



Curso de Extensão
Tecnologias Microsoft



INF-0996

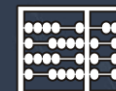
Desenvolvimento de Interface de Usuário

AULA 2

Prof. Dr. Rodrigo Bonacin

RBonacin@unicamp.br

12 de Novembro de 2022



INSTITUTO DE
COMPUTAÇÃO




1. Ferramentas para XAML
2. *Guidelines* de design Microsoft (em *WinUI 3*)
3. Padrão MVVM (*Model–View–ViewModel*)
 - Exemplo
4. MVVM Toolkit
 - Exemplo

The background features a network of gray lines connecting various colored circles (orange, yellow, light blue, green) in a non-linear fashion. A dark blue horizontal band with a light blue border is positioned across the middle of the slide.

Ferramentas para XAML

Ferramentas para XAML - Uno Platform



Uno Platform v0.4.0 Preview

Uno Platform | 5.513 | ★★★★★ (3)

Uno Platform Projects XAML Completion and Hot Reload support for VS Code

[Install](#) ⚙️

[Details](#) [Feature Contributions](#) [Dependencies](#)

Uno Platform

Welcome to the Uno Platform extension for Code!

This extension is used with Uno Platform C#/XAML projects for WebAssembly and Skia based targets.

It provides:

- XAML Code Completion

Ferramentas para XAML - Uno Platform



Target platform coverage by IDE on Windows

	Windows 10/11 (UWP/WinUI)	Android	iOS	Web (WebAssembly)	macOS (Xamarin)	macOS (Skia-Gtk)	Linux (Skia- Gtk)	Windows 7+ (Skia-WPF)
Visual Studio	✓	✓	✓†	✓	✗	✓	✓	✓
VS Code	✗	✗	✗	✓	✗	✓	✓	✓
Codespaces / Gitpod	✗	✗	✗	✓	✗	✓	✓	✓
JetBrains Rider	✓	✓	✓†	✓	✗	✓	✓	✓



Get Started on VS Code

This guide will walk you through the set-up process for building WebAssembly and Gtk+ apps with Uno under Windows, Linux, or macOS.

See these sections for information about using Uno Platform with:

- [Codespaces](#)
- [Gitpod](#)

Prerequisites

- [Visual Studio Code](#)
- [.NET SDK](#)
 - [.NET 6.0 SDK \(version 6.0 \(SDK 6.0.100\) or later\)](#)

Use `dotnet --version` from the terminal to get the version installed.

- 
- The [Uno Platform Visual Studio Code Extension](#)
 - For Windows, install the [GTK+ 3 runtime](#) (See [this uno-check issue](#))

<https://platform.uno/docs/articles/get-started-vscode.html>



Finalize your environment

Windows

macOS

Linux

1. Open a command-line prompt, Windows Terminal if you have it installed, or else Command Prompt or Windows Powershell from the Start menu.
2. a. Install the tool by running the following command from the command prompt:

➔ `dotnet tool install -g uno.check`

- b. To update the tool, if you already have an existing one:

➔ `dotnet tool update -g uno.check`

3. Run the tool from the command prompt with the following command:

➔ `uno-check`

4. Follow the instructions indicated by the tool

<https://platform.uno/docs/articles/get-started-vscode.html>

Ferramentas para XAML - Uno Platform



- Modelos de Projetos

```
PS D:\Bonacin\ICunicamp\Curso_Samsung_MS\XAML\WP1> dotnet new unoapp --search
Procurando modelos...
Correspondências da origem do modelo: NuGet.org
Esses modelos correspondem à sua entrada: 'unoapp'
```

Nome do modelo	Nome Curto	Autor	Idioma	Pacote	Downloads
Cross-Platform App (Prism)	unoapp-prism	Uno Platform		Uno.ProjectTemplates.Dotnet	448k
Cross-Platform UI Tests Library	unoapp-uitest	Uno Platform		Uno.ProjectTemplates.Dotnet	448k
Multi-Platform App (.NET 6, UWP)	unoapp-uwp-net6	Uno Platform		Uno.ProjectTemplates.Dotnet	448k
Multi-Platform App (.NET 6, Windows App SDK)	unoapp	Uno Platform		Uno.ProjectTemplates.Dotnet	448k
Multi-Platform App (Xamarin, UWP)	unoapp-uwp	Uno Platform		Uno.ProjectTemplates.Dotnet	448k
Multi-Platform App (Xamarin, Windows App SDK)	unoapp-winui-xamarin	Uno Platform		Uno.ProjectTemplates.Dotnet	448k
Multi-Platform App (Uno.Extensions)	unoapp-extensions	Uno Platform		Uno.Extensions.Templates	14k

```
PS D:\Bonacin\ICunicamp\Curso_Samsung_MS\XAML\WP1> dotnet new --install Uno.ProjectTemplates.Dotnet
Os seguintes pacotes de modelo serão instalados:
  Uno.ProjectTemplates.Dotnet
```

Êxito: Uno.ProjectTemplates.Dotnet::4.5.12 instalou os seguintes modelos:

Nome do modelo	Nome Curto	Idioma	Tags
Cross-Platform App (Prism)	unoapp-prism	[C#]	Cross-platform/Uno Platform/Prism/Android/iOS/windows/macOS/WebAssembly/dotnet-new
Cross-Platform Library	unolib	[C#]	Cross-platform/Uno Platform/Library/Android/iOS/windows/macOS/Linux/WebAssembly/dotnet-new
Cross-Platform UI Tests Library	unoapp-uitest	[C#]	Cross-platform/Uno Platform/UITest/Android/iOS/WebAssembly/dotnet-new
Cross-Runtime Library	unolib-crossruntime	[C#]	Cross-platform/Uno Platform/Library/Android/iOS/windows/macOS/Linux/WebAssembly/dotnet-new
Multi-Platform App	unoapp-uwp	[C#]	Multi-platform/Uno Platform/Android/iOS/windows/macOS/Linux/Tizen/WebAssembly/dotnet-new
Multi-Platform App (net6)	unoapp-uwp-net6	[C#]	Multi-platform/Uno Platform/Android/iOS/windows/macOS/Linux/WebAssembly/dotnet-new
Multi-Platform App (WinUI)	unoapp-winui-xamarin	[C#]	Multi-platform/Uno Platform/Android/iOS/windows/macOS/Linux/WebAssembly/winUI/dotnet-new
Multi-Platform App net6 (WinUI)	unoapp	[C#]	Multi-platform/Uno Platform/Android/iOS/windows/macOS/Linux/WebAssembly/winUI/dotnet-new
Uno Platform WebAssembly Head for Xam...	wasmxthead	[C#]	Cross-platform/Uno Platform/Xamarin.Forms/WebAssembly/dotnet-new



- Criando um APP

Create the project

In the terminal, type the following to create a new project:

```
dotnet new unoapp -o MyApp -mobile=false --skia-wpf=false --skia-linux-fb=false --vscode
```

`MyApp` is the name you want to give to your project.

This will create a solution that only contains the WebAssembly and Skia+GTK platforms support.



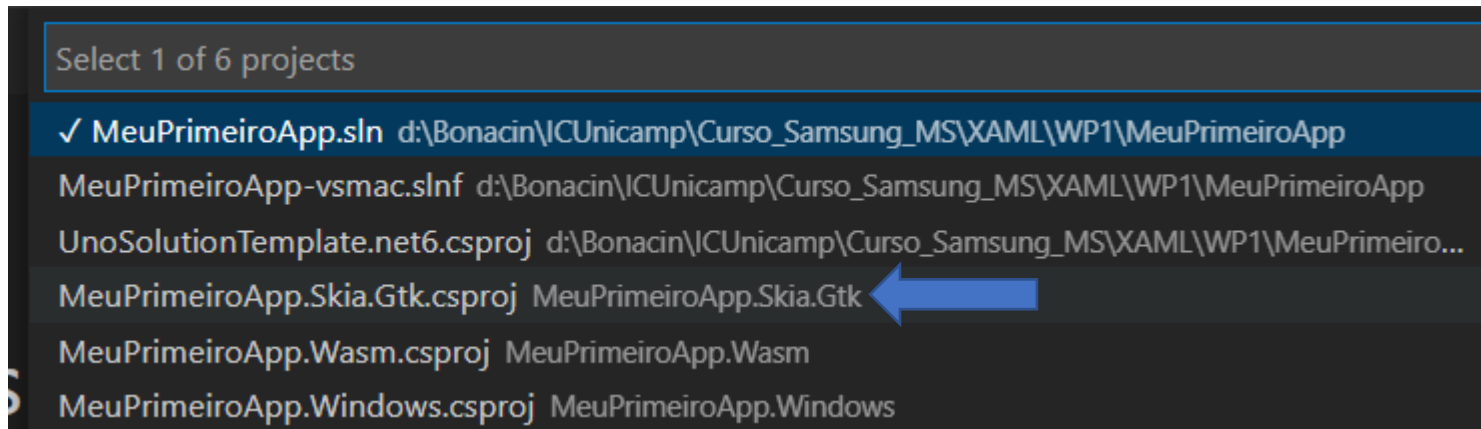
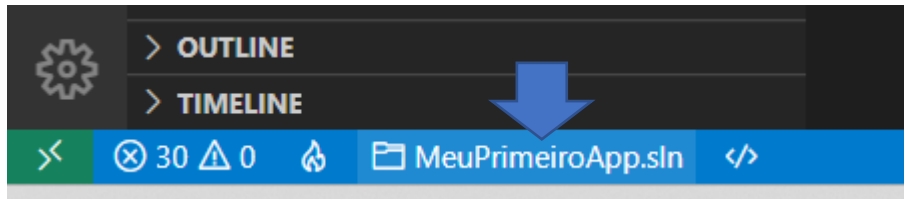
```
dotnet new unoapp -o MeuPrimeiroApp -mobile=false --skia-wpf=false --skia-linux-fb=false --vscode
```

Ferramentas para XAML - Uno Platform



- Abrindo aplicação *WebAssembly* ou *Skia.Gtk*

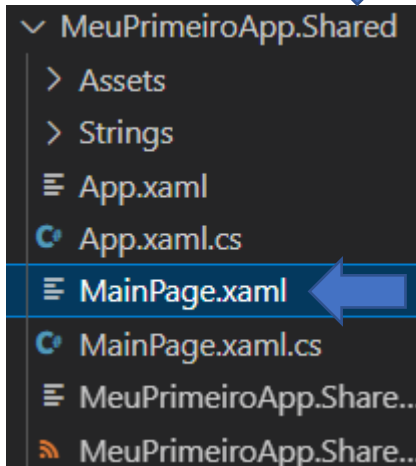
```
D:\Bonacin\ICUnicamp\Curso_Samsung_MS\XAML\WP1> code .\MeuPrimeiroApp
```



Ferramentas para XAML - Uno Platform



- MainPage



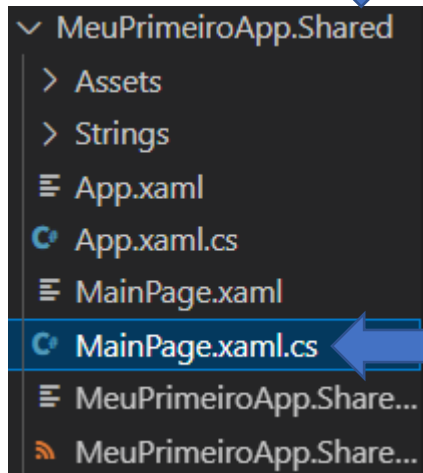
```
<Page
  x:Class="MeuPrimeiroApp.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:local="using:MeuPrimeiroApp"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  mc:Ignorable="d"
  Background="{ThemeResource ApplicationPageBackgroundThemeBrush}"

<StackPanel>
  <TextBlock x:Name="texto"
    Text="Olá Alunos -> INF-0996"
    VerticalAlignment = "Top"
    Margin="20"
    FontSize="30" />
  <Button Content="Olá"
    Click="{x:Bind OnClickOla}"
    Margin="20"
    FontSize="20"/>
  <Button Content="Tchau"
    Click="{x:Bind OnClickTchau}"
    Margin="20"
    FontSize="20"/>
</StackPanel>
</Page>
```

Ferramentas para XAML - Uno Platform



- MainPage



```
namespace MeuPrimeiroApp
{
    5 references
    public sealed partial class MainPage : Page
    {
        0 references
        public MainPage()
        {
            this.InitializeComponent();
        }
        1 reference
        private void OnClickOla()
        {
            texto.Text = "Olá";
        }
        1 reference
        private void OnClickTchau()
        {
            texto.Text = "Tchau";
        }
    }
}
```

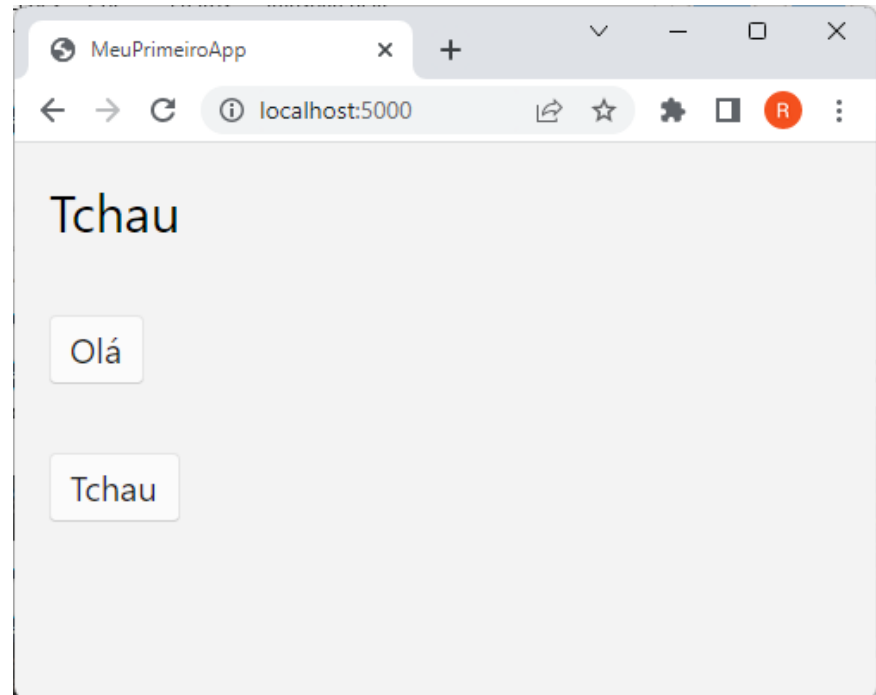
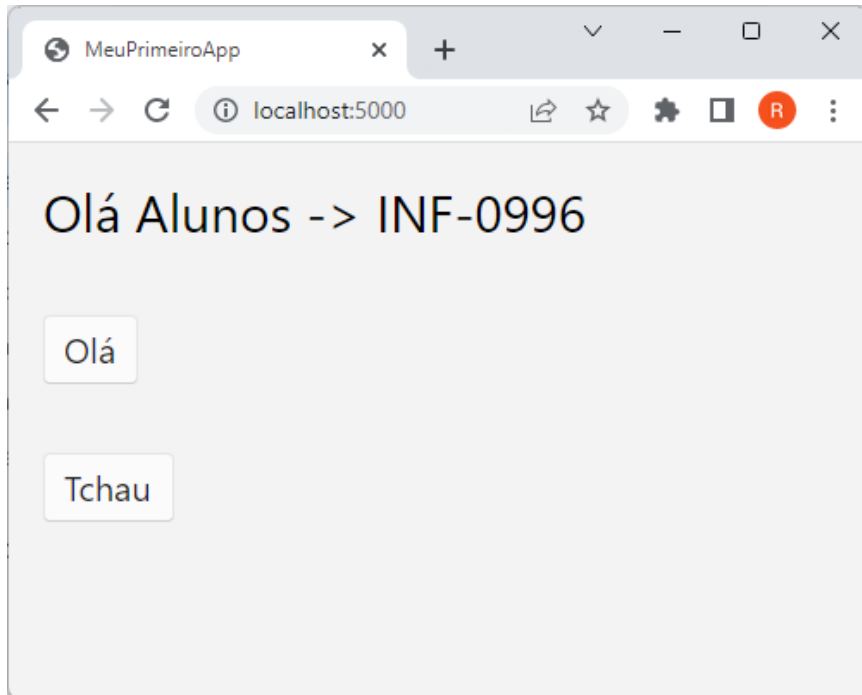
Ferramentas para XAML - Uno Platform



- Executando Web:



```
PS D:\Bonacin\ICUnicamp\Curso_Samsung_MS\XAML\Tools\MeuPrimeiroApp> cd .\MeuPrimeiroApp.Wasm\  
PS D:\Bonacin\ICUnicamp\Curso_Samsung_MS\XAML\Tools\MeuPrimeiroApp\MeuPrimeiroApp.Wasm> dotnet run  
D:\Bonacin\ICUnicamp\Curso_Samsung_MS\XAML\Tools\MeuPrimeiroApp\MeuPrimeiroApp.Wasm\MeuPrimeiroApp.Wasm.csproj : warning NU1504: Duplicate 'PackageReference' i  
tems found. Remove the duplicate items or use the Update functionality to ensure a consistent restore behavior. The duplicate 'PackageReference' items are: Mic  
rosoft.Windows.Compatibility 5.0.0, Microsoft.Windows.Compatibility 5.0.0.  
Hosting environment: Development  
Content root path: D:\Bonacin\ICUnicamp\Curso_Samsung_MS\XAML\Tools\MeuPrimeiroApp\MeuPrimeiroApp.Wasm  
Now listening on: http://localhost:5000  
Now listening on: https://localhost:5001  
Application started. Press Ctrl+C to shut down.
```



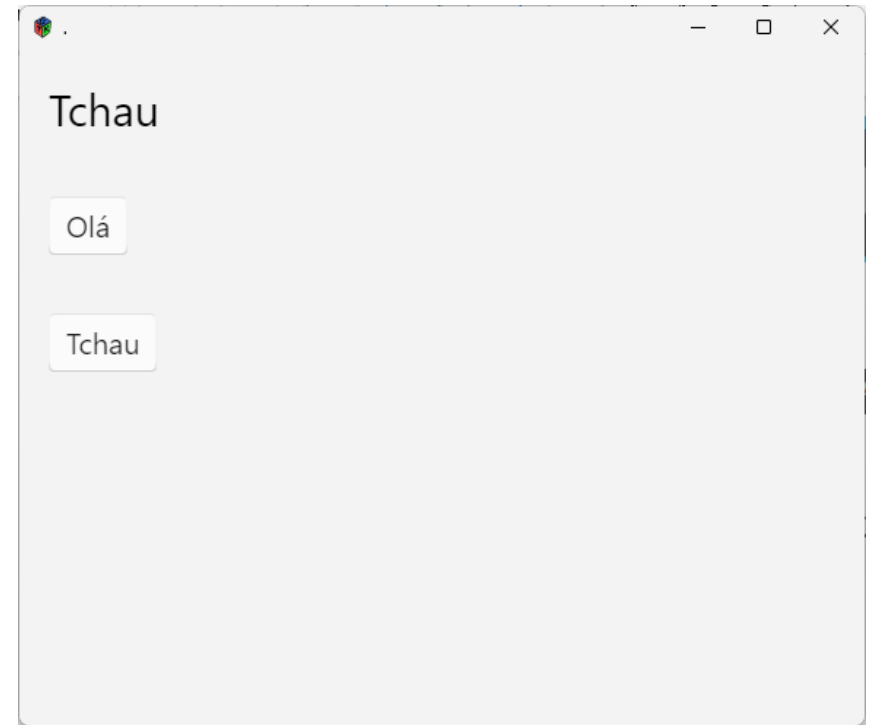
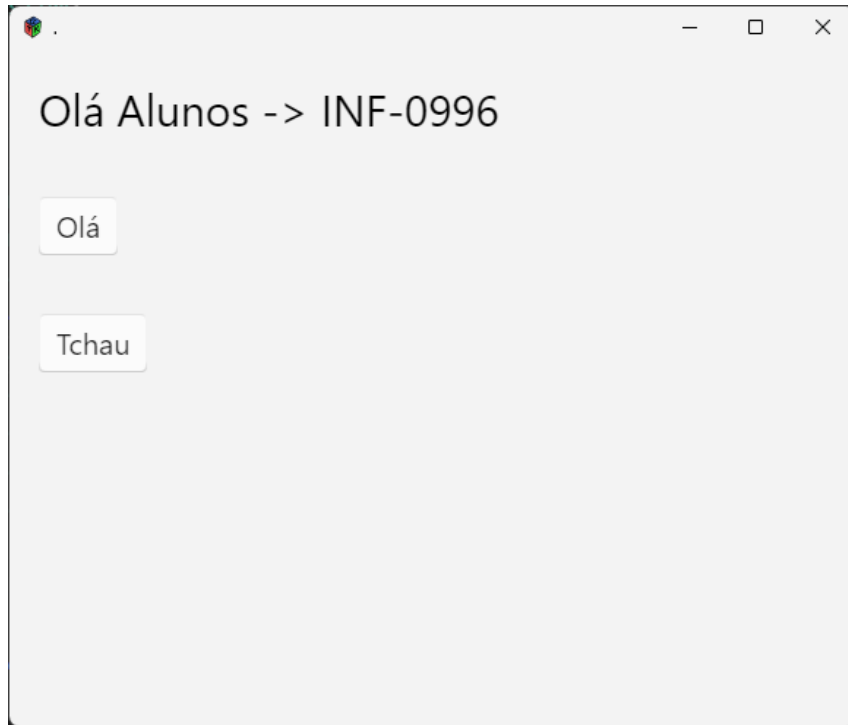
Ferramentas para XAML - Uno Platform



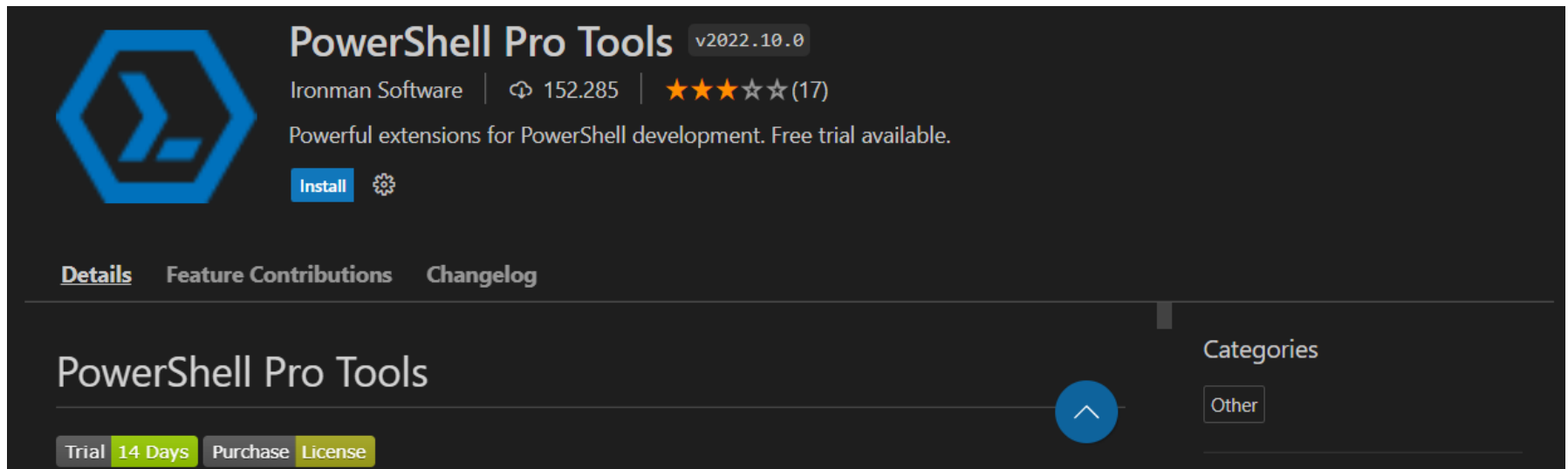
- Executando Web:



```
\MeuPrimeiroApp\MeuPrimeiroApp.Skia.Gtk> dotnet run
```



Ferramentas XAML + PowerShell



The image shows the Visual Studio Marketplace page for the PowerShell Pro Tools extension. The header includes the extension's icon (a blue hexagon with a white 'P'), the name 'PowerShell Pro Tools', the version 'v2022.10.0', the publisher 'Ironman Software', a download count of 152,285, and a rating of 4.5 stars from 17 reviews. Below this is a description: 'Powerful extensions for PowerShell development. Free trial available.' and an 'Install' button with a settings gear icon. The page has tabs for 'Details', 'Feature Contributions', and 'Changelog'. The main content area shows the extension name 'PowerShell Pro Tools' and a 'Trial 14 Days' badge. To the right, there is a 'Categories' section with an 'Other' button. A blue circular button with an upward arrow is also visible.

PowerShell Pro Tools v2022.10.0
Ironman Software | 152.285 | ★★★★★ (17)
Powerful extensions for PowerShell development. Free trial available.

[Install](#) ⚙️

[Details](#) [Feature Contributions](#) [Changelog](#)

PowerShell Pro Tools

Trial 14 Days Purchase License

Categories
Other

```
Get-ExecutionPolicy -Scope CurrentUser  
Set-ExecutionPolicy -ExecutionPolicy Bypass -Scope CurrentUser
```

Ferramentas XAML + PowerShell



> window.ps1
≡ window.xaml

```
<Window
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Width="525"
  Height="350"
  Title="Hello, World">
  <Grid>
    <StackPanel>
      <TextBlock x:Name="texto"
        Text="Ola Alunos -> INF-0996"
        VerticalAlignment = "Top"
        Margin="20"
        FontSize="30" />
      <Button x:Name="ButtonOla"
        Content="Ola"
        Margin="20"
        FontSize="20" />
      <Button x:Name="ButtonTchau"
        Content="Tchau"
        Margin="20"
        FontSize="20" />
    </StackPanel>
  </Grid>
</Window>
```


Ferramentas XAML + PowerShell



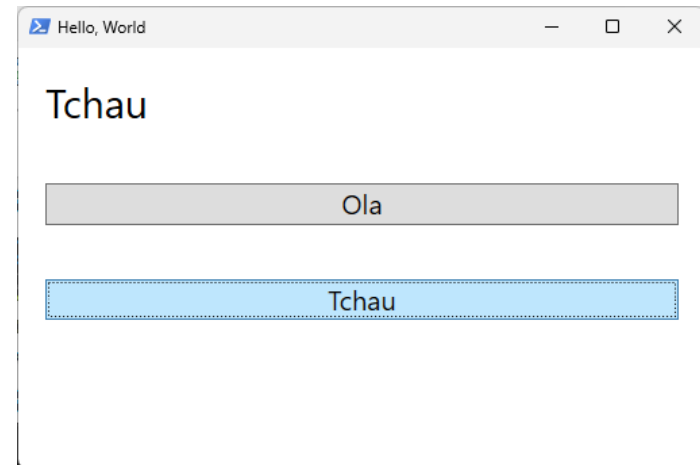
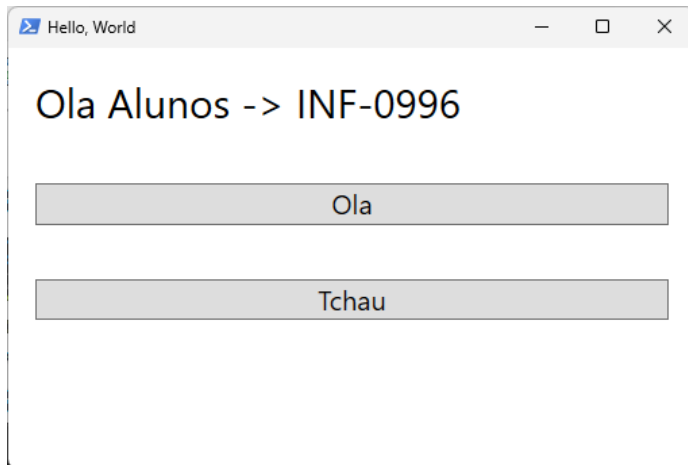
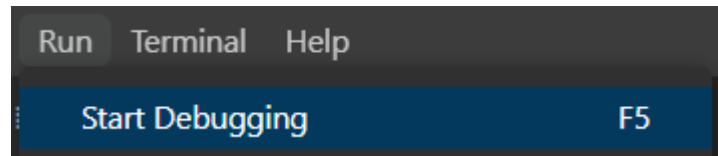
> window.ps1
≡ window.xaml

```
[System.Reflection.Assembly]::LoadWithPartialName("PresentationFramework") | Out-Null

1 reference
function Import-Xaml {
    [xml]$xaml = Get-Content -Path $PSScriptRoot\window.xaml
    $manager = New-Object System.Xml.XmlNamespaceManager -ArgumentList $xaml.NameTable
    $manager.AddNamespace("x", "http://schemas.microsoft.com/winfx/2006/xaml");
    $xamlReader = New-Object System.Xml.XmlNodeReader $xaml
    [Windows.Markup.XamlReader]::Load($xamlReader)
}
$Window = Import-Xaml

$Texto = $Window.FindName("texto")
$ButtonOla = $Window.FindName('ButtonOla')
$ButtonTchau = $Window.FindName('ButtonTchau')
$ButtonOla.add_Click({
    $Texto.Text = "Ola"
})
$ButtonTchau.add_Click({
    $Texto.Text = "Tchau"
})
$Window.ShowDialog()
```

Ferramentas XAML + PowerShell



The background features a network of gray lines connecting various colored circles (orange, yellow, light blue, green) in a non-linear fashion. A dark blue horizontal band with a light blue border is positioned across the middle of the slide.

Guidelines de Design Microsoft

Guidelines Windows



Criar aplicativos da área de trabalho para Windows

Esta documentação fornece as diretrizes mais recentes sobre como criar aplicativos da área de trabalho para Windows 11 e Windows 10.

Prepare-se para o desenvolvimento

COMEÇAR AGORA

[Instalar ferramentas para o SDK do Aplicativo Windows](#)

[Habilitar seu dispositivo para desenvolvimento](#)

[Recursos e depuração do modo de desenvolvedor](#)

[Crie uma conta de desenvolvedor](#)

Começar a criar aplicativos

VISÃO GERAL

[Comece a usar aplicativos da área de trabalho](#)

[Criar aplicativos usando o SDK do Aplicativo Windows](#)

[Criar seu primeiro projeto da WinUI 3](#)

[Aprimore seus aplicativos no Windows 11](#)

REFERÊNCIA

[Modelos do Visual Studio para aplicativos do Windows](#)

Design e interface do usuário

VISÃO GERAL

[Criar e codificar a interface do usuário do seu aplicativo](#)

[Biblioteca de Interface do Usuário do Windows \(WinUI\)](#)

[Princípios de design do Windows 11](#)

REFERÊNCIA

[Referência de API do WinUI](#)

Desenvolver

VISÃO GERAL

[Recursos e tecnologias para aplicativos do Windows](#)

[SDK do Aplicativo do Windows](#)

[Modernize seus aplicativos da área de trabalho](#)

REFERÊNCIA

[API do WinRT para o SDK do Windows](#)

[API do Win32 para o SDK do Windows](#)

[API do WinRT para o SDK do de Aplicativo do Windows](#)

[API do Win32 para o SDK do de Aplicativo do Windows](#)

Implantar

VISÃO GERAL

[Visão geral da implantação](#)

[Implantar aplicativos que usam o SDK do Aplicativo Windows](#)

[Documentação do MSIX](#)

Comunidade e suporte

VISÃO GERAL

[SDK de Aplicativo do Windows no GitHub ¹²](#)

[WinUI \(Biblioteca de Interface do Usuário do Windows\) no GitHub ¹²](#)

[@WindowsDocs no Twitter ¹²](#)

[OneDevMinute no YouTube](#)

[Suporte Developer do Windows](#)

<https://learn.microsoft.com/pt-br/windows/apps/desktop/>



- Exemplos estão em WinUI 3 ... (mas é parecido com WPF)

Win32 **WPF** Windows Forms UWP

O WPF é a plataforma estabelecida para aplicativos gerenciados do Windows com acesso ao .NET ou ao .NET Framework. Ele também usa a marcação XAML para separar a interface do usuário do código. Essa plataforma é adequada para aplicativos da área de trabalho que exigem uma interface do usuário sofisticada, personalização de estilos e cenários com uso intensivo de gráficos. As habilidades de desenvolvimento do WPF são semelhantes às habilidades de desenvolvimento da WinUI 3, portanto, a migração do WPF para WinUI 3 é mais fácil do que a migração de Windows Forms.

Introdução ao WPF

Você também tem acesso aos recursos modernos da plataforma do Windows e às APIs fornecidas pelo SDK do Aplicativo Windows. Para obter mais informações, consulte [Modernize seus aplicativos da área de trabalho](#).

Guidelines Windows - Design



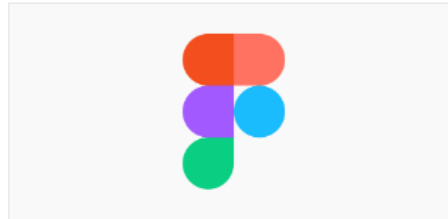
Design e código de aplicativos do Windows

Artigo • 27/09/2022 • 2 minutos para o fim da leitura • 4 colaboradores

[Comentários](#)

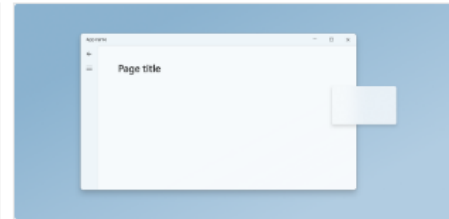
Diretrizes do design e exemplos de códigos da interface do usuário para criação de experiências de aplicativo do Windows.

Downloads de design



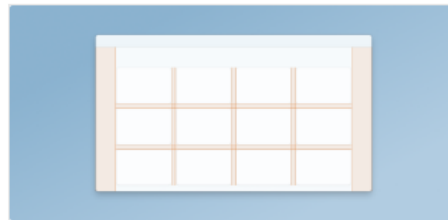
Obtenha kits de ferramentas de design e exemplos.

Noções básicas de design



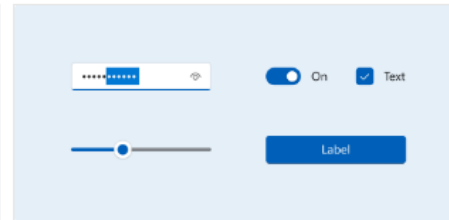
Uma introdução ao design de aplicativo e ao sistema Fluent Design.

Layout



Dicas e APIs responsivas para uma interface do usuário com aparência incrível em todos os tamanhos de tela.

Controles e padrões

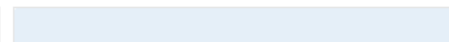


Todos os blocos de construção de interface do usuário necessários, desde botões de opção até controles de navegação.

Estilo



Movimento



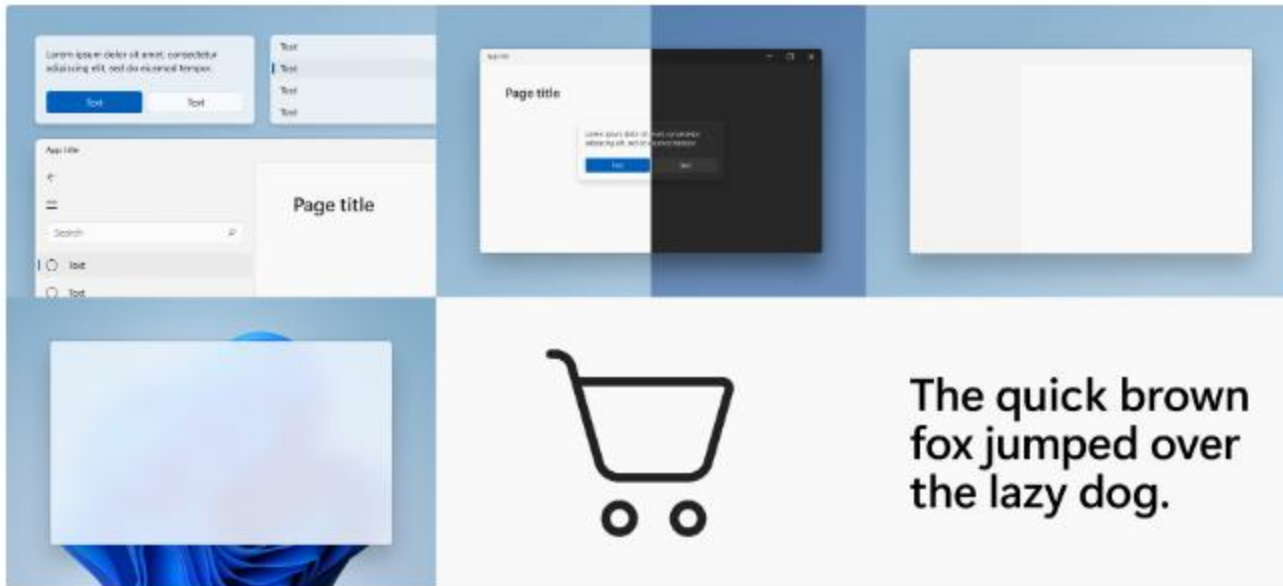
<https://learn.microsoft.com/pt-br/windows/apps/design/>



Princípios de design do Windows 11

Artigo • 27/09/2022 • 2 minutos para o fim da leitura • 2 colaboradores

 Comentários



<https://learn.microsoft.com/pt-br/windows/apps/design/>

Guidelines Windows - Design



Sem esforço

O Windows 11 é mais rápido e intuitivo. É fácil fazer o que eu quero, com foco e precisão.

Simples

O Windows 11 é mais suave e organizado. Ele esmaece em segundo plano para ajudar a manter a atenção e o foco. A experiência parece acolhedora, etérea e acessível.

Pessoal

O Windows 11 se adapta perfeitamente à maneira como uso meu dispositivo. Ele se adapta às minhas necessidades e preferências individuais para que eu possa realmente me expressar.

Familiar

O Windows 11 equilibra uma aparência nova e atualizada com a familiaridade do Windows que eu já conheço. Não há nenhuma curva de aprendizado; basta começar a usar.

Completo + coerente

O Windows 11 oferece uma experiência visualmente perfeita entre plataformas. Posso trabalhar em muitas plataformas e ainda ter uma experiência Windows consistente.

<https://learn.microsoft.com/pt-br/windows/apps/design/>



Desenvolver aplicativos da área de trabalho do Windows

Artigo • 24/10/2022 • 2 minutos para o fim da leitura • 5 colaboradores

 Comentários

Esta seção da documentação fornece informações sobre APIs e recursos que você pode usar ao desenvolver aplicativos da área de trabalho do Windows. Alguns desses recursos estão disponíveis por meio de APIs no SDK de Aplicativo do Windows. Outros recursos estão disponíveis pelo uso de APIs no sistema operacional Windows (por meio do SDK do Windows) e pelo .NET e não exigem o uso do SDK de Aplicativo do Windows.

<https://learn.microsoft.com/pt-br/windows/apps/develop/>



Biblioteca de Interface do Usuário do Windows (WinUI)

Artigo • 31/10/2022 • 4 minutos para o fim da leitura • 10 colaboradores

 Comentários



A Biblioteca WinUI (Interface do Usuário do Windows) é uma estrutura de UX (experiência do usuário) nativa para aplicativos UWP e de área de trabalho do Windows.

Incorporando o [Sistema Fluent Design](#) em todas as experiências e em todos os controles e estilos, o WinUI fornece experiências uniformes, intuitivas e acessíveis usando os padrões de interface do usuário mais recentes.

Com suporte para aplicativos UWP e de área de trabalho, é possível fazer criações completas com o WinUI ou migrar gradualmente os aplicativos MFC, WinForms ou WPF existentes usando linguagens conhecidas como C++, C#, Visual Basic e Javascript (por meio do [React Native para Windows](#)).

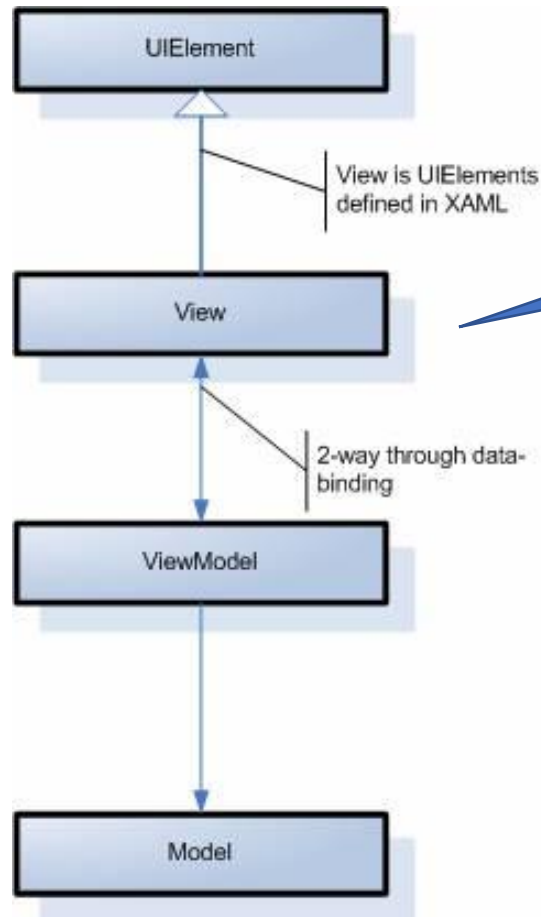
<https://learn.microsoft.com/pt-br/windows/apps/develop/>

The background features a network of abstract elements: several circles in orange, yellow, light blue, and light green, connected by thin grey lines. Some lines are thicker and colored to match the circles they connect. A dark blue horizontal band with a thin light blue border runs across the middle of the slide, containing the title text.

Model-View-ViewModel (MVVM)

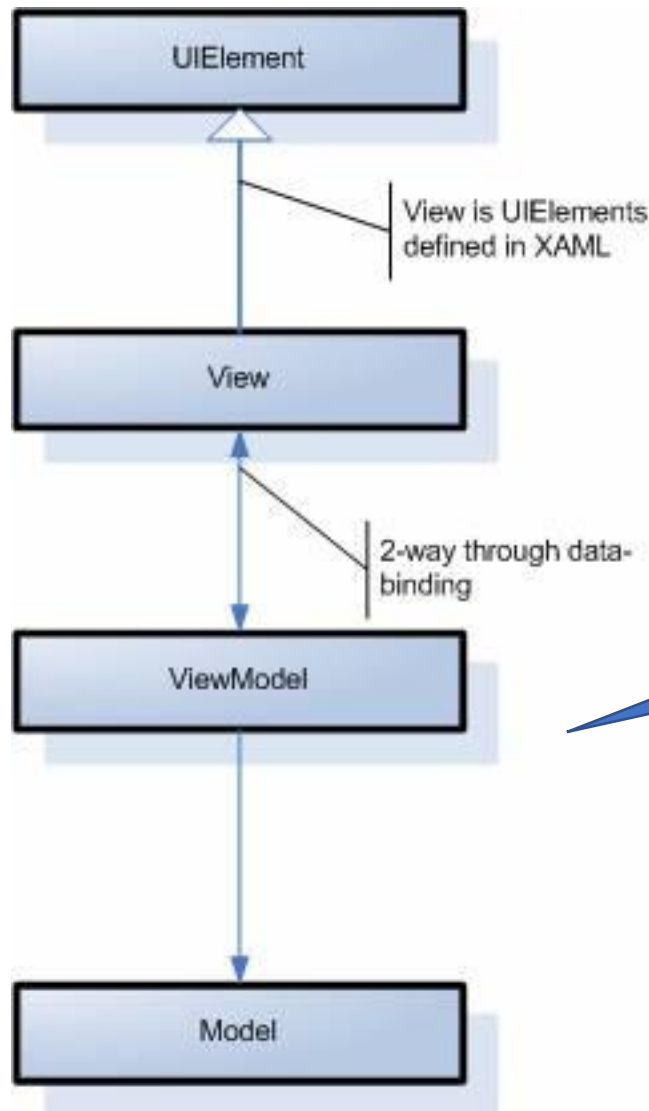


- Visão geral do padrão MVVM



É uma “instanciação” do MVP (*Model View Presenter*) para o WPF

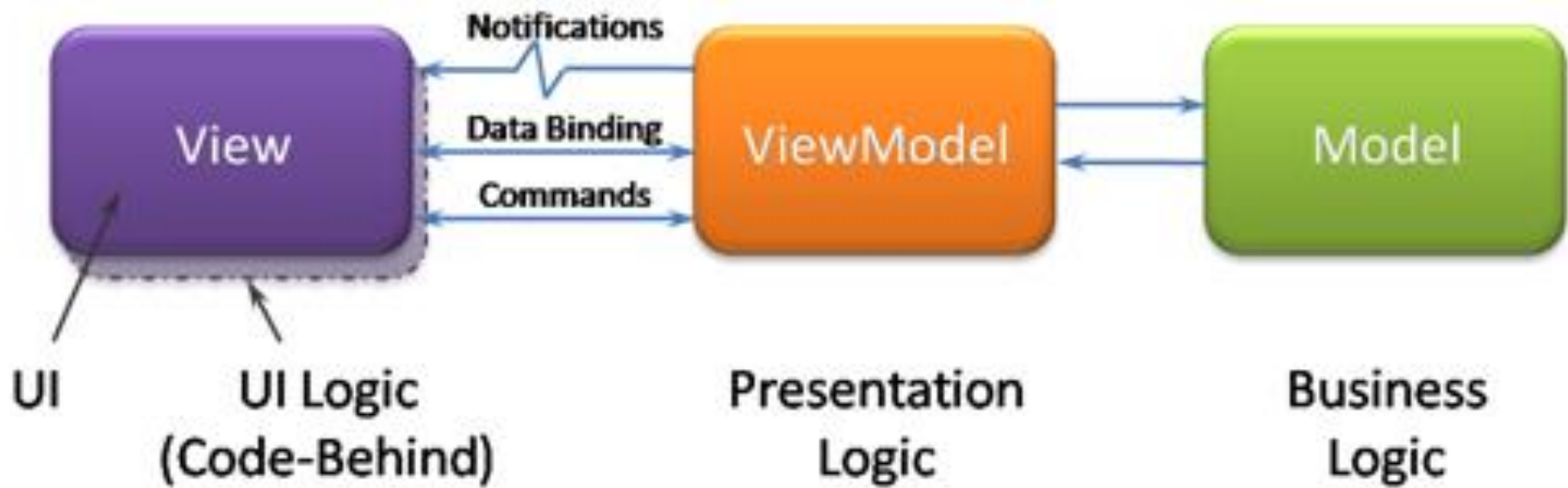
Conceitos MVVM



Camada **Model** (Modelo) não conhece a **View** (Camada de apresentação) e vice-versa. Na verdade a **View** conhece a **ViewModel** e se comunica com ela através do mecanismo de **binding**.



- Visão geral do padrão MVVM



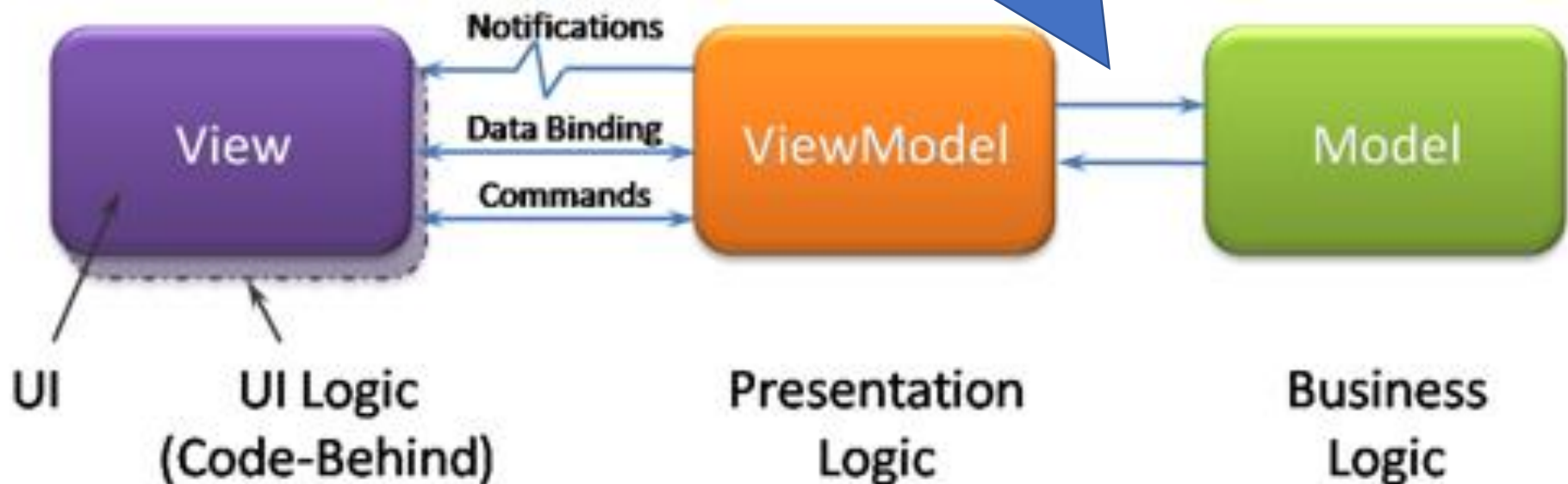
<https://www.devmedia.com.br/entendendo-o-pattern-model-view-viewmodel-mvvm/18411>

Conceitos MVVM



- Visão geral do padrão MVVM

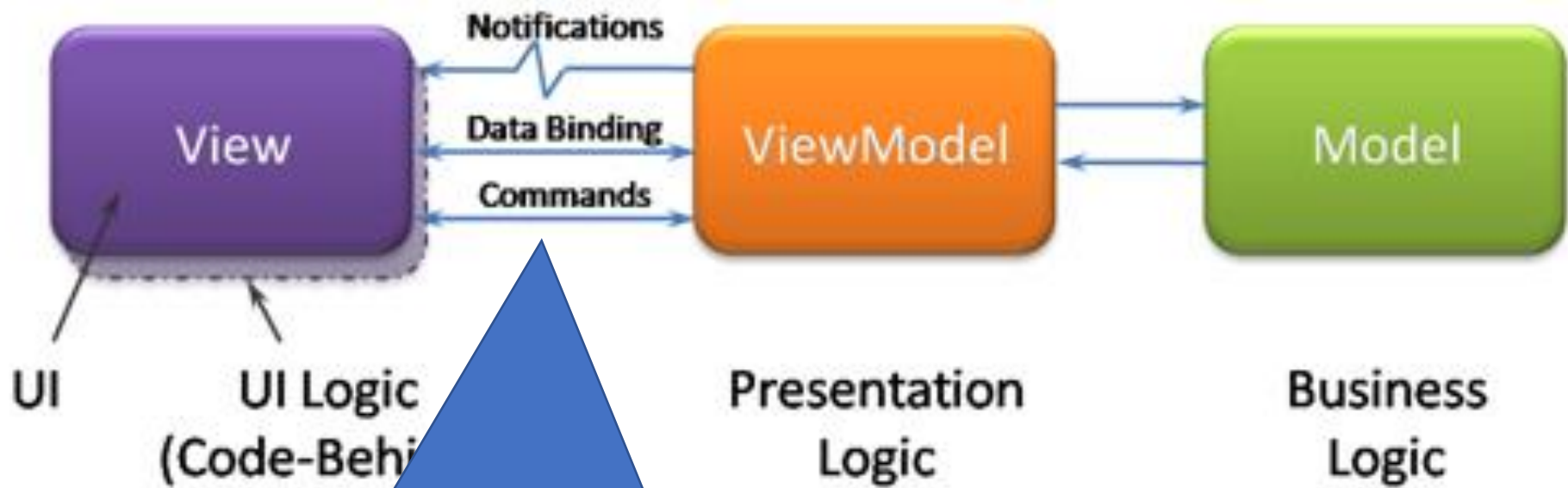
A comunicação entre Model e ViewModel ocorre via chamada de interfaces



<https://www.devmedia.com.br/entendendo-o-pattern-model-view-viewmodel-mvvm/18411>



- Visão geral do padrão MVVM



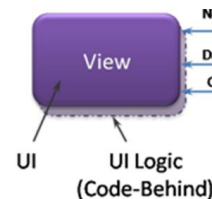
View através do **data binding** interage com a ViewModel notificando a ocorrência de eventos e o disparo de comandos (XAML e C#)

<https://www>

11



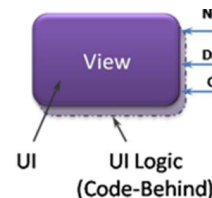
- View - define a aparência ou estrutura da interface
 - O ideal é que o codebehind da view, contenha apenas a chamada ao método `InitializeComponent` dentro do construtor
 - Em alguns casos, código que manipule os controles visuais, ou crie animações
- View se liga ao ViewModel, através da propriedade DataContext
 - Setada para a classe ViewModel correspondente à aquela View





- View – Características comuns

- A View é um elemento visual, como um objeto Window, Page, UserControl ou DataTemplate
- A View referencia a ViewModel através da propriedade DataContext. Os controles da View são preenchidos com propriedades ou comandos, expostos pela ViewModel
- O codebehind da view, define comportamentos visuais difíceis de expressar em XAML



<https://www.devmedia.com.br/entendendo-o-pattern-model-view-viewmodel-mvvm/18411>



- **ViewModel** – disponibiliza uma lógica de apresentação
 - A ViewModel não tem nenhum conhecimento específico sobre a view, ou como ela implementada, nem o seu tipo
 - A ViewModel implementa propriedades e comandos, para que a View possa preencher seus controles e notifica a mesma
 - Também pode implementar a lógica de validação, para garantir a consistência dos dados



Presentation
Logic

<https://www.devmedia.com.br/entendendo-o-pattern-model-view-viewmodel-mvvm/18411>



- ViewModel – Características comuns
 - É uma classe não visual, que expõe para a View uma lógica de apresentação
 - É testável, independentemente da View ou Model
 - Coordena as intenções entre a View e o Model
 - Não referencia a View, na verdade não tem nenhum conhecimento sobre a mesma
 - Implementa as interfaces INotifyPropertyChanged



Presentation
Logic

<https://www.devmedia.com.br/entendendo-o-pattern-model-view-viewmodel-mvvm/18411>



- **ViewModel – Características comuns**
 - Expõe propriedade e comando, para que a View possa utilizar para preencher seus controles; e notifica a View quando o estado de uma determinada propriedade muda, via implementação da interface INotifyPropertyChanged ou INotifyCollectionChanged
 - Pode conter a lógica de validação, através da implementação da interfaces IDataErrorInfo ou INotifyDataErrorInfo



Presentation
Logic

<https://www.devmedia.com.br/entendendo-o-pattern-model-view-viewmodel-mvvm/18411>



- **Model** – Encapsula a lógica de negócios e os dados
 - É o Modelo do domínio de uma aplicação, ou seja, as classes de negócio que serão utilizadas em uma determinada aplicação
 - Também contém os papéis e também a validação dos dados de acordo com o negócio, cuja aplicação em questão visa atender

Não é foco desta disciplina se aprofundar no model – regras de negócio, acesso a banco de dados, etc



Business
Logic

<https://www.devmedia.com.br/entendendo-o-pattern-model-view-viewmodel-mvvm/18411>



- Model – Características comuns
 - Classes que encapsulam a lógica de negócios e os dados
 - Não referencia diretamente a View ou ViewModel
 - Provê eventos de notificação de mudança de estado, através das interfaces INotifyPropertyChanged e INotifyCollectionChanged
 - Contém validação de dados e reporta os erros através da interface INotifyDataErrorInfo
 - Geralmente é utilizado, com um repositório (pode ser o Repository Pattern) ou serviço (não abordados nesta disciplina)



Business
Logic

<https://www.devmedia.com.br/entendendo-o-pattern-model-view-viewmodel-mvvm/18411>

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

MVVM - Exemplo

Conceitos MVVM – Prática - Agenda



ContatoMVVM

Novo Editar Deletar Sair

Nome	Sobrenome	Email	Telefone	
Rodrigo	Bonacin	rbonacin@unicamp.br	019-9999-9999	
José	Silva	Josesilv@email.com	019-9999-9999	
John	Snow	johnSnow@email.com	019-9999-9999	

Exemplo modificado de: <http://www.andrealveslima.com.br/blog/index.php/2017/11/15/exemplo-de-crud-no-wpf-com-mvvm/>

Conceitos MVVM – Prática - Agenda



Neste primeiro exemplo não utilizaremos um framework MVVM com classes prontas

```
MVVM_Exemplo> dotnet new wpf
```

```
▼ MVVM
  C# BaseCommand.cs
  C# BaseNotifyPropertyChanged.cs
```

Vamos criar classes de base para não precisar criar a cada classe

```
using System;
using System.Windows.Input;

namespace MVVM_Exemplo.MVVM
{
    4 references
    public abstract class BaseCommand : ICommand
    {
        1 reference
        public event EventHandler CanExecuteChanged;

        0 references
        public virtual bool CanExecute(object parameter) => true;
        0 references
        public abstract void Execute(object parameter);

        2 references
        public void RaiseCanExecuteChanged()
        {
            CanExecuteChanged?.Invoke(this, EventArgs.Empty);
        }
    }
}
```

Interface ICommand
(Próximo Slide)

Exemplo modificado de: <http://www.andrealveslima.com.br/blog/index.php/2017/11/15/exemplo-de-crud-no-wpf-com-mvvm/>



- `System.Windows.Input.Icommand`
 - Fornecem o comportamento de comando para elementos de interface do usuário (ex: Botão em XAML)

Métodos

`CanExecute(Object)`

Define o método que determina se o comando pode ser executado em seu estado atual.

`Execute(Object)`

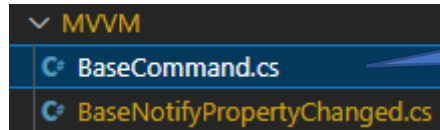
Define o método a ser chamado quando o comando for invocado.

Eventos

`CanExecuteChanged`

Ocorre em caso de alterações que afetam se o comando deve ser executado ou não.

Conceitos MVVM - Prática - Agenda



Vamos criar classes de base para não precisar criar a cada classe

```
using System;
using System.Windows.Input;

namespace MVVM_Exemplo.MVVM
{
    4 references
    public abstract class BaseCommand : ICommand
    {
        1 reference
        public event EventHandler CanExecuteChanged;

        0 references
        public virtual bool CanExecute(object parameter) => true;
        0 references
        public abstract void Execute(object parameter);

        2 references
        public void RaiseCanExecuteChanged()
        {
            CanExecuteChanged?.Invoke(this, EventArgs.Empty);
        }
    }
}
```

Método virtual, se não for sobrescrito retorna True. EX: se false desabilita o botão.

Método abstrato (deve ser sobrescrito para viabilizar a execução)

Dispara evento de mudança do "CanExecute"

Exemplo modificado de: <http://www.andrealveslima.com.br/blog/index.php/2017/11/15/exemplo-de-crud-no-wpf-com-mvvm/>

Conceitos MVVM - Prática - Agenda



▼ MVVM
BaseCommand.cs
BaseNotifyPropertyChanged.cs

Vamos criar classes de base para não precisar criar a cada classe

```
using System.Collections.Generic;

namespace MVVM_Exemplo.MVVM
{
    2 references
    public abstract class BaseNotifyPropertyChanged : System.ComponentModel.INotifyPropertyChanged
    {
        1 reference
        public event System.ComponentModel.PropertyChangedEventHandler PropertyChanged;

        5 references
        protected void SetField<T>(ref T field,
                                   T value,
                                   [System.Runtime.CompilerServices.CallerMemberName] string propertyName = null)
        {
            if (!EqualityComparer<T>.Default.Equals(field, value))
            {
                field = value;
                RaisePropertyChanged(propertyName);
            }
        }

        1 reference
        protected void RaisePropertyChanged(string propertyName)
        {
            PropertyChanged?.Invoke(this, new System.ComponentModel.PropertyChangedEventArgs(propertyName));
        }
    }
}
```

Interface
INotifyPropertyChanged
(Próximo Slide)



- `System.ComponentModel.INotifyPropertyChanged`
 - É usada para notificar clientes, normalmente associando clientes, que um valor de propriedade foi alterado.

Eventos

`PropertyChanged`

Ocorre quando um valor de propriedade é alterado.

Conceitos MVVM - Prática - Agenda



▼ MVVM
BaseCommand.cs
BaseNotifyPropertyChanged.cs

Vamos criar classes de base para não precisar criar a cada classe

```
using System.Collections.Generic;

namespace MVVM_Exemplo.MVVM
{
    2 references
    public abstract class BaseNotifyPropertyChanged : System.ComponentModel.INotifyPropertyChanged
    {
        1 reference
        public event System.ComponentModel.PropertyChangedEventHandler PropertyChanged;

        5 references
        protected void SetField<T>(ref T field,
                                   T value,
                                   [System.Runtime.CompilerServices.CallerMemberName] string propertyName = null)
        {
            if (!EqualityComparer<T>.Default.Equals(field, value))
            {
                field = value;
                RaisePropertyChanged(propertyName);
            }
        }

        1 reference
        protected void RaisePropertyChanged(string propertyName)
        {
            PropertyChanged?.Invoke(this, new System.ComponentModel.PropertyChangedEventArgs(propertyName));
        }
    }
}
```

Altera a propriedade e chama método *RaisePropertyChanged*

Gera evento de propriedade alterada

Conceitos MVVM - Prática - Agenda

Vamos criar o modelo (classe de negócio) “simples”

Model
Contato.cs

```
using System;
using MVVM_Exemplo.MVVM;

namespace MVVM_Exemplo.Model
{
    9 references
    public class Contato : BaseNotifyPropertyChanged, ICloneable
    {
        2 references
        private string _nome;
        2 references
        private string _sobrenome;
        2 references
        private string _telefone;
        2 references
        private string _email;
        5 references
        public string Nome
        {
            get { return _nome; }
            set
            {
                SetField(ref _nome, value);
            }
        }
        5 references
        public string Email
        {
            get { return _email; }
            set
            {
                SetField(ref _email, value);
            }
        }
    }
}
```

```
5 references
public string Sobrenome
{
    get { return _sobrenome; }
    set
    {
        SetField(ref _sobrenome, value);
    }
}
5 references
public string Telefone
{
    get { return _telefone; }
    set
    {
        SetField(ref _telefone, value);
    }
}
1 reference
public object Clone()
{
    return this.MemberwiseClone();
}
}
```


Conceitos MVVM - Prática - Agenda



Model

Contato.cs

Sempre que altera notifica

Implementa ICloneable

5 references
protected void SetField<T>(ref T field,

Clona o objeto

Conceitos MVVM - Prática - Agenda



▼ ViewModel

ContatoViewModel.cs

DeletarCommand.cs

EditarCommand.cs

NovoCommand.cs

SairCommand.cs

Desenvolver ViewModel

```
using System.Collections.ObjectModel;
using MVVM_Exemplo.Model;
using MVVM_Exemplo.MVVM;
namespace MVVM_Exemplo.ViewModel
{
```

7 references

```
public class ContatoViewModel : BaseNotifyPropertyChanged
```

```
{
```

8 references

```
public ObservableCollection<Contato> listaContato { get; set; }
```

```
// Objeto que lida com o comando deletar
```

1 reference

```
public DeletarCommand Deletar { get; private set; } = new DeletarCommand();
```

```
// Objeto que lida com o comando novo
```

0 references

```
public NovoCommand Novo { get; private set; } = new NovoCommand();
```

```
// Objeto que lida com o comando de editar
```

1 reference

```
public EditarCommand Editar { get; private set; } = new EditarCommand();
```

```
// Objeto que lida com o comando de sair
```

0 references

```
public SairCommand Sair { get; private set; } = new SairCommand();
```

Notifica alterações nas propriedades

Lista com todos os contatos

Objeto para tratar botão Deletar

Objeto para tratar botão Novo

Objeto para tratar botão Editar

Objeto para tratar botão Sair

Conceitos MVVM - Prática - Agenda



▼ ViewModel

- ContatoViewModel.cs
- DeletarCommand.cs
- EditarCommand.cs
- NovoCommand.cs
- SairCommand.cs

Desenvolver ViewModel

Contato selecionado na lista

```
2 references
private Model.Contato _contatoSelecionado;

11 references
public Model.Contato ContatoSelecionado
{
    get { return _contatoSelecionado; }
    set { SetField(ref _contatoSelecionado, value);
          Deletar.RaiseCanExecuteChanged();
          Editar.RaiseCanExecuteChanged(); }
}

1 reference
public ContatoViewModel()
{
    PreparaContatoCollection();
}
```

Quando o item selecionado muda verifica que o botão Deletar e Editar estarão habilitado

Chama método para popular a coleção

Desenvolver ViewModel



Cria Coleção

ViewModel

- ContatoViewModel.cs
- DeletarCommand.cs
- EditarCommand.cs
- NovoCommand.cs
- SairCommand.cs

Insere Contatos

Seleciona o primeiro

```
private void PreparaContatoCollection()
{
    listaContato = new ObservableCollection<Contato>();
    var Contato1 = new Contato
    {
        Nome = "Rodrigo",
        Sobrenome = "Bonacin",
        Email = "rbonacin@unicamp.br",
        Telefone = "019-9999-9999"
    };
    this.listaContato.Add(Contato1);
    var Contato2 = new Contato
    {
        Nome = "José",
        Sobrenome = "Silva",
        Email = "Josesilv@email.com",
        Telefone = "019-9999-9999"
    };
    this.listaContato.Add(Contato2);
    var Contato3 = new Contato
    {
        Nome = "John",
        Sobrenome = "Snow",
        Email = "johnSnow@email.com",
        Telefone = "019-9999-9999"
    };
    this.listaContato.Add(Contato3);
    ContatoSelecionado = this.listaContato[0];
}
```

Conceitos MVVM - Prática - Agenda



▼ ViewModel

ContatoViewModel.cs

DeletarCommand.cs

EditarCommand.cs

NovoCommand.cs

SairCommand.cs

```
using System.Linq;  
using MVVM_Exemplo.MVVM;
```

```
namespace MVVM_Exemplo.ViewModel  
{
```

2 references

```
public class DeletarCommand : BaseCommand
```

```
{
```

```
    // Só é possível deletar se tiver um contato selecionado
```

0 references

```
    public override bool CanExecute(object parameter)
```

```
    {
```

```
        var viewModel = parameter as ContatoViewModel;
```

```
        return viewModel != null && viewModel.ContatoSelecioneado != null;
```

```
    }
```

```
    // Execução do Deletar
```

0 references

```
    public override void Execute(object parameter)
```

```
    {
```

```
        var viewModel = (ContatoViewModel)parameter;
```

```
        viewModel.listaContato.Remove(viewModel.ContatoSelecioneado);
```

```
        viewModel.ContatoSelecioneado = viewModel.listaContato.FirstOrDefault();
```

```
    }
```

```
}
```

```
}
```

Estende a classe BaseCommand

O botão só fica habilitado se tiver um contato selecionado

Implementação do método para apagar contato selecionado

Conceito

```
using MVVM_Exemplo.MVVM;
```

```
namespace MVVM_Exemplo.ViewModel  
{
```

2 references

```
public class EditarCommand : BaseCommand
```

```
{
```

0 references

```
public override bool CanExecute(object parameter)
```

```
{
```

```
    var viewModel = parameter as ContatoViewModel;
```

```
    return viewModel != null && viewModel.ContatoSelecionado != null;
```

```
}
```

0 references

```
public override void Execute(object parameter)
```

```
{
```

```
    var viewModel = (ContatoViewModel)parameter;
```

```
    var cloneContato = (Model.Contato)viewModel.ContatoSelecionado.Clone();
```

```
    var fw = new ContatoWindow();
```

```
    fw.DataContext = cloneContato;
```

```
    fw.ShowDialog();
```

```
    if (fw.DialogResult.HasValue && fw.DialogResult.Value)
```

```
    {
```

```
        viewModel.ContatoSelecionado.Nome = cloneContato.Nome;
```

```
        viewModel.ContatoSelecionado.Sobrenome = cloneContato.Sobrenome;
```

```
        viewModel.ContatoSelecionado.Telefone = cloneContato.Telefone;
```

```
        viewModel.ContatoSelecionado.Email = cloneContato.Email;
```

```
    }
```

```
}
```

```
}
```

```
}
```

Estende a classe BaseCommand

ViewModel

ContatoViewModel

DeletarCommand.cs

EditarCommand.cs

NovoCommand.cs

SairCommand.cs

O botão só fica habilitado se tiver um contato selecionado

Clona o objeto selecionado, cria uma janela, seta o DataContext e exibe

Atribui os campos preenchidos para o objeto selecionado

Conceitos



Estende a classe BaseCommand

```
using MVVM_Exemplo.MVVM;

namespace MVVM_Exemplo.ViewModel
{
    2 references
    public class NovoCommand : BaseCommand
    {
        0 references
        public override bool CanExecute(object parameter)
        {
            return parameter is ContatoViewModel;
        }
        0 references
        public override void Execute(object parameter)
        {
            var viewModel = (ContatoViewModel)parameter;
            var contato = new Model.Contato();

            var fw = new MVVM_Exemplo.ContatoWindow();
            fw.DataContext = contato;

            fw.ShowDialog();

            if (fw.DialogResult.HasValue && fw.DialogResult.Value)
            {
                viewModel.listaContato.Add(contato);
                viewModel.ContatoSelecionado = contato;
            }
        }
    }
}
```

O botão fica habilitado

Cria janela par
inclusão e atribui
DataContext

Inclui o contato novo na
lista

Conceitos MVVM - Prática - Agenda



▼ ViewModel

- ContatoViewModel.cs
- DeletarCommand.cs
- EditarCommand.cs
- NovoCommand.cs
- SairCommand.cs

```
using MVVM_Exemplo.MVVM;

namespace MVVM_Exemplo.ViewModel
{
    2 references
    public class SairCommand : BaseCommand
    {
        // Não sobrecremos a CanExecute -
        // ela poderia verificar condições de sair (caso existam)

        // Execução do Sair (fecha aplicação)
        0 references
        public override void Execute(object parameter)
        {
            System.Windows.Application.Current.Shutdown();
        }
    }
}
```

Estende a classe BaseComand

Conceitos MVVM - Prática - Agenda



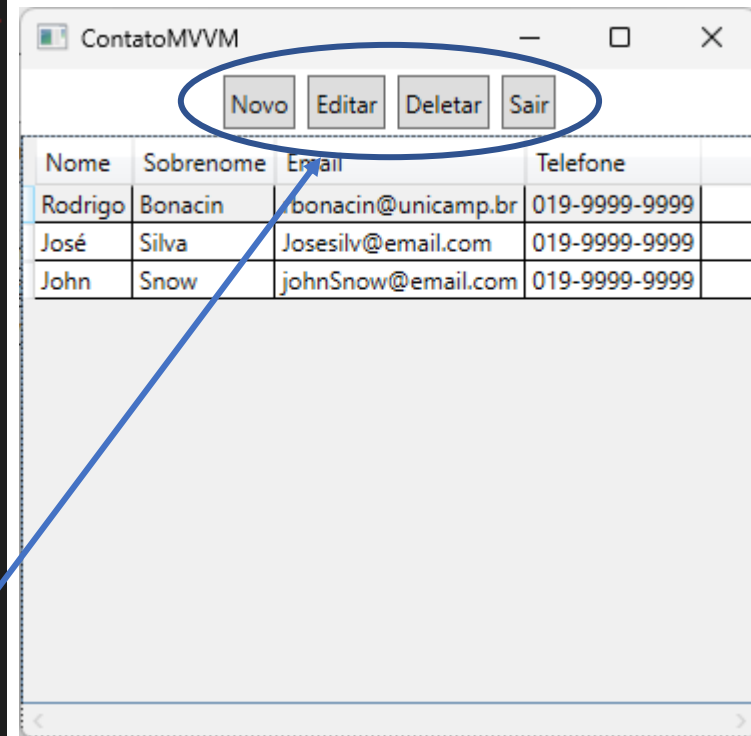
Janela principal

MainWindow.xaml

MainWindow.xaml.cs

```
<Window x:Class="MVVM_Exemplo.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:MVVM_Exemplo"
        mc:Ignorable="d"
        Title="ContatoMVVM" Height="450" Width="800">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="*" />
            <RowDefinition Height="9*" />
        </Grid.RowDefinitions>
        <StackPanel Orientation="Horizontal" HorizontalAlignment="Center">
            <Button Margin="3" Padding="3"
                    CommandParameter="{Binding}"
                    Command="{Binding Novo}"
                    Content="Novo" />
            <Button Margin="3" Padding="3"
                    CommandParameter="{Binding}"
                    Command="{Binding Editar}"
                    Content="Editar" />
            <Button Margin="3" Padding="3"
                    CommandParameter="{Binding}"
                    Command="{Binding Deletar}"
                    Content="Deletar" />
            <Button Margin="3" Padding="3"
                    IsCancel="true"
                    Command="{Binding Sair}"
                    Content="Sair" />
        </StackPanel>
    </Grid>
</Window>
```

Botões com
Binding para as
classes "Novo,
Editar, Deletar e
Sair"



Conceitos MVVM - Prática - Agenda



Janela principal

MainWindow.xaml

MainWindow.xaml.cs

DataGrid com Binding para a listaContato (*ObservableCollection*)

```
<ScrollView Grid.Row="1"
    HorizontalScrollBarVisibility="Visible"
    VerticalScrollBarVisibility="Hidden">
    <DataGrid ItemsSource="{Binding listaContato}"
        AutoGenerateColumns="False"
        IsReadOnly="True"
        SelectedItem="{Binding ContatoSelecioneado}">
        <DataGrid.Columns>
            <DataGridTextColumn Header="Nome" Binding="{Binding Nome}"/>
            <DataGridTextColumn Header="Sobrenome" Binding="{Binding Sobrenome}"/>
            <DataGridTextColumn Header="Email" Binding="{Binding Email}"/>
            <DataGridTextColumn Header="Telefone" Binding="{Binding Telefone}"/>
        </DataGrid.Columns>
    </DataGrid>
</ScrollView>
</Grid>
</Window>
```

ContatoMVVM

Novo Editar Deletar Sair

Nome	Sobrenome	Email	Telefone
Rodrigo	Bonacin	rbonacin@unicamp.br	019-9999-9999
José	Silva	Josesilv@email.com	019-9999-9999
John	Snow	johnSnow@email.com	019-9999-9999

Conceitos MVVM - Prática - Agenda



MainWindow.xaml
MainWindow.xaml.cs

View com o mínimo de código

```
using System.Windows;

namespace MVVM_Exemplo
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    0 references
    public partial class MainWindow : Window
    {
        0 references
        public MainWindow()
        {
            InitializeComponent();
            DataContext = new ViewModel.ContatoViewModel();
        }
    }
}
```

DataContext é o ViewModel

Conceitos MVVM - Prática - Agenda



☰ ContatoWindow.xaml
☑ ContatoWindow.xaml.cs

Janela para inclusão e alteração de contato

```
<Window x:Class="MVVM_Exemplo.ContatoWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  xmlns:local="clr-namespace:MVVM_Exemplo"
  mc:Ignorable="d"
  Title="Contato" Height="326.178" Width="300" WindowStyle="ToolWindow">
  <Grid Margin="3">
    <StackPanel Orientation="Vertical">
      <TextBlock Text="Nome"/>
      <TextBox Text="{Binding Nome}"/>
      <TextBlock Text="Sobrenome"/>
      <TextBox Text="{Binding Sobrenome}"/>
      <TextBlock Text="Telefone"/>
      <TextBox Text="{Binding Telefone}"/>
      <TextBlock Text="Email"/>
      <TextBox Text="{Binding Email}"/>
    </StackPanel>
    <Grid>
      <Grid.ColumnDefinitions>
        <ColumnDefinition Width="5*"/>
        <ColumnDefinition Width="5*"/>
      </Grid.ColumnDefinitions>
      <Button Name="OKButton"
        Grid.Column="0"
        Content="OK"
        Margin="3"
        IsDefault="True"
        Click="OKButton_Click"/>
      <Button Grid.Column="1"
        Content="Cancelar"
        Margin="3"
        IsCancel="True"/>
    </Grid>
  </Grid>
</Window>
```

Binding com campos do contato (Model). Se for seguir “na risca” o MVVM deveria ter mais um *viewmodel* aqui

Botão de OK e cancelar

Contato

Nome
Rodrigo

Sobrenome
Bonacin

Telefone
019-9999-9999

Email
rbonacin@unicamp.br

OK Cancelar

Conceitos MVVM - Prática - Agenda



- Temos três casos que **violam** o MVVM:

ContatoWindow.xaml

```
<StackPanel Orientation="Vertical">
  <TextBlock Text="Nome"/>
  <TextBox Text="{Binding Nome}"/>
  <TextBlock Text="Sobrenome"/>
  <TextBox Text="{Binding Sobrenome}"/>
  <TextBlock Text="Telefone"/>
  <TextBox Text="{Binding Telefone}"/>
  <TextBlock Text="Email"/>
  <TextBox Text="{Binding Email}"/>
</StackPanel>
```

Views não
devem
comunicar
diretamente
com os
Models.

EditarCommand.cs

```
public override void Execute(object parameter)
{
    var viewModel = (ContatoViewModel)parameter;
    var cloneContato = (Model.Contato)viewModel.ContatoSelecionado.Clone();
    var fw = new ContatoWindow();
    fw.DataContext = cloneContato;
    fw.ShowDialog();
    if (fw.DialogResult.HasValue && fw.DialogResult.Value)
    {
        viewModel.ContatoSelecionado.Nome = cloneContato.Nome;
        viewModel.ContatoSelecionado.Sobrenome = cloneContato.Sobrenome;
        viewModel.ContatoSelecionado.Telefone = cloneContato.Telefone;
        viewModel.ContatoSelecionado.Email = cloneContato.Email;
    }
}
```

NovoCommand.cs

```
public override void Execute(object parameter)
{
    var viewModel = (ContatoViewModel)parameter;
    var contato = new Model.Contato();

    var fw = new MVVM_Exemplo.ContatoWindow();
    fw.DataContext = contato;

    fw.ShowDialog();

    if (fw.DialogResult.HasValue && fw.DialogResult.Value)
    {
        viewModel.listaContato.Add(contato);
        viewModel.ContatoSelecionado = contato;
    }
}
```

ViewModel não deve
saber nada das Views

- Ver discussão em:

<http://www.andrealveslima.com.br/blog/index.php/2017/11/15/exemplo-de-crud-no-wpf-com-mvvm/>

- **Exercício extra**: deixar 100% MVVM (sem framework)

Conceitos MVVM - Prática - Agenda



- Funcionamento/Teste:

1

2

4

3

5

7

8

6

ContatoMVVM

Novo Editar Deletar Sair

Nome	Sobrenome	Email	Telefone
Rodrigo	Bonacin	rbonacin@unicamp.br	019-9999-9999
José	Silva	Josesilv@email.com	019-9999-9999
John	Snow	johnSnow@email.com	019-9999-9999

Contato

Nome: Maria

Sobrenome: Silva

Telefone: (19) 9999-9999

Email: email@email.com

OK Cancelar

ContatoMVVM

Novo Editar Deletar Sair

Nome	Sobrenome	Email	Telefone
Rodrigo	Bonacin	rbonacin@unicamp.br	019-9999-9999
José	Silva	Josesilv@email.com	019-9999-9999
John	Snow	johnSnow@email.com	019-9999-9999
Maria	Silva	email@email.com	(19) 9999-9999

Contato

Nome: José

Sobrenome: Silva

Telefone: 019-9999-9999

Email: Josesilva@email.com

OK Cancelar

ContatoMVVM

Novo Editar Deletar Sair

Nome	Sobrenome	Email	Telefone
Rodrigo	Bonacin	rbonacin@unicamp.br	019-9999-9999
José	Silva	Josesilv@email.com	019-9999-9999
John	Snow	johnSnow@email.com	019-9999-9999
Maria	Silva	email@email.com	(19) 9999-9999

ContatoMVVM

Novo Editar Deletar Sair

Nome	Sobrenome	Email	Telefone
José	Silva	Josesilva@email.com	019-9999-9999
John	Snow	johnSnow@email.com	019-9999-9999
Maria	Silva	email@email.com	(19) 9999-9999



MVVM – MVVM Toolkit



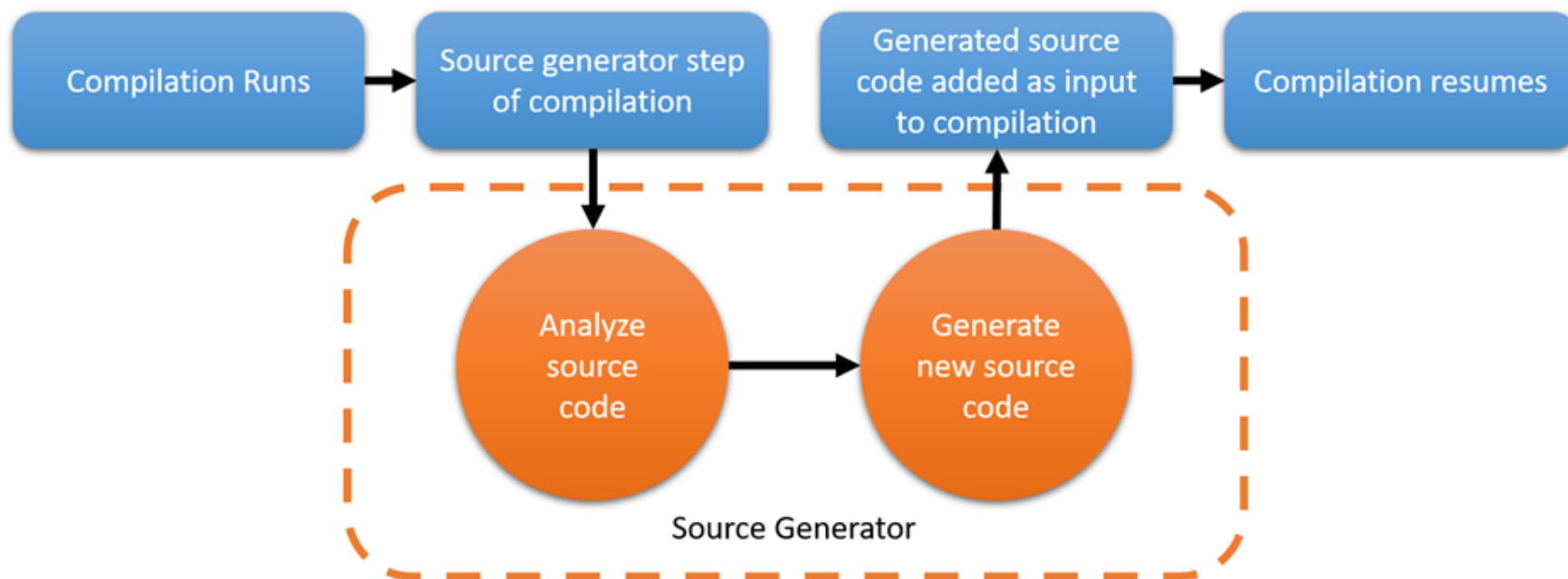
- Existem frameworks e implementações de “classes de base” que facilitam o desenvolvimento em MVVM, ex:
 - MVVM Toolkit (CommunityToolkit)
- O Kit de Ferramentas MVVM é mantido e publicado pela Microsoft e faz parte do .NET Foundation
 - Pode ser usado em qualquer plataforma de aplicativo: UWP, WinForms, WPF, Xamarin, Uno ...

Instalação do MVVM Toolkit

```
[MVVM_Exemplo_Toolkit]> dotnet add package CommunityToolkit.Mvvm
```

<https://learn.microsoft.com/pt-br/dotnet/communitytoolkit/mvvm/>

MVVM – MVVM Toolkit



<https://learn.microsoft.com/pt-br/dotnet/communitytoolkit/mvvm/>



- Objetos Observáveis

- *ObservableObject*

- objetos que podem ser observados implementando as interfaces `INotifyPropertyChanging` as `INotifyPropertyChanged`

- *ObservableRecipient*

- objetos observáveis que também atuam como destinatários de mensagens

- *ObservableValidator*

- Implementa a `INotifyDataErrorInfo` interface, fornecendo suporte para validar propriedades expostas a outros módulos de aplicativo



- Injeção de dependência
 - *CommunityToolkit.Mvvm.DependencyInjection*
 - objetos que podem ser observados implementando as interfaces `INotifyPropertyChanging` as `INotifyPropertyChanged`
 - Criação de vários serviços que são injetados em classes de back-end (ou seja, passados como parâmetros para os construtores viewmodel)
 - Permite que o código que usa esses serviços não dependa dos detalhes de implementação desses serviços e também facilita a troca das implementações concretas desses serviços
 - Facilita a disponibilização de recursos específicos da plataforma para o código de back-end, abstraindo-os por meio de um serviço que é injetado quando necessário



- Mensageiro
 - *IMessenger*
 - Tipos que podem ser usados para trocar mensagens entre objetos diferentes
 - *WeakReferenceMessenger*
 - Implementação de referência para o *IMessenger*
 - *StrongReferenceMessenger*
 - Implementação de referência para o *IMessenger*, com referência forte (é necessário “desregistrar” os recipientes de mensagens manualmente)



- Mensageiro

- *IRecipient<TMessage>*

- Interface para um destinatário que declara um registro para um tipo de mensagem específico

- *MessageHandler<TRecipient, TMessage>*

- Um *delegate* usado para representar ações a serem invocadas quando uma mensagem é recebida



- *PropertyChangedMessage<T>*
 - Mensagem usada para transmitir alterações de propriedade em objetos observáveis
- *RequestMessage<T>*
 - Mensagens de solicitação - usada diretamente ou por meio de classes derivadas
- *AsyncRequestMessage<T>*
 - Mensagens de solicitação assíncrona, que podem ser usadas diretamente ou por meio de classes derivadas
- *CollectionRequestMessage<T>* e *AsyncCollectionRequestMessage<T>*
 - Mensagens de solicitação que podem receber várias respostas
- *MessageHandler<TRecipient, TMessage>*
 - Representam ações a serem invocadas quando uma mensagem é recebida



- Comando

- `CommunityToolkit.Mvvm.Input`

- `RelayCommand`
- `RelayCommand<T>`
- `AsyncRelayCommand`
- `AsyncRelayCommand<T>`
- `IRelayCommand`
- `IRelayCommand<T>`
- `IAsyncRelayCommand`
- `IAsyncRelayCommand<T>`

As implementações `RelayCommand` e `RelayCommand<T>` são `IRelayCommand` que podem expor um método ou delegar ao modo de exibição.

Esses tipos atuam como uma maneira de associar comandos entre os elementos viewmodel e de interface do usuário.

(**não** é necessário criar classes individuais como no exemplo anterior)

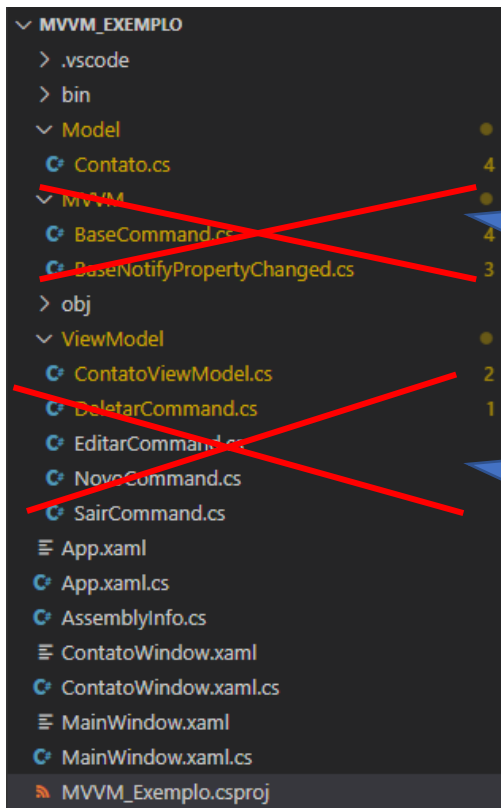


MVVM Toolkit - Exemplo

MVVM Toolkit - Exemplo

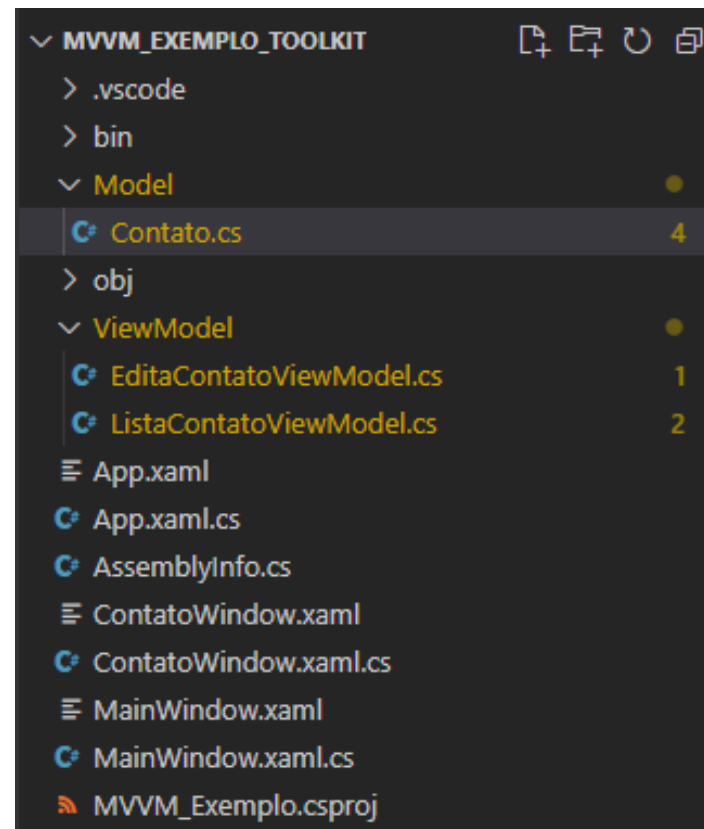


- Vamos atualizar nossa agenda para usar o Toolkit (CommunityToolkit) e ficar mais aderente ao MVVM



As classes de base já estão implementadas

Vamos usar RelayCommand



MVVM Toolkit - Exemplo



C# Contato.cs

```
namespace MVVM_Exemplo_Toolkit.Model
{
    10 references
    public class Contato : ObservableObject, ICloneable
    {
        2 references
        private string _nome;
        2 references
        private string _sobrenome;
        2 references
        private string _telefone;
        2 references
        private string _email;
        5 references
        public string Nome
        {
            get { return _nome; }
            set { SetProperty(ref _nome, value); }
        }
        5 references
        public string Email
        {
            get { return _email; }
            set { SetProperty(ref _email, value); }
        }
        5 references
        public string Sobrenome
        {
            get { return _sobrenome; }
            set { SetProperty(ref _sobrenome, value); }
        }
        5 references
        public string Telefone
        {
            get { return _telefone; }
            set { SetProperty(ref _telefone, value); }
        }
        1 reference
        public object Clone()
        {
            return this.MemberwiseClone();
        }
    }
}
```

Classe base já implementada

Usa outro método de Set

MVVM Toolkit - Exemplo



ListaContatoViewModel.cs

```
using System.Collections.ObjectModel;
using MVVM_Exemplo_Toolkit.Model;
using CommunityToolkit.Mvvm.Input;
using CommunityToolkit.Mvvm.ComponentModel;
using CommunityToolkit.Mvvm.Messaging;
using CommunityToolkit.Mvvm.Messaging.Messages;
```

```
namespace MVVM_Exemplo_Toolkit.ViewModel
```

```
{
```

```
    1 reference
```

```
    public class ListaContatoViewModel : ObservableObject
```

```
    {
```

```
        // Coleção de Objetos
```

```
        9 references
```

```
        public ObservableCollection<Contato> listaContato { get; set; }
```

```
        // Contato selecionado
```

```
        2 references
```

```
        private Model.Contato _contatoSelecionado;
```

```
        // Objeto que lida com o comando novo
```

```
        1 reference
```

```
        public RelayCommand Novox {get; set; }
```

```
        // Objeto que lida com o comando deletar
```

```
        2 references
```

```
        public RelayCommand Deletarx {get; set; }
```

```
        // Objeto que lida com o comando editar
```

```
        2 references
```

```
        public RelayCommand Editarx {get; set; }
```

```
        // Objeto que lida com o comando sair
```

```
        1 reference
```

```
        public RelayCommand Sairx {get; set; }
```

Classe base já implementada

Coleção observável

Um objeto para cada botão
(não precisa criar classes)

MVVM Toolkit - Exemplo



ListaContatoViewModel.cs

```
// Executa comando de novo contato
1 reference
private void NovoCMD()
{
    var contatoViewModel = new EditarContatoViewModel();

    WeakReferenceMessenger.Default.Send(new OpenWindowMessage(contatoViewModel));
    if (contatoViewModel.contato != null)
    {
        this.listaContato.Add(contatoViewModel.contato);
        this.ContatoSelecionado = contatoViewModel.contato;
    }
}
```

Tratamento do botão “Novo”

Cria uma View para a Janela “filha”

Envia mensagem que vai ser tratada pela view da janela principal (é ela que abre a outra janela)

Caso o novo contato seja preenchido insere ele

MVVM Toolkit - Exemplo



ListaContatoViewModel.cs

Tratamento do botão “Editar”

Cria uma View para a Janela “filha”

```
// Executa comando editar
```

```
1 reference
```

```
private void EditarCMD()
```

```
{
```

```
    var contatoViewModel = new EditarContatoViewModel();
```

```
    var cloneContato = (Model.Contato)this.ContatoSelecionado.Clone();
```

```
    contatoViewModel.contato = cloneContato;
```

Envia mensagem que vai ser tratada pela view da janela principal (é ela que abre a outra janela)

```
    WeakReferenceMessenger.Default.Send(new OpenWindowMessage(contatoViewModel));
```

```
    if (contatoViewModel.contato != null)
```

```
    {
```

```
        this.ContatoSelecionado.Nome = cloneContato.Nome;
```

```
        this.ContatoSelecionado.Sobrenome = cloneContato.Sobrenome;
```

```
        this.ContatoSelecionado.Telefone = cloneContato.Telefone;
```

```
        this.ContatoSelecionado.Email = cloneContato.Email;
```


```
    }
```

```
}
```

Caso o seja editado altera ele

MVVM Toolkit - Exemplo



 ListaContatoViewModel.cs

```
// Executa comando deletar
1 reference
private void DeletarCMD()
{
    this.listaContato.Remove(this.ContatoSelecioneado);
    if (this.listaContato.Count > 0)
        this.ContatoSelecioneado = this.listaContato[0];
    else
        this.ContatoSelecioneado = null;
}
```

```
// Executa comando sair
1 reference
private void SairCMD()
{
    System.Windows.Application.Current.Shutdown();
}
```

MVVM Toolkit - Exemplo



C# ListaContatoViewModel.cs

```
//Executa can deletar
1 reference
private bool CanDeletarCMD()
{
    return this.ContatoSelecioneado != null;
}
```

```
//Executa can deletar
1 reference
private bool CanEditarCMD()
{
    return this.ContatoSelecioneado != null;
}
```

```
// Quando o selecionado muda notificar CanExecute dos botões Editar e Deletar
12 references
public Contato ContatoSelecioneado
{
    get { return _contatoSelecioneado; }
    set
    {
        SetProperty(ref _contatoSelecioneado, value);
        Deletarx.NotifyCanExecuteChanged();
        Editarx.NotifyCanExecuteChanged();
    }
}
```

Verifica se tem um contato selecionado (para habilitar ou desabilitar botões)

MVVM Toolkit - Exemplo



ListaContatoViewModel.cs

Construtor, aonde são criados e associados os *RelayCommand*

```
// Construtor
1 reference
public ListaContatoViewModel()
{
    Novox = new RelayCommand(NovoCMD);
    Deletarx = new RelayCommand(DeletarCMD, CanDeletarCMD);
    Editarx = new RelayCommand(EditarCMD, CanEditarCMD);
    Sairx = new RelayCommand(SairCMD);
    listaContato = new ObservableCollection<Contato>();
    PreparaContatoCollection();
}
```


MVVM Toolkit - Exemplo



C# ListaContatoViewModel.cs

```
private void PreparaContatoCollection()
{
    var Contato1 = new Contato
    {
        Nome = "Rodrigo",
        Sobrenome = "Bonacin",
        Email = "rbonacin@unicamp.br",
        Telefone = "019-9999-9999"
    };
    this.listaContato.Add(Contato1);
    var Contato2 = new Contato
    {
        Nome = "José",
        Sobrenome = "Silva",
        Email = "Josesilv@email.com",
        Telefone = "019-9999-9999"
    };
    this.listaContato.Add(Contato2);
    var Contato3 = new Contato
    {
        Nome = "John",
        Sobrenome = "Snow",
        Email = "johnSnow@email.com",
        Telefone = "019-9999-9999"
    };
    this.listaContato.Add(Contato3);
    ContatoSelecionado = this.listaContato[0];
}
```

Não foi alterado ... em uma situação real, por exemplo, poderia interagir com o Model que implementaria um padrão de acesso a um banco de dados

MVVM Toolkit - Exemplo



C# ListaContatoViewModel.cs

Criar no mesmo arquivo a classe
OpenWindowMessage que implementa
uma mensagem de “abrir janela”

```
// Cria uma classe de mensagem OpenWindowMessage
3 references
public class OpenWindowMessage : ValueChangedMessage<EditaContatoViewModel>
{
    2 references
    public OpenWindowMessage(EditaContatoViewModel contatoViewModel) : base(contatoViewModel)
    {
    }
}
}
```

MVVM Toolkit - Exemplo



EditaContatoViewModel.cs

```
using CommunityToolkit.Mvvm.ComponentModel;
using MVVM_Exemplo_Toolkit.Model;
using CommunityToolkit.Mvvm.Input;
using CommunityToolkit.Mvvm.Messaging;
using CommunityToolkit.Mvvm.Messaging.Messages;

namespace MVVM_Exemplo_Toolkit.ViewModel
{
    5 references
    public class EditaContatoViewModel : ObservableObject
    {
        7 references
        public Contato contato { get; set; }
        // Objeto que lida com o comando OK
        1 reference
        public RelayCommand OK { get; set; }
        // Objeto que lida com o comando Cancelar
        1 reference
        public RelayCommand Cancelar { get; set; }
        // Executa comando de OK
        1 reference
    }
}
```

Criar um ViewModel
associado à Janela de
Editar/Novo

Agora é a ViewModel
que se comunica com o
Model

Objetos para lidar com o
botão OK e Cancelar

MVVM Toolkit - Exemplo



EditaContatoViewModel.cs

Métodos para execução
do OK e cancelar

```
// Executa comando de OK
1 reference
private void OkCMD()
{
    bool comando = true;
    WeakReferenceMessenger.Default.Send(new CloseWindowMessage(comando));
}

// Executa comando de Cancelar
1 reference
private void CancelarCMD()
{
    bool comando = false;
    WeakReferenceMessenger.Default.Send(new CloseWindowMessage(comando));

    // Anula alteração
    contato = null;
}
```

Envia evento de janela
Fechada para a View

MVVM Toolkit - Exemplo



EditaContatoViewModel.cs

```
// Construtor
3 references
public EditarContatoViewModel()
{
    OK = new RelayCommand(OkCMD);
    Cancelar = new RelayCommand(CancelarCMD);
    contato = new Contato();
}
}
```

Construtor onde são associados o RelayCommand aos botões

MVVM Toolkit - Exemplo



EditaContatoViewModel.cs

Criar no mesmo arquivo a classe CloseWindowMessage que implementa uma mensagem de “fechar janela”

```
// Cria uma classe de mensagem OpenWindowMessage
3 references
public class CloseWindowMessage : ValueChangedMessage<bool>
{
    2 references
    public CloseWindowMessage(bool value) : base(value)
    {
    }
}
}
```

MVVM Toolkit - Exemplo



ContatoWindow.xaml

```
<Window x:Class="MVVM_Exemplo_Toolkit.ContatoWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:MVVM_Exemplo_Toolkit"
        mc:Ignorable="d"
        Title="Contato" Height="326.178" Width="300" WindowStyle="ToolWindow">
    <Grid Margin="3">
        <StackPanel Orientation="Vertical">
            <TextBlock Text="Nome"/>
            <TextBox Text="{Binding contato.Nome}"/>
            <TextBlock Text="Sobrenome"/>
            <TextBox Text="{Binding contato.Sobrenome}"/>
            <TextBlock Text="Telefone"/>
            <TextBox Text="{Binding contato.Telefone}"/>
            <TextBlock Text="Email"/>
            <TextBox Text="{Binding contato.Email}"/>
            <Grid>
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="5*"/>
                    <ColumnDefinition Width="5*"/>
                </Grid.ColumnDefinitions>
                <Button Name="OKButton"
                        Grid.Column="0"
                        Content="OK"
                        Margin="3"
                        IsDefault="True"
                        CommandParameter="{Binding}"
                        Command="{Binding OK}" />
                <Button Grid.Column="1"
                        Content="Cancelar"
                        Margin="3"
                        IsCancel="True"
                        CommandParameter="{Binding}"
                        Command="{Binding Cancelar}" />
            </Grid>
        </StackPanel>
    </Grid>
</Window>
```

Agora não há Binding com o Model diretamente ... e sim com ModelView

Binding com RelayCommand OK e Cancelar do ModelView

MVVM Toolkit - Exemplo



C# ContatoWindow.xaml.cs

```
using System.Windows;
using CommunityToolkit.Mvvm.Messaging;
using MVVM_Exemplo_Toolkit.ViewModel;

namespace MVVM_Exemplo_Toolkit
{
    // C#
    1 reference
    public partial class ContatoWindow : Window
    {
        1 reference
        public ContatoWindow()
        {
            InitializeComponent();

            WeakReferenceMessenger.Default.Register<CloseWindowMessage>(this, (r, m) =>
            {
                this.Hide();
            });

            DataContext = new EditaContatoViewModel();
        }
    }
}
```

A view se registra para receber o evento para fechar a janela (quem “cuida” da interface agora é a view)

O DataContext da interface é a ViewModel (agora não há ligação direta da View com o Model)

MVVM Toolkit - Exemplo



≡ MainWindow.xaml

```
<Window x:Class="MVVM_Exemplo_Toolkit.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:MVVM_Exemplo_Toolkit"
        mc:Ignorable="d"
        Title="ContatoMVVM" Height="450" Width="800">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="*" />
            <RowDefinition Height="9*" />
        </Grid.RowDefinitions>
        <StackPanel Orientation="Horizontal" HorizontalAlignment="Center">
            <Button Margin="3" Padding="3"
                    CommandParameter="{Binding}"
                    Command="{Binding Novox}"
                    Content="Novo" />
            <Button Margin="3" Padding="3"
                    CommandParameter="{Binding}"
                    Command="{Binding Editarx}"
                    Content="Editar" />
            <Button Margin="3" Padding="3"
                    CommandParameter="{Binding}"
                    Command="{Binding Deletarx}"
                    Content="Deletar" />
            <Button Margin="3" Padding="3"
                    IsCancel="true"
                    Command="{Binding Sairx}"
                    Content="Sair" />
        </StackPanel>
```

Binding com
RelayCommands do
ModelView

MVVM Toolkit - Exemplo



≡ MainWindow.xaml

```
<ScrollView Grid.Row="1"
    HorizontalScrollBarVisibility="Visible"
    VerticalScrollBarVisibility="Hidden">
    <DataGrid ItemsSource="{Binding listaContato}"
        AutoGenerateColumns="False"
        IsReadOnly="True"
        SelectedItem="{Binding ContatoSelecionado}">
        <DataGrid.Columns>
            <DataGridTextColumn Header="Nome" Binding="{Binding Nome}"/>
            <DataGridTextColumn Header="Sobrenome" Binding="{Binding Sobrenome}"/>
            <DataGridTextColumn Header="Email" Binding="{Binding Email}"/>
            <DataGridTextColumn Header="Telefone" Binding="{Binding Telefone}"/>
        </DataGrid.Columns>
    </DataGrid>
</ScrollView>
</Grid>
</Window>
```

MVVM Toolkit - Exemplo



C# MainWindow.xaml.cs

```
using System.Windows;
using CommunityToolkit.Mvvm.Messaging;
using MVVM_Exemplo_Toolkit.ViewModel;

namespace MVVM_Exemplo_Toolkit
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    0 references
    public partial class MainWindow : Window
    {
        0 references
        public MainWindow()
        {
            InitializeComponent();
            // Register a message in some module
            WeakReferenceMessenger.Default.Register<OpenWindowMessage>(this, (r, m) =>
            {
                var fw = new MVVM_Exemplo_Toolkit.ContatoWindow();

                // Passa o contato
                fw.DataContext = m.Value;

                fw.ShowDialog();
            });

            DataContext = new ViewModel.ListaContatoViewModel();
        }
    }
}
```

A view se registra para receber o evento para abrir a janela de edição (quem “cuida” da interface é a view)

O DataContext da interface é a ViewModel (não há ligação direta da View com o Model)

MVVM Toolkit - Exemplo



ContatoMVVM

Novo Editar Deletar Sair

Nome	Sobrenome	Email	Telefone
Rodrigo	Bonacin	rbonacin@unicamp.br	019-9999-9999
José	Silva	Josesilv@email.com	019-9999-9999
John	Snow	johnSnow@email.com	019-9999-9999

Isso tudo é só a “Ponta do Iceberg”

