

programação cross-platform com

xamarin

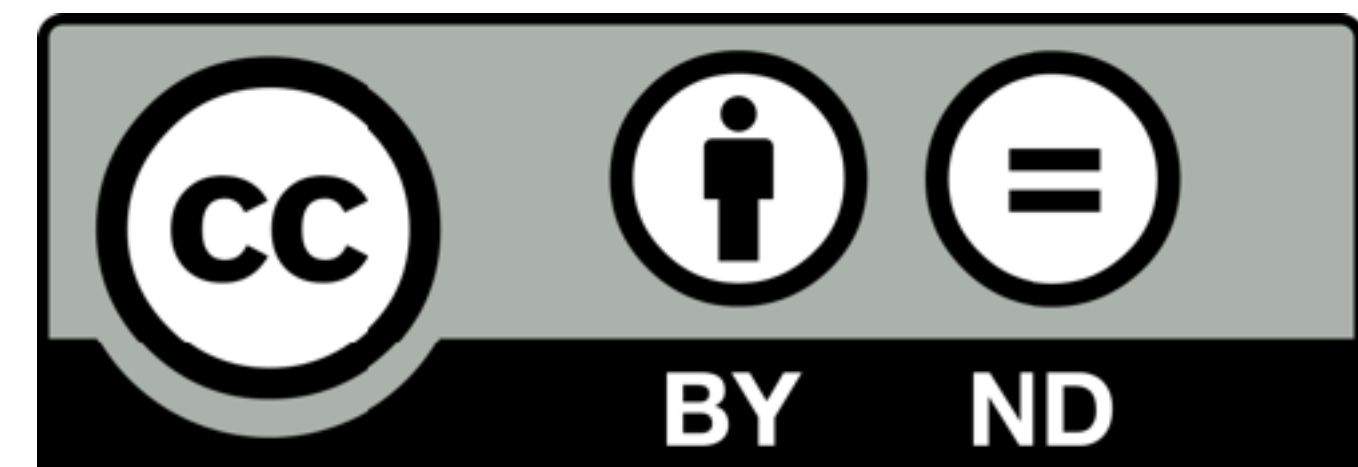


daniel ferreira
@dfsdaniel

Licença



- **Compartilhar:**
 - Você tem o direito de copiar e redistribuir o material em qualquer suporte ou formato para qualquer fim, mesmo que comercial.
- **Atribuição:**
 - Você deve dar o crédito apropriado, prover um link para a licença e indicar se mudanças foram feitas. Você deve fazê-lo em qualquer circunstância razoável, mas de maneira alguma que sugira ao licenciante a apoiar você ou o seu uso.
- **Sem derivações:**
 - Se você remixar, transformar ou criar a partir do material, você não pode distribuir o material modificado.
- **Licença CC BY-ND 4.0:**
 - https://creativecommons.org/licenses/by-nd/4.0/deed.pt_BR





Textos

Multiline



- Basta utilizar o texto normalmente dentro da propriedade **Text**.
- É possível utilizar **código unicode** dentro da string.
- Label possui uma propriedade **LineBreakMode**.
- Para quebras de linhas, recomenda-se utilizar **Environment.NewLine**.

```
Content = new Label
{
    VerticalOptions = LayoutOptions.Center,
    Text =
        "Mr. Sherlock Holmes, who was usually very late in " +
        "the mornings, save upon those not infrequent " +
        "occasions when he was up all night, was seated at " +
        "the breakfast table. I stood upon the hearth-rug " +
        "and picked up the stick which our visitor had left " +
        "behind him the night before. It was a long, thick " +
        "piece of wood, bulbous-headed, of the sort which " +
        "is known as a \"Penang lawyer.\" Just " +
        "under the head was a broad silver band, nearly an " +
        "inch across, \"To James Mortimer, M.R.C.S., " +
        "from his friends of the C.C.H.,\" was engraved " +
        "upon it, with the date \"1884.\" It was " +
        "just such a stick as the old-fashioned family " +
        "practitioner used to carry—dignified, solid, " +
        "and reassuring."
};
```

Cores



- É possível propriedades como **TextColor** e **BackgroundColor** através da estrutura **Color**.
- **Color** possui **17 campos estáticos** que definem cores comuns:

Color Fields	Color	Red	Green	Blue	Hue	Saturation	Luminosity
White		255	255	255	0	0	1.00
Silver		192	192	192	0	0	0.75
Gray		128	128	128	0	0	0.50
Black		0	0	0	0	0	0
Red		255	0	0	1.00	1	0.50
Maroon		128	0	0	1.00	1	0.25
Yellow		255	255	0	0.17	1	0.50
Olive		128	128	0	0.17	1	0.25
Lime		0	255	0	0.33	1	0.50
Green		0	128	0	0.33	1	0.25
Aqua		0	255	255	0.50	1	0.50
Teal		0	128	128	0.50	1	0.25
Blue		0	0	255	0.67	1	0.50
Navy		0	0	128	0.67	1	0.25
Pink		255	102	255	0.83	1	0.70
Fuchsia		255	0	255	0.83	1	0.50
Purple		128	0	128	0.83	1	0.25

```
Content = new Label
{
    Text = "Usando cores no Xamarin",
    VerticalTextAlignment = TextAlignment.Center,
    BackgroundColor = Color.Yellow,
    TextColor = Color.Blue
};
```

Cores



- Existe diversas formas para instanciar uma cor.

```
new Color(double grayShade)
new Color(double r, double g, double b)
new Color(double r, double g, double b, double a)
```

```
Color.FromRgb(double r, double g, double b)
Color.FromRgb(int r, int g, int b)
Color.FromRgba(double r, double g, double b, double a)
Color.FromRgba(int r, int g, int b, int a)
Color.FromHsla(double h, double s, double l, double a)
```

```
Color.FromRgb(1, 0, 0)
```



```
Color.Default
Color.Accent
```


Tamanhos e Atributos



- Os atributos de texto que podemos alterar são:
 - **FontFamily**
 - **FontSize**
 - Use o método **Device.GetNamedSize()** para obter o tamanho correspondente.
 - **FontAttributes** (None, Bold, Italic)

```
Content = new Label
{
    Text = "Tamanhos e atributos",
    HorizontalOptions = LayoutOptions.Center,
    VerticalOptions = LayoutOptions.Center,
    FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Label)),
    FontAttributes = FontAttributes.Bold | FontAttributes.Italic
};
```

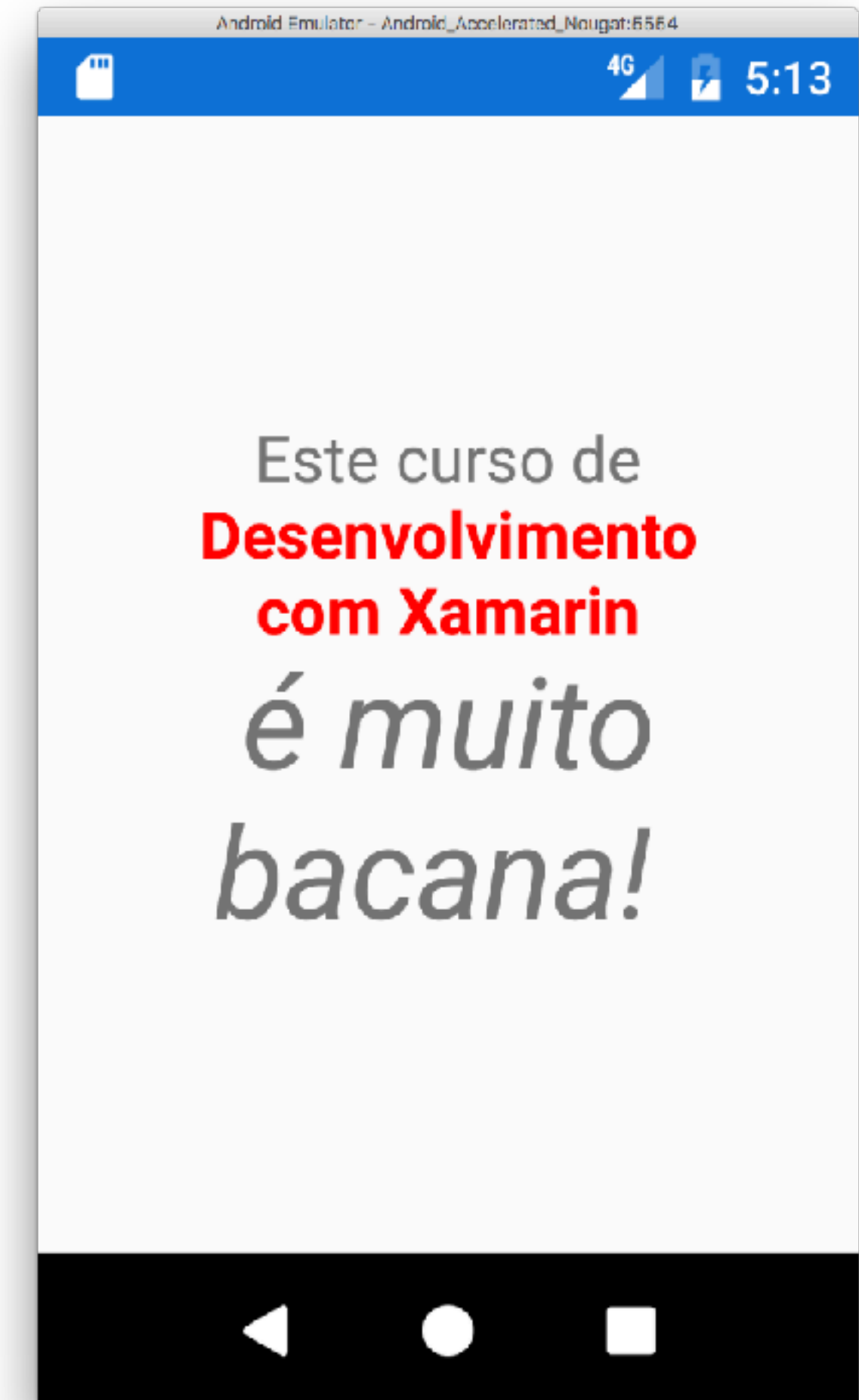
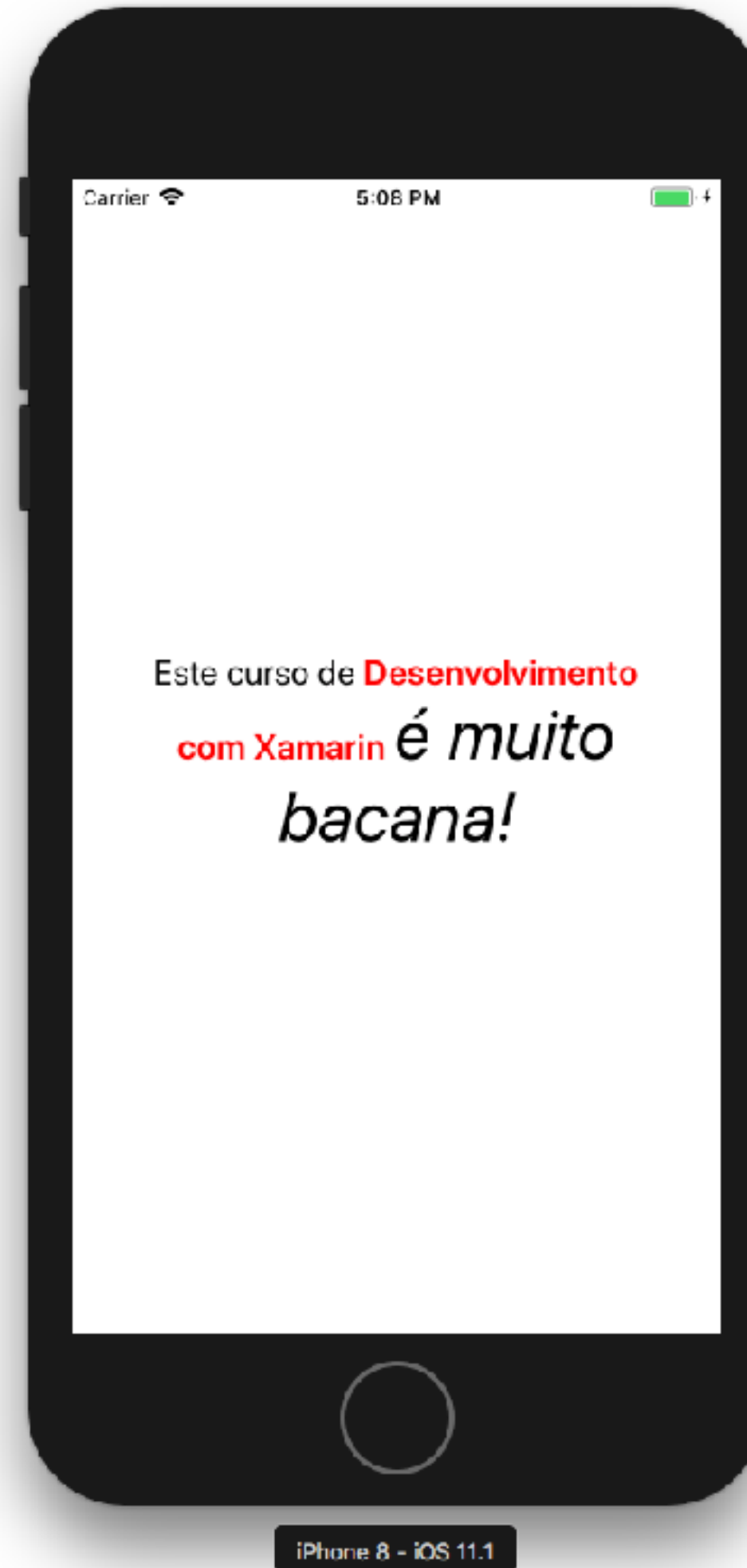


Recebe mais de um valor!

Exercício



- Reproduza a tela ao lado utilizando apenas um **único elemento Label**.
- Dica: Utilize a propriedade **FormattedText**.





Layouts

Tipos de Layout



- Uma página no Xamarin deriva da classe **ContentPage**, que define uma propriedade **Content**. Porém este **Content** recebe apenas um único componente.
- Para que sua página possua mais elementos, é necessário utilizar uma classe que derive da classe **Layout<T>** (abstrata):
 - **AbsoluteLayout**
 - **Grid**
 - **RelativeLayout**
 - **StackLayout**

StackLayout

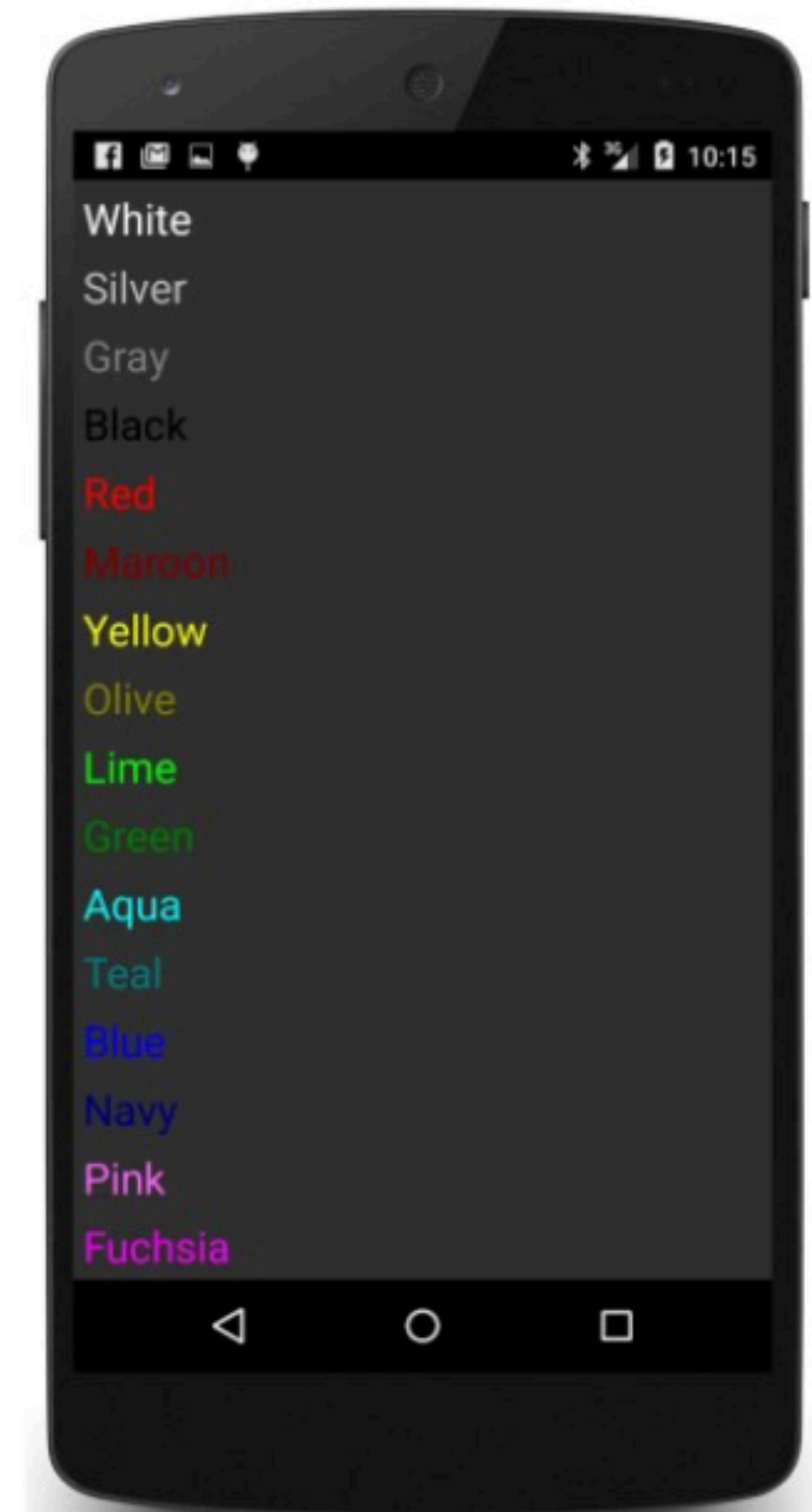
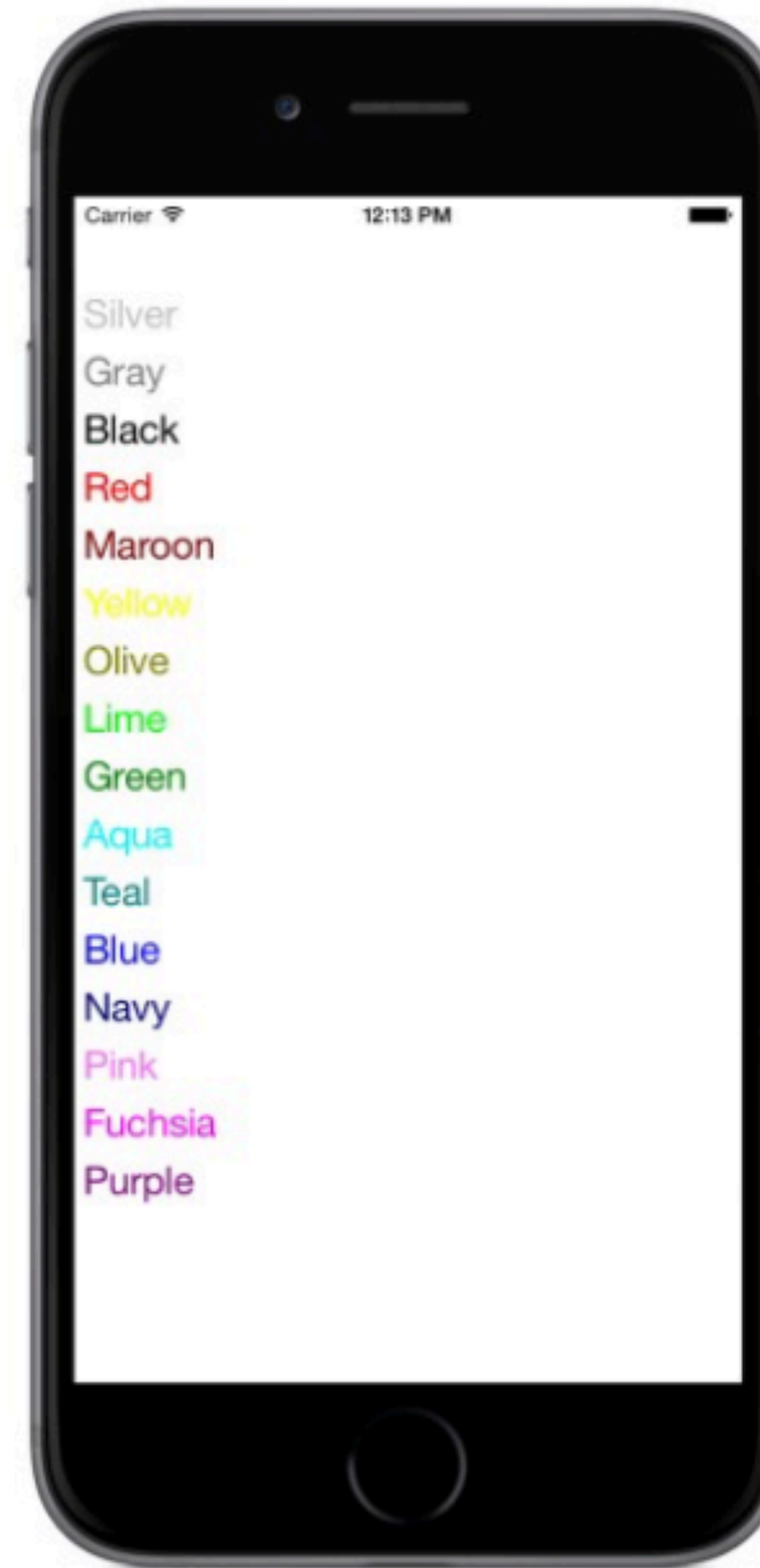


- Permite “empilhar” as views horizontalmente ou verticalmente.
- Propriedades:
 - Children: `ICollection<T>`. Coleção de views.
 - Orientation: `StackOrientation.Vertical` ou `StackOrientation.Horizontal`. Orientação de como as views serão empilhadas.
 - Spacing: `Double`. Espaço entre os filhos (padrão é 6).

Exercício



- Crie uma aplicação que exiba a lista com as 17 cores disponíveis na estrutura **Color**.
- A aplicação deve **permitir scroll** caso a lista seja maior que o tamanho da tela disponível.
- Dica 1: Utilize o elemento **ScrollView**.
- Dica 2: Utilize o método **.GetRuntimeFields()** e **.GetRuntimeProperties()**.



Expands



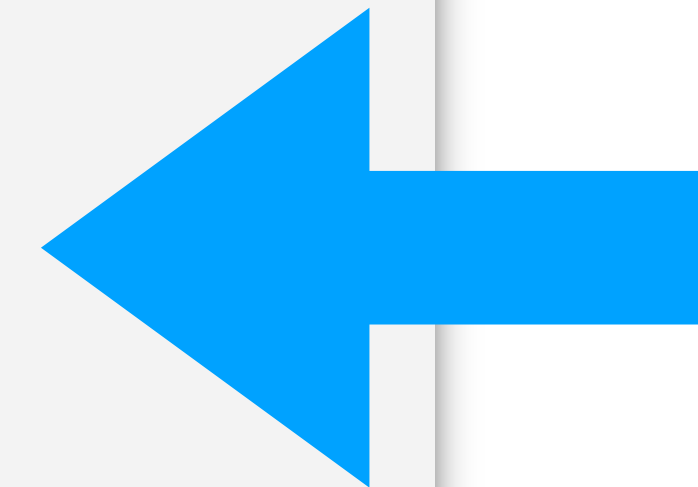
- Elemento se expande (verticalmente) de acordo com o espaço do container disponível.
- É um dos tipos do **VerticalOptions**.
- Funciona apenas dentro do **StackLayout**.
- Para fazer efeito, é necessário:
 - A altura total de todos os elementos do **StackLayout** precisa ser menor que a altura do **StackLayout**.
 - A opção **VerticalOptions** do **StackLayout** não pode ser **Start, Center ou End**, pois isso faria ele ser adaptável ao conteúdo.
 - Pelo menos um dos elementos do **StackLayout** precisa ter **VerticalOptions** utilizando o **Expands**.

Expands



```
LayoutOptions.Start  
LayoutOptions.Center  
LayoutOptions.End  
LayoutOptions.Fill
```

```
LayoutOptions.StartAndExpand  
LayoutOptions.CenterAndExpand  
LayoutOptions.EndAndExpand  
LayoutOptions.FillAndExpand
```



Frame e BoxView



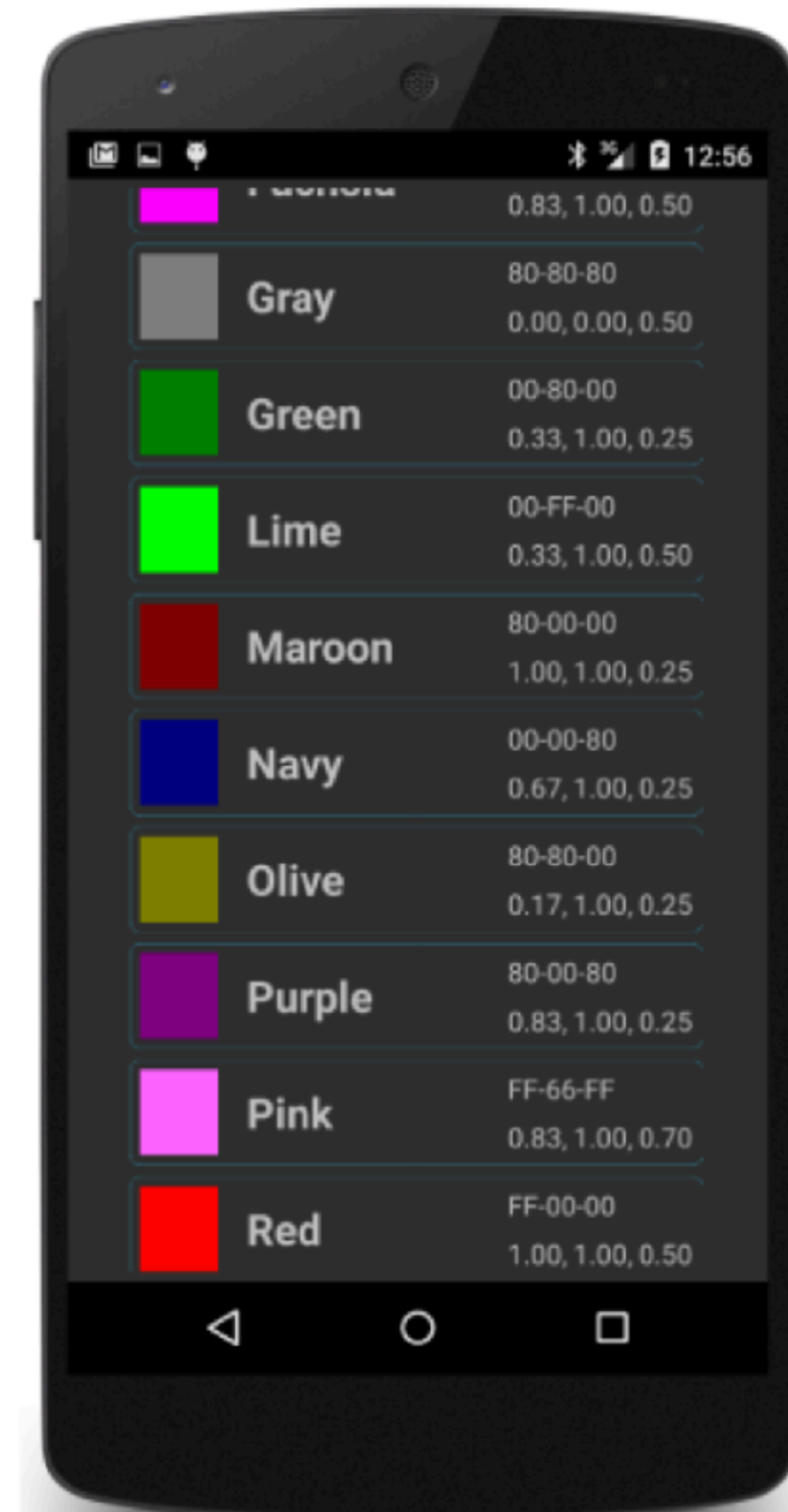
- Retângulos simples.
- **BoxView:**
 - Retângulo preenchido.
 - Possui apenas a propriedade **Color** (padrão é transparente).
- **Frame:**
 - Borda retangular ao redor de outro elemento.
 - Propriedades:
 - Content (deriva da classe ContentView).
 - Padding (padrão é 20).
 - BackgroundColor (padrão é branco no iOS e transparente no Android e Windows)
 - HasShadow (padrão é true).
 - OutlineColor (padrão é transparente e não é visível no iOS)

Para definir tamanhos fixos:

- **WidthRequest**
- **HeightRequest**

Exercício

- Modifique a aplicação do exercício anterior para que exiba a lista de cores utilizando os elementos **Frame** e **BoxView**.





Eventos

Regra Geral



- Os métodos que são utilizados para “assinar” os eventos são chamados de **Event Handler** e são “do tipo” **EventHandler**.
- O tipo **EventHandler** possui 2 argumentos:
 - **sender**: Objeto que disparou o evento.
 - **e**: Informação de contexto do evento.
- Nomenclatura:
 - On + objeto + evento que ocorreu (nominalizado)

SizeChanged += OnInitialPageSizeChanged;

```
void OnInitialPageSizeChanged(object sender, EventArgs e)
{
    label.Text = String.Format("{0} \u00D7 {1}", Width, Height);
}
```

Button

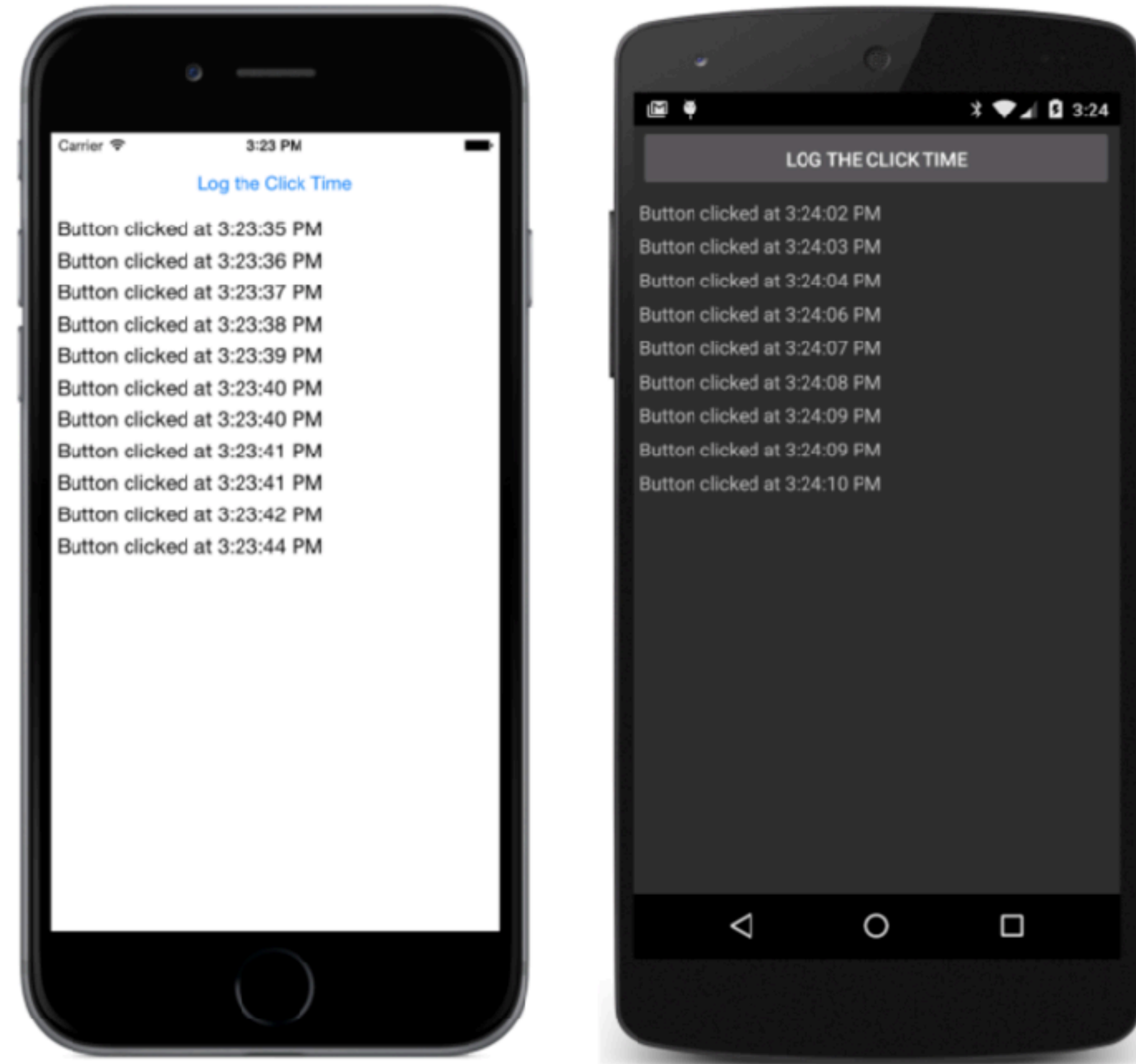


- Um dos principais elementos de interação nas aplicações.
- Permite ser apenas textual ou com imagens.
- Assim como os outros elementos, é renderizado nativamente em cada plataforma (UIButton, Button, etc...)
- Propriedades interessantes:
 - FontFamily (string)
 - FontSize (double)
 - FontAttributes (FontAttributes)
 - TextColor (Color)
 - BorderColor (Color)
 - BorderWidth (double)
 - BorderRadius (double)
 - Image (veremos mais tarde)

Button



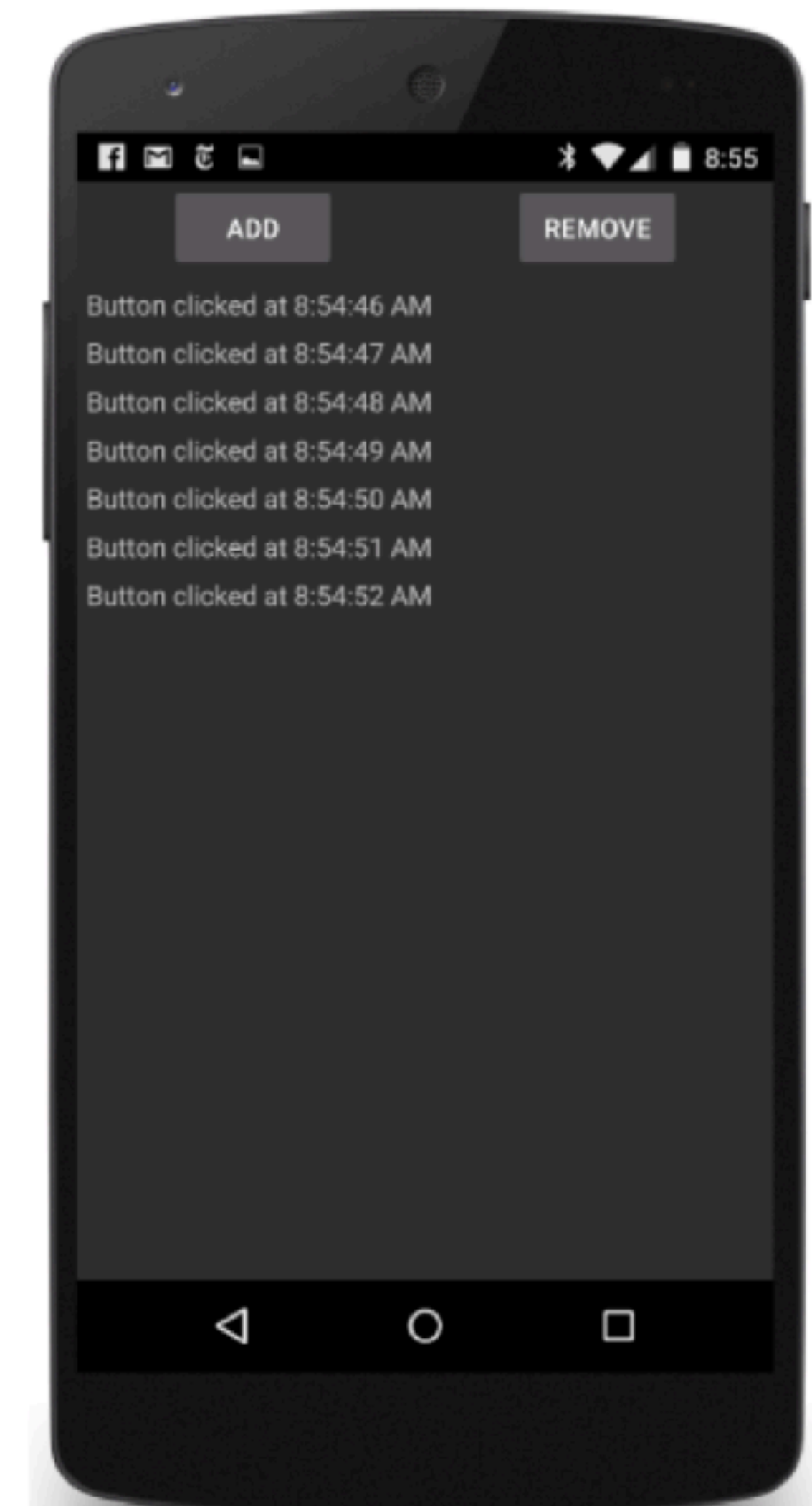
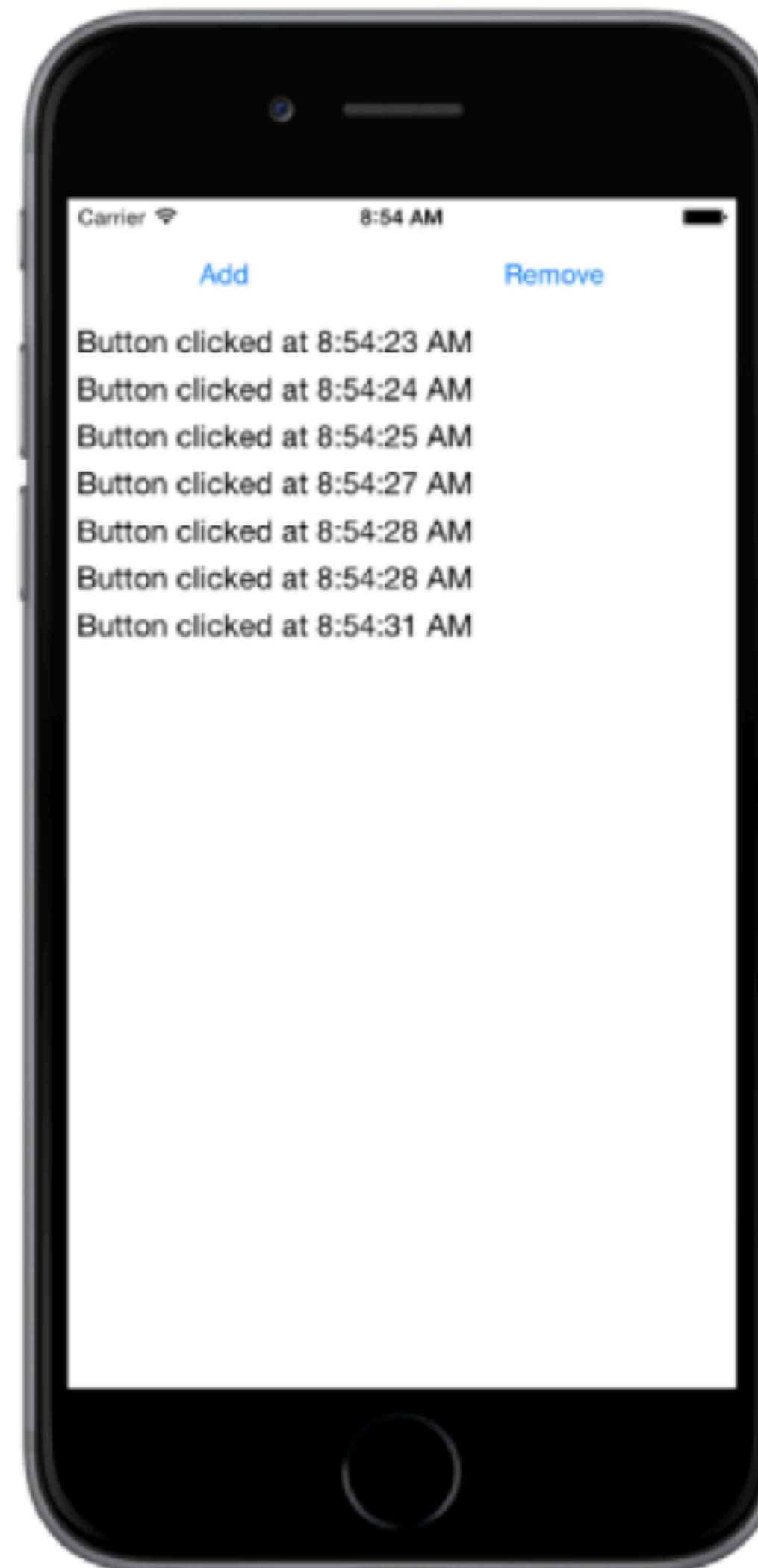
- Lembrete:
 - Os elementos se comportam ligeiramente diferente em cada plataforma!
 - Utilizar o **OnPlatform/****RuntimePlatform** se necessário.



Exercício



- Crie uma aplicação que possui 2 botões:
 - **Add:** adiciona o log assim como no exemplo anterior
 - **Remove:** Remove a primeira entrada da lista de logs
- Apenas um **EventHandler** deve ser criado
- O botão Remove deve estar **desabilitado** caso não haja nada para remover.



Lambda Functions



- Muitas vezes é útil definir um método/função dentro do escopo de outro método ou bloco de código.
- Recurso também conhecido como **anonymous event handlers**.
- Vantagem: Variáveis do método pai são acessíveis localmente.
- Desvantagem: Não poder ser compartilhado com outros objetos.

```
double number = 1;

Button timesButton = new Button
{
    Text = "Double",
};

timesButton.Clicked += (sender, args) =>
{
    number *= 2;
    label.Text = number.ToString();
};
```

```
Button.Clicked += (object sender, EventArgs e) => {};
```

StyleId



- Permite identificar elementos através de um "id".
- Tipo **string**.
- Xamarin.Forms não utiliza esta propriedade internamente.

```
Button saveButton = new Button
{
    Text = "Salvar",
    StyleId = "btSalvar"
};
```

Exercício

- Utilizando a propriedade **StyleId**, crie um aplicativo de teclado número, onde o clique em cada botão exibe o número que está sendo digitado.
- Deve possuir uma tecla "BackSpace"
 - Código: `\u21E6`
 - Deve utilizar apenas um **EventHandler** para os botões numéricos.



Persistindo Dados



- É importante manter o estado das telas sempre que o usuário sair e retornar da aplicação.
- Podemos utilizar a propriedade **Application.Properties**, um tipo de dicionário que recebe uma **string** como chave é um **object** como valor.
 - Permite salvar pequenos objetos para recuperar posteriormente.
 - Para acessar fora da classe **Application**, utilizamos a propriedade **Current**.
- É possível combinar o uso com os eventos da class **App**:
 - **OnStart, OnSleep, OnResume**.

Application.Properties



```
Application.Current.Properties["displayLabelText"] = displayLabel.Text;
```

```
IDictionary<string, object> properties = Application.Current.Properties;  
  
if (properties.ContainsKey("displayLabelText"))  
{  
    displayLabel.Text = properties["displayLabelText"] as string;  
    backspaceButton.IsEnabled = displayLabel.Text.Length > 0;  
}
```


OnStart, OnResume, OnSleep



- **OnStart:**
 - Disparado quando a aplicação inicia, **logo após a primeira página ser aberta.**
- **OnResume:**
 - Disparado quando a aplicação volta a ser ativa.
- **OnSleep:**
 - Disparado quando a aplicação deixa de ser ativa.
 - **Obs:** A aplicação pode ser fechado após este estado!
- **Ordem dos eventos:**
 - 1º: Construtor da App.
 - 2º: Construtor da primeira página.
 - 3º: OnStart

Exercício

- Modifique o último exercício para que tudo o que for digitado seja armazenado para quando o usuário retornar a aplicação.
- Utilize o método **OnSleep** e **Properties**.





XAML

XAML



- Extensible Application Markup Language (vem do XML).
- Linguagem declarativa para **instanciar e inicializar** objetos.
- Base do WPF (Windows Presentation Foundation), Silverlight, Windows Phone 7 e 8, Windows 8 e 10.
- Permite separação entre lógica da aplicação e design de telas.
- Possui **parsers** que permitem deixar o código mais direto e limpo.
- É fortemente tipada.
- Tudo feito em XAML pode ser feito com C#.

XAML vs Code



```
new Label
{
    Text = "Hello from Code!",
    IsVisible = true,
    Opacity = 0.75,
    HorizontalTextAlignment = TextAlignment.Center,
    VerticalOptions = LayoutOptions.CenterAndExpand,
    TextColor = Color.Blue,
    BackgroundColor = Color.FromRgb(255, 128, 128),
    FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Label)),
    FontAttributes = FontAttributes.Bold | FontAttributes.Italic
};
```



```
<Label Text="Hello from XAML!"
        IsVisible="True"
        Opacity="0.75"
        HorizontalTextAlignment="Center"
        VerticalOptions="CenterAndExpand"
        TextColor="Blue"
        BackgroundColor="#FF8080"
        FontSize="Large"
        FontAttributes="Bold,Italic" />
```



Classe precisa:

- Ter um construtor sem parâmetros
- Propriedades precisam ser públicas (set)

Type Converters



Enum.Parse

```
<Label Text="Hello from XAML!"
  IsVisible="True"
  Opacity="0.75"
  HorizontalTextAlignment="Center"
  VerticalOptions="CenterAndExpand"
  TextColor="Blue"
  BackgroundColor="#FF8080"
  FontSize="Large"
  FontAttributes="Bold,Italic" />
```

```
[TypeConverter(typeof(LayoutOptionsConverter))]
public struct LayoutOptions
{
    ...
}
```

```
[TypeConverter(typeof(ColorTypeConverter))]
public struct Color
{
    ...
}
```

```
[Flags]
public enum FontAttributes
{
    None = 0,
    Bold = 1,
    Italic = 2
}
```


Sintaxe de Elemento



- Em alguns casos não será possível utilizar a **sintaxe de atributo**.
- A sintaxe de elemento permite atribuir elementos inteiros e complexos em propriedades.
- Sintaxe:
 - **<TIPO DO ELEMENTO>.<NOME DA PROPRIEDADE>**
- XML continua sendo válido.
- Qualquer propriedade pode ser utilizada com a **sintaxe de elemento**.
 - Não é possível utilizar em conjunto com a sintaxe de atributo.

```
new Frame
{
    OutlineColor = Color.Accent,
    HorizontalOptions = LayoutOptions.Center,
    VerticalOptions = LayoutOptions.Center,
    Content = new Label
    {
        Text = "Greetings, Xamarin.Forms!"
    }
};
```

```
<Frame OutlineColor="Accent"
      HorizontalOptions="Center"
      VerticalOptions="Center"
      Content=" e agora josé? " />
```

```
<Frame OutlineColor="Accent"
      HorizontalOptions="Center"
      VerticalOptions="Center">
    <Frame.Content>
        <Label Text="Greetings, Xamarin.Forms!" />
    </Frame.Content>
</Frame>
```

Sintaxe de Elemento



```
<Frame HorizontalOptions="Center">
  <Frame.VerticalOptions>
    Center
  </Frame.VerticalOptions>
  <Frame.OutlineColor>
    Accent
  </Frame.OutlineColor>
  <Frame.Content>
    <Label>
      <Label.Text>
        Greetings, Xamarin.Forms!
      </Label.Text>
    </Label>
  </Frame.Content>
</Frame>
```

Tudo pode estar em sintaxe de elemento!

USE COM MODERAÇÃO

Sintaxe de Coleções

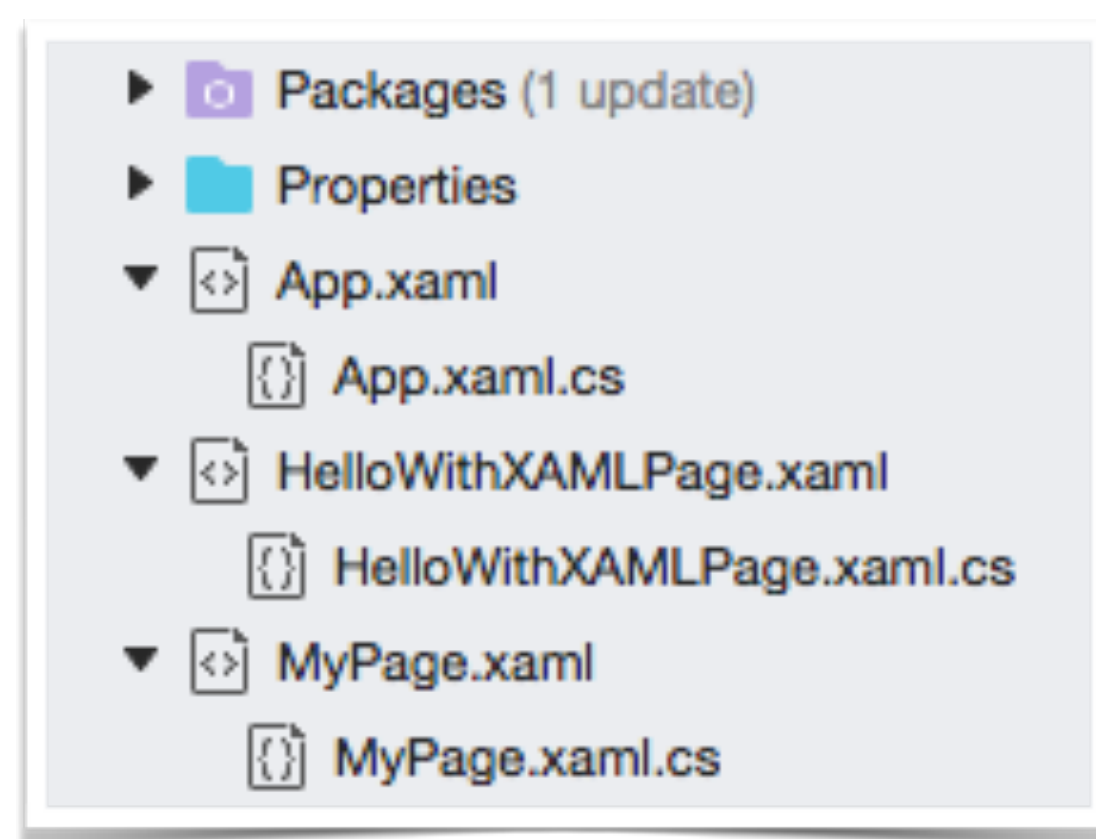


```
<StackLayout>
  <StackLayout.Children>
    <StackLayout Orientation="Horizontal">
      <StackLayout.Children>
        <BoxView Color="Red" />
        <Label Text="Red"
              VerticalOptions="Center" />
      </StackLayout.Children>
    </StackLayout>
    <StackLayout Orientation="Horizontal">
      <StackLayout.Children>
        <BoxView Color="Green" />
        <Label Text="Green"
              VerticalOptions="Center" />
      </StackLayout.Children>
    </StackLayout>
  </StackLayout.Children>
</StackLayout>
```

Code behind



- Ao criar um arquivo **.xaml**, um outro arquivo **.cs** também é gerado.
 - Este arquivo contém o código C# que dará suporte ao XAML.
- Os dois arquivos trabalham em conjunto para definição e **comportamento da tela**.



```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
              xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
              x:Class="HelloWithXAML.MyPage">
    <ContentPage.Content>
    </ContentPage.Content>
</ContentPage>
```

namespaces



code behind

Code behind



MyPage.xaml.g.cs

```
//-----  
// <auto-generated>  
// This code was generated by a tool.  
// Runtime Version:4.0.30319.42000  
//  
// Changes to this file may cause incorrect behavior and will be lost if  
// the code is regenerated.  
// </auto-generated>  
//-----  
  
namespace HelloXAML {  
    public partial class MyPage : global::Xamarin.Forms.ContentPage {  
        private void InitializeComponent() {  
            global::Xamarin.Forms.Xaml.Extensions.LoadFromXaml(this, typeof(MyPage));  
        }  
    }  
}
```

MyPage.xaml.cs

```
using Xamarin.Forms;  
  
namespace HelloXAML  
{  
    public partial class MyPage : ContentPage  
    {  
        public Mypage()  
        {  
            InitializeComponent();  
        }  
    }  
}
```

Se desejar adicionar elementos em tempo de execução, melhor colocar após este método!

OnPlatform



- Permite definir valores de propriedades específicas para cada plataforma.
- É tipado de acordo com a propriedade que se deseja utilizar.

Sintaxe de elemento

```
<ContentPage.Padding>  
  <OnPlatform x:TypeArguments="Thickness"  
    iOS="0, 20, 0, 0"  
    Android="0"  
    WinPhone="0" />  
</ContentPage.Padding>
```

namespace x

Content Property



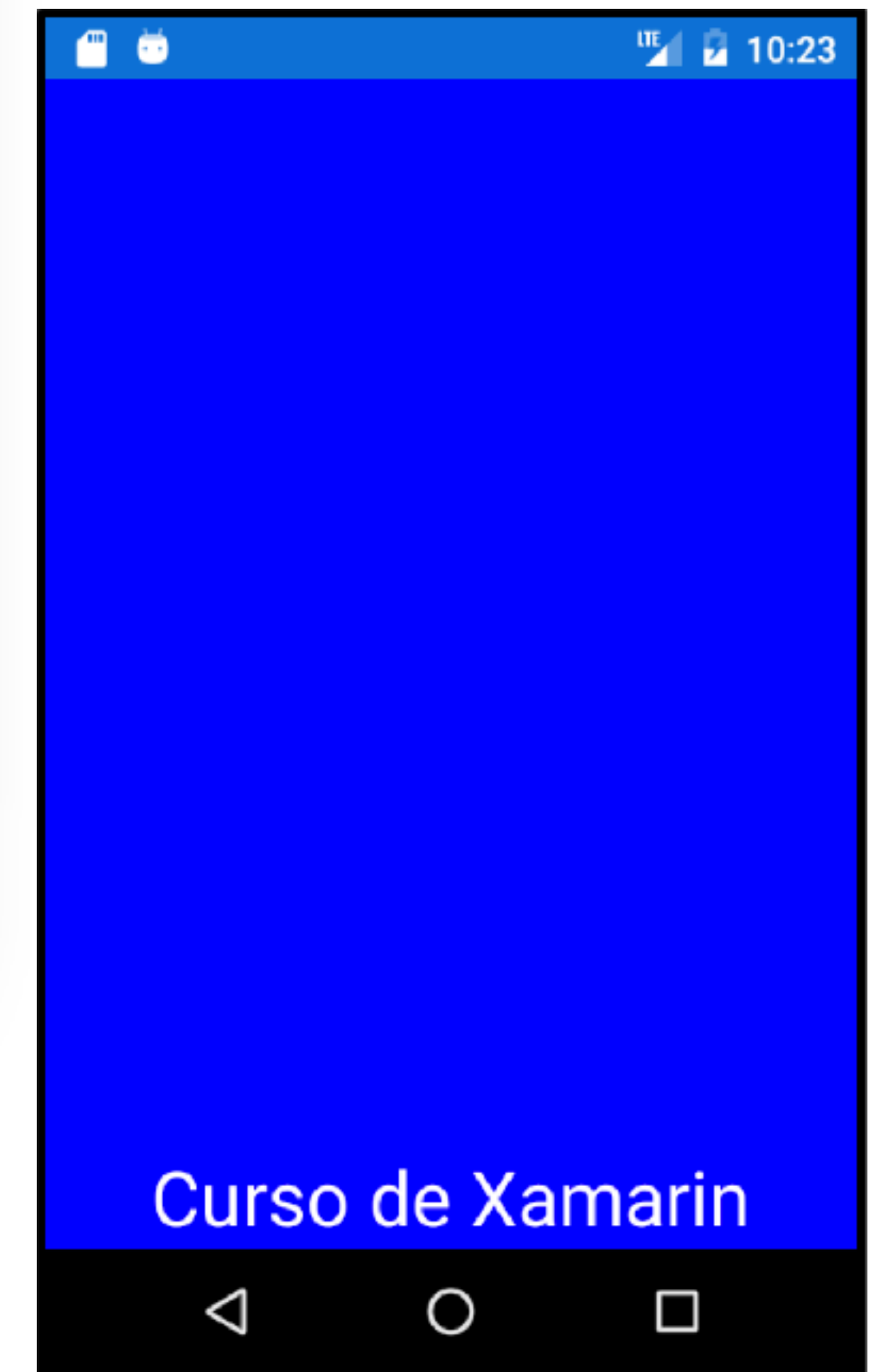
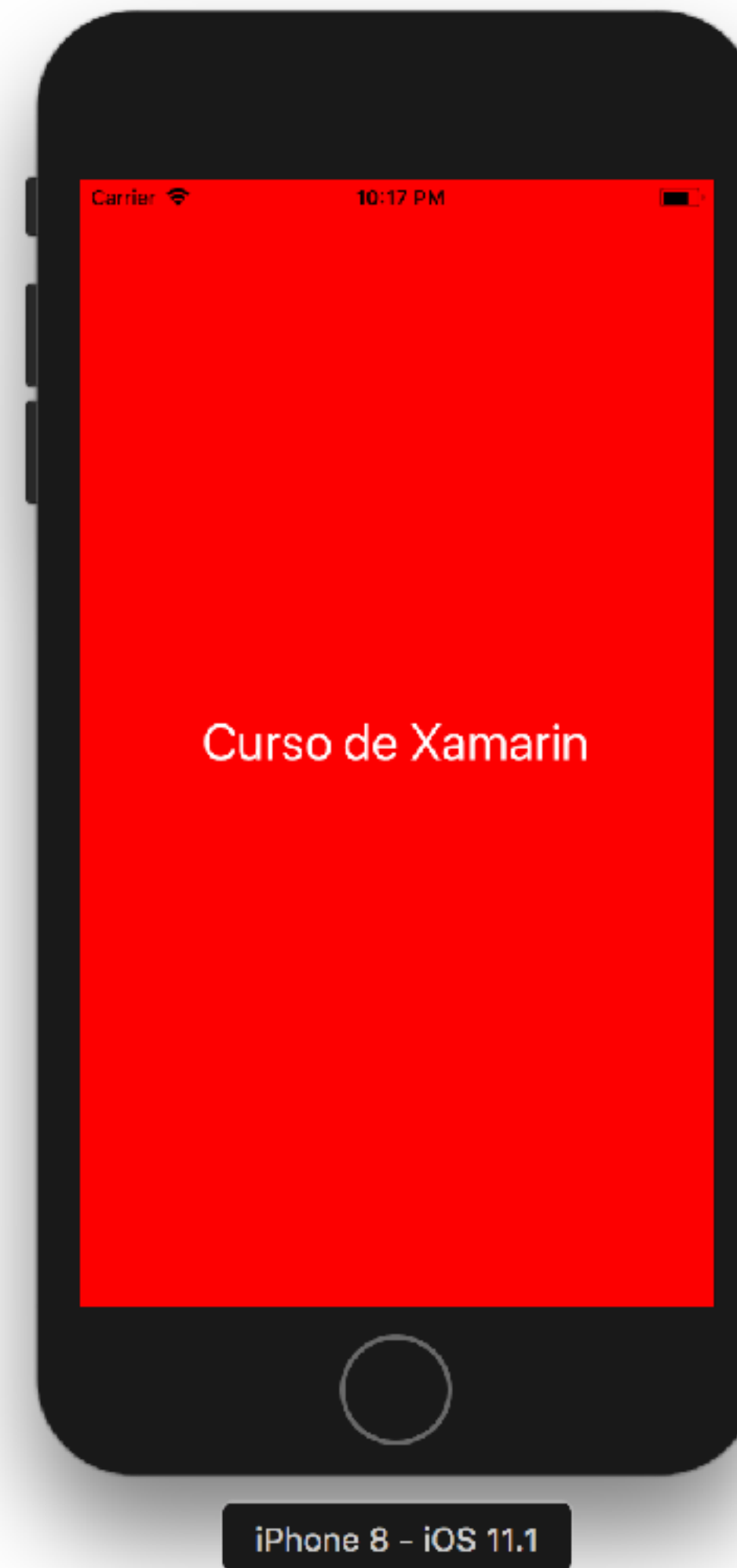
- Propriedade do elemento que pode ser eliminada no XAML.
- Também chamada de **default property** da classe.
- Exemplos:
 - ContentPage: **Content**
 - StackLayout: **Children** (vem da classe Layout<T>)
 - Frame: **Content**

```
[Xamarin.Forms.ContentProperty("Content")]  
public class ContentPage : TemplatedPage
```

```
[Xamarin.Forms.ContentProperty("Children")]  
public abstract class Layout<T> : Layout,  
IViewContainer<T> where T : View
```

Exercício

- Crie uma aplicação utilizando apenas o XAML, onde a cor de fundo para o **iOS é vermelha**, e para **Android é azul**.
- A aplicação deve conter um label com o texto “Curso de Xamarin”, que no **iOS é centralizado na tela**, e no **Android fica centralizado porém na parte de baixo da tela**.





XAML + Code

XAML + Code



- Na Xamarin, XAML e código C# sempre funcionam em pares.
- É necessário saber acessar elementos do XAML dentro do código C#.
 - E também como acessar recursos do C# no XAML.
- Exemplo:
 - Como instanciar classes no XAML mesmo que essas classes só possuam construtores com argumentos?
 - Como instanciar classes no XAML utilizando métodos estáticos das classes que criam objetos (factories)?

Passando Argumentos



- Lembram dos namespaces utilizados no XAML?

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
              xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
              x:Class="HelloWithXAML.MyPage">
    <ContentPage.Content>
    </ContentPage.Content>
</ContentPage>
```

- O **namespace x** declara alguns atributos importantes para uso de argumentos no XAML:
 - **x:Arguments**
 - **x:FactoryMethod**

Tipos Básicos



- O namespace `x` também declara os tipos básicos do .NET Framework para serem utilizados no XAML:
 - `x:Object`
 - `x:Boolean`
 - `x:Byte`
 - `x:Int16`
 - `x:Int32`
 - `x:Int64`
 - `x:Single`
 - `x:Double`
 - `x:Decimal`
 - `x:Char`
 - `x:String`
 - `x:TimeSpan`
 - `x:Array`
 - `x:DateTime*`

x:Arguments



- Os argumentos precisam estar dentro da tag `<x:Arguments></x:Arguments>`.
- Cada argumento precisa ser uma **tag diferente**.
- A **ordem e tipos** dos argumentos precisam ser a mesma da assinatura do construtor ou método utilizado.

```
<ContentPage.BackgroundColor>
  <Color>
    <x:Arguments>
      <x:Double>1</x:Double>
      <x:Double>0</x:Double>
      <x:Double>0</x:Double>
    </x:Arguments>
  </Color>
</ContentPage.BackgroundColor>
```


```
<ContentPage.BackgroundColor>
  <Color>
    <x:Arguments>
      <x:Double>0.5</x:Double>
    </x:Arguments>
  </Color>
</ContentPage.BackgroundColor>
```

x:FactoryMethod




- É possível fazer chamadas a métodos no XAML deste que eles sejam:
 - Estáticos e públicos.
 - Retornem o tipo do objeto que se deseja instanciar no XAML.
 - São geralmente conhecidos como **Factory Methods** ou **Creation Methods**.
- O **x:FactoryMethod** é utilizado como atributo da classe que possui o método, e geralmente é utilizado em conjunto com o **x:Arguments**.

```
Color bg = Color.FromRgb(255, 255, 0);
```




```
<BoxView.Color>  
  <Color x:FactoryMethod="FromRgb">  
    <x:Arguments>  
      <x:Int32>255</x:Int32>  
      <x:Int32>255</x:Int32>  
      <x:Int32>0</x:Int32>  
    </x:Arguments>  
  </Color>  
</BoxView.Color>
```



x:Name



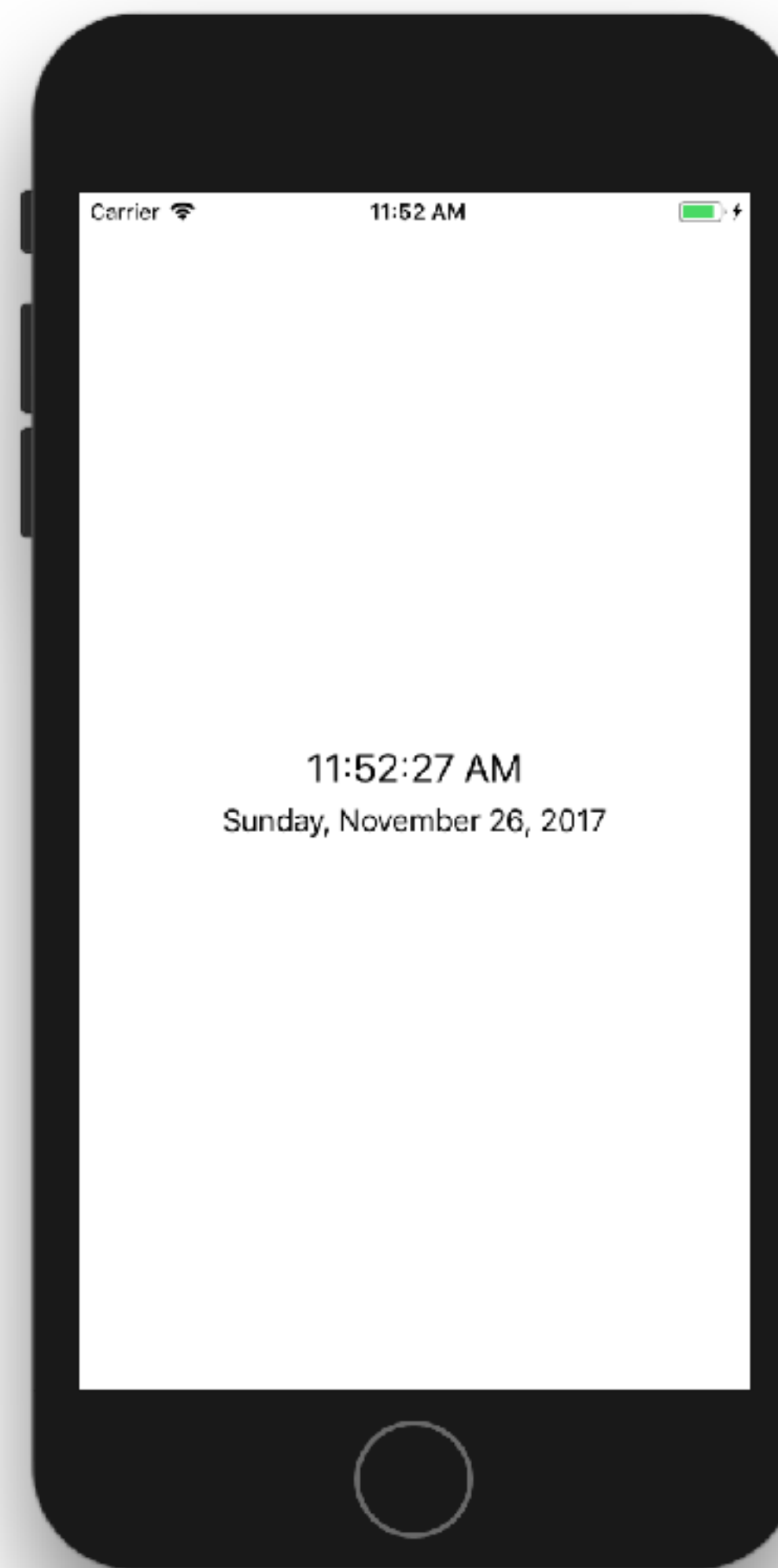
- Atributo importante que define o nome da variável para o elemento no C#.
- Ou seja, faz com que o elemento seja acessível mais facilmente dentro do código C#.
- Também implica que o nome escolhido precisa seguir as regras de nomenclatura de variáveis do C#.
- Precisa ser único, começar com letra ou **underscore** e deve conter apenas letras, números e **underscores**.
- Também funciona sem o prefixo x.



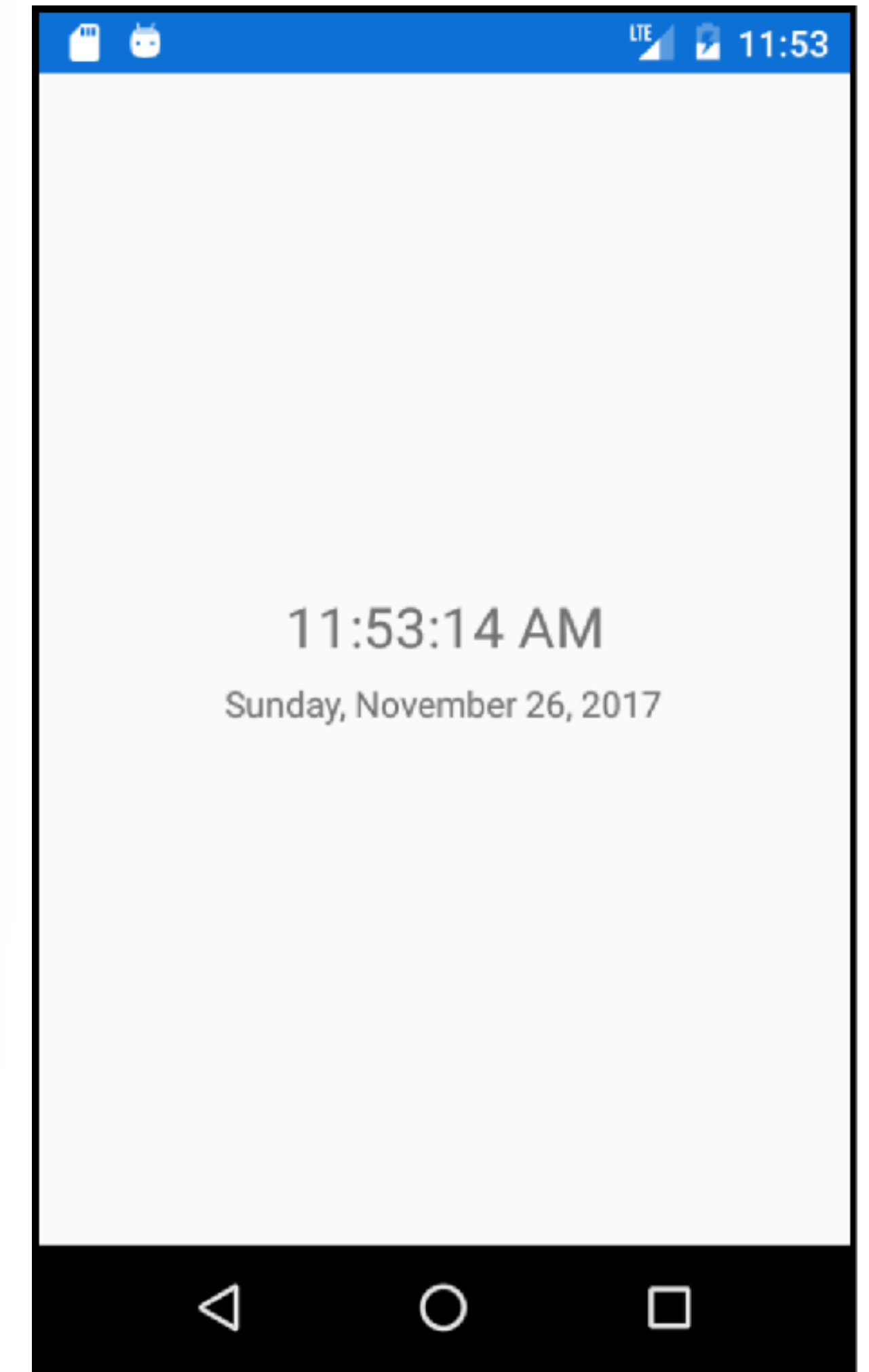
```
<Label x:Name="timeLabel"  
    Text="Usando o atributo Name"  
    FontSize="Large"  
    HorizontalOptions="Center"  
    VerticalOptions="Center" />
```

Exercício

- Utilizando XAML e C#, crie uma aplicação que exibe um relógio digital (atualizado a cada 1 segundo) e que também exibe a data atual no formato exibido ao lado:
- Deve possuir um botão para iniciar e parar o relógio.



iPhone 8 - iOS 11.1



Ainda Sobre o x:Name



- Observe o arquivo gerado automaticamente da página:




```
[global::System.CodeDom.Compiler.GeneratedCodeAttribute("Xamarin.Forms.Build.Tasks.XamlG", "0.0.0.0")]
private global::Xamarin.Forms.Label timeLabel;

[global::System.CodeDom.Compiler.GeneratedCodeAttribute("Xamarin.Forms.Build.Tasks.XamlG", "0.0.0.0")]
private global::Xamarin.Forms.Label dateLabel;


[global::System.CodeDom.Compiler.GeneratedCodeAttribute("Xamarin.Forms.Build.Tasks.XamlG", "0.0.0.0")]
private void InitializeComponent() {
    global::Xamarin.Forms.Xaml.Extensions.LoadFromXaml(this, typeof(XamlClockPage));
    timeLabel = global::Xamarin.Forms.NameScopeExtensions.FindByName<global::Xamarin.Forms.Label>(this, "timeLabel");
    dateLabel = global::Xamarin.Forms.NameScopeExtensions.FindByName<global::Xamarin.Forms.Label>(this, "dateLabel");
}
```

- As variáveis são declaradas, mas somente após a chamada do **InitializeComponent** elas recebem os valores

Cuidado com o OnPlatform



```
<OnPlatform x:TypeArguments="View">
  <OnPlatform.iOS>
    <Label x:Name="deviceLabel"
      Text="This is an iOS device"
      HorizontalOptions="Center"
      VerticalOptions="Center" />
  </OnPlatform.iOS>
  <OnPlatform.Android>
    <Label x:Name="deviceLabel"
      Text="This is an Android device"
      HorizontalOptions="Center"
      VerticalOptions="Center" />
  </OnPlatform.Android>
</OnPlatform>
```



```
<Label x:Name="deviceLabel"
  HorizontalOptions="Center"
  VerticalOptions="Center">
  <Label.Text>
    <OnPlatform x:TypeArguments="x:String"
      iOS="This is an iOS device"
      Android="This is an Android device" />
  </Label.Text>
</Label>
```


Custom XAML



- É possível criar seus próprios componentes personalizados com XAML + código C#.
- Existe uma classe específica em XAML que serve justamente como **container** para outros controles e componentes, a **ContentView**.
- Existe um template no Visual Studio (Mac) e no Xamarin Studio para criar este tipo de classe.
 - Botão direito no projeto > Add > New File... > Forms > **Forms ContentView Xaml**.

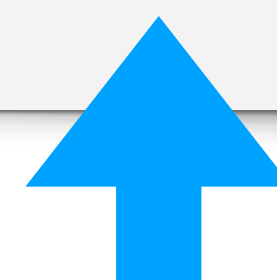
```
<?xml version="1.0" encoding="UTF-8"?>
<ContentView xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class="ColorViewList.ColorView">
</ContentView>
```

Custom XAML



- As propriedades que o componente irá possuir devem ser definidas no code-behind.
- Para utilizar o componente customizado criado, é necessário importar o namespace no XAML:

```
xmlns:local="clr-namespace:ColorViewList;assembly=ColorViewList"
```



Opcional se estiver no mesmo assembly

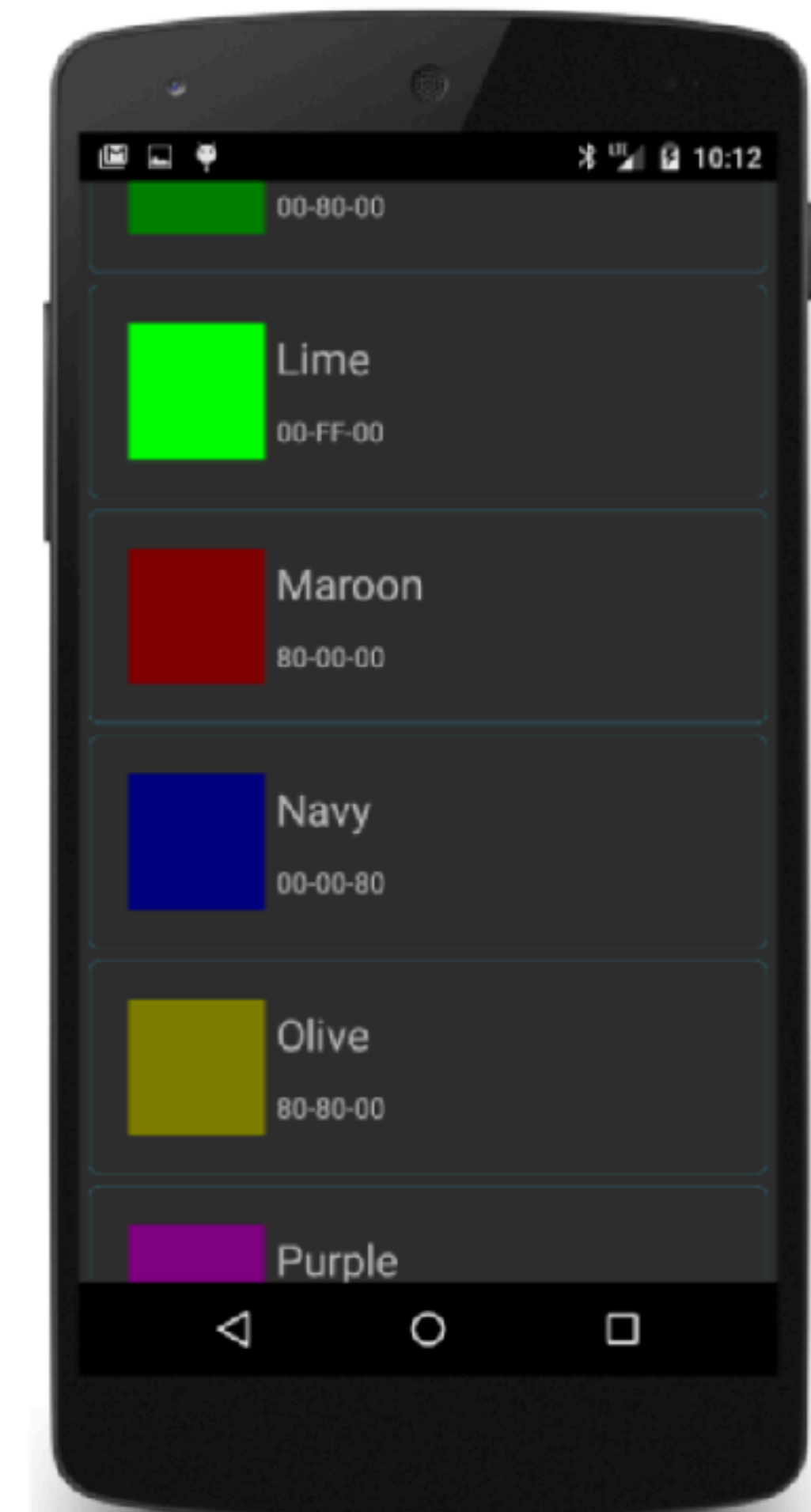
Exercício



- Crie um componente customizado para exibir cores seguindo o template abaixo.
- Crie uma aplicação que utilize este novo componente diretamente no XAML.



Template




Eventos



- Para assinar eventos dos elementos no XAML, devemos utilizar a mesma forma que utilizamos com propriedades (sintaxe de atributo).
- O valor que passamos é o nome do método que será disparado quando o evento acontecer.
 - Este método precisa existir no code-behind.
 - Este método precisa ter a assinatura requerida pelo evento (**EventHandler**).

XAML



```
<Button x:Name="backspaceButton"  
    Text="&#x21E6;"  
    Font="Large"  
    IsEnabled="False"  
    Clicked="OnBackspaceButtonClicked" />
```

```
void OnBackspaceButtonClicked(object sender, EventArgs e)  
{  
    ...  
}
```

C#

Gestos de TAP



C#

- Para escutar os eventos de TAP que podem acontecer em algum elemento, podemos utilizar a classe **TapGestureRecognizer**.
 - Define uma propriedade **NumberOfTapsRequired** (padrão: 1).
- Todos os elementos possuem uma coleção (**GestureRecognizers**). É nela que devemos adicionar os **TapGestureRecognizer** que desejarmos.
- Para gerar eventos de Tap, os elementos precisam ter:
 - **IsEnabled=true.**
 - **IsVisible=true.**
 - **InputTransparent=false.**

```
BoxView boxView = new BoxView
{
    Color = Color.Blue
};

TapGestureRecognizer tapGesture = new TapGestureRecognizer
{
    NumberOfTapsRequired = 3
};
tapGesture.Tapped += TapGesture_Tapped;

boxView.GestureRecognizers.Add(tapGesture);
```

XAML

```
<BoxView Color="Blue">
    <BoxView.GestureRecognizers>
        <TapGestureRecognizer Tapped="OnBoxViewTapped"
            NumberOfTapsRequired="3" />
    </BoxView.GestureRecognizers>
</BoxView>
```