

Macoratti.net Xamarin Android - Criando sua primeira Aplicação Android : Activity (Conceitos)



Neste artigo vou mostrar como criar uma aplicação **Android** usando os recursos do **Xamarin** no Visual Studio 2015 abordando os principais conceitos relacionados e o conceito de Activity.

Em uma aplicação Android o ponto de entrada de uma aplicação deve ser uma classe que herda da classe **Activity** e sobrescreve o método **OnCreate()**. Este método seria o equivalente ao método **Main()** da classe **Program** em uma aplicação C#.

A classe **Activity** é responsável por gerenciar a interface do usuário em aplicações Android. (Cada tela em uma aplicação é representada por uma Activity)

Assim, uma **Activity** é um componente de aplicativo que fornece uma tela com a qual os usuários podem interagir para fazer algo, como discar um número no telefone, tirar uma foto, enviar um e-mail ou ver um mapa.

Cada atividade recebe uma janela que exibe a interface do usuário. Geralmente, a janela preenche a tela, mas pode ser menor que a tela e flutuar sobre outras janelas.

Quase todas as atividades interagem com o usuário, de modo que a classe **Activity** cuida de criar uma janela para você em que você pode colocar sua interface com **setContentView(View)**.

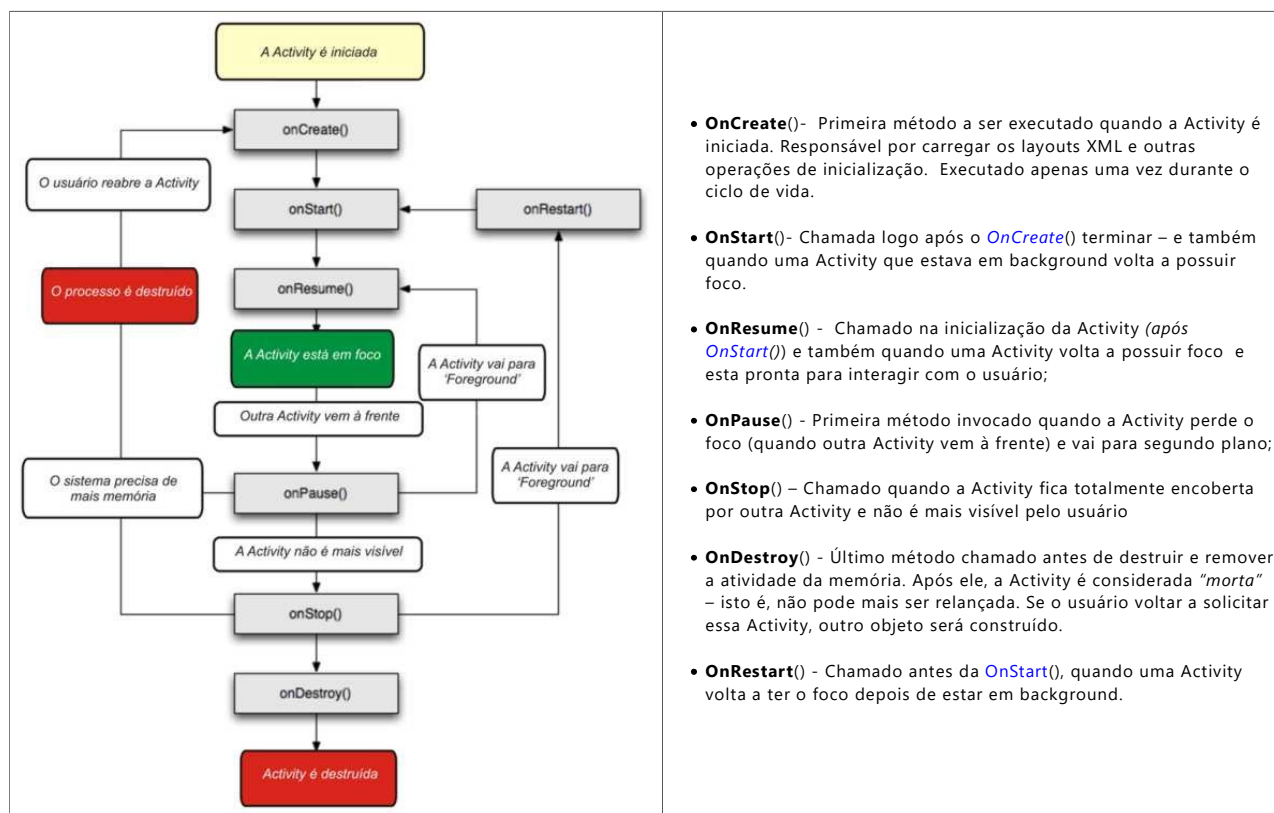
Embora as atividades sejam apresentadas para o usuário como janelas em tela cheia, eles também podem ser usadas em outras formas: como janelas flutuantes (via um tema com o conjunto **android.support.design.widget.WindowInsets**) ou incorporadas dentro de outra atividade (usando **ActivityGroup**).

Existem dois métodos que quase todas as subclasses de **Activity** devem implementar:

onCreate(Bundle) - é onde você inicia sua atividade. Aqui você vai normalmente chamar **setContentView(int)** com um recurso de layout para definir a sua interface, e a seguir usar **findViewById(int)** para recuperar os **widgets** na **IU** com a qual você precisa interagir via código.

onPause() - é onde você lida com o usuário quando ele deixa a sua atividade. Todas as mudanças feitas pelo usuário devem neste momento ser persistidas (geralmente para o **ContentProvider**)

Na verdade existem outros métodos e na figura abaixo vemos um diagrama que ilustra os **métodos** durante o ciclo de vida de uma **Activity**:



Dessa forma podemos concluir que :

- 1- O **tempo de vida completo de uma atividade** acontece entre a primeira chamada ao método **onCreate()** até a chamada do método **onDestroy()**;
- 2- A **visibilidade de uma atividade** ocorre entre a chamada do método **onStart()** até o correspondente **onStop()**;
- 3- O **tempo de vida do primeiro plano** de uma atividade ocorre entre a chamada do **onResume()** e o correspondente **onPause()**;

O ciclo de vida de uma atividade é implementado como um conjunto de métodos que o SO chama durante todo o ciclo de vida de uma atividade. Estes métodos permitem aos desenvolvedores implementar a funcionalidade necessária para satisfazer as exigências do estado e de gestão de recursos de

suas aplicações.

Muitos dispositivos Android têm dois botões distintos: o botão "**Voltar**" e o botão "**Home**".

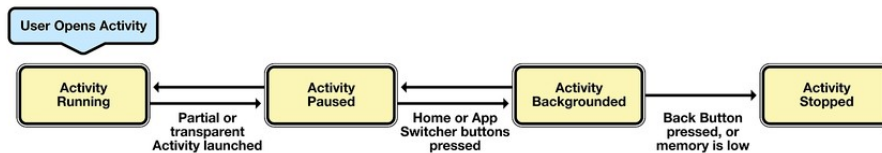
Há uma sutil diferença entre os dois botões, mesmo que eles pareçam ter o mesmo efeito de colocar um aplicativo em *segundo plano* (*background*).

1. **Voltar (Back)** - Quando um usuário clica no botão **Voltar**, ele está dizendo ao Android que a atividade foi concluída. O Android vai *destruir* a atividade.
2. **Home** - Quando o usuário clica no botão **Home** a atividade é simplesmente colocada em segundo plano, e, o Android não vai *matar* a atividade.

Os estados de uma Activity

O sistema operacional Android arbitra as atividades com base no seu respectivo estado. Isso ajuda o Android identificar as atividades que não estão mais em uso, permitindo que o sistema operacional recupere memória e recursos.

Abaixo temos o diagrama que ilustra os estados de uma Activity:



Dessa forma uma atividade pode possuir quatro estados:

1. **Active ou Running** - Se uma atividade está no primeiro plano (**foreground**) da tela (*no topo da pilha*), ela está ativa ou em execução;
2. **Paused** - Se uma atividade perdeu o foco, mas ainda está visível (*isto é, uma nova atividade, sem usar a tela cheia ou ser transparente, tem o foco sobre a atividade*), ele está em pausa. Uma atividade pausada está viva (*ela mantém todas as informações de estado e informação dos membros e permanece ligada ao gerenciador de janelas*), mas pode ser 'morta' pelo sistema em situações extremas de falta de memória;
3. **Stopped** - Se uma atividade está oculta por outra atividade, o seu estado é parado. Ela ainda mantém todas as informações de estado e membros, no entanto, ela não é mais visível para o usuário, e a sua janela está oculta e, muitas vezes, e poderá ser 'morta' pelo sistema quando a houver falta de memória;
4. **Restarted** - Se uma atividade é pausada ou parada, o sistema pode remover a atividade da memória, seja solicitando a ela que encerre, seja simplesmente 'matando' seu processo. Quando ela for exibida novamente para o usuário, deve ser completamente reiniciada e restaurada ao seu estado anterior;

Vamos criar nossa primeira aplicação **Android** e analisar em detalhes o seu código usando o [Visual Studio 2015](#) e o [Xamarin](#).

Recursos usados:

- [Visual Studio Community 2015](#)
- [Xamarin](#)
- Emulador Android virtual ou físico ([veja como emular usando o Vysor](#))

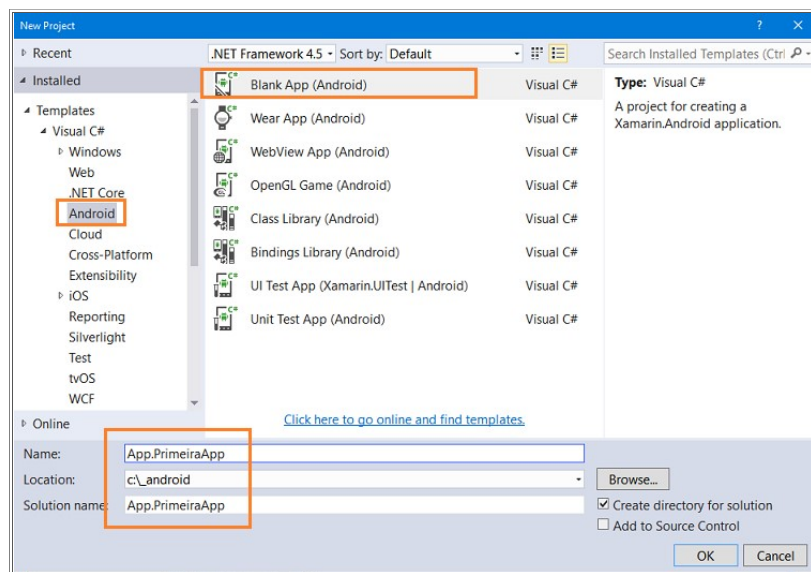
Nota: Baixe e use a versão **Community 2015** do VS ela é grátis e é equivalente a versão **Professional**.

Criando o projeto no VS Community com Xamarin

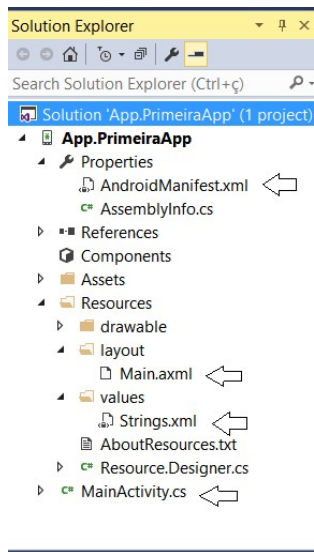
Abra o [VS Community 2015](#) e clique em **New Project**;

Selecione a linguagem Visual C# e o template **Android -> Blank App (Android)**;

Informe o nome **App.PrimeiraApp** e clique no botão OK:



Será criada uma solução com a seguinte estrutura:



- **Properties** - Contém o arquivo [AndroidManifest.xml](#) que descreve as funcionalidades e requisitos da sua aplicação Android, e o arquivo [AssemblyInfo.cs](#) contém informação sobre o projeto como número de versão e build.

- **References** - Contém as bibliotecas [Mono.Android](#), [System.Core](#) e todas as bibliotecas usadas no seu projeto;

- **Components** - Contém componentes de terceiros ou desenvolvidos por você usados no seu projeto.

A maioria dos componentes está disponíveis diretamente do **Xamarin Component Store** e são **free** (*não todos*) e prontos para serem usados; (Para incluir um componente clique com o botão direito sobre **Components** e a seguir em [Get More Components](#));

- **Assets e Resources** - Contém arquivos que não são código, como imagens, sons, arquivos XML e qualquer outro recurso que sua aplicação for usar. Os arquivos externos colocados na pasta **Assets** são facilmente acessíveis em tempo de execução através do [Asset Manager](#).

Já os arquivos colocados na pasta **Resources** precisam ser declarados e mantidos em uma lista com os **IDs** dos recursos que você deseja usar em tempo de execução.

De forma geral, todas as imagens, ícones, sons e outros arquivos externos são colocados na pasta **Resources** enquanto que dicionários e arquivos XML são postos na pasta **Assets**;

Na subpasta **layout** temos os arquivos **.xml** que definem as **views** usadas no projeto;

Na subpasta **values** temos o arquivo **Strings.xml** onde definimos as strings usadas no projeto;

Nota : A pasta **Drawable** contém recursos como imagens png, jpg, etc., usadas no aplicativo. Ela contém múltiplas pastas específicas para cada resolução possível em uma aplicação Android. Numa aplicação típica Android você vai acabar encontrando as pastas: [Drawable-LDPI](#), [Drawable-mdpi](#), [Drawable-hdpi](#), [Drawable-xhdpi](#), [Drawable-xxhdpi](#), etc.

O arquivo **MainActivity.cs**, como o nome já sugere, é onde esta definida a **Activity** da aplicação Android.

Abrendo o arquivo **MainActivity.cs** veremos o seguinte código:

```
using Android.App;
using Android.OS;
using Android.Widget;

namespace App.PrimeiraApp
{
    [Activity(Label = "App.PrimeiraApp", MainLauncher = true, Icon = "@drawable/icon")]
    public class MainActivity : Activity
    {
        int count = 1;

        protected override void OnCreate(Bundle bundle)
        {
            base.OnCreate(bundle);

            // Set our view from the "main" layout resource
            SetContentView(Resource.Layout.Main);

            // Get our button from the layout resource,
            // and attach an event to it
            Button button = FindViewById<Button>(Resource.Id.MyButton);

            button.Click += delegate { button.Text = string.Format("{0} clicks!", count++); };
        }
    }
}
```

Este é um código na linguagem C#. Vamos entendê-lo :

1- As declarações dos namespaces usados no projeto

```
using Android.App;
using Android.OS;
using Android.Widget;
```

2- O namespace : **namespace App.PrimeiraApp**

Cada aplicação Android possui um namespace e um arquivo manifest : **AndroidManifest.xml** definido em **Properties**.

Cada aplicação tem que ter um arquivo **AndroidManifest.xml** que tem as seguintes funções :

- fornecer o nome para o pacote que serve como identificador único da aplicação.
- Declarar quais permissões a aplicação precisa ter afim de acessar partes protegidas da API e interagir com outras aplicações
- Declarar o nível mínimo e máximo da API Android que a aplicação precisa para funcionar
- Declara as bibliotecas que a aplicação precisa usar

3- **[Activity(Label = "App.PrimeiraApp", MainLauncher = true, Icon = "@drawable/icon")]**

- Declara o **título da barra** que vai aparecer na aplicação (**App.PrimeiraApp**)
- Determina que a atividade deve aparecer na tela do aplicativo (**MainLauncher= true**) (*Por padrão a Activity não aparece na tela do App*)
- Define o arquivo **icon** presente na pasta **drawable** como ícone da App;

Em tempo de compilação os **assemblies** são vasculhados a procura de classes que herdem da classe **Activity** e possuam o atributo **[Activity]** declarado.

O Compilado então usa essas classes e os atributos para construir o **manifest** gerando o arquivo **AndroidManifest.xml**.

4- A seguir declaramos uma classe chamada **MainActivity** que herda da classe **Activity** e define a atividade principal da App Android.

5- É definida uma variável **count** do tipo `int` : `int count = 1`

6- O método **OnCreate** é definido:

O método **OnCreate(Bundle bundle)** é o primeiro método a ser executado quando uma **Activity** é iniciada, sendo executado somente uma vez durante o ciclo de vida da Activity.

Ele é a responsável por carregar os layouts XML e outras operações de inicialização e gerenciar o estado da aplicação.

Este método leva um parâmetro **bundle** do tipo **Bundle** que é um dicionário para armazenar e transmitir informações do estado e objetos entre as atividades. Se **bundle** não for nulo, isso indica que a atividade está reiniciando e deve restaurar o estado da instância anterior.

7- **SetContentView(Resource.Layout.Main);**

O arquivo `Resource.designer.cs` é um arquivo C# na pasta `Resources` que é gerado pelo `Xamarin.Android` e contém definições de ID para todos os recursos na App.

O método **SetContentView** define o conteúdo da atividade (**activity**) para uma **view** explícita.

No exemplo, estamos carregando a interface do usuário (a *view*) definida no arquivo **Main.axml** presente na pasta **Resources/layout**.

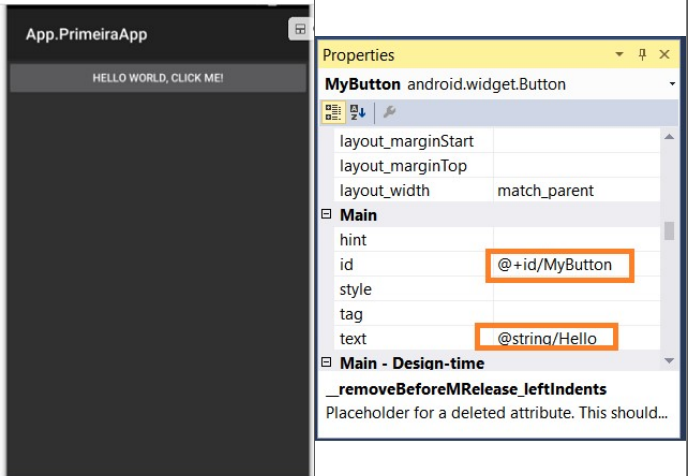
Abaixo podemos ver o código do arquivo **Main.axml**, a view que será renderizada e exibida ao usuário e a janela de propriedades do **Button** exibindo os valores das propriedades **id** e **text**;

Nota: o arquivo .axml é na verdade um arquivo XML

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <Button
        android:id="@+id/MyButton"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/Hello" />
</LinearLayout>
```

wrap_content - informa ao [componente/view](#) que ele deve ocupar apenas o espaço necessário na tela. A view fica com o seu tamanho natural.

match_parent - informa ao [componente/view](#) para preencher automaticamente todo o conteúdo de seu layout pai.



The screenshot shows the Android Studio IDE. On the left, the `Main.axml` file is open, displaying the XML code for the layout. On the right, the `Properties` window is open, showing the properties for the `MyButton` widget. The `id` property is set to `@+id/MyButton` and the `text` property is set to `@string/Hello`. The `Main` layout is also visible in the background.

Observe no código a definição da **string** chamada **Hello** que pode ser vista no arquivo **Strings.xml** que esta na pasta **Resources/values** :

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="Hello">Hello World, Click me!</string>
    <string name="ApplicationName">App.PrimeiraApp</string>
</resources>
```

O texto definido na string será exibido no botão.

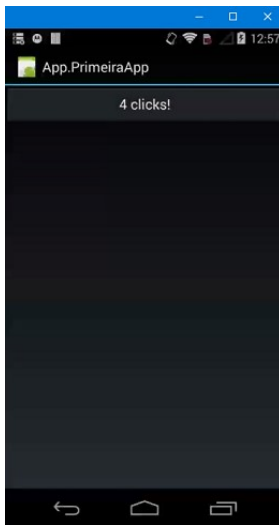
8 - `Button button = FindViewById<Button>(Resource.Id.MyButton);`

Definição de um objeto do tipo **Button** chamado **button** localizado pelo método **FindViewById** pelo Id **MyButton** a partir do arquivo XML que foi processado no método **OnCreate**.

9 - `button.Click += delegate { button.Text = string.Format("{0} clicks!", count++); };`

Define o evento **Click** do botão **button** usando um *delegate* e altera sua propriedade **Text** toda vez que o botão for clicado exibindo a mensagem : **<número> clicks !**

Executando a aplicação e dando alguns cliques no botão iremos obter:

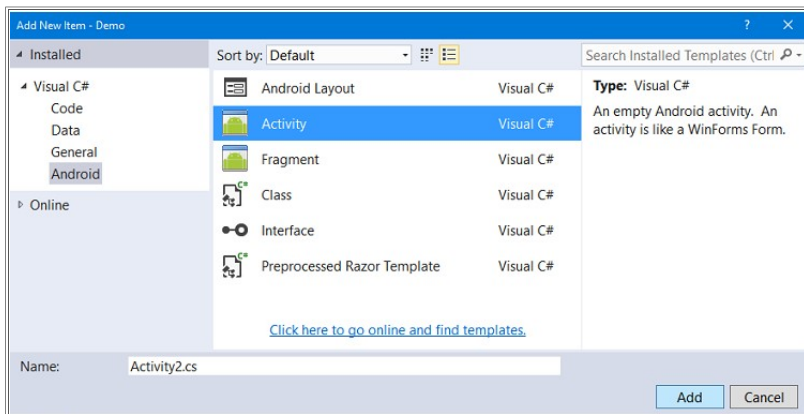


Incluindo uma nova Activity no projeto

Vamos agora criar uma nova Activity em nossa aplicação a invocá-la usando o método **StartActivity** passando o tipo de atividade a iniciar.

No menu **Project** clique em **Add New Item**;

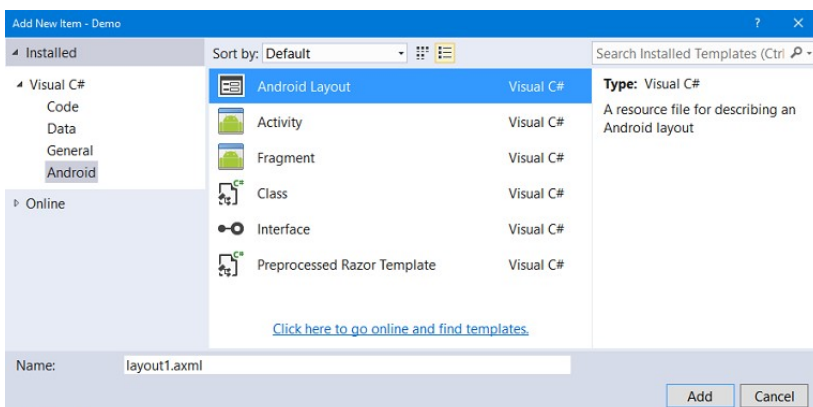
A seguir selecione o template **Android** e **Activity** informando o nome **Activity2.cs** e clique no botão **Add**;



Será criado o arquivo **Activity2.cs** no projeto. Vamos agora criar um novo arquivo para representar a view que vamos carregar na segunda atividade.

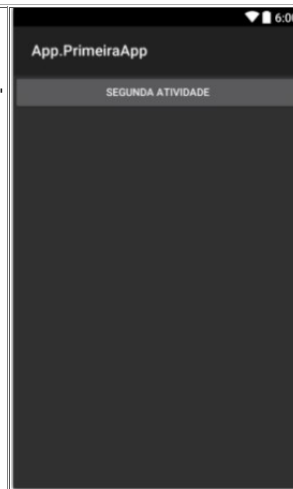
Clique com o botão direito sobre a pasta layout e a seguir em **Add -> New Item**;

Selecione o template **Android -> Android Layout** e informe o nome **layout1.axml** e clique no botão **Add**;



A seguir abra este arquivo e defina o código conforme abaixo:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:minWidth="25px"
    android:minHeight="25px">
    <Button
        android:text="Segunda Atividade"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/macButton" />
</LinearLayout>
```



Observe que definimos o texto fixo e um Button identificado por **macButton**.

Agora abra o arquivo **Activity2.cs** e inclua o código abaixo:

```
using Android.App;
using Android.OS;
using Android.Widget;

namespace App.PrimeiraApp
{
    [Activity(Label = "Segunda Atividade")]
    public class Activity2 : Activity
    {
        protected override void OnCreate(Bundle savedInstanceState)
        {
            base.OnCreate(savedInstanceState);
            // Create your application here
            SetContentView(Resource.Layout.layout1);

            Button button = FindViewById<Button>(Resource.Id.macButton);

            button.Click += delegate
            {
                StartActivity(typeof(MainActivity));
            };
        }
    }
}
```

Neste código estamos carregando o arquivo **layout1.axml** que representa a view da nossa atividade e definimos no evento **Click** do botão **macButton** o código para acionar a primeira atividade - **MainActivity** usando o método **StartActivity()**.

Finalmente abra o arquivo **MainActivity.cs** e inclua o código no evento **Click** do botão de comando que vai chamar a segunda **Activity**: **StartActivity(typeof(Activity2));**

```
using Android.App;
using Android.OS;
using Android.Widget;

namespace App.PrimeiraApp
{
    [Activity(Label = "App.PrimeiraApp", MainLauncher = true, Icon = "@drawable/icon")]
    public class MainActivity : Activity
    {
        int count = 1;

        protected override void OnCreate(Bundle bundle)
        {
            base.OnCreate(bundle);

            // Set our view from the "main" layout resource
            SetContentView(Resource.Layout.Main);

            // Get our button from the layout resource,
            // and attach an event to it
            Button button = FindViewById<Button>(Resource.Id.MyButton);

            button.Click += delegate
            {

```

```

        button.Text = string.Format("{0} clique!", count++);
        StartActivity(typeof(Activity2));
    };
}
}
}

```

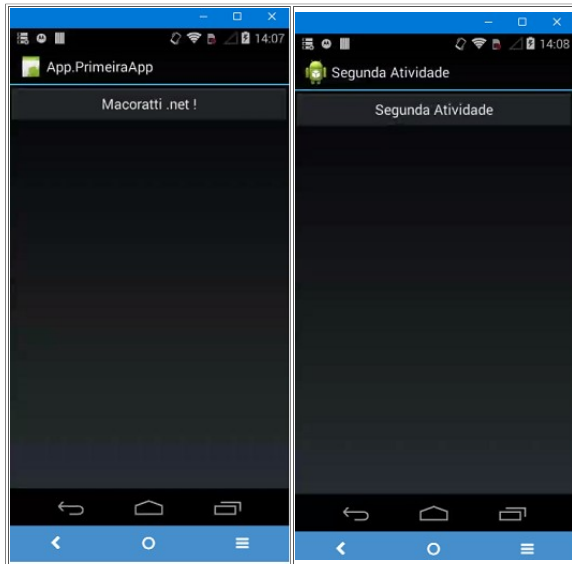
Agora temos duas atividades que usam dois layouts distintos:

- **MainActivity - Main.axml**
- **Activity2 - layout1.axml**

Executando o projeto e emulando no meu celular **Motorola** (veja o artigo : [Xamarin - Espelhando o dispositivo físico no Windows 10 como Vysor](#))

Temos o resultado abaixo:

- 1- Na primeira atividade (**MainActivity**) ao clicar no botão de comando **MyButton** chamamos a segunda atividade que exibe a view **layout1.axml** ;
- 2- Na segunda atividade (**Activity2**) ao clicar no botão **MacButton** voltamos para a primeira atividade que exibe a view **Main.axml**;



E assim, vimos alguns conceitos importantes das atividades nas aplicações Android criadas no **Visual Studio com Xamarin**: como ciclo de vida, seus eventos e estados.

É extremamente importante que o desenvolvedor analise os requisitos de cada atividade para determinar quais os métodos expostos pelo ciclo de vida da atividade precisam ser implementados.

Não fazer isso pode resultar em instabilidade no aplicativo, falhas, desperdício de recursos e instabilidade no SO.

Na [segunda parte do artigo](#) veremos como passar informações entre atividades distintas usando o conceito de **Intent**.

Pegue o projeto completo aqui : [📁 App.PrimeiraApp.zip](#)

Porque a palavra da cruz é loucura para os que perecem; mas para nós, que somos salvos, é o poder de Deus.

1 Coríntios 1:18

[Veja os Destaques e novidades do SUPER DVD Visual Basic \(sempre atualizado\) : clique e confira !](#)

Quer migrar para o VB .NET ?

- Veja mais sistemas completos para a plataforma .NET no [Super DVD .NET](#) , confira...
- [Curso Básico VB .NET - Vídeo Aulas](#)

Quer aprender C# ??

- Chegou o [Super DVD C#](#) com exclusivo material de suporte e vídeo aulas com curso básico sobre C#.
- [Curso C# Basico - Vídeo Aulas](#)

Quer aprender os conceitos da Programação Orientada a objetos ?

- [Curso Fundamentos da Programação Orientada a Objetos com VB .NET](#)

NEW

Quer aprender o gerar relatórios com o ReportViewer no VS 2013 ?

- [Curso - Gerando Relatórios com o ReportViewer no VS 2013 - Vídeo Aulas](#) NEW

Referências:

- [Seção VB .NET do Site Macoratti.net](#)
- [Super DVD .NET - A sua porta de entrada na plataforma .NET](#)
- [Super DVD Vídeo Aulas - Vídeo Aula sobre VB .NET, ASP .NET e C#](#)
- [Super DVD C# - Recursos de aprendizagens e vídeo aulas para C#](#)
- [Seção C# do site Macoratti.net](#)
- [Seção ASP .NET do site Macoratti .net](#)
- [Curso Básico VB .NET - Vídeo Aulas](#)
- [Curso C# Básico - Vídeo Aulas](#)
- [Curso Fundamentos da Programação Orientada a Objetos com VB .NET](#) NEW
- [Macoratti .net | Facebook](#)
- [macoratti - YouTube](#)
- [Jose C Macoratti \(@macorati\) | Twitter](#)
- [Xamarin - Desenvolvimento Multiplataforma com C# ... - Macoratti.net](#)
- [Xamarin - Apresentando Xamarin.Forms - Macoratti.net](#)
- [Xamarin.Forms - Olá Mundo - Criando sua primeira ... - Macoratti.net](#)
- [Xamarin.Forms - Olá Mundo - Anatomia da aplicação - Macoratti.net](#)

[José Carlos Macoratti](#)