



Flutter Avançado - Aula 2

Exercícios 5 a 7



- Flutter Releases [Ver mais](#)



- Com o objeto *MediaQueryData*, nós conseguimos recuperar diversas informações do dispositivo
- Além de tamanho de tela, conseguimos verificar a orientação do dispositivo
- Dados de acessibilidade

```
Widget build(BuildContext context) {  
  MediaQueryData details =  
  MediaQuery.of(context);  
}
```





Tipo	Atributo	Descrição
bool	accessibleNavigation	Indica se o usuário está usando um serviço de acessibilidade
bool	alwaysUse24HourFormat	Indica se a hora usa o formato de 24 horas
bool	boldText	Indica se a plataforma solicita fonte em negrito
double	devicePixelRatio	Indica o número de pixels no dispositivo para cada pixel lógico
bool	disableAnimations	Indica se a plataforma solicita desabilitar animações
bool	invertColors	Indica se as cores estão sendo invertidas
Orientation	orientation	Indica a orientação : retrato ou paisagem

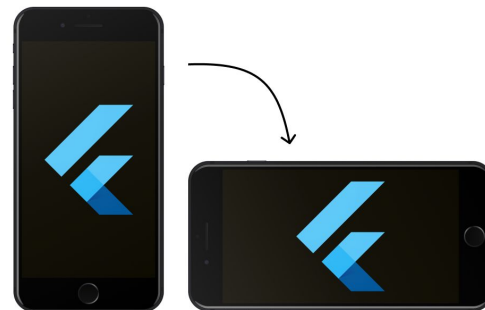


Tipo	Atributo	Descrição
Brightness	platformBrightness	Retorna o nível de brilho do display
Size	size	Retorna Tamanho da mídia
double	textScaleFactor	Indica o número de pixels da fonte para cada pixel lógico
EdgeInsets	viewInsets	Indica as partes do display que estão completamente obscurecidos pela UI do sistema, como o teclado



- Com o *OrientationBuilder*, conseguimos controlar o layout do widget conforme a orientação do dispositivo

```
OrientationBuilder(  
  builder: (context, orientation) {  
    if (orientation ==  
Orientation.portrait) {  
      return Text("Retrato");  
    } else {  
      return Text("Paisagem");  
    }  
  },
```

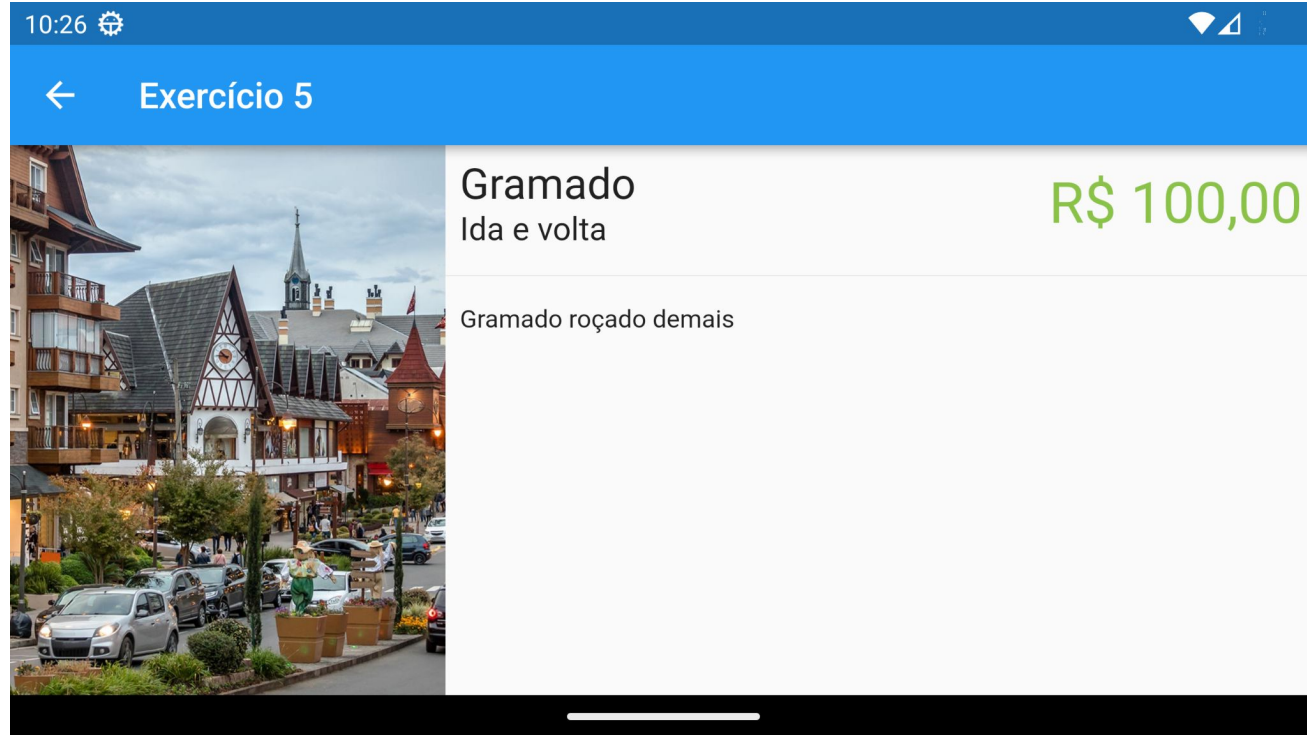




- Ainda reaproveitando nosso app de Viagens
- Ao exibir os detalhes da viagem, eles devem ser exibidos conforme a orientação da tela (paisagem ou retrato)

Exercício 5 (cont)

[Ver mais](#)





- Um widget que delega o tamanho do filho e pode determinar o posicionamento do mesmo
- Para isso, é necessário criarmos um delegate nosso, herdado de *SingleChildLayoutDelegate*
- Nesse delegate determinamos o tamanho do filho do widget



getSize

O método `getSize` determina o tamanho do *CustomSingleChildLayout*, não o do seu filho. Por parâmetro, recebemos as *constraints* do objeto e o método devolve o tamanho, utilizando o objeto *Size*.

Caso o método não seja alterado, será considerado o maior tamanho possível.

```
Size getSize(BoxConstraints constraints) => constraints.biggest;
```



getConstraintsForChild

Para definir as constraints do filho, basta sobrescrever esse método. As constraints retornadas pelo método são repassados para o filho.

```
BoxConstraints getConstraintsForChild(BoxConstraints constraints) =>  
constraints;
```



getPositionForChild

Para determinar a posição do filho, é necessário sobrescrever esse método. Recebemos nesse método dois parâmetros: O tamanho do pai e o tamanho do filho.

```
Offset getPositionForChild(Size size, Size childSize) => Offset.zero;
```



shouldRelayout

Se esse método retornar true, executará novamente os métodos de definição de layout (os métodos citados anteriormente).

```
bool shouldRelayout(covariant SingleChildLayoutDelegate oldDelegate);
```

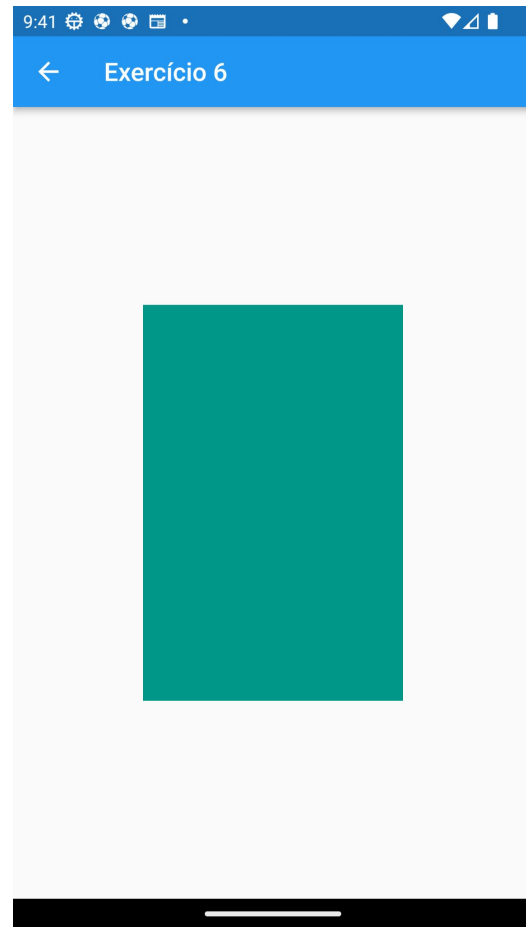


Ordem de execução do *delegate*

1. **getSize** -> Para definir o tamanho do CustomSingleChildLayout
2. **getConstraintsForChild** -> Para definir o tamanho do filho
3. **getPositionForChild** -> Para definir a posição do filho

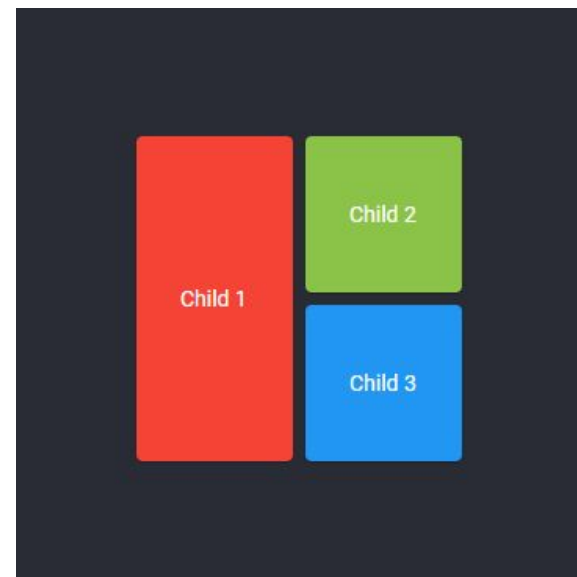


- Utilizando um *CustomSingleChildLayout*, desenhe um quadrado
- O quadrado deve estar posicionado no centro da tela
- Ele deve possuir metade do tamanho da tela, em altura e largura





- Semelhante ao *CustomSingleChildLayout*, mas com vários componentes
- Também possui um *delegate*
- Cada filho deve estar dentro de um *LayoutID*
- O *LayoutID* serve para identificar cada filho no delegate





performLayout

Recebemos por parâmetro nesse método o tamanho do widget pai, e nele definiremos o tamanho e a posição dos filhos.

```
void performLayout(Size size) {  
  // TODO: implement performLayout  
  if (hasChild(1)) {  
    layoutChild(1, BoxConstraints(maxHeight: (size.height / 3), maxWidth:  
size.width));  
  
    // X - horizontal Y - vertical  
    positionChild(1, Offset(0, 0));  
  }  
}
```



shouldRelayout

Se esse método retornar true, executará novamente os métodos de definição de layout (os métodos citados anteriormente).

```
bool shouldRelayout(covariant MultiChildLayoutDelegate oldDelegate) {  
}
```



- Utilizando um *CustomMultiChildLayout*
- Adicione três *containers* em tela, cada um de uma cor
- Cada Container deve ocupar $\frac{1}{3}$ da tela
- Eles devem ficar posicionados um embaixo do outro

