05

# Margin e DisplayAlert

### Transcrição

Aprendemos a utilizar o `ListView` para implementar uma listagem em nossa aplicação. Porém, ainda temos alguns problemas no layout a serem resolvidos, como o texto que está "colando" nas margens esquerda e superior.

Queremos um layout mais agradável e com margens maiores, além de exibir `R$` antes dos valores. Para isto, pausaremos o programa, abriremos `MainPage.xaml` e, em `ListView`, definiremos uma margem de `25`.

```
<ListView x:Name="listViewVeiculos" ItemsSource="{Binding Veiculos}" Margin="25">
```

Rodaremos a aplicação novamente, e em seguida, verificaremos que as margens foram devidamente implementadas. Para incluirmos o símbolo de reais antes dos preços, modificaremos a classe `Veiculo` para que ela comporte uma propriedade pública chamada `PrecoFormatado`. Em `MainPage.xaml.cs`, teremos:

```
namespace TestDrive
{
    public class Veiculo
    {
        public string Nome { get; set; }
        public decimal Preco { get; set; }
        public string PrecoFormatado
        {
            get { return string.Format("R$ {0}", Preco); }
        }
    }
}
```

Além disto, precisamos informar ao XAML para que se utilize o preço formatado como origem do `Binding` no `Label`. Trocaremos `Preco` por `PrecoFormatado` em `MainPage.xaml`:

```
<Label Text="{Binding PrecoFormatado}" FontAttributes="Bold" VerticalTextAlignment="Center"></La
```

Feito isto, rodaremos a app, e veremos que tudo funciona conforme gostaríamos. Continuando, qual é a utilidade da lista na aplicação? Ela serve para o usuário selecionar o veículo a ser utilizado no *Test Drive*. Como esta seleção é feita? Usando o toque em um item.

Para implementarmos um evento que responda ao toque do usuário em um item da listagem, abriremos `MainPage.xaml`, incluindo o nome de um evento no `ListView` chamado `ItemTapped` (que em inglês significa "item que foi tocado"). Ao colocarmos as aspas e apertarmos "*Enter*", o Visual Studio criará um novo *handler* de evento que no *code behind* do C# será o nome de uma função a receber os parâmetros deste evento, tratando-o para fazer o que quisermos.

```
<ListView x:Name="listViewVeiculos" ItemsSource="{Binding Veiculos}" Margin="25" ItemTapped="lis
```

Se formos ao nosso *code behind* ( `MainPage.xaml.cs` ), aparecerá um método que receberá os parâmetros do item que foi tocado no `ListView` . Nestes parâmetros, temos `ItemTappedEventArgs` , cujo parâmetro `e` irá conter informações importantes utilizados para informar qual item foi tocado.

`e.Item` trará as informações para a renderização da listagem que se encontra no `ListView` . Acrescentaremos a declaração de uma variável, que chamaremos de `veiculo` :

```
private void listViewVeiculos_ItemTapped(object sender, ItemTappedEventArgs e)
{
    var veiculo = e.Item;
}
```

No entanto, `e.Item` é um `object` , um objeto genérico. Não queremos lidar com objetos, e sim com `veiculos` . Para a conversão de um objeto em `veiculos` , faremos um *cast*. A partir daí, é possível utilizarmos as propriedades do `veiculo` da forma como quisermos, alterando `Preco` , `PrecoFormatado` , e afins.

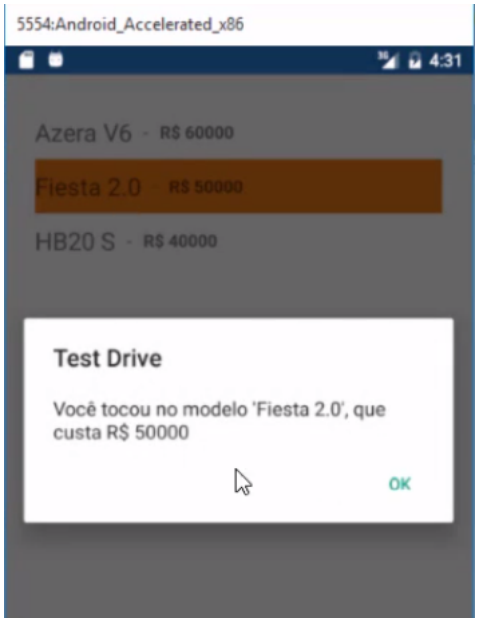
Para mostrar ao usuário por meio de uma caixa de mensagem que um item foi tocado, usaremos um alerta, com `DisplayAlert` , dentro do qual teremos alguns parâmetros, cujo primeiro será o título ("*Test Drive*", o nome da nossa aplicação), e o segundo será a mensagem propriamente dita, uma `string` .

Esta mensagem inclui o nome do modelo, seu preço em posição de índice `1` , e os parâmetros a serem passados a esta `string` : `veiculo.Nome` e `veiculo.PrecoFormatado` . Por fim, colocaremos o texto que irá no botão para fechar a mensagem ("ok").

```
private void listViewVeiculos_ItemTapped(object sender, ItemTappedEventArgs e)
{
    var veiculo = (Veiculo)e.Item;

    DisplayAlert("Test Drive", string.Format("Você tocou no modelo '{0}', que custa {1}", veicu
}
```

Vamos verificar o comportamento da aplicação rodando-a no emulador. Clicando no modelo "Fiesta 2.0", abre-se uma caixa de diálogo com a mensagem que acabamos de configurar.



Aprendemos a utilizar uma caixa de mensagem a ser exibida ao usuário, como incluirmos margem aos controles (neste caso em `ListView`), utilizando o `StackLayout` para distribuição dos controles na horizontal. Vimos também como utilizar corretamente o `Binding` no `ListView`, para trazer a origem de dados do *code behind*, exibindo-se a listagem.