06

Binding On, SwitchCell, TwoWay e OnPropertyChanged

Transcrição

As configurações definidas até aqui ainda não estão funcionando de forma completamente satisfatória, pois quando ativamos o freio ABS, por exemplo, o valor total exibido na tela não muda.

Precisamos que o `SwitchCell` seja avisado quanto ao cálculo total do valor, quando este sofre alguma alteração. Isto é possível por conta do `Binding`. Como vimos antes, a propriedade do `SwitchCell` que indica seu funcionamento é o `On`. Abrindo o arquivo `DetalheView.xaml`, então, teremos:

```
<SwitchCell Text="{Binding TextoFreioABS}" On="{Binding TemFreioABS}"></SwitchCell>
```

A definição do `Binding` possui alguma propriedade que ainda não foi criada (`TemFreioABS`), o que será feito também no *code behind*, sendo de tipo específico (boolean) para suportar os valores de ativação e desativação do `SwitchCell`.

Expandiremos esta propriedade pois queremos saber quando ela for ativada ou desativada, exibindo-se uma mensagem ao usuário.

```
bool temFreioABS;
public bool TemFreioABS { get
    {
        return temFreioABS;
    }
    set
    {
        temFreioABS = value;

        if (temFreioABS)
            DisplayAlert("Freio ABS", "Ligado!", "Ok");
        else
            DisplayAlert("Freio ABS","Desligado!", "Ok");
    }
}
```

Deste modo, pode-se rodar a aplicação e verificar o que se exibe na tela. Os alertas aparecem corretamente, tanto desligado quando ligado.

Para fazermos o `Binding` do valor total em `DetalheView.xaml` (em que há um `TextCell` com o texto "Total: R$ 5000"), substituiremos este texto por outro `Binding` que trará a soma total a partir da propriedade `ValorTotal`.

```
<TextCell Text="{Binding ValorTotal}"></TextCell>
```

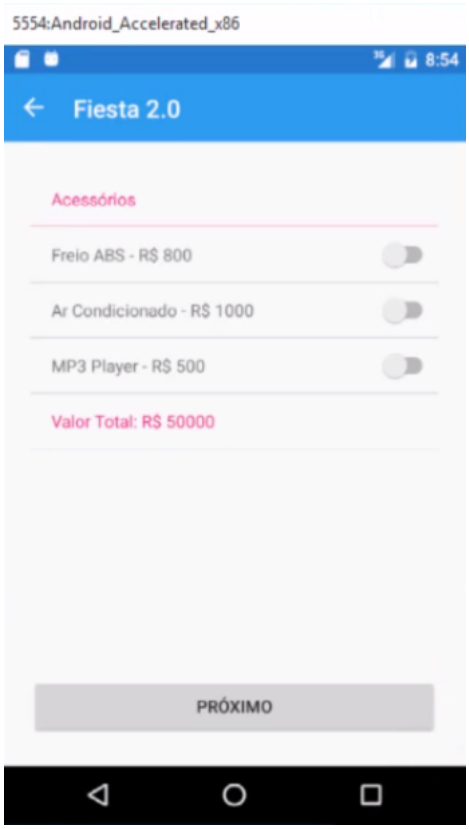
Inicialmente, criaremos apenas a propriedade denominada `ValorTotal`, de tipo `decimal` e, no acessor `get`, colocaremos simplesmente o valor total do veículo.

```
public decimal ValorTotal
{
    get
    {
        return Veiculo.Preco;
    }
}
```

Com isto, esperamos visualizar o valor total na tela, e de fato conseguimos. Além deste valor, exibiremos um texto complementar, para não deixá-lo "solto". Em `DetalheView.xaml.cs`, acrescentaremos uma `string` antes de `Veiculo.Preco`, lembrando de trocar `decimal` por `string`:

```
public string ValorTotal
{
    get
    {
        return string.Format("Valor Total: R$ {0}",Veiculo.Preco);
    }
}
```

Feito isto, rodaremos de novo a aplicação, de que obteremos este resultado:



Agora este valor total precisa mudar conforme vamos acrescentando acessórios, como o freio ABS. Para isto, voltaremos ao `ValorTotal` no arquivo XAML para que se calcule, assim como o preço do veículo, o valor do acessório, quando ativado.

Poderíamos utilizar condicionais, como em `+ if TemFreioABS FREIO_ABS else 0`, para trazermos um valor de acordo com a nossa chave (`SwitchCell`). No entanto, para uma linha, substituiremos `if e else` com os operadores ternários `?` e `:`, respectivamente:

```
public string ValorTotal
{
    get
    {
        return string.Format("Valor Total: R$ {0}",Veiculo.Preco
        + (TemFreioABS ? FREIO_ABS : 0));
    }
}
```

Isto permitirá que se calcule corretamente o valor total para *Test Drive*, de acordo com a chave de configuração. Vamos rodar a app, testando a ativação de uma das chaves para ver se isto funciona.

O valor total não mudou, independentemente das chaves estarem ativas ou não. Qual será o motivo para o valor total não está sendo atualizado? Vamos inserir um *breakpoint* na linha de retorno do valor total (em `DetalheView.xaml.cs`) e rodar a app de novo. Quando a chave do freio ABS não é ativada, `TemFreioABS` é **falso**, portanto o valor a ser somado será `0`.

Ativando-se a chave deste mesmo acessório e desativando-o em seguida repetidas vezes, vemos que o valor total não está sendo acionado. O que quer dizer que a nossa *view* não está se perguntando qual o valor deverá ser exibido na tela.

Pelo que vimos colocando o *breakpoint*, o valor total é acionado apenas no início, no momento em que a página é carregada, pegando-se o texto e, a partir daí, ele não se altera mais. Portanto, precisaremos modificar o app para que ele entenda que houve uma mudança no `SwitchCell`.

Notificaremos o Xamarin Forms de que houve mudança em uma das propriedades que estão sendo acessadas via `Binding`. Qual propriedade é esta? Trata-se da propriedade boolean `temFreioABS`.

Apesar de exibirmos ao usuário que o freio ABS está selecionado (ligado) ou não, por meio de caixas de mensagem, não estamos de fato notificando o Xamarin Forms. Teremos que utilizar uma estratégia de notificação, de que uma propriedade está sendo modificada.

Faremos isto chamando um método que já faz parte de uma página do Xamarin Forms, chamado `OnPropertyChanged()`, que podemos passar sem nenhum parâmetro, e ele notificará a página de que houve mudança em uma propriedade pertencente ao `Binding`.

```
bool temFreioABS;
public bool TemFreioABS
{
    get
    {
        return temFreioABS;
    }
    set
    {
        temFreioABS = value;
        OnPropertyChanged();
    }
}
```

Além disto, é bom modificarmos em `DetalheView.xaml` o tipo de `Binding` que estamos utilizando. Até agora, vimos o simples, que recebe o valor do *code behind*, do código C#, modificando a interface gráfica de acordo com isso.

No entanto, o caminho inverso também precisa ocorrer: nosso código C# tem de ser notificado de que houve uma mudança na página. Isto será feito trocando-se o modo como fazemos o `Binding`, definindo-o como `TwoWay`, que é a **notificação de duas vias**.

Desta maneira, a página é atualizada com o valor que vem do código C#, bem como a página notifica o código C# sobre a atualização. Incluiremos este modo `TwoWay` nas outras propriedades também:

```
<TableSection Title="Acessórios">
    <SwitchCell Text="{Binding TextoFreioABS}" On="{Binding TemFreioABS, Mode=TwoWay}"></SwitchC
    <SwitchCell Text="{Binding TextoArCondicionado, Mode=TwoWay}"></SwitchCell>
    <SwitchCell Text="{Binding TextoMP3Player, Mode=TwoWay}"></SwitchCell>
    <TextCell Text="{Binding ValorTotal, Mode=TwoWay}"></SwitchCell>
</TableSection>
```

Rodaremos novamente a aplicação, que pausa no *breakpoint*. Ativaremos o freio ABS, porém o valor total continua inalterado.

Quando modificamos a chave do freio ABS, o programa deveria parar novamente no *breakpoint*, o que não ocorre. Em `DetalheView.xaml.cs`, então, precisamos fazer uma nova notificação informando que a propriedade `ValorTotal` também está sendo atualizada quando o freio ABS é modificado.

Colocaremos outro `OnPropertyChanged()`, inserindo o nome da propriedade `ValorTotal`, o qual poderíamos colocar entre aspas (`"ValorTotal"`), porém há uma maneira melhor para isto, o `nameof`. Ele pegará o nome de um tipo que já existe no sistema, sem que nos preocupemos com erros no texto. Podemos utilizar o ***IntelliSense*** para nos auxiliar a não errar o nome da propriedade que utilizaremos em `OnPropertyChanged`.

```
public bool TemFreioABS
{
    get
    {
        return temFreioABS;
    }
    set
    {
        temFreioABS = value;
        OnPropertyChanged();
        OnPropertyChanged(nameof(ValorTotal));
    }
}
```

Vamos rodar a app de novo e ver o que acontece; ele pausa no *breakpoint*, depois rodaremos novamente e selecionaremos o freio ABS. Agora que alteramos o `SwitchCell` marcando-o como verdadeiro (ou seja, ativando-o), o programa pediu novamente à propriedade `ValorTotal` seu valor atualizado.

Tiraremos o *breakpoint*, voltando a rodar a aplicação em seguida, e o que aparece é o valor total finalmente modificado, com acréscimo do preço do freio (o acessório selecionado pelo usuário).

Após todo este trabalho, conseguimos finalmente fazer o cálculo do valor final utilizando-se o `Binding`, refletindo-o em uma propriedade do nosso *code behind*, que está sendo exibido na tela.