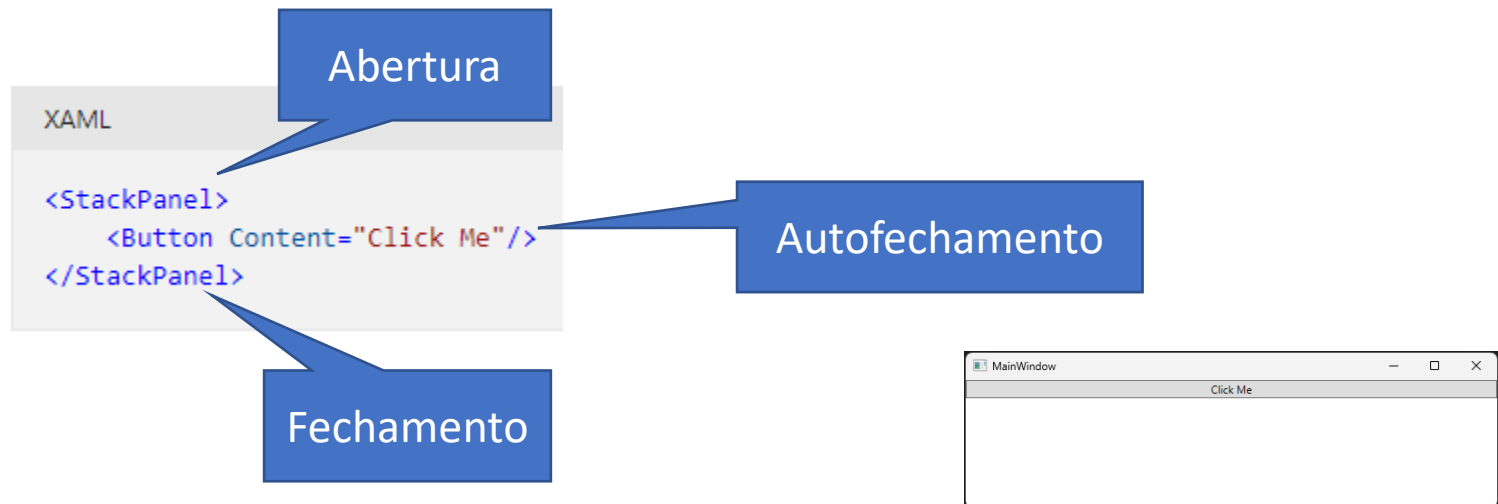
```
1 <Window x:Class="PrimeiroExemplo.MainWindow"
2       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4       xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
5       xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
6       xmlns:local="clr-namespace:PrimeiroExemplo"
7       mc:Ignorable="d"
8       Title="MainWindow" Height="450" Width="800">
9     <Grid>
10      <Button x:Name="button" Content="Olá Alunos INF0996"
11             HorizontalAlignment="Center" VerticalAlignment="Center"/>
12    </Grid>
13 </Window>
```



Conceitos Básicos - WPF - XAML



- O XAML - *eXtensible Application Markup Language*
 - Linguagem de marcação declarativa
- XAML são arquivos XML que geralmente tem a extensão .xaml



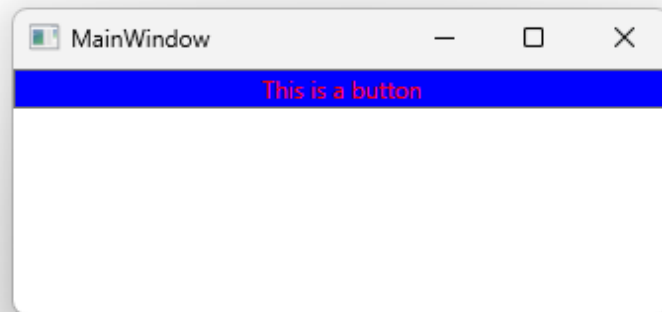
<https://learn.microsoft.com/pt-br/dotnet/desktop/wpf/xaml/?view=netdesktop-6.0>



- O XAML – Propriedades

XAML

```
<Button Background="Blue" Foreground="Red" Content="This is a button"/>
```

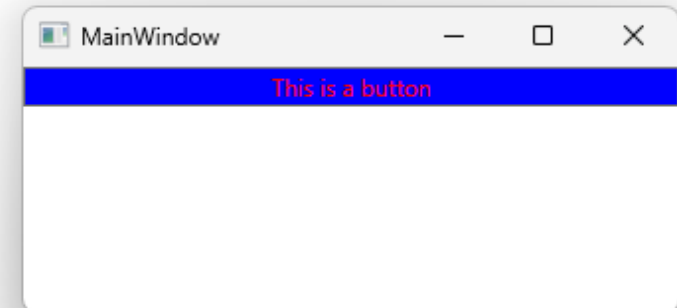
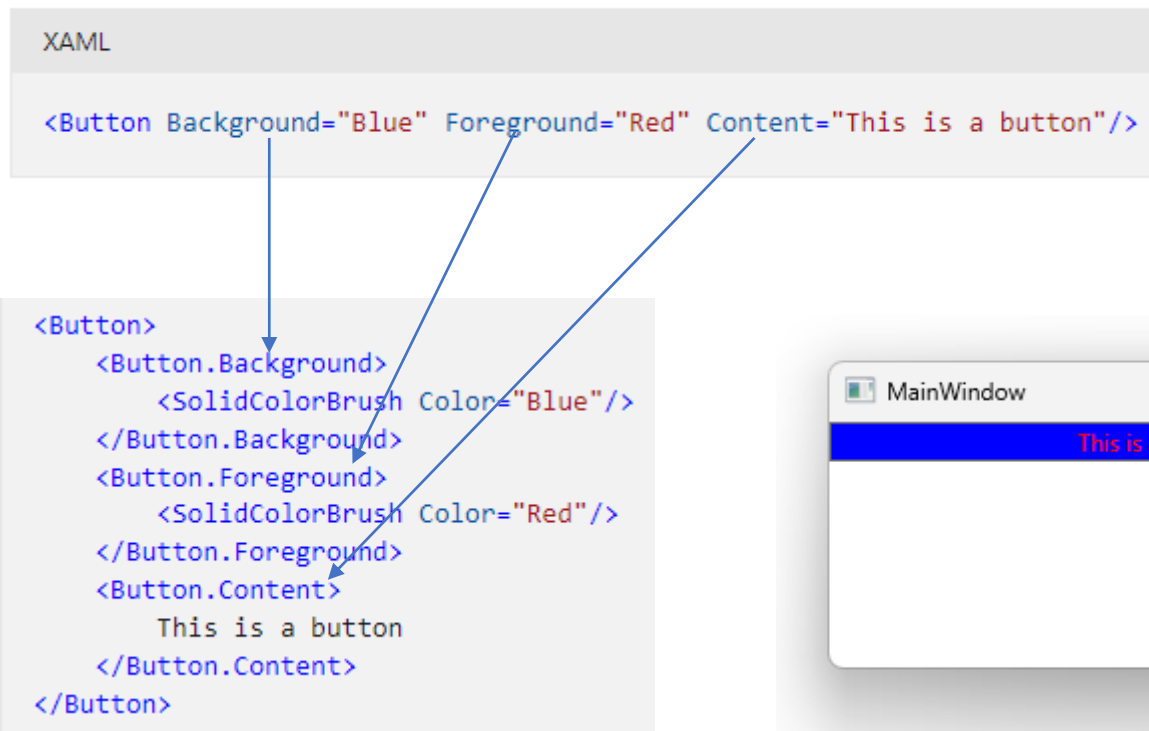


<https://learn.microsoft.com/pt-br/dotnet/desktop/wpf/xaml/?view=netdesktop-6.0>

Conceitos Básicos - WPF - XAML



- O XAML – Sintaxe de elemento de propriedade



<https://learn.microsoft.com/pt-br/dotnet/desktop/wpf/xaml/?view=netdesktop-6.0>



- O XAML – Sintaxe de coleção

Coleção

```
Rectangle Width="200" Height="100">  
  <Rectangle.Fill>  
    <LinearGradientBrush StartPoint="0,0" EndPoint="1,1">  
      <GradientStop Offset="0.0" Color="Red" />  
      <GradientStop Offset="1.0" Color="Blue" />  
    </LinearGradientBrush>  
  </Rectangle.Fill>  
</Rectangle>
```

Itens



<https://learn.microsoft.com/pt-br/dotnet/desktop/wpf/xaml/?view=netdesktop-6.0>

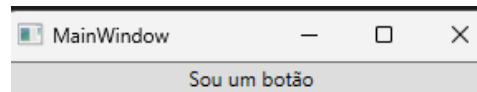
Conceitos Básicos - WPF - XAML



- O XAML – Propriedades de conteúdo XAML

Conteúdo

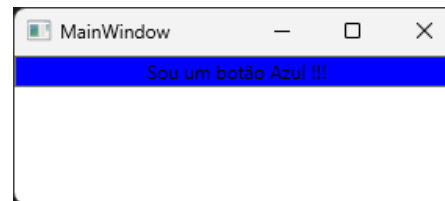
```
<Button>Sou um botão </Button>
```



```
<Border>  
    <TextBox Width="300"/>  
</Border>  
<!--explicit equivalent-->  
<Border>  
    <Border.Child>  
        <TextBox Width="300"/>  
    </Border.Child>  
</Border>
```

Conteúdo de uma borda
pode ser um TextBox

```
<Button>Sou um botão Azul !!!  
    <Button.Background>Blue</Button.Background>  
</Button>
```

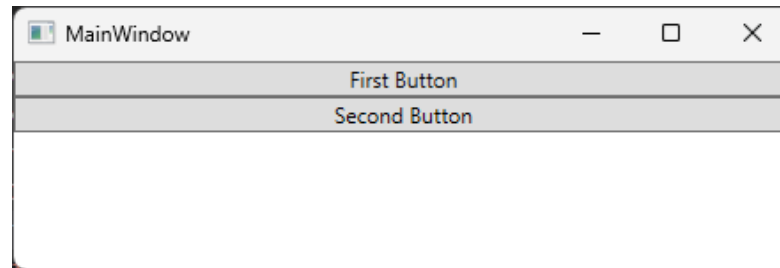


<https://learn.microsoft.com/pt-br/dotnet/desktop/wpf/xaml/?view=netdesktop-6.0>



- O XAML – Propriedades de conteúdo e sintaxe de coleção combinadas

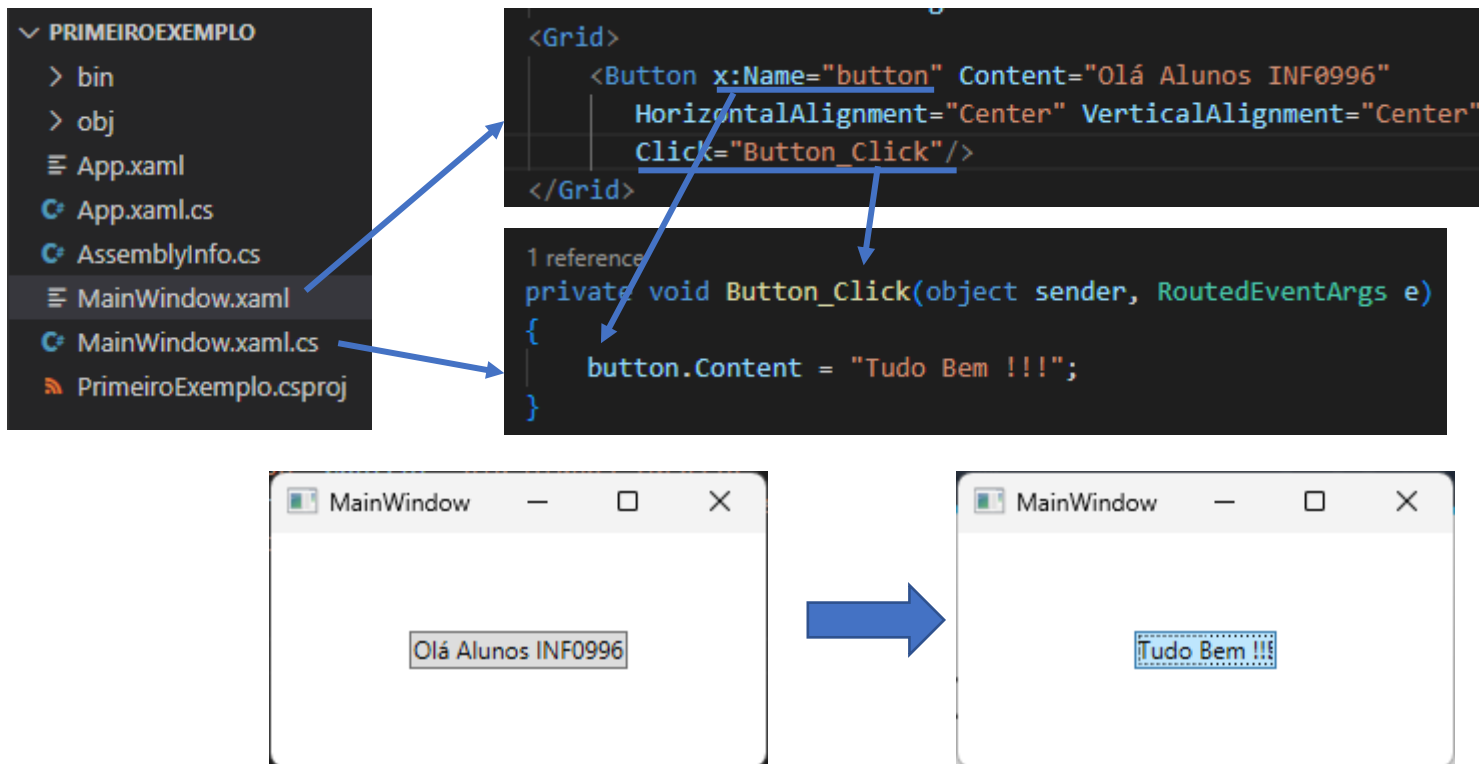
```
<StackPanel>  
    <Button>First Button</Button>  
    <Button>Second Button</Button>  
</StackPanel>
```



Conceitos Básicos - WPF - XAML



- O XAML – Sintaxe de atributo (eventos)
 - Exemplo, evento "Click" no botão



<https://learn.microsoft.com/pt-br/dotnet/desktop/wpf/xaml/?view=netdesktop-6.0>



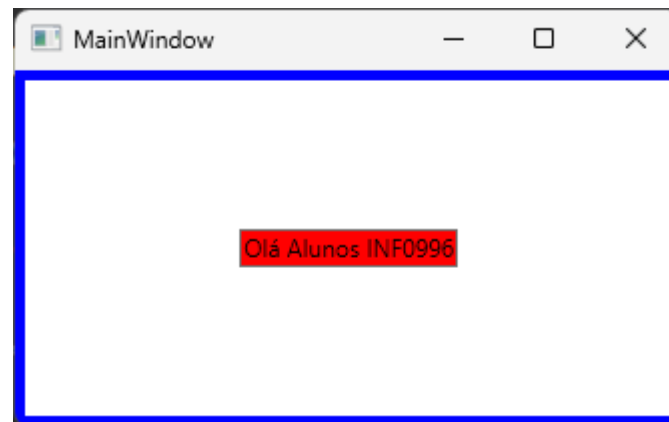
- Maiúsculas e minúsculas e espaço em branco no XAML
 - O XAML diferencia maiúsculas de minúsculas
- Ignora ou remove espaço em branco e normaliza qualquer espaço em branco não significativo
 - O XAML converte caracteres de espaço, de linha e de tabulação em espaços e preserva um espaço se encontrado em qualquer extremidade de uma cadeia de caracteres contígua
 - `xml:space="preserve"` -> Especifique esse atributo no nível do elemento onde a preservação de espaço em branco é desejada
 - `xml:space="default"`
 - Mais detalhes em: <https://learn.microsoft.com/en-us/dotnet/desktop/xaml-services/white-space-processing>

Conceitos Básicos - WPF - XAML



- Extensões de marcação
 - Quando usamos chaves ({ }) nos valores dos atributos
 - Referências StaticResource (em tempo de compilação) e DynamicResource (em tempo de execução)

```
<Window.Resources>
  <SolidColorBrush x:Key="MyBrush" Color="Gold"/>
  <Style TargetType="Border" x:Key="PageBackground">
    <Setter Property="BorderBrush" Value="Blue"/>
    <Setter Property="BorderThickness" Value="5" />
  </Style>
  <Style TargetType="Button" x:Key="BotaoPadrao">
    <Setter Property="HorizontalAlignment" Value="Center"/>
    <Setter Property="VerticalAlignment" Value="Center"/>
    <Setter Property="Background" Value="Red"/>
  </Style>
</Window.Resources>
<Border Style="{StaticResource PageBackground}">
  <Grid>
    <Button x:Name="button" Content="Olá Alunos INF0996"
      Style="{StaticResource BotaoPadrao}"
      Click="Button_Click"/>
  </Grid>
</Border>
```



<https://learn.microsoft.com/pt-br/dotnet/desktop/wpf/xaml/?view=netdesktop-6.0>



- Conversores de Tipo
 - O XAML utiliza strings que são processadas e convertidas em diferentes tipos

```
<Button Margin="10,20,10,30" Content="Click me"/>
```

São equivalentes



```
<Button Content="Click me">  
  <Button.Margin>  
    <Thickness Left="10" Top="20" Right="10" Bottom="30"/>  
  </Button.Margin>  
</Button>
```

- Detalhes sobre conversores de tipos: <https://learn.microsoft.com/pt-br/dotnet/desktop/wpf/advanced/typeconverters-and-xaml?view=netframeworkdesktop-4.8&viewFallbackFrom=netdesktop-6.0>

<https://learn.microsoft.com/pt-br/dotnet/desktop/wpf/xaml/?view=netdesktop-6.0>



- Elementos raiz e *namespaces*

```
MainWindow.xaml
1 <Window x:Class="PrimeiroExemplo.MainWindow"
2       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4       xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
5       xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
6       xmlns:local="clr-namespace:PrimeiroExemplo"
7       mc:Ignorable="d"
8       Title="MainWindow" Height="450" Width="800">
```



Conceitos Básicos - WPF - XAML



- Elementos Raiz e *namespaces*
 - Um arquivo XAML deve ter apenas um elemento raiz
 - Por exemplo, [Window](#) ou [Page](#) para uma página, [ResourceDictionary](#) para um dicionário externo ou [Application](#) para a definição do aplicativo

```
App.xaml
1 <Application x:Class="PrimeiroExemplo.App"
2   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4   xmlns:local="clr-namespace:PrimeiroExemplo"
5   StartupUri="MainWindow.xaml">
6   <Application.Resources>
7
8   </Application.Resources>
9 </Application>
```

```
MainWindow.xaml
1 <Window x:Class="PrimeiroExemplo.MainWindow"
2   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4   xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
5   xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
6   xmlns:local="clr-namespace:PrimeiroExemplo"
7   mc:Ignorable="d"
8   Title="MainWindow" Height="450" Width="800">
```

<https://learn.microsoft.com/pt-br/dotnet/desktop/wpf/xaml/?view=netdesktop-6.0>



- Elementos Raiz e *namespaces*
 - O elemento raiz também contém os atributos xmlns e xmlns:x
 - O atributo xmlns indica especificamente o namespace XAML padrão
 - Dentro do namespace XAML padrão, os elementos de objeto na marcação podem ser especificados sem um prefixo.
 - O atributo xmlns:x indica um namespace XAML adicional, que mapeia o namespace

```
MainWindow.xaml
1 <Window x:Class="PrimeiroExemplo.MainWindow"
2   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4   xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
5   xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
6   xmlns:local="clr-namespace:PrimeiroExemplo"
7   mc:Ignorable="d"
8   Title="MainWindow" Height="450" Width="800">
```



- Elementos Raiz - O prefixo **x:**
 - O prefixo x: é usado para mapear o namespace XAML nos modelos dos projetos.
 - ***x:Key***: define uma chave exclusiva para cada recurso em um ResourceDictionary

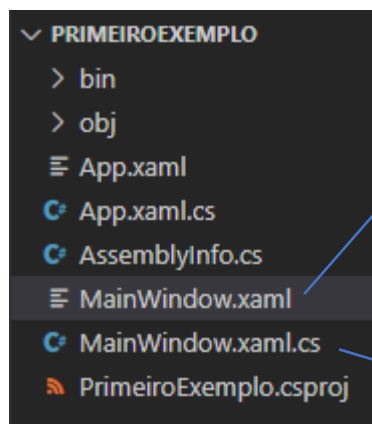
```
<Window.Resources>
  <SolidColorBrush x:Key="MyBrush" Color="Gold"/>
  <Style TargetType="Border" x:Key="PageBackground">
    <Setter Property="BorderBrush" Value="Blue"/>
    <Setter Property="BorderThickness" Value="5" />
  </Style>
  <Style TargetType="Button" x:Key="BotaoPadrao">
    <Setter Property="HorizontalAlignment" Value="Center"/>
    <Setter Property="VerticalAlignment" Value="Center"/>
    <Setter Property="Background" Value="Red"/>
  </Style>
</Window.Resources>
<Border Style="{StaticResource PageBackground}">
  <Grid>
    <Button x:Name="button" Content="Olá Alunos INF0996"
      Style="{StaticResource BotaoPadrao}"
      Click="Button_Click"/>
  </Grid>
</Border>
```

<https://learn.microsoft.com/pt-br/dotnet/desktop/wpf/xaml/?view=netdesktop-6.0>

Conceitos Básicos - WPF - XAML



- Elementos Raiz - O prefixo **x:**
 - **x:Class:** especifica o namespace CLR (*common language runtime*) e o nome da classe que fornece *code-behind* para uma página XAML. Você deve ter uma classe desse tipo para dar suporte a *code-behind* segundo o modelo de programação do WPF.



```
MainWindow.xaml
1 <Window x:Class="PrimeiroExemplo.MainWindow"
2       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
```

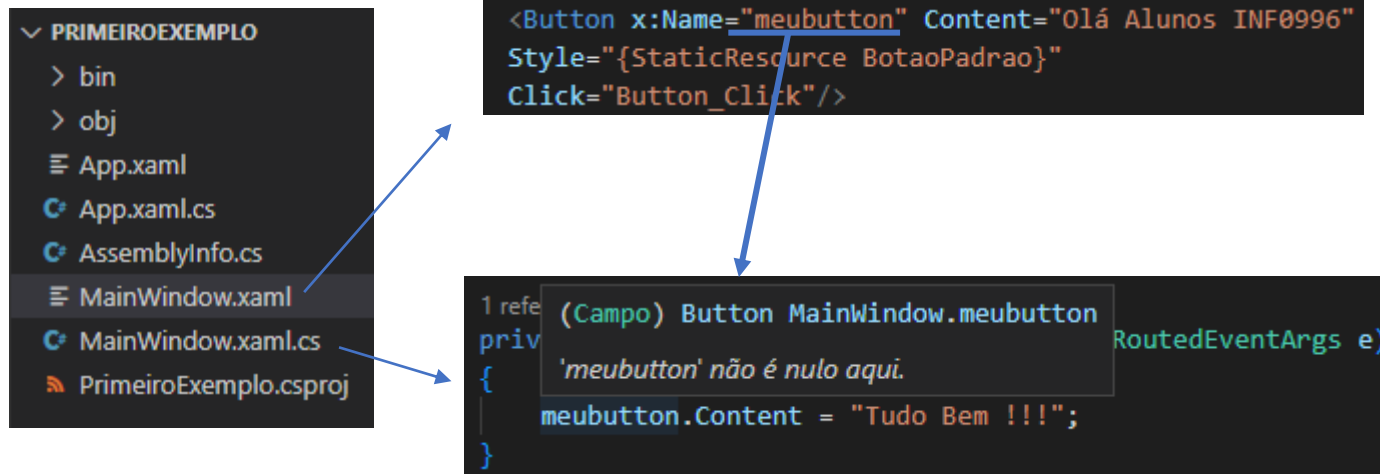
```
namespace PrimeiroExemplo
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    0 references
    public partial class MainWindow : Window
    {
        0 references
    }
}
```

<https://learn.microsoft.com/pt-br/dotnet/desktop/wpf/xaml/?view=netdesktop-6.0>

Conceitos Básicos - WPF - XAML



- Elementos Raiz - O prefixo **x:**
 - **x:Name:** Identifica exclusivamente elementos definidos pelo XAML em um namespace XAML. Essas propriedades são mapeadas especificamente para uma propriedade de suporte CLR e, portanto, são mais convenientes para programação de aplicativos.





- Elementos Raiz - O prefixo **x:**
 - **x:Static**: Faz referência a qualquer entidade de código de valor estático definida em uma maneira compatível com Common Language Specification (CLS).

```
<object property="{x:Static prefix:typeName.staticMemberName}" .../>
```

A entidade de código que é referenciada deve ser uma das seguintes:

- Uma constante
 - Uma propriedade estática
 - Um campo
 - Um valor de enumeração
- O namespace XAML padrão que você usa para a programação do WPF não contém muitas propriedades estáticas úteis, e a maioria das propriedades estáticas úteis têm suporte como conversores de tipo que facilitam o uso sem a necessidade {x:Static}



- Elementos Raiz - O prefixo **x:**
 - **x:Type:** constrói uma referência de [Type](#) com base em um nome de tipo. Isso é usado para especificar atributos que levam [Type](#), como [Style.TargetType](#), embora frequentemente a propriedade tenha conversão de cadeia de caracteres nativa para [Type](#) de forma que o uso da extensão de marcação [x:Type](#) seja opcional.

```
<Style TargetType="{x:Type Button}" x:Key="BotaoPadrao">
  <Setter Property="HorizontalAlignment" Value="Center"/>
  <Setter Property="VerticalAlignment" Value="Center"/>
  <Setter Property="Background" Value="Red"/>
</Style>
```



Equivalentes

```
<Style TargetType="Button" x:Key="BotaoPadrao">
  <Setter Property="HorizontalAlignment" Value="Center"/>
  <Setter Property="VerticalAlignment" Value="Center"/>
  <Setter Property="Background" Value="Red"/>
</Style>
```



- Elementos Raiz - O prefixo **x:**

```
<x:Array Type="typeName">  
    arrayContents  
</x:Array>
```

```
<object xml:lang="rfc3066lang" />
```

Outros prefixos: <https://learn.microsoft.com/pt-br/dotnet/desktop/xaml-services/namespace-language-features?view=netdesktop-6.0>




- Prefixos personalizados e tipos personalizados
 - Você pode especificar o *assembly* (ex: *.dll*) como parte de um mapeamento de xmlns personalizado. Você pode então referenciar tipos desse *assembly* em seu XAML, desde que esse tipo esteja corretamente implementado

```
<Page
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:custom="clr-namespace:NumericUpDownCustomControl;assembly=CustomLibrary"
>
  <StackPanel Name="LayoutRoot">
    <custom:NumericUpDown Name="numericCtrl1" Width="100" Height="60"/>
    ...
  </StackPanel>
</Page>
```



- Eventos e *code-behind* XAML
 - Em um projeto, o XAML é escrito como um arquivo *.xaml* e uma linguagem CLR, como Microsoft Visual Basic ou C#, é usada para escrever um arquivo *code-behind*.

```
<Window x:Class="index.Window2"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Window2" Height="450" Width="800">
    <StackPanel>
        <Button Click="Button_Click">Click me</Button>
    </StackPanel>
</Window>
```



```
private void Button_Click(object sender, RoutedEventArgs e)
{
    var buttonControl = (Button)e.Source;
    buttonControl.Foreground = Brushes.Red;
}
```



- Eventos roteados
 - Eventos roteados habilitam um elemento para manipular um evento que foi acionado por um elemento diferente, desde que os elementos estejam conectados por meio de uma relação de árvore
 - Ao especificar o tratamento de eventos com um atributo XAML, o evento roteado pode ser escutado e manipulado em qualquer elemento, incluindo elementos que não listam esse evento específico na tabela de membros da classe
 - Por exemplo, o StackPanel pai no exemplo de StackPanel / Button em andamento pode registrar um manipulador para o evento Click do botão de elemento filho especificando o atributo Button

<https://learn.microsoft.com/pt-br/dotnet/desktop/wpf/xaml/?view=netdesktop-6.0>



- Elementos nomeados
 - Por padrão, objeto XAML não tem um identificador exclusivo ou uma referência de objeto.
 - Você pode definir o valor do atributo *x:Name* em qualquer elemento de objeto. Em seu *code-behind*, o identificador escolhido é equivalente a uma variável

```
<StackPanel Name="buttonContainer">  
    <Button Click="RemoveThis_Click">Click to remove this button</Button>  
</StackPanel>
```

```
private void RemoveThis_Click(object sender, RoutedEventArgs e)  
{  
    var element = (FrameworkElement)e.Source;  
  
    if (buttonContainer.Children.Contains(element))  
        buttonContainer.Children.Remove(element);  
}
```

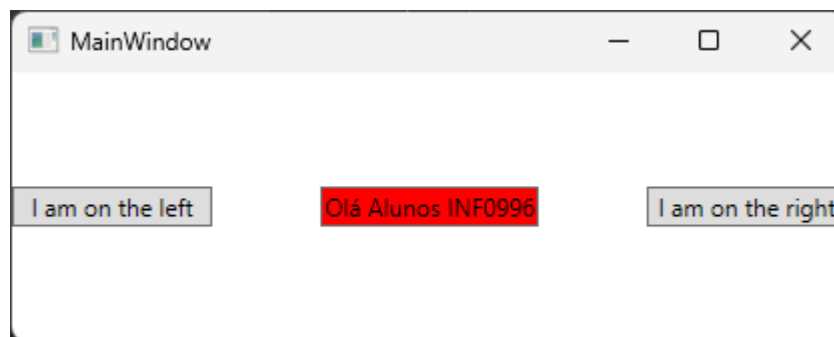
<https://learn.microsoft.com/pt-br/dotnet/desktop/wpf/xaml/?view=netdesktop-6.0>



- Propriedades anexadas e eventos anexados
 - Permite que determinadas propriedades ou eventos sejam especificados em qualquer elemento, mesmo que a propriedade ou evento não exista nas definições do tipo para o elemento em que está sendo definido.

```
<DockPanel>
  <Button DockPanel.Dock="Left" Width="100" Height="20">I am on the left</Button>
  <Button DockPanel.Dock="Right" Width="100" Height="20">I am on the right</Button>
  <Button x:Name="meubutton" Content="Olá Alunos INF0996"
    Style="{StaticResource BotaoPadrao}"
    Click="Button_Click"/>
</DockPanel>
```

Propriedade anexada. A classe DockPanel define os "acessadores" para DockPanel.Dock e possui a propriedade anexada.



<https://learn.microsoft.com/pt-br/dotnet/desktop/wpf/xaml/?view=netdesktop-6.0>



- Tipos de base
 - O XAML do WPF subjacente e seu namespace XAML são uma coleção de tipos que correspondem a objetos CLR e elementos de marcação para XAML
 - Classes base, incluindo aquelas abstratas, são importantes para o desenvolvimento de XAML, porque cada um dos elementos XAML concretos herda os membros de alguma classe base na sua hierarquia
 - [FrameworkElement](#) é a classe de interface do usuário base concreta do WPF no nível da estrutura do WPF

<https://learn.microsoft.com/pt-br/dotnet/api/system.windows.frameworkelement?view=windowsdesktop-6.0>

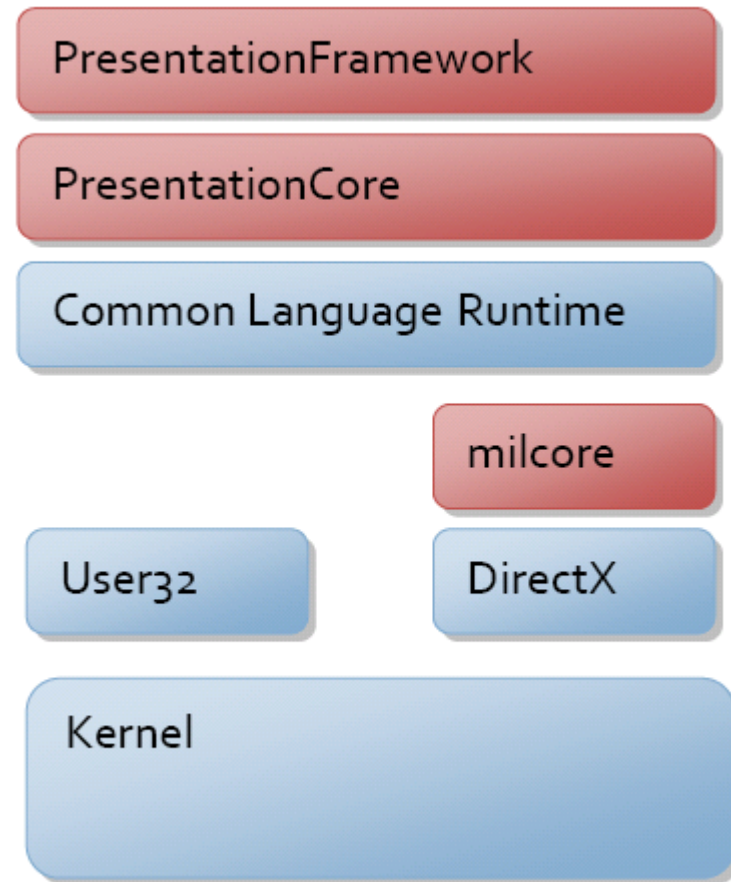
<https://learn.microsoft.com/pt-br/dotnet/desktop/wpf/xaml/?view=netdesktop-6.0>

Conceitos Básicos - WPF - Arquitetura



- Arquitetura - *System.Object*

- As seções vermelhas do diagrama (*PresentationFramework*, *PresentationCore* e *milcore*) são as principais partes de código do WPF
- O *milcore* é escrito em código não gerenciado integrado o DirectX.
- Toda a exibição no WPF é feita por meio do mecanismo DirectX.





- Arquitetura - *System.Threading.DispatcherObject*
 - A maioria dos objetos no WPF deriva de *DispatcherObject*, que fornece as construções básicas para lidar com simultaneidade e threading
 - Você tem objetos com threading STA (afinidade de thread único – para thread físico) que precisa de uma maneira para se comunicar entre threads e validar que você está no thread correto
 - O dispatcher é um sistema básico de despacho de mensagens, com várias filas priorizadas

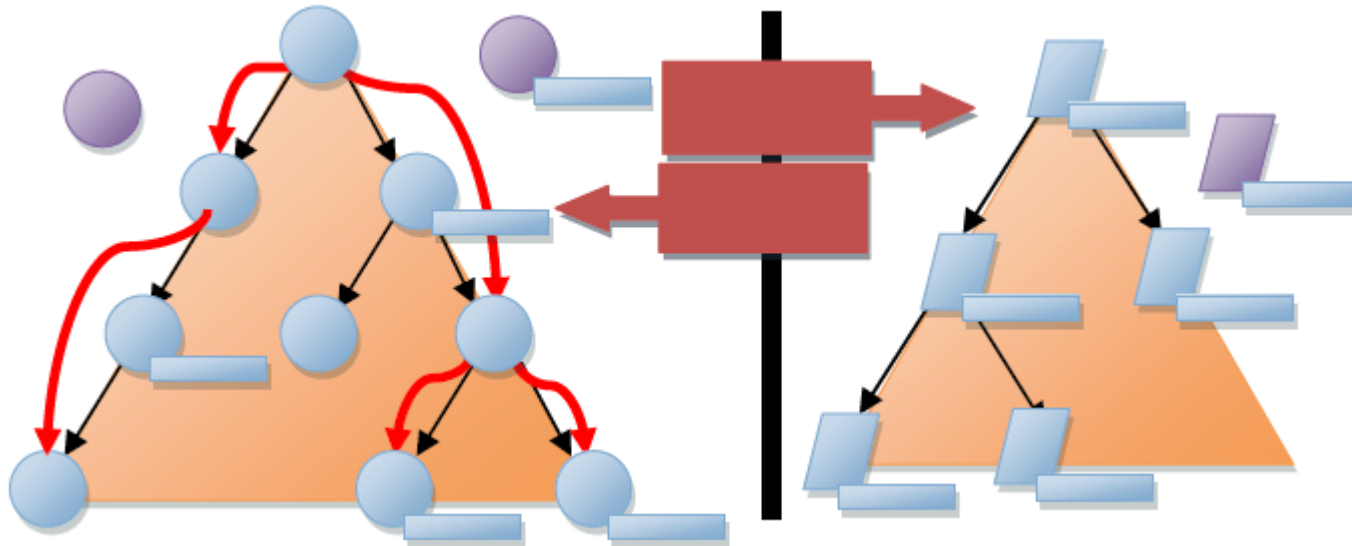


- Arquitetura - *System.Windows.DependencyObject*
 - Uma das principais filosofias arquitetônicas usadas na construção do WPF é a preferência por propriedades em relação a métodos ou eventos
 - O WPF fornece um sistema de propriedades mais avançado, derivado do *DependencyObject* que:
 - Controla as dependências entre expressões de propriedade e revalida automaticamente valores
 - Por exemplo, se você tiver uma propriedade que herda (como *FontSize*), o sistema será atualizado automaticamente se a propriedade for alterada em um pai de um elemento que herda o valor
 - Fornece um modelo de expressão de propriedade
 - Propriedades anexadas

Conceitos Básicos - WPF - Arquitetura



- Arquitetura - *System.Windows.Media.Visual*
 - A classe *Visual* fornece a criação de uma árvore de objetos visuais, cada uma contendo opcionalmente instruções de desenho e metadados sobre como renderizar essas instruções (recorte, transformação etc.).





- Arquitetura - *System.Windows.UIElement*
 - UIElement define subsistemas principais, incluindo Layout, Entrada e Eventos
 - Layout flexível e extensível controlado por valores, em duas fases:
 - Measure permite que um componente determine quanto tamanho ele gostaria de ter
 - A fase Arrange permite que um pai posicione e determine o tamanho final de cada filho
 - A entrada se origina como um sinal em um driver de dispositivo de modo kernel e é roteada para o processo e thread
 - Cada evento de entrada é convertido em pelo menos dois eventos – um evento de "visualização" e o evento real

Conceitos Básicos - WPF - Arquitetura



- Arquitetura - *System.Windows.FrameworkElement*
 - Apresenta um conjunto de políticas e personalizações nos subsistemas introduzidos em camadas inferiores do WPF
- Arquitetura - *System.Windows.Controls.Control*
 - Um *ControlTemplate* não é nada mais do que um script para criar um conjunto de elementos filhos, com associações às propriedades oferecidas pelo controle

The background features a network of gray lines connecting various colored circles (orange, yellow, green, blue) and a central dark blue horizontal band with a white circuit-like pattern.

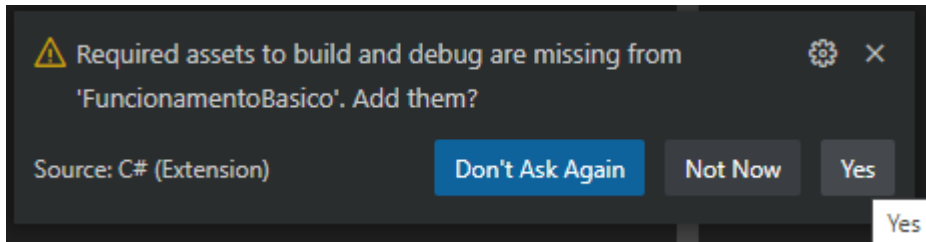
Atividades Práticas com WPF

Práticas com WPF – Funcionamento Básico

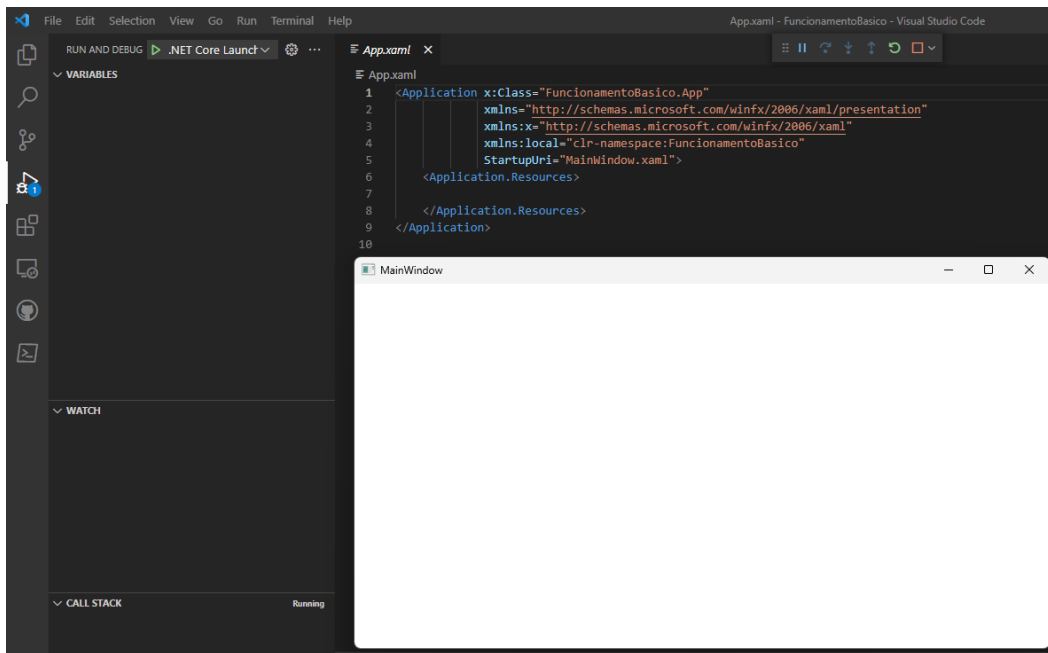


```
\FuncionamentoBasico> dotnet new wpf
```

Cria o projeto
no terminal



Debug (F5)



Execução no
terminal

```
\FuncionamentoBasico> dotnet run
```

Práticas com WPF – Funcionamento Básico



App.xaml

```
1 <Application x:Class="FuncionamentoBasico.App"
2   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4   xmlns:local="clr-namespace:FuncionamentoBasico"
5   StartupUri="AppInf0996.xaml">
```

Modificar janela
inicial

AppInf0996.xaml

AppInf0996.xaml.cs

Arquivo C#

Título, tamanho da janela
quando minimizada e estado
inicial (*fullscreen*)

AppInf0996.xaml

```
1 <Window x:Class="FuncionamentoBasico.AppInf0996"
2   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4   xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
5   xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
6   xmlns:local="clr-namespace:FuncionamentoBasico"
7   mc:Ignorable="d"
8   Title="Janela Principal - AppInf0996" Width="1024" Height="768" WindowState="Maximized">
9   <Grid>
10
11   </Grid>
12 </Window>
```

Layout do tipo
"Grid"

Práticas com WPF – Funcionamento Básico



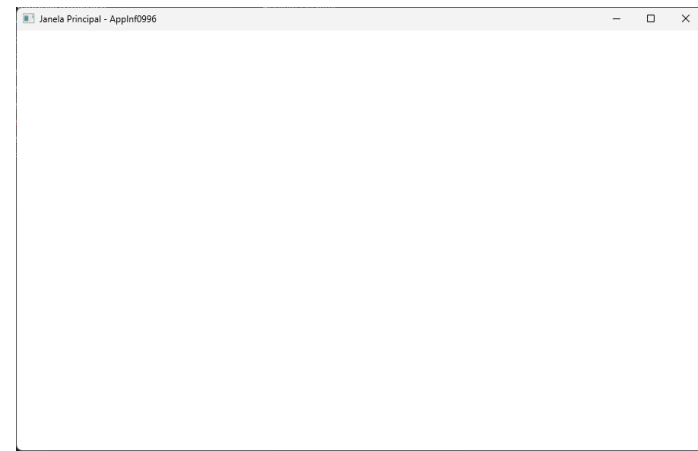
C# AppInf0996.xaml.cs

```
<Window x:Class="FuncionamentoBasico.AppInf0996">
```

C# - code-behind

```
namespace FuncionamentoBasico
{
    /// <summary>
    /// Interaction logic for AppInf0996.xaml
    /// </summary>
    0 references
    public partial class AppInf0996 : Window
    {
        0 references
        public AppInf0996()
        {
            InitializeComponent();
        }
    }
}
```

```
class System.Windows.Window
```





- Os painéis de layout são contêineres que permitem organizar e agrupar elementos da interface do usuário no aplicativo
- Existem vários containers, cada um com uma característica específica e mais adequado para certas finalidades, ex:
 - Grid: permite organizar os controles em linhas e colunas
 - StackPanel: organiza todos os seus objetos filhos de forma horizontal ou vertical
 - DockPanel: alinhar os controles colocados em seu interior em posições fixas: no topo, na base, nas laterais, ou no centro
 - Canvas: posiciona explicitamente os objetos
 - VirtualizingStackPanel: os controles filho são virtualizados e organizados em uma única linha que é orientado horizontalmente ou verticalmente
 - WrapPanel: os controles filho são posicionados na ordem da esquerda para a direita e agrupados na próxima linha quando houver não há espaço suficiente. na linha atual.

Práticas com WPF – *Painéis de layout*



A primeira coluna tem uma Largura de "3*", enquanto a segunda tem "5*", dividindo o espaço horizontal entre as duas colunas a uma proporção de 3:5.

```
<Grid>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="3*" />
    <ColumnDefinition Width="5*" />
  </Grid.ColumnDefinitions>
  <Grid.RowDefinitions>
    <RowDefinition Height="2*" />
    <RowDefinition Height="*" />
  </Grid.RowDefinitions>
  <Border Background="#2f5cb6" />
  <Border Grid.Column="1" Background="#1f3d7a" />
  <Border Grid.Row="1" Grid.ColumnSpan="2" Background="#152951" />
</Grid>
```

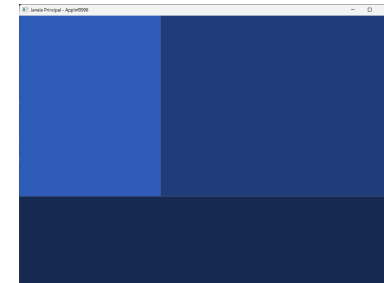
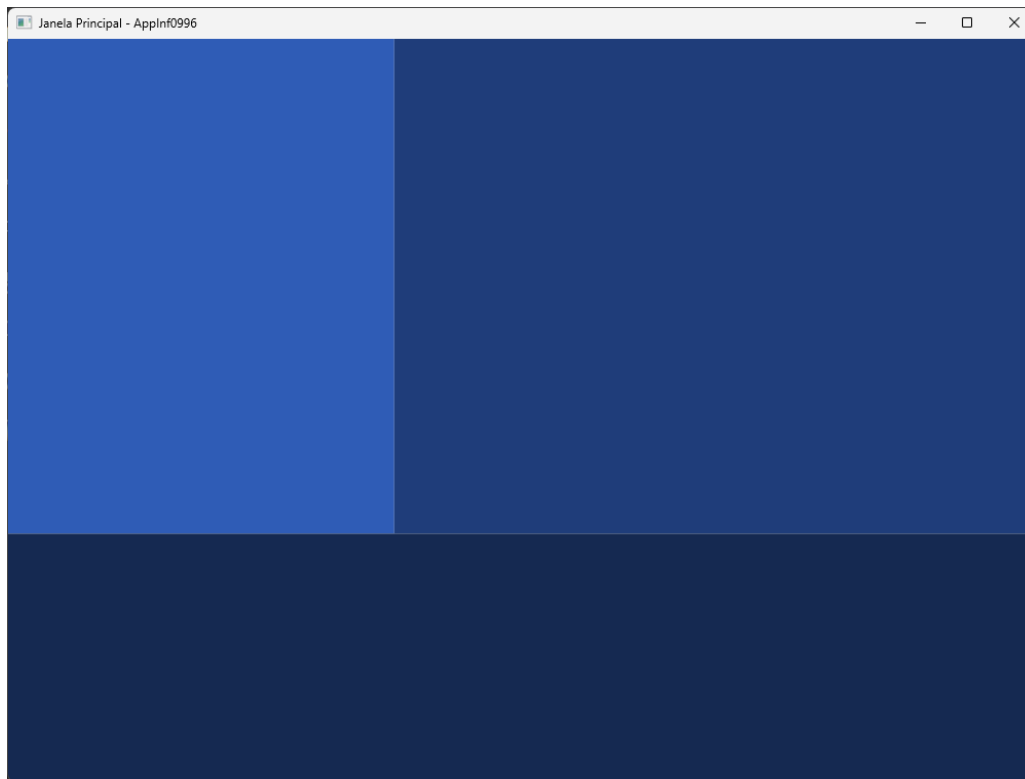
As duas linhas têm uma altura de "2*" e "*", respectivamente, portanto, a Grade aloca duas vezes mais espaço para a primeira linha do que para a segunda ("*" é igual a "1*")

Definição de Cores e bordas
(a coluna começa em 0)

Práticas com WPF – *Painéis de layout*



- Testando ... (proporções são mantidas em qualquer tamanho de tela)



Práticas com WPF – *Painéis de layout*



Inserir StackPanel
dentro do Grid

```
<StackPanel Grid.Column="1" Margin="40,0,0,0" VerticalAlignment="Center">
  <TextBlock Foreground="White" FontSize="25" Text="Temperatura Hoje - 25° C"/>
  <TextBlock Foreground="White" FontSize="25" Text="Parcialmente Nublado"/>
  <TextBlock Foreground="White" FontSize="25" Text="Chuva: 25%"/>
</StackPanel>
<StackPanel Grid.Row="1" Grid.ColumnSpan="2" Orientation="Horizontal"
  HorizontalAlignment="Center" VerticalAlignment="Center">
  <TextBlock Foreground="White" FontSize="25" Text="Máxima: 33°" Margin="0,0,20,0"/>
  <TextBlock Foreground="White" FontSize="25" Text="Mínima: 17°" Margin="0,0,20,0"/>
  <TextBlock Foreground="White" FontSize="25" Text="Sensação Térmica: 26°"/>
</StackPanel>
```

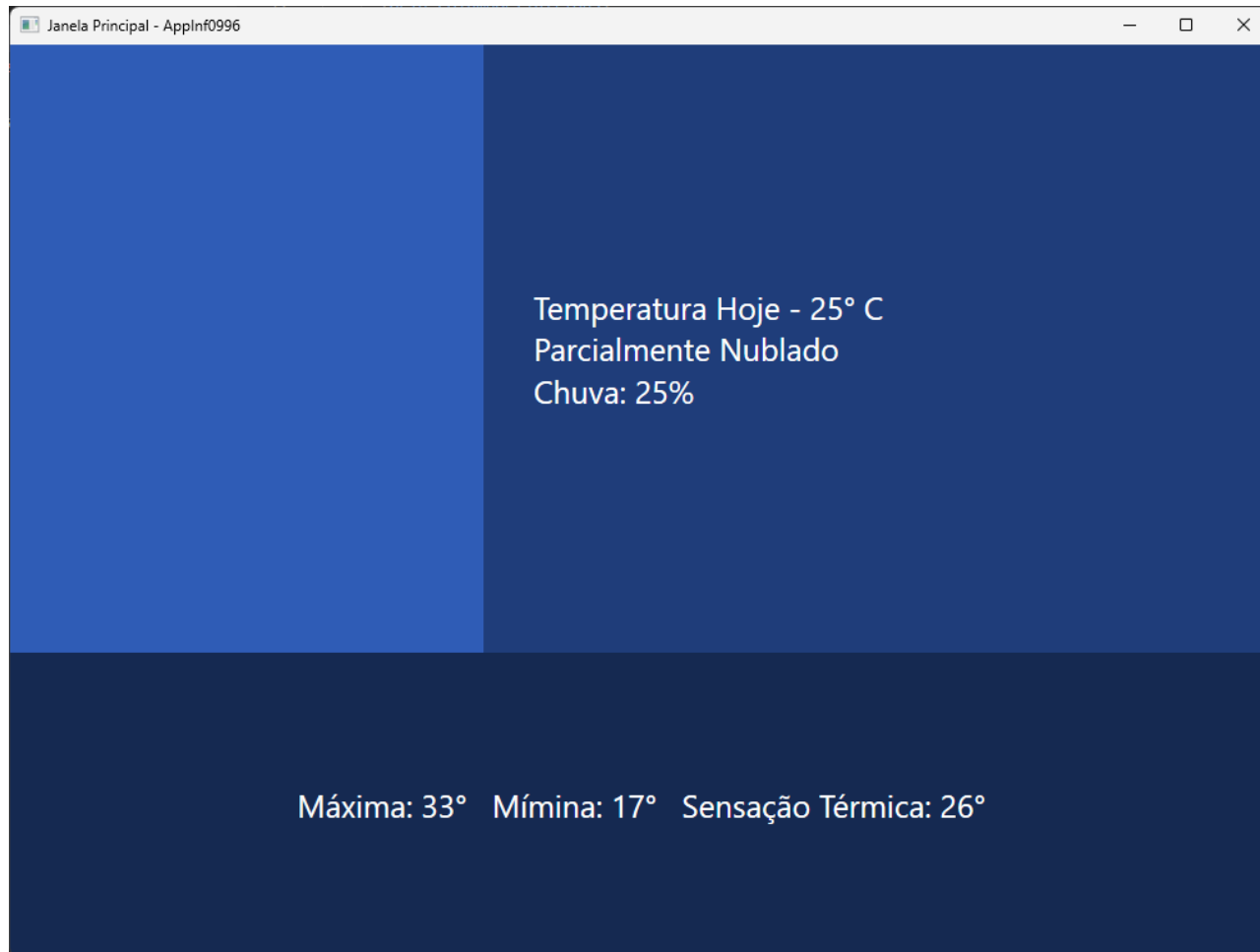
TextBlocks dentro
do StackPanel em
alinhamento
Vertical

TextBlocks dentro
do StackPanel em
alinhamento
Horizontal

Práticas com WPF – *Painéis de layout*



- Testando ...



<https://learn.microsoft.com/pt-br/windows/apps/design/layout/layout-panels> - (Winui 3)

Práticas com WPF – *Painéis de layout*



Adicionar na
sequência

```
<Image Grid.Column="0" Grid.Row="0" Margin="20"  
Source="D:\Bonacin\ICUnicamp\Curso_Samsung_MS\XAML\WPF1\FuncionamentoBasico\Assets\partially-cloudy.png"/>
```



<https://learn.microsoft.com/pt-br/windows/apps/design/layout/layout-panels> - (Winui 3)

Práticas com WPF – *Botão*



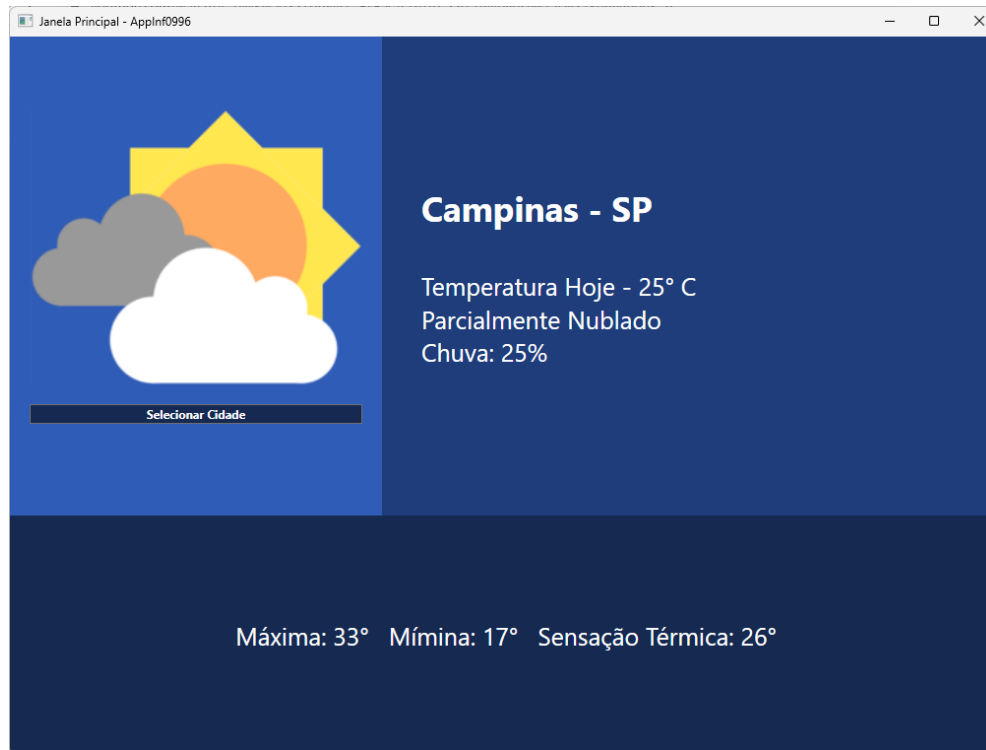
Nome da Cidade

```
<StackPanel Grid.Column="1" Margin="40,0,0,0" VerticalAlignment="Center">
  <TextBlock Foreground="White" Margin="0,0,0,40" FontSize="35" FontWeight="Bold"
    Name="CidadeTempo"
    Text="Campinas - SP"/>
  <TextBlock Foreground="White" FontSize="25" Text="Temperatura Hoje - 25° C"/>
  <TextBlock Foreground="White" FontSize="25" Text="Parcialmente Nublado"/>
  <TextBlock Foreground="White" FontSize="25" Text="Chuva: 25%"/>
</StackPanel>

<StackPanel Grid.Row="1" Grid.ColumnSpan="2" Orientation="Horizontal"
  HorizontalAlignment="Center" VerticalAlignment="Center">
  <TextBlock Foreground="White" FontSize="25" Text="Máxima: 33°" Margin="0,0,20,0"/>
  <TextBlock Foreground="White" FontSize="25" Text="Mínima: 17°" Margin="0,0,20,0"/>
  <TextBlock Foreground="White" FontSize="25" Text="Sensação Térmica: 26°"/>
</StackPanel>

<StackPanel Grid.Column="0" Grid.Row="0" Margin="0,0,0,0" VerticalAlignment="Center">
  <Image Margin="20,0,20,0"
    Source="D:\Bonacin\ICUnicamp\Curso_Samsung_MS\XAML\WPF1\FuncionamentoBasico\Assets\partially-cloudy.png"/>
  <Button Margin="20,20,20,20" Name="button" Foreground="White"
    FontWeight="Bold" Background="#152951" >Selecionar Cidade</Button>
</StackPanel>
```

Inclusão de painel
com botão de
seleção da cidade





Menu no canto
esquerdo superior

```
<MenuItem Grid.Column = "1" Click="MenuSuperior_Click" x:Name="MenuSuperiorEsquerdo"
    Background="#1f3d7a" VerticalAlignment="Top" HorizontalAlignment="Right"
    Height="50">
    <MenuItem.Header>
        <StackPanel Orientation="Horizontal" VerticalAlignment="Top" HorizontalAlignment="Right">
            <Image Source="D:\Bonacin\ICUnicamp\Curso_Samsung_MS\XAML\WPF1\FuncionamentoBasico\Assets\menuicon.png" Height="30"
                Margin="20,0,0,0" />
        </StackPanel>
    </MenuItem.Header>
    <MenuItem.ContextMenu>
        <ContextMenu Background="#1f3d7a" BorderThickness="0" Padding="0">
            <MenuItem Background="#1f3d7a" BorderThickness="0" FontWeight="Bold" FontSize="14" Header="Opções">
                <MenuItem.Icon>
                    <Image Source="D:\Bonacin\ICUnicamp\Curso_Samsung_MS\XAML\WPF1\FuncionamentoBasico\Assets\opcoes.png" Width="24" />
                </MenuItem.Icon>
            </MenuItem>
            <MenuItem Background="#1f3d7a" BorderThickness="0" FontWeight="Bold" FontSize="14" Header="Sair" Click="Sair_Click">
                <MenuItem.Icon>
                    <Image Source="D:\Bonacin\ICUnicamp\Curso_Samsung_MS\XAML\WPF1\FuncionamentoBasico\Assets\sair.png" Width="24" />
                </MenuItem.Icon>
            </MenuItem>
        </ContextMenu>
    </MenuItem.ContextMenu>
</MenuItem>
```

Práticas com WPF – Menu



Ícone quando o menu está fechado

```
<MenuItem Grid.Column = "1" Click="MenuSuperior_Click" x:Name="MenuSuperiorEsquerdo"
    Background="#1f3d7a" VerticalAlignment="Top" HorizontalAlignment="Right"
    Height="50">
    <MenuItem.Header>
        <StackPanel Orientation="Horizontal" VerticalAlignment="Top" HorizontalAlignment="Right">
            <Image Source="D:\Bonacin\ICUnicamp\Curso_Samsung_MS\XAML\WPF1\FuncionamentoBasico\Assets\menuicon.png" Height="30"
                Margin="20,0,0,0" />
        </StackPanel>
    </MenuItem.Header>
    <MenuItem.ContextMenu>
        <ContextMenu Background="#1f3d7a" BorderThickness="0" Padding="0">
            <MenuItem Background="#1f3d7a" BorderThickness="0" FontWeight="Bold" FontSize="14" Header="Opções">
                <MenuItem.Icon>
                    <Image Source="D:\Bonacin\ICUnicamp\Curso_Samsung_MS\XAML\WPF1\FuncionamentoBasico\Assets\opcoes.png" Width="24" />
                </MenuItem.Icon>
            </MenuItem>
            <MenuItem Background="#1f3d7a" BorderThickness="0" FontWeight="Bold" FontSize="14" Header="Sair" Click="Sair_Click">
                <MenuItem.Icon>
                    <Image Source="D:\Bonacin\ICUnicamp\Curso_Samsung_MS\XAML\WPF1\FuncionamentoBasico\Assets\sair.png" Width="24" />
                </MenuItem.Icon>
            </MenuItem>
        </ContextMenu>
    </MenuItem.ContextMenu>
</MenuItem>
```



```
<MenuItem Grid.Column = "1" Click="MenuSuperior_Click" x:Name="MenuSuperiorEsquerdo"
    Background="#1f3d7a" VerticalAlignment="Top" HorizontalAlignment="Right"
    Height="50">
    <MenuItem.Header>
        <StackPanel Orientation="Horizontal" VerticalAlignment="Top" HorizontalAlignment="Right">
            <Image Source="D:\Bonacin\ICUnicamp\Curso_Samsung_MS\XAML\WPF1\FuncionamentoBasico\Assets\menuicon.png" Height="30"
                Margin="20,0,0,0" />
        </StackPanel>
    </MenuItem.Header>
    <MenuItem.ContextMenu>
        <ContextMenu Background="#1f3d7a" BorderThickness="0" Padding="0">
            <MenuItem Background="#1f3d7a" BorderThickness="0" FontWeight="Bold" FontSize="14" Header="Opções">
                <MenuItem.Icon>
                    <Image Source="D:\Bonacin\ICUnicamp\Curso_Samsung_MS\XAML\WPF1\FuncionamentoBasico\Assets\opcoes.png" Width="24" />
                </MenuItem.Icon>
            </MenuItem>
            <MenuItem Background="#1f3d7a" BorderThickness="0" FontWeight="Bold" FontSize="14" Header="Sair" Click="Sair_Click">
                <MenuItem.Icon>
                    <Image Source="D:\Bonacin\ICUnicamp\Curso_Samsung_MS\XAML\WPF1\FuncionamentoBasico\Assets\sair.png" Width="24" />
                </MenuItem.Icon>
            </MenuItem>
        </ContextMenu>
    </MenuItem.ContextMenu>
</MenuItem>
```

Opções do Menu

Práticas com WPF – *Menu - Code-behind*



Método para tratar
click no menu

```
<MenuItem Grid.Column = "1" Click="MenuSuperior_Click" x:Name="MenuSuperiorEsquerdo"
    Background="#1f3d7a" VerticalAlignment="Top" HorizontalAlignment="Right"
    Height="50">
    <MenuItem.Header>
        <StackPanel Orientation="Horizontal" VerticalAlignment="Top" HorizontalAlignment="Right">
            <Image Source="D:\Bonacin\ICUnicamp\Curso_Samsung_MS\XAML\WPF1\FuncionamentoBasico\Assets\menuicon.png" Height="30"
                Margin="20,0,0,0" />
        </StackPanel>
    </MenuItem.Header>
    <MenuItem.ContextMenu>
        <ContextMenu Background="#1f3d7a" BorderThickness="0" Padding="0">
            <MenuItem Background="#1f3d7a" BorderThickness="0" FontWeight="Bold" FontSize="14" Header="Sair" Click="Sair_Click">
                <MenuItem.Icon>
                    <Image Source="D:\Bonacin\ICUnicamp\Curso_Samsung_MS\XAML\WPF1\FuncionamentoBasico\Assets\sair.png" Width="24" />
                </MenuItem.Icon>
            </MenuItem>
        </ContextMenu>
    </MenuItem.ContextMenu>
</MenuItem>
```

Método para tratar
opção sair

Práticas com WPF – *Menu - Code-behind*



Click="MenuSuperior_Click"

```
// Exibe Menu
1 reference
private void MenuSuperior_Click(object sender, RoutedEventArgs e)
{
    var addButton = sender as FrameworkElement;
    if (addButton != null)
    {
        addButton.ContextMenu.IsOpen = true;
    }
}
```

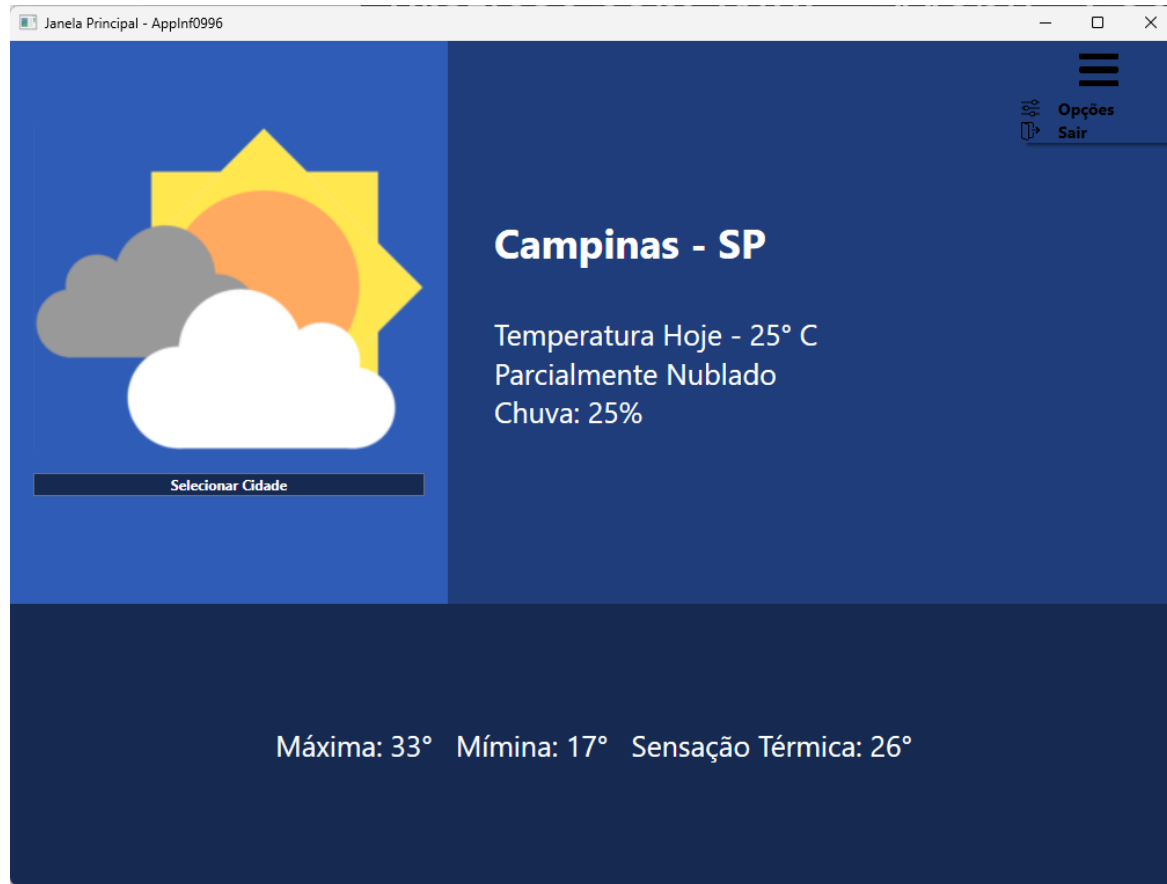
Abre o menu

Click="Sair_Click"

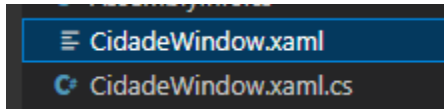
```
// Sai da Aplicação
1 reference
private void Sair_Click(object sender, RoutedEventArgs e)
{
    Application.Current.Shutdown();
}
```

Fecha a aplicação

Práticas com WPF – *Menu - Code-behind*



Práticas com WPF – *Janela Filha*



```
CidadeWindow.xaml
1 <Window x:Class="FuncionamentoBasico.CidadeWindow"
2       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4       xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
5       xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
6       xmlns:local="clr-namespace:FuncionamentoBasico"
7       mc:Ignorable="d"
8       Title="Seleção de Cidade" Height="450" Width="800">
9     <Grid>
10       <StackPanel Background="#2f5cb6">
11         <Label Foreground="White" Content="Cidade Window" HorizontalAlignment="Center" Margin="0,22,0,0" Name="label1" VerticalAlignment="Top" />
12
13         <Button Margin="20,20,20,20" Name="button" Foreground="White"
14             FontWeight="Bold" Background="#152951" Click="FecharJanela_Click" >Fechar</Button>
15       </StackPanel>
16     </Grid>
17 </Window>
```

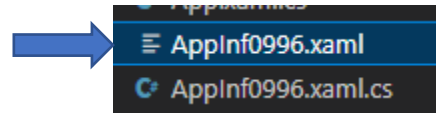
Práticas com WPF – *Janela Filha*



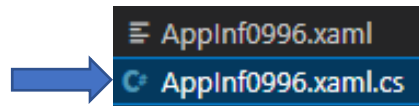
CidadeWindow.xaml
CidadeWindow.xaml.cs

```
16 namespace FuncionamentoBasico
17 {
18     /// <summary>
19     /// Interaction logic for AppInf0996.xaml
20     /// </summary>
21     2 references
22     public partial class CidadeWindow : Window
23     {
24         1 reference
25         public CidadeWindow()
26         {
27             InitializeComponent();
28
29             // Sai da Aplicação
30             1 reference
31             private void FecharJanela_Click(object sender, RoutedEventArgs e)
32             {
33                 this.Hide();
34             }
35         }
36     }
```

Práticas com WPF – *Janela Filha*



```
<StackPanel Grid.Column="0" Grid.Row="0" Margin="0,0,0,0" VerticalAlignment="Center">
    <Image Margin="20,0,20,0"
        Source="D:\Bonacin\ICUnicamp\Curso_Samsung_MS\XAML\WPF1\FuncionamentoBasico\Assets\partially-cloudy.png"/>
    <Button Margin="20,20,20,20" Name="button" Foreground="White"
        FontWeight="Bold" Background="#152951" Click="Cidade_Click" >Selecionar Cidade</Button>
</StackPanel>
```

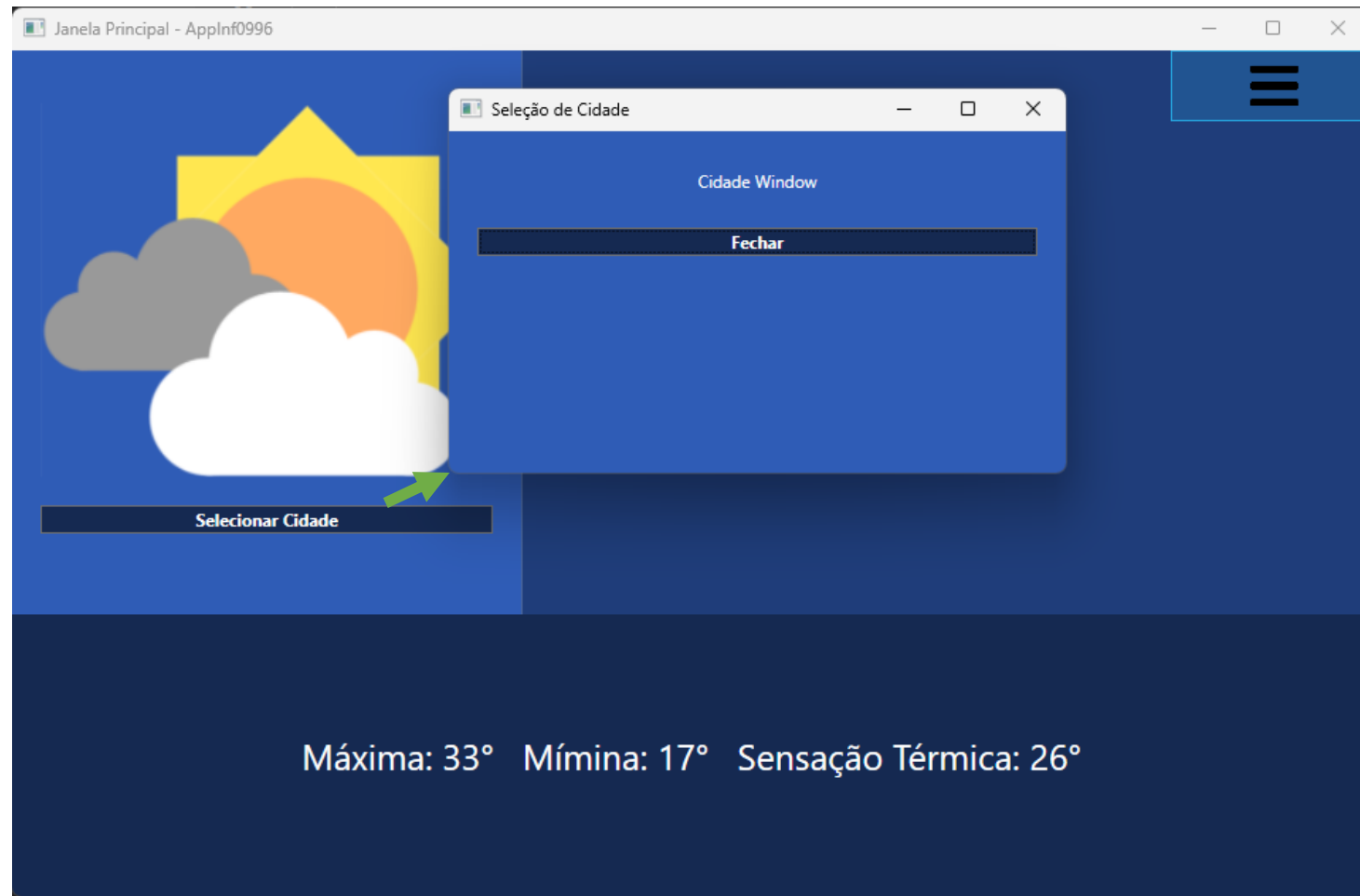


```
public partial class AppInf0996 : Window
{
    3 references
    CidadeWindow Cw;

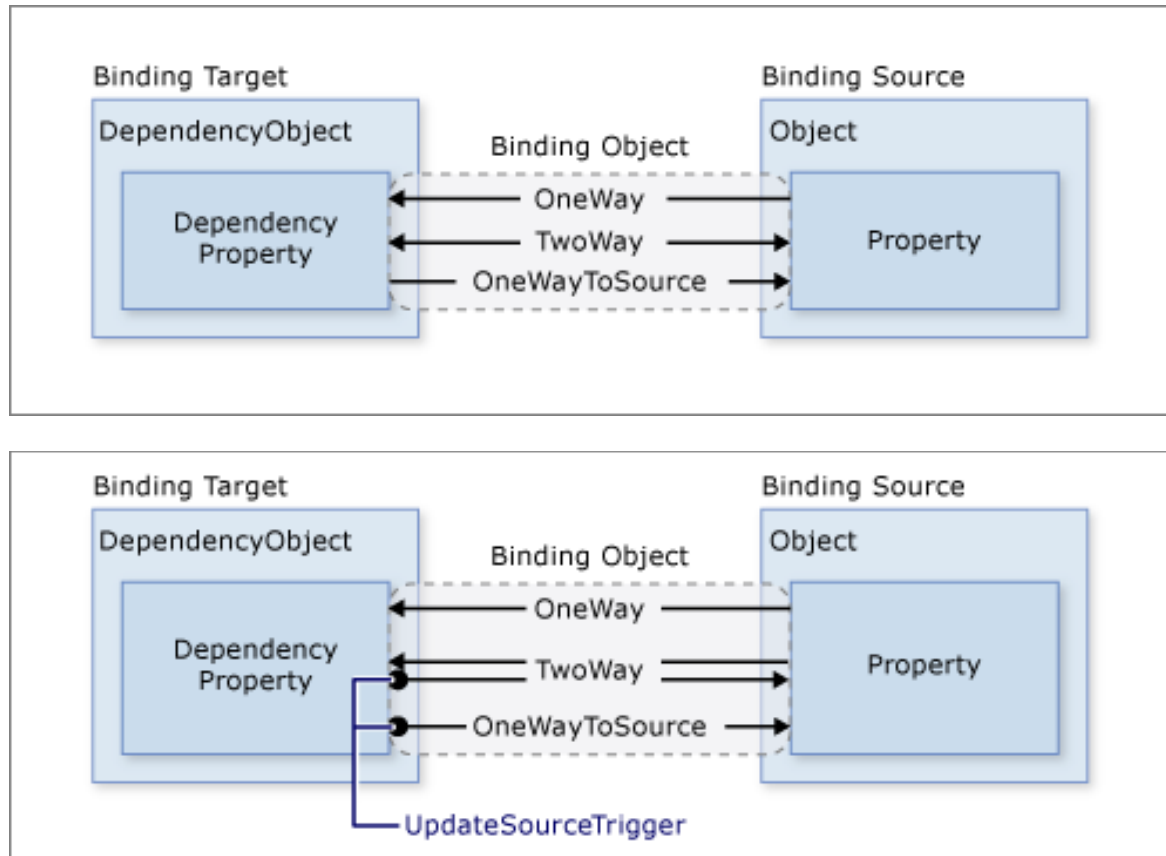
    0 references
    public AppInf0996()
    {
        Cw = new CidadeWindow();
        InitializeComponent();
    }
}
```

```
// Abre Janela para Selecionar a Cidade
1 reference
private void Cidade_Click(object sender, RoutedEventArgs e)
{
    Cw.ShowInTaskbar=false;
    Cw.Show();
}
```

Práticas com WPF – *Janela Filha*



Práticas com WPF – *Data binding*



<https://learn.microsoft.com/en-us/dotnet/desktop/wpf/data/?view=netdesktop-6.0>



- Componentes do Data Binding
 - **Target (Destino)** - É o objeto que vai usar o resultado da vinculação (binding)
 - **Target Property (Propriedade de destino)** - A propriedade de destino do objeto que irá utilizar o resultado
 - **Source (Origem)** - O objeto que fornece um valor para o objeto de destino usar
 - **Path (Caminho)** - Um caminho que localiza o valor dentro do objeto de origem

https://www.macoratti.net/11/06/wpf_dbt1.htm



- Vários tipos de *data binding*, Exemplos:
 - *Binding* com uma fonte relativa
 - *Binding* com classes no *code-behind*
 - *Binding* com classes em XAML
 - *Binding* com coleções de dados (em XAML e Code-Behind)
 - *Binding* usando *templates* ...

Nosso exemplo

https://www.macoratti.net/11/06/wpf_dbt1.htm



Estado.cs

Classe Estado com atributos sigla e nome

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace FuncionamentoBasico
{
    29 references
    public class Estado
    {
        3 references
        private string sigla="";
        3 references
        private string nome="";

        0 references
        public string Sigla { get => sigla; set => sigla = value; }

        0 references
        public string Nome { get => nome; set => nome = value; }

        27 references
        public Estado(string sg, string nm) => (this.sigla, this.nome) = (sg,nm);
    }
}
```



Cidade.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace FuncionamentoBasico
{
    35 references
    public class Cidade
    {
        3 references
        private string codCid="";
        3 references
        private string nome="";
        3 references
        private string estado="";

        3 references
        public string CodCid { get => codCid; set => codCid = value; }

        1 reference
        public string Nome { get => nome; set => nome = value; }

        2 references
        public string Estado { get => estado; set => estado = value; }

        30 references
        public Cidade(string codCid, string nm, string std) =>
            (this.codCid, this.nome, this.estado) = (codCid,nm,std);
    }
}
```

Classe Cidade com atributos codCid (IBGE), nome e estado

Práticas com WPF – *Data binding* – Classes C#



CidadesList.cs

Classe CidadeList com a lista de cidades

Construtor que instancia capitais e cidades com mais de 1 milhão de habitantes

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace FuncionamentoBásico
{
    2 references
    public class CidadesList
    {

        34 references
        public List<Cidade> Cidades { get; set; }

        // Construtor com lista de cidades
        0 references
        public CidadesList(List<Cidade> cd) {
            this.Cidades = cd;
        }
    }
}
```

```
// Construtor que cria lista de cidades
1 reference
public CidadesList() {
    this.Cidades = new List<Cidade>();
    this.Cidades.Add(new Cidade("1100205", "Porto Velho", "RO"));
    this.Cidades.Add(new Cidade("1302603", "Manaus", "AM"));
    this.Cidades.Add(new Cidade("1200401", "Rio Branco", "AC"));
    this.Cidades.Add(new Cidade("5002704", "Campo Grande", "MS"));
    this.Cidades.Add(new Cidade("1600303", "Macapá", "AP"));
    this.Cidades.Add(new Cidade("5300108", "Brasília", "DF"));
    this.Cidades.Add(new Cidade("1400100", "Boa Vista", "RR"));
    this.Cidades.Add(new Cidade("5103403", "Cuiabá", "MT"));
    this.Cidades.Add(new Cidade("1721000", "Palmas", "TO"));
    this.Cidades.Add(new Cidade("3550308", "São Paulo", "SP"));
    this.Cidades.Add(new Cidade("2211001", "Teresina", "PI"));
    this.Cidades.Add(new Cidade("3304557", "Rio de Janeiro", "RJ"));
    this.Cidades.Add(new Cidade("1200401", "Belém", "PA"));
    this.Cidades.Add(new Cidade("5208707", "Goiânia", "GO"));
    this.Cidades.Add(new Cidade("2927408", "Salvador", "BA"));
    this.Cidades.Add(new Cidade("4205407", "Florianópolis", "SC"));
    this.Cidades.Add(new Cidade("2111300", "São Luís", "MA"));
    this.Cidades.Add(new Cidade("2704302", "Maceió", "AL"));
    this.Cidades.Add(new Cidade("4314902", "Porto Alegre", "RS"));
    this.Cidades.Add(new Cidade("4106902", "Curitiba", "PR"));
    this.Cidades.Add(new Cidade("3106200", "Belo Horizonte", "MG"));
    this.Cidades.Add(new Cidade("2304400", "Fortaleza", "CE"));
    this.Cidades.Add(new Cidade("2611606", "Recife", "PE"));
    this.Cidades.Add(new Cidade("2507507", "João Pessoa", "PB"));
    this.Cidades.Add(new Cidade("2800308", "Aracaju", "SE"));
    this.Cidades.Add(new Cidade("2408102", "Natal", "RN"));
    this.Cidades.Add(new Cidade("3205309", "Vitória", "ES"));
    this.Cidades.Add(new Cidade("3509502", "Campinas", "SP"));
    this.Cidades.Add(new Cidade("3518800", "Guarulhos", "SP"));
    this.Cidades.Add(new Cidade("3304904", "São Gonçalo", "RJ"));
}
}
```



Práticas com WPF – *Data binding* – XAML



CidadeWindow.xaml

CidadeWindow.xaml.cs

Modificar para selecionar
estado e cidade

```
<Grid>
    <StackPanel Background="#2f5cb6">
        <Label Foreground="White" Content="Selecione o Estado:" HorizontalAlignment="Center"
            Margin="0,22,0,0" Name="label1" VerticalAlignment="Top" />

        <ComboBox ItemsSource="{Binding Estados}"
            DisplayMemberPath="Nome"
            SelectedValuePath="Sigla"
            Width="200"
            DropDownClosed="Estados_DropDownClosed"
            Name="ComboEstados" />

        <Label Foreground="White" Content="Selecione a Cidade:" HorizontalAlignment="Center"
            Margin="0,22,0,0" Name="label2" VerticalAlignment="Top" />

        <ComboBox ItemsSource="{Binding CidadesSelecionadas}"
            DisplayMemberPath="Nome"
            SelectedValuePath="CodCid"
            Width="200"
            Name="ComboCidades" />

        <StackPanel Orientation="Horizontal" HorizontalAlignment="Center">
            <Button Margin="20,40,20,20" Name="buttonSelecionar" Foreground="White" Width="120"
                FontWeight="Bold" Background="#152951" Click="Seleciona_Click" >Selecionar</Button>

            <Button Margin="20,40,20,20" Name="buttonFechar" Foreground="White" Width="120"
                FontWeight="Bold" Background="#152951" Click="FecharJanela_Click" >Fechar</Button>
        </StackPanel>
    </StackPanel>
</Grid>
```

Práticas com WPF – *Data binding* – XAML



CidadeWindow.xaml

CidadeWindow.xaml.cs

Label “Selecione o Estado:”

```
<Grid>
    <StackPanel Background="#2f5cb6">
        <Label Foreground="White" Content="Selecione o Estado:" HorizontalAlignment="Center"
            Margin="0,22,0,0" Name="label1" VerticalAlignment="Top" />

        <ComboBox ItemsSource="{Binding Estados}"
            DisplayMemberPath="Nome"
            SelectedValuePath="Sigla"
            Width="200"
            DropDownClosed="Estados_DropDownClosed"
            Name="ComboEstados" />

        <Label Foreground="White" Content="Selecione a Cidade:" HorizontalAlignment="Center"
            Margin="0,22,0,0" Name="label2" VerticalAlignment="Top" />

        <ComboBox ItemsSource="{Binding CidadesSelecionadas}"
            DisplayMemberPath="Nome"
            SelectedValuePath="CodCid"
            Width="200"
            Name="ComboCidades" />

        <StackPanel Orientation="Horizontal" HorizontalAlignment="Center">
            <Button Margin="20,40,20,20" Name="buttonSelecionar" Foreground="White" Width="120"
                FontWeight="Bold" Background="#152951" Click="Seleciona_Click" >Selecionar</Button>

            <Button Margin="20,40,20,20" Name="buttonFechar" Foreground="White" Width="120"
                FontWeight="Bold" Background="#152951" Click="FecharJanela_Click" >Fechar</Button>
        </StackPanel>
    </StackPanel>
</Grid>
```

Práticas com WPF – *Data binding* – XAML



ComboBox para seleção de Estado

Binding para propriedade Estados (lista de estados a ser criada na classe CidadeWindow.xaml.cs)

DisplayMemberPath: É o nome (atributo da classe Estado) que é exibida na lista do ComboBox.
SelectedValuePath: É a sigla (atributo da classe Estado) que é o valor selecionado no ComboBox.

Método para tratamento do evento de seleção e fechamento do ComboBox

```
<Grid>
    <StackPanel Orientation="Vertical" HorizontalAlignment="Center">
        <ComboBox ItemsSource="{Binding Estados}"
            DisplayMemberPath="Nome"
            SelectedValuePath="Sigla"
            Width="200"
            DropDownClosed="Estados_DropDownClosed"
            Name="ComboEstados" />
        <Label Foreground="White" Content="Selecionar Estado"
            Margin="0,22,0,0" Name="label2" VerticalAlignment="Top" />
        <ComboBox ItemsSource="{Binding CidadesSelecionadas}"
            DisplayMemberPath="Nome"
            SelectedValuePath="CodCid"
            Width="200"
            Name="ComboCidades" />
    </StackPanel>
    <StackPanel Orientation="Horizontal" HorizontalAlignment="Center">
        <Button Margin="20,40,20,20" Name="buttonSelecionar" Foreground="White" Width="120"
            FontWeight="Bold" Background="#152951" Click="Seleciona_Click" >Selecionar</Button>
        <Button Margin="20,40,20,20" Name="buttonFechar" Foreground="White" Width="120"
            FontWeight="Bold" Background="#152951" Click="FecharJanela_Click" >Fechar</Button>
    </StackPanel>
</Grid>
```


Práticas com WPF – *Data binding* – XAML



CidadeWindow.xaml

CidadeWindow.xaml.cs

```
<Grid>
    <StackPanel Background="#2f5cb6">
        <Label Foreground="White" Content="Selecione o Estado:" HorizontalAlignment="Center"
            Margin="0,22,0,0" Name="label1" VerticalAlignment="Top" />

        <ComboBox ItemsSource="{Binding Estados}"
            DisplayMemberPath="Nome"
            SelectedValue
            Width="200"
            DropDownClos
            Name="ComboEstados" />

        <Label Foreground="White" Content="Selecione a Cidade:" HorizontalAlignment="Center"
            Margin="0,22,0,0" Name="label2" VerticalAlignment="Top" />

        <ComboBox ItemsSource="{Binding CidadesSelecionadas}"
            DisplayMemberPath="Nome"
            SelectedValuePath="CodCid"
            Width="200"
            Name="ComboCidades" />

        <StackPanel Orientation="Horizontal" HorizontalAlignment="Center">
            <Button Margin="20,40,20,20" Name="buttonSelecionar" Foreground="White" Width="120"
                FontWeight="Bold" Background="#152951" Click="Seleciona_Click" >Selecionar</Button>

            <Button Margin="20,40,20,20" Name="buttonFechar" Foreground="White" Width="120"
                FontWeight="Bold" Background="#152951" Click="FecharJanela_Click" >Fechar</Button>
        </StackPanel>
    </StackPanel>
</Grid>
```

Label "Selecione a Cidade:"

Práticas com WPF – *Data binding* – XAML



CidadeWindow.xaml

CidadeWindow.xaml.cs

```
<Grid>
    <StackPanel Background="#2f5cb6">
        <Label Foreground="White" Content="Selecione o Estado:" HorizontalAlignment="Center"
            Margin="0,22,0,0" Name="label1" VerticalAlignment="Top" />

        <ComboBox ItemsSource="{Binding Estados}"
            DisplayMemberPath="Nome"
            SelectedValuePath="Sigla"
            Width="200"
            DropDownClosed="Estados_DropDownClosed"
            Name="ComboEstados" />

        <Label Foreground="White" Content="Selecione a Cidade:" HorizontalAlignment="Center"
            Margin="0,22,0,0" Name="label2" VerticalAlignment="Top" />

        <ComboBox ItemsSource="{Binding CidadesSelecionadas}"
            DisplayMemberPath="Nome"
            SelectedValuePath="CodCid"
            Width="200"
            Name="ComboCidades" />

        <StackPanel Orientation="Horizontal">
            <Button Margin="20,40,20,20"
                FontWeight="Bold" Background="White" />

            <Button Margin="20,40,20,20" Name="FecharJanela" Foreground="White" Width="120"
                FontWeight="Bold" Background="#152951" Click="FecharJanela_Click" >Fechar</Button>
        </StackPanel>
    </StackPanel>
</Grid>
```

ComboBox para seleção de Cidade

Binding para propriedade CidadesSelecionadas (lista de cidades de um estado selecionado na classe CidadeWindow.xaml.cs)

DisplayMemberPath: É o nome (atributo da classe Cidade) que é exibida na lista do ComboBox.
SelectedValuePath: É a CodCid (atributo da classe Cidade) que é o valor selecionado no ComboBox.

Práticas com WPF – *Data binding* – XAML



CidadeWindow.xaml

CidadeWindow.xaml.cs

```
<Grid>
    <StackPanel Background="#2f5cb6">
        <Label Foreground="White" Content="Selecione o Estado:" HorizontalAlignment="Center"
            Margin="0,22,0,0" Name="label1" VerticalAlignment="Top" />

        <ComboBox ItemsSource="{Binding Estados}"
            DisplayMemberPath="Nome"
            SelectedValuePath="Sigla"
            Width="200"
            DropDownClosed="Estados_DropDownClosed"
            Name="ComboEstados" />

        <Label Foreground="White" Content="Selecione a Cidade:" HorizontalAlignment="Center"
            Margin="0,22,0,0" Name="label2" VerticalAlignment="Top" />

        <ComboBox ItemsSource="{Binding CidadesSelecionadas}"
            DisplayMemberPath="Nome"
            Name="ComboCidades" />

        <StackPanel Orientation="Horizontal" HorizontalAlignment="Center">
            <Button Margin="20,40,20,20" Name="buttonSelecionar" Foreground="White" Width="120"
                FontWeight="Bold" Background="#152951" Click="Seleciona_Click" >Selecionar</Button>

            <Button Margin="20,40,20,20" Name="buttonFechar" Foreground="White" Width="120"
                FontWeight="Bold" Background="#152951" Click="FecharJanela_Click" >Fechar</Button>
        </StackPanel>
    </StackPanel>
</Grid>
```

Botões “Selecionar” e “Fechar”

Método `Seleciona_Click` (classe `CidadeWindow.xaml.cs`) que tratará o evento deste botão

Método `FecharJanela_Click` (classe `CidadeWindow.xaml.cs`) que tratará o evento deste botão

Práticas com WPF – *Data binding* – C#



CidadeWindow.xaml

CidadeWindow.xaml.cs

```
namespace FuncionamentoBasico
{
    /// <summary>
    /// Interaction logic for AppInf0996.xaml
    /// </summary>
    2 references
    public partial class CidadeWindow : Window
    {
        3 references
        private AppInf0996 appWin;
        28 references
        public List<Estado> Estados { get; set; }
        6 references
        public List<Cidade> CidadesSelecionadas { get; set; }
        3 references
        private CidadesList ListaCidades;

        1 reference
        public CidadeWindow(AppInf0996 appW)
        {
            appWin = appW;
            InicListaEstados();
            ListaCidades = new CidadesList();
            CidadesSelecionadas = ListaCidades.Cidades;
            // Set the data context for this window.
            DataContext = this;
            InitializeComponent();
        }
    }
}
```

Referência para a janela principal, para atualizar cidade com a previsão de tempo

Lista de estados para *binding* com ComboBox

```
<ComboBox ItemsSource="{Binding Estados}"
    DisplayMemberPath="Nome"
    SelectedValuePath="Sigla"
    Width="200"
    DropDownClosed="Estados_DropDownClosed"
    Name="ComboEstados" />
```

Práticas com WPF – *Data binding* – C#



CidadeWindow.xaml

CidadeWindow.xaml.cs

```
namespace FuncionamentoBasico
{
    /// <summary>
    /// Interaction logic for AppInf0996.xaml
    /// </summary>
    2 references
    public partial class CidadeWindow : Window
    {
        3 references
        private AppInf0996 appWin;
        28 references
        public List<Estado> Estados { get; set; }
        6 references
        public List<Cidade> CidadesSelecionadas { get; set; }
        3 references
        private CidadesList ListaCidades;

        1 reference
        public CidadeWindow(AppInf0996 appW)
        {
            appWin = appW;
            InicListaEstados();
            ListaCidades = new CidadesList();
            CidadesSelecionadas = ListaCidades.Cidades;
            // Set the data context for this window.
            DataContext = this;
            InitializeComponent();
        }
    }
}
```

```
<ComboBox ItemsSource="{Binding CidadesSelecionadas}"
    DisplayMemberPath="Nome"
    SelectedValuePath="CodCid"
    Width="200"
    Name="ComboCidades" />
```

Lista de cidades para *binding* com ComboBox, são as cidades de um estado selecionado no outro ComboBox

Atributo da classe CidadesList (que tem as capitais e cidades com mais de 1 milhão de habitantes)

Práticas com WPF – *Data binding* – C#



CidadeWindow.xaml

CidadeWindow.xaml.cs

```
namespace FuncionamentoBasico
{
    /// <summary>
    /// Interaction logic for AppInf0996.xaml
    /// </summary>
    2 references
    public partial class CidadeWindow : Window
    {
        3 references
        private AppInf0996 appWin;
        28 references
        public List<Estado> Estados { get; set; }
        6 references
        public List<Cidade> CidadesSelecionadas { get; set; }
        3 references
        private CidadesList ListaCidades;

        1 reference
        public CidadeWindow(AppInf0996 appW)
        {
            appWin = appW;
            InicListaEstados();
            ListaCidades = new CidadesList();
            CidadesSelecionadas = ListaCidades.Cidades;
            // Set the data context for this window.
            DataContext = this;
            InitializeComponent();
        }
    }
}
```

Construtor passando a janela principal

Chama método para criar lista de estado

Inicializa lista de cidades com todas no ComboBox

Necessário para dizer que este objeto contem os dados da janela CidadeWindow

Práticas com WPF – *Data binding* – C#



CidadeWindow.xaml

CidadeWindow.xaml.cs

```
private void InicListaEstados() {  
    Estados = new List<Estado>();  
    Estados.Add(new Estado("AC", "Acre"));  
    Estados.Add(new Estado("AL", "Alagoas"));  
    Estados.Add(new Estado("AP", "Amapá"));  
    Estados.Add(new Estado("AM", "Amazonas"));  
    Estados.Add(new Estado("BA", "Bahia"));  
    Estados.Add(new Estado("CE", "Ceará"));  
    Estados.Add(new Estado("DF", "Distrito Federal"));  
    Estados.Add(new Estado("ES", "Espírito Santo"));  
    Estados.Add(new Estado("GO", "Goiás"));  
    Estados.Add(new Estado("MA", "Maranhão"));  
    Estados.Add(new Estado("MT", "Mato Grosso"));  
    Estados.Add(new Estado("MS", "Mato Grosso do Sul"));  
    Estados.Add(new Estado("MG", "Minas Gerais"));  
    Estados.Add(new Estado("PA", "Pará"));  
    Estados.Add(new Estado("PB", "Paraíba"));  
    Estados.Add(new Estado("PR", "Paraná"));  
    Estados.Add(new Estado("PE", "Pernambuco"));  
    Estados.Add(new Estado("PI", "Piauí"));  
    Estados.Add(new Estado("RJ", "Rio de Janeiro"));  
    Estados.Add(new Estado("RN", "Rio Grande do Norte"));  
    Estados.Add(new Estado("RS", "Rio Grande do Sul"));  
    Estados.Add(new Estado("RO", "Rondônia"));  
    Estados.Add(new Estado("RR", "Roraima"));  
    Estados.Add(new Estado("SC", "Santa Catarina"));  
    Estados.Add(new Estado("SP", "São Paulo"));  
    Estados.Add(new Estado("SE", "Sergipe"));  
    Estados.Add(new Estado("TO", "Tocantins"));  
}
```

Inicializa lista de estados

Práticas com WPF – *Data binding* – C#



CidadeWindow.xaml

CidadeWindow.xaml.cs

Evento de fechamento (após seleção) do ComboBox para seleção de Estados

```
// Evento quando o estado é selecionado
1 reference
private void Estados_DropDownClosed(object sender, EventArgs e)
{
    this.CidadesSelecionadas = (List<Cidade>)ListaCidades.Cidades.Where(
        item => item.Estado.Equals(ComboEstados.SelectedValue)).ToList();
    ComboCidades.ItemsSource = this.CidadesSelecionadas;
}

// Seleciona a Cidade do ComboBox
1 reference
private void Seleciona_Click(object sender, RoutedEventArgs e)
{
    // Atualiza a previsão do Tempo (TO DO)
    if (CidadesSelecionadas.Find(item => item.CodCid.Equals(ComboCidades.SelectedValue)) != null)
    {
        this.appWin.atualizaPrevisao(CidadesSelecionadas.Find(item => item.CodCid.Equals(ComboCidades.SelectedValue)).Nome,
            CidadesSelecionadas.Find(item => item.CodCid.Equals(ComboCidades.SelectedValue)).Estado);
        this.Hide();
    }
    else
    {
        MessageBox.Show("Selecione uma Cidade");
    }
}
```

Filtra as cidades que pertencem ao estado selecionado

Atualiza o ComboBox de cidades com as selecionadas

Práticas com WPF – *Data binding* – C#



CidadeWindow.xaml

CidadeWindow.xaml.cs

```
// Evento quando o estado é selecionado
1 reference
private void Estados_DropDownClosed(object sender, EventArgs e)
{
    this.CidadesSelecionadas = (List<Cidade>)ListaCidades.Cidades.Where(
        item => item.Estado.Equals(ComboEstados.SelectedValue)).ToList();
    ComboCidades.ItemsSource = this.CidadesSelecionadas;
}

// Seleciona a Cidade do ComboBox
1 reference
private void Seleciona_Click(object sender, RoutedEventArgs e)
{
    // Atualiza a previsão do Tempo (TO DO)
    if (CidadesSelecionadas.Find(item => item.CodCid.Equals(ComboCidades.SelectedValue)) != null)
    {
        this.appWin.atualizaPrevisao(CidadesSelecionadas.Find(item => item.CodCid.Equals(ComboCidades.SelectedValue)).Nome,
            CidadesSelecionadas.Find(item => item.CodCid.Equals(ComboCidades.SelectedValue)).Estado);
        this.Hide();
    }
    else
    {
        MessageBox.Show("Selecione uma Cidade");
    }
}
```

Evento de Click no botão seleciona

Verifica se tem uma cidade selecionada

Esconde janela de seleção

Mensagem, caso a cidade não esteja selecionada

Atualiza previsão do tempo → nesta versão só muda o *label* com a Cidade e sigla do estado

Práticas com WPF – *Data binding* – C#



CidadeWindow.xaml

CidadeWindow.xaml.cs

Evento de Click no botão fechar

```
// Fecha Janela
1 reference
private void FecharJanela_Click(object sender, RoutedEventArgs e)
{
    this.Hide();
}

// Sobrescreve evento de fechar para só esconder a janela
0 references
protected override void OnClosing(CancelEventArgs e)
{
    this.Hide();
    e.Cancel = true;
}
```

Sobrescreve evento de fechar, para não matar a janela, ela é escondida e pode ser exibida novamente

Práticas com WPF – *Data binding*



- Atualizações no código: [AppInf0996.xaml.cs](#)

```
public partial class AppInf0996 : Window
{
    3 references
    CidadeWindow Cw;

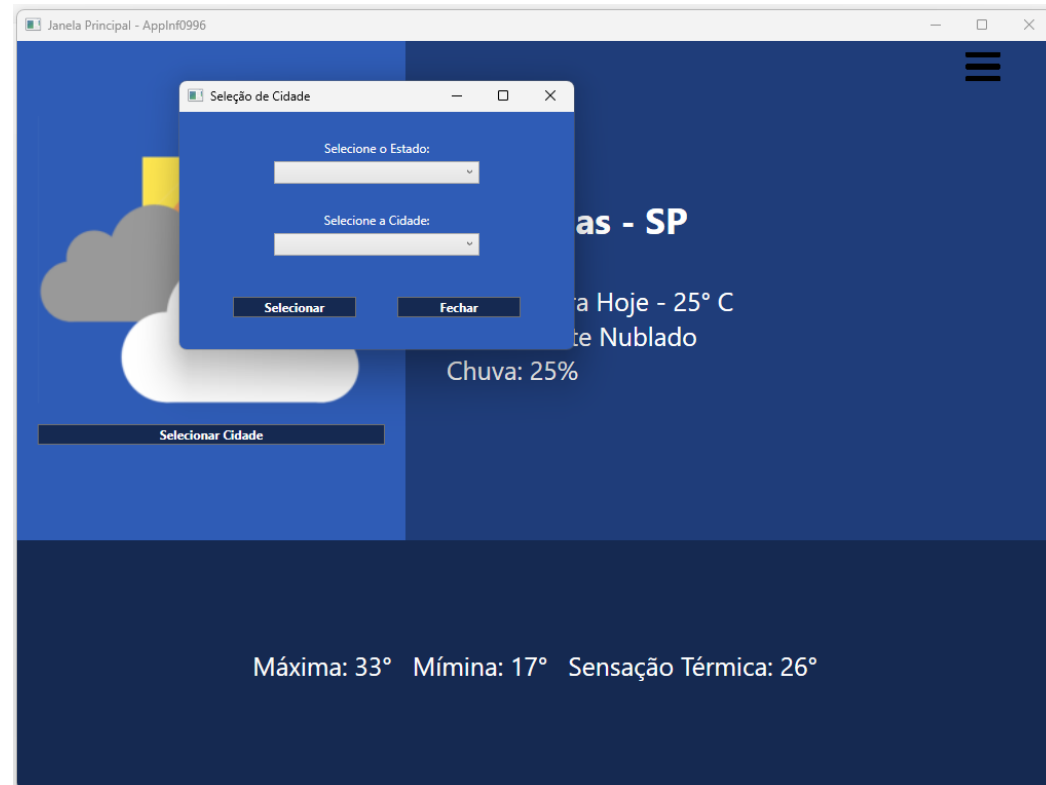
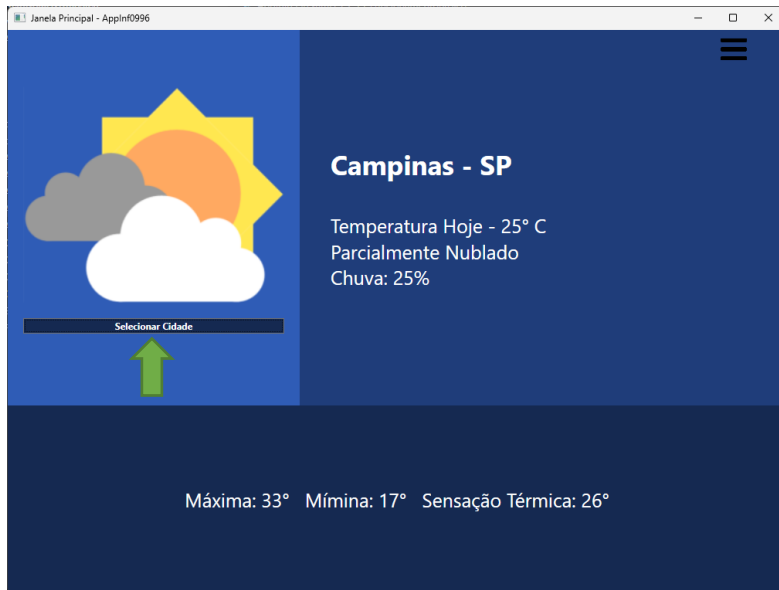
    0 references
    public AppInf0996()
    {
        Cw = new CidadeWindow(this);
        InitializeComponent();
    }
}
```

Método para atualizar previsão do tempo –
Nesta versão apenas muda o nome da cidade e sigla do estado

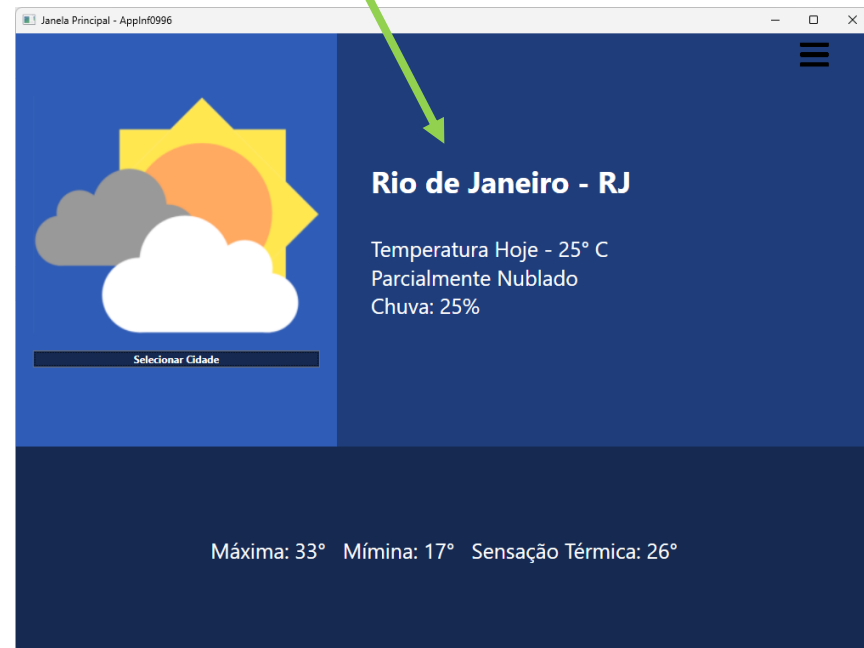
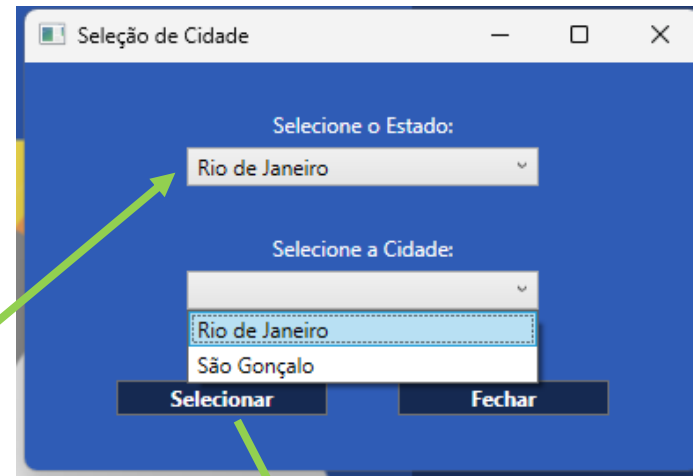
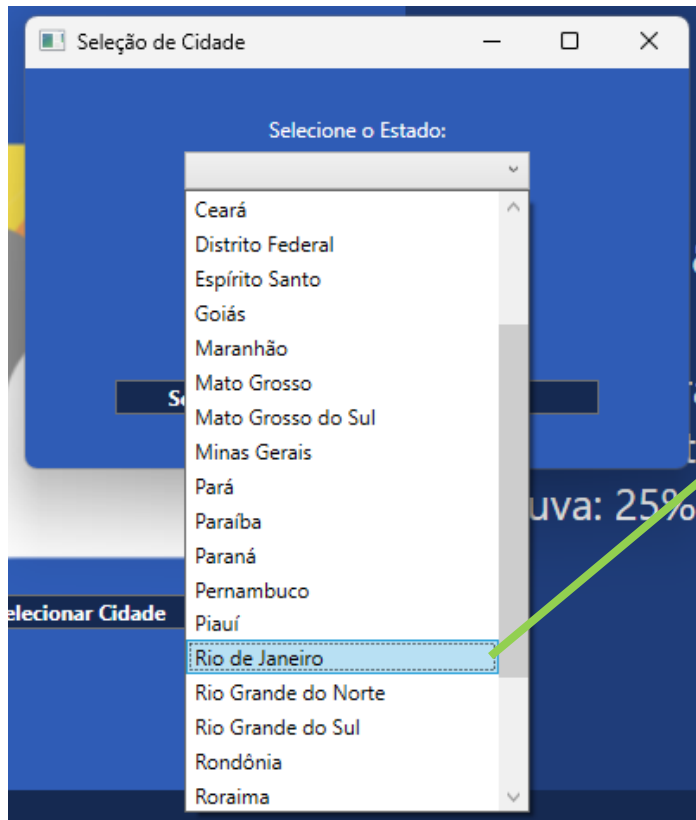
```
// Atualiza Previsão do Tempo para Cidade (TO DO - FAKE)
1 reference
public void atualizaPrevisao(string Cidade, string Estado) {
    CidadeTempo.Text = Cidade + " - " + Estado;
}
```

Passar como parâmetro a referência da janela principal

Práticas com WPF – Funcionamento



Práticas com WPF – Funcionamento



The background features a network of gray lines connecting various colored circles (orange, yellow, green, blue) and a central dark blue horizontal band with a repeating circuit pattern.

Trabalho



INF-0996 – Desenvolvimento de Interface de Usuário

Curso de Extensão Tecnologias Microsoft

Universidade Estadual de Campinas – UNICAMP

Prof. Rodrigo Bonacin

rbonacin@unicamp.br

05/novembro/2022

TRABALHO

1 Objetivos

Este trabalho tem como objetivo desenvolver habilidades de programação utilizando o Windows Presentation Foundation (WPF) com XAML e C#. Este trabalho compõe a avaliação da disciplina INF0996, e deverá ser realizado em grupos de até quatro pessoas.





2 Atividade

Desenvolver tocador de mídias (áudio e vídeo) utilizando o WPF com XAML e C#. O design da solução é aberto, sendo parte integrante do trabalho a construção de uma interface com boa usabilidade e um conjunto de recursos relevantes aos usuários.

Os grupos são livres para criar seus próprios aplicativos, contanto que possuam recursos básicos que permitam os usuários selecionar mídias (áudio e vídeo) e manipular controles (*Tocar, Pausar, Parar, Pular, ...*). Recursos avançados como criar, armazenar e compartilhar *playlists* (por exemplo), efeitos visuais (ex: transparência, animação, etc), perfil do usuário, para citar algumas alternativas, são de livre escolha dos grupos de acordo com a concepção do aplicativo. Esses recursos adicionarão nota ao trabalho.

Boas práticas de design de interfaces (como as dos *Guidelines* do Windows e WPF) e boas práticas arquiteturais (como o MVVM - *Model-View-ViewModel*, MVC, ...) devem ser adotadas, bem como boas práticas de programação .net/C#.



3 Entrega

Os seguintes itens devem ser entregues:

1. Relatório “.pdf” entregue no Moodle (um arquivo por grupo) com:
 - a. Capa com identificação dos alunos que compõe o grupo;
 - b. Descrição do conteúdo e organização dos arquivos no GitHub;
 - c. Link para repositório no GitHub com:
 - i. Código Fonte;
 - ii. Código Executável;
 - iii. Instruções para compilação e execução;
 - iv. Descrição da arquitetura da aplicação (com classes principais);
 - v. Descrição do funcionamento do sistema.
 - d. Link para vídeo com (a duração do vídeo é livre):
 - i. Descrição do funcionamento do sistema.
 - ii. Instruções para compilação e execução.
 - iii. Descrição da arquitetura, implementação e demais itens que o grupo julgar pertinente.



4 Critérios de Avaliação

A nota será atribuída de acordo com critérios de avaliação do trabalho, que incluem:

- (1) *Qualidade da interface*: a interface tem boa usabilidade geral, facilidade de uso, faz uso de recursos avançados e é aderente a guidelines.
- (2) *Funcionalidades e recursos*: quantidade, qualidade e complexidade dos recursos disponíveis para o usuário.
- (3) *Qualidade do código*: Padrões de arquitetura e projeto adotado (ex: MVC, MVVM, etc), boas práticas de programação e ausência de bugs.
- (4) *Qualidade do relatório e apresentação*. Qualidade da documentação entregue, incluindo relatório, descrições do sistema, vídeo/apresentação e organização no GitHub.

A nota do trabalho é dada pela média aritmética simples das notas (0 a 10) destes critérios.



DATA FINAL DE ENTREGA: 18/11/2022

