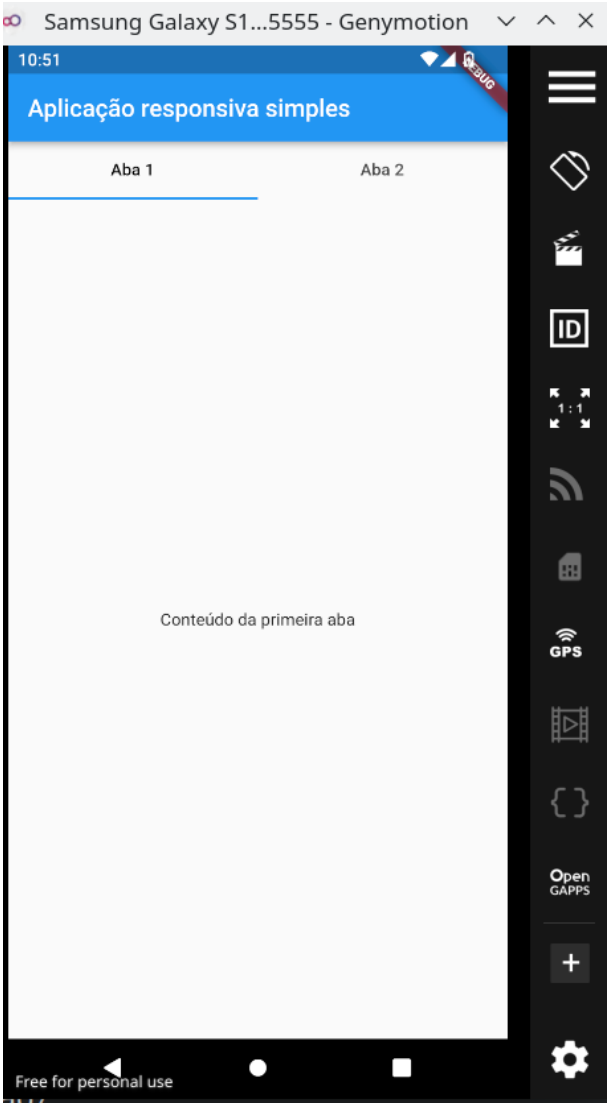
1 Introdução

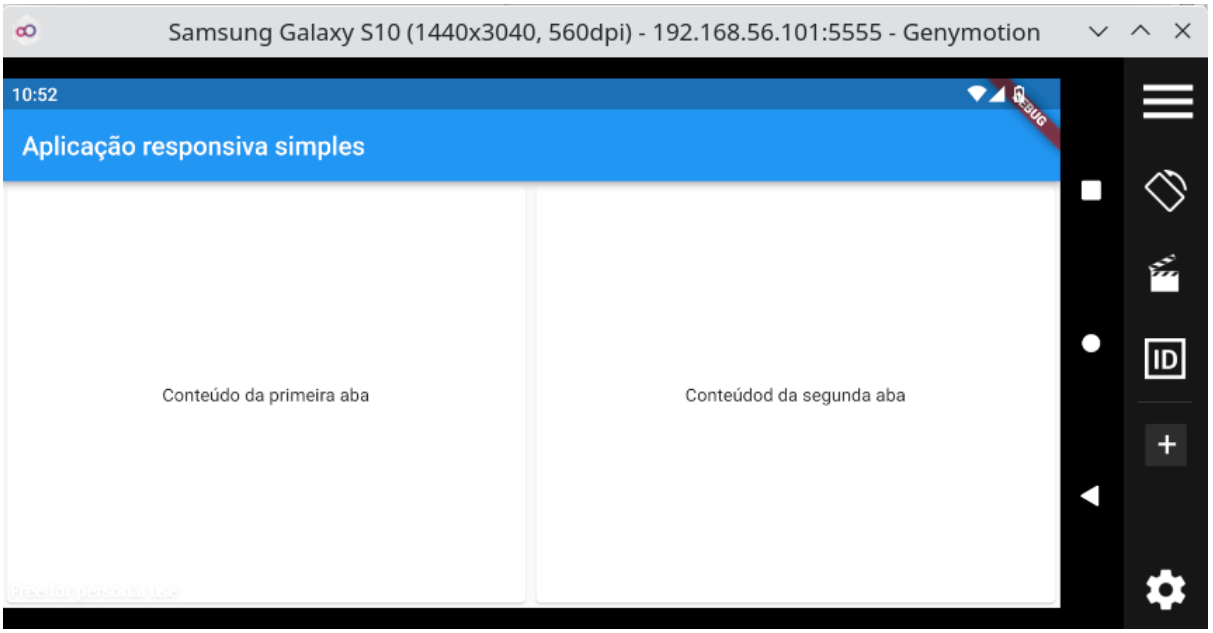
Neste material, vamos desenvolver a seguinte aplicação. Ela mostra como fazer uma tela responsiva em Flutter.

Para telas que tenham largura de, no máximo, 768 pixels, ela será apresentada assim.



Para telas maiores, ela será apresentada assim.

Professor Rodrigo Bossini <professorbossini@gmail.com>
Produção: setembro de 2023



2 Desenvolvimento

2.1 (Workspace, novo projeto Flutter, VS Code, emulador Genymotion) Comece criando uma pasta para desempenhar o papel de Workspace. Ou seja, uma pasta que abriga subpastas, cada qual representando um projeto Flutter. No Windows, você pode usar algo assim:

C:\Users\seuUsuario\Documents\dev

A seguir, abra um terminal e vincule-o ao diretório que acabou de criar com

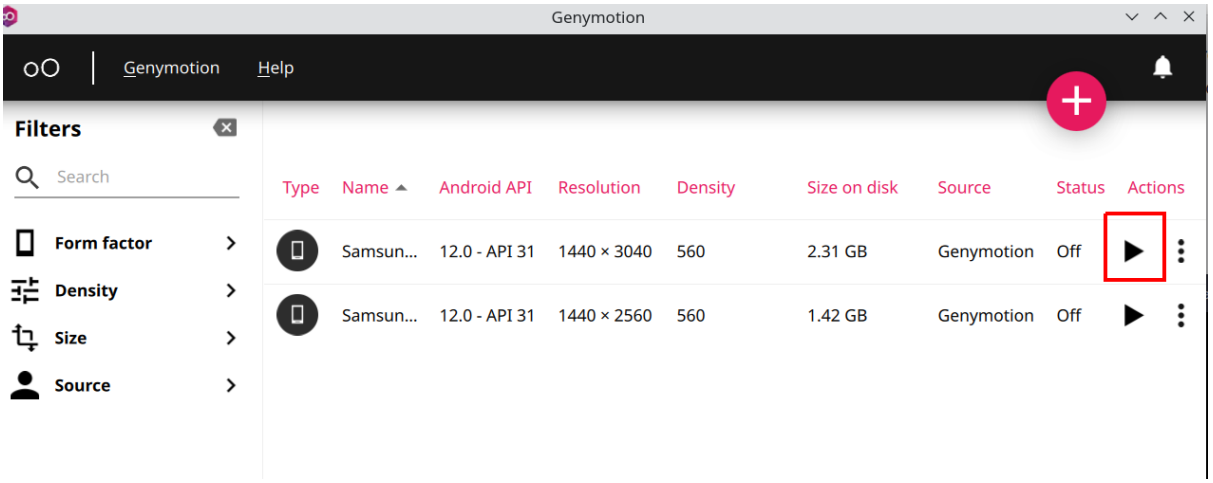
C:\Users\seuUsuario\Documents\dev

Crie o projeto Flutter com

flutter create responsividade

Abra o VS Code e clique em **File >> Open Folder**. Navegue até a pasta recém-criada. Abra o arquivo **lib/main.dart e apague todo o seu conteúdo**.

Clique **Terminal >> New terminal** para abrir um terminal interno do VS Code. Caso possua o emulador Genymotion instalado, abra o aplicativo agora e coloque um de seus dispositivos virtuais em execução.



Caso não possua o Genymotion, você pode executar a aplicação no navegador normalmente.

2.2 Três Widgets: Aquele para telas pequenas, aquele para telas "não pequenas" e aquele que decide qual deve ser exibido. Nossa aplicação terá três widgets

- MobileLayout: ele constrói a tela a ser exibida em dispositivos de largura de, no máximo, 768 pixels.

- WebLayout: ele constrói a tela a ser exibida em dispositivos de largura maior do que 768 pixels.

- MeuLayoutResponsivo: utiliza a classe LayoutBuilder para decidir qual Widget exibir em função da largura da tela.

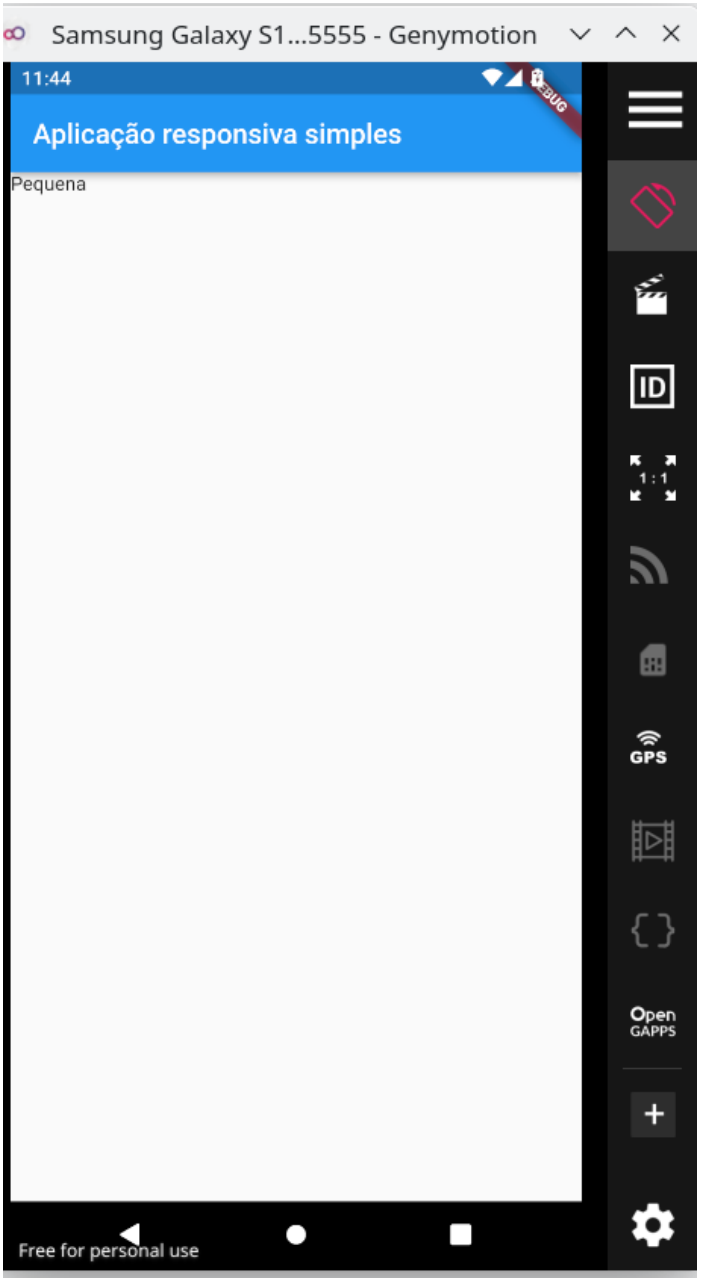
Além disso, ela terá também o Widget principal, cuja definição aparece a seguir.

```dart
import 'package:flutter/material.dart';

void main() => runApp(App());

class App extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Aplicação responsiva simples',
      //um ThemeData armazena dados de um tema, como estilos de
      texto, cores, estilo da AppBar etc
      theme: ThemeData(
        primarySwatch: Colors.blue //swatch significa algo como
        "amostra"
      ),
      //essa classe decide qual Widget usar
      home: Text("Começando"),
    );
  }
}
```

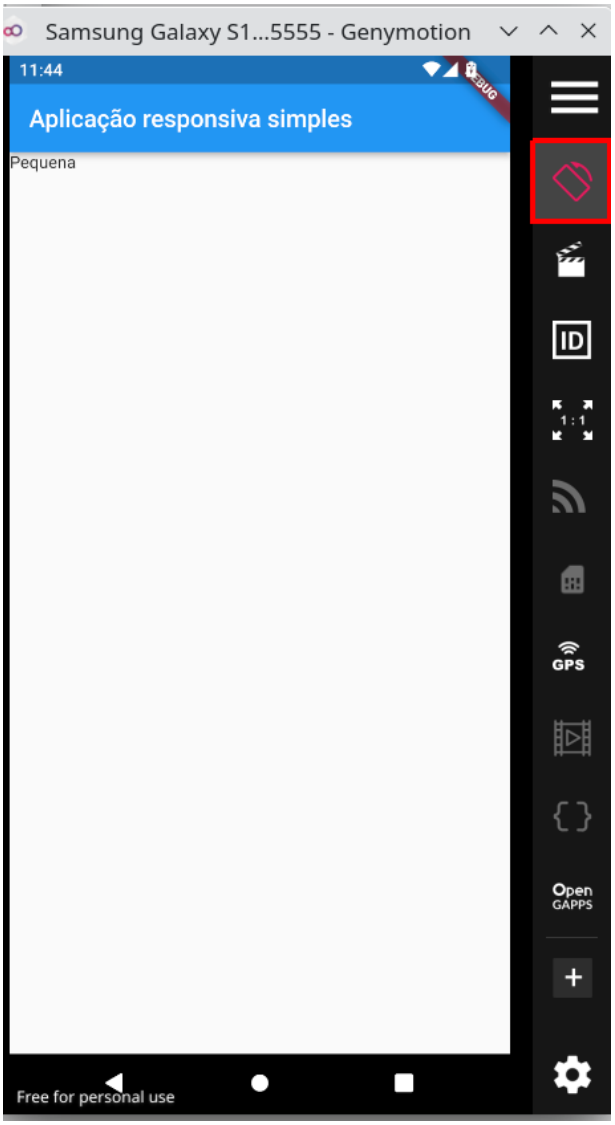
A seguir, escrevemos o Widget MeuLayoutResponsivo. Ele define um Scaffold e, em função da largura da tela, obtida por meio da class LayoutBuilder, decide o que exibir.

```dart
import 'package:flutter/material.dart';

void main() => runApp(App());

class App extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Aplicação responsiva simples',
      //um ThemeData armazena dados de um tema, como estilos de texto, cores, estilo
      da AppBar etc
      theme: ThemeData(
        primarySwatch: Colors.blue //swatch significa algo como "amostra"
      ),
      //essa classe decide qual Widget usar
      home: MeuLayoutResponsivo(),
    );
  }
}

//essa classe decide qual Widget usar, de acordo com o tamanho da tela
class MeuLayoutResponsivo extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text("Aplicação responsiva simples")),
      body: LayoutBuilder(builder: (context, constraints) {
        //breakpoint que definimos para dizer que a tela é pequena
        return constraints.maxWidth <= 578 ? Text("Pequena") : Text("Grande");
      }),
    );
  }
}
```

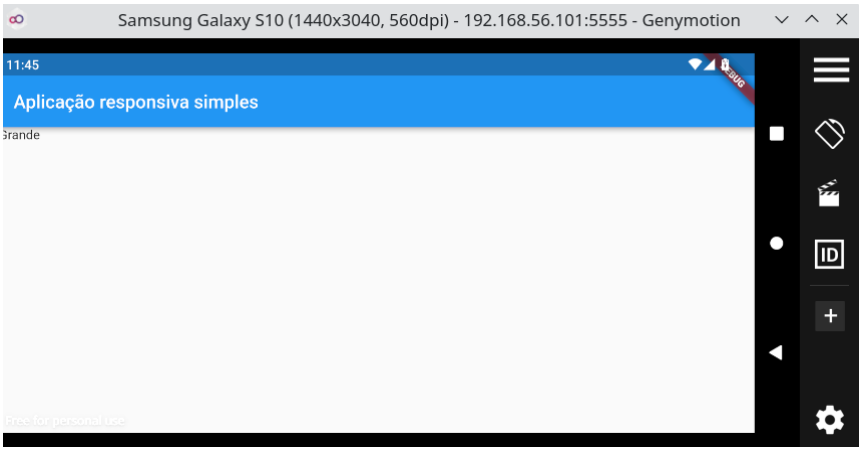
No emulador, veja o resultado quando ele está em modo retrato. Lembre-se de fazer **hot restart antes**, apertando **SHIFT+R no terminal em que você executou o comando flutter run**.



Você pode clicar no botão destacado a seguir para ver o resultado quando o dispositivo está em modo paisagem.



Veja o resultado.



2.3 Dart: O recurso chamado Mixins Dart possui um recurso chamado Mixin. Um Mixin permite utilizarmos código sem termos de utilizar herança. Veja algumas características dos mixins.

- São definidos como classes. Mas trocamos a palavra class por mixin.

- Quando uma classe decide utilizar um mixin que definimos, ela o faz utilizando a palavra **with**, em vez de **extends**.

- Mixins não podem ser instanciados diretamente.

> **Nota**. Uma tradução direta da palavra mixin poderia ser algo como "**misturar**" ou "**combinar**". No ambiente de desenvolvimento de software, geralmente se mantém a palavra original, sem tradução mesmo.

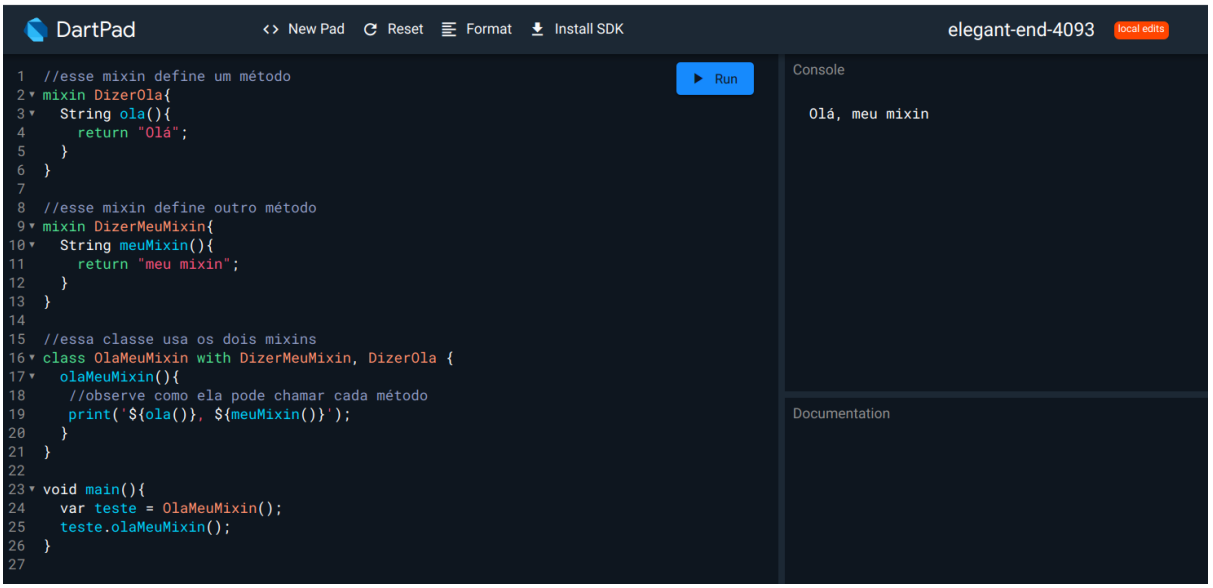
Visite o DartPad em

https://dartpad.dev/

e faça o seguinte teste. Leia os comentários com atenção, ok?

```
//esse mixin define um método
mixin DizerOla{
  String ola(){
    return "Olá";
  }
}

//esse mixin define outro método
mixin DizerMeuMixin{
  String meuMixin(){
    return "meu mixin";
  }
}

//essa classe usa os dois mixins
class OlaMeuMixin with DizerOla, DizerMeuMixin {
  olaMeuMixin(){
    //observe como ela pode chamar cada método
    print('${ola()}, ${meuMixin()}');
  }
}

void main(){
  var teste = OlaMeuMixin();
  teste.olaMeuMixin();
}
```

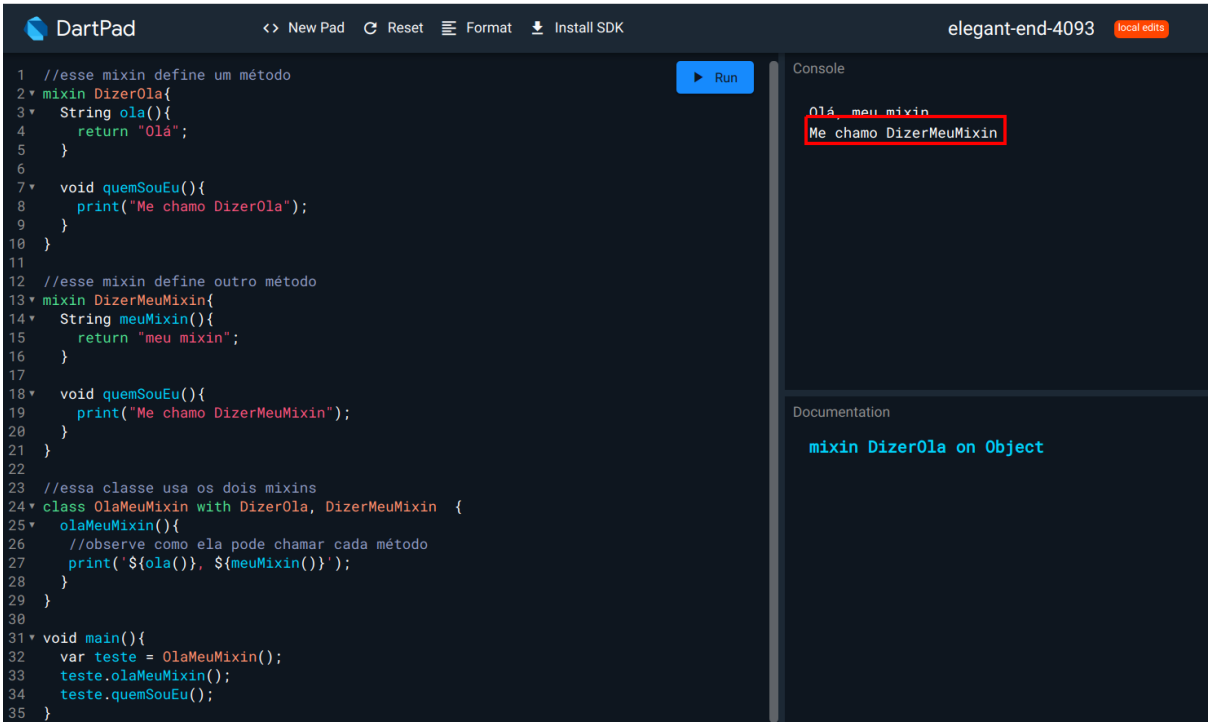
Veja o resultado esperado.



Ocorre que dois mixins podem definir métodos de nomes iguais. O que acontece quando uma classe utiliza ambos? Veja o exemplo a seguir.

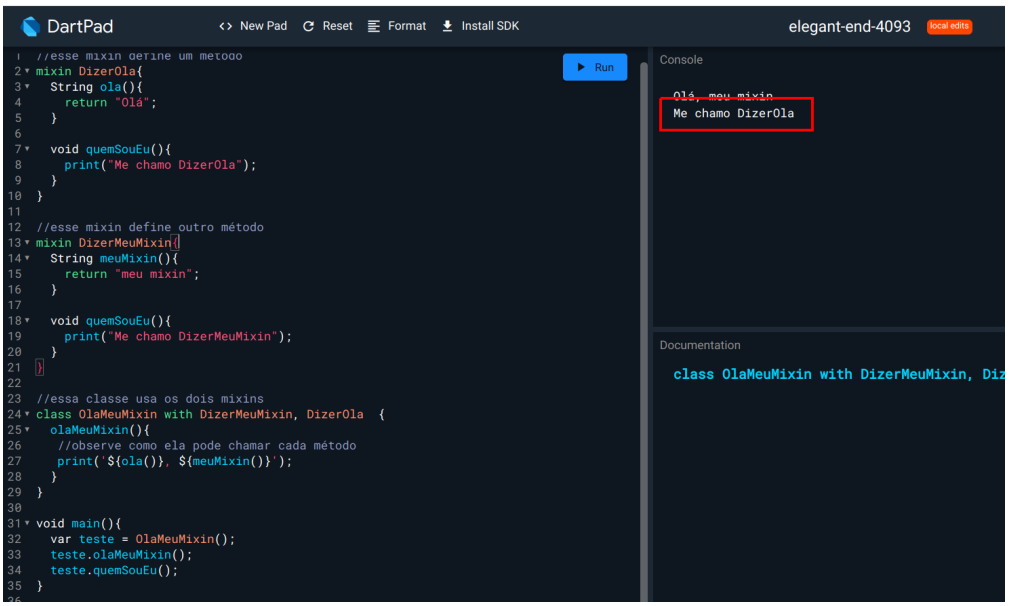
```
//esse mixin define um método
mixin DizerOla{
  String ola(){
    return "Olá";
  }
  void quemSouEu(){
    print("Me chamo DizerOla");
  }
}

//esse mixin define outro método
mixin DizerMeuMixin{
  String meuMixin(){
    return "meu mixin";
  }
  void quemSouEu(){
    print("Me chamo DizerMeuMixin");
  }
}

//essa classe usa os dois mixins
class OlaMeuMixin with DizerOla, DizerMeuMixin {
  olaMeuMixin(){
    //observe como ela pode chamar cada método
    print('${ola()}, ${meuMixin()}');
  }
}

void main(){
  var teste = OlaMeuMixin();
  teste.olaMeuMixin();
  teste.quemSouEu();
}
```

Veja o resultado esperado.



Observe que aparece o texto produzido pelo mixin "DizerMeuMixin". Por que isso aconteceu? É simples. Ele foi o último, da esquerda para a direita, definido na lista de mixins da classe, após a palavra with. Inverta a ordem e confira o resultado.

```
class OlaMeuMixin with DizerMeuMixin, DizerOla{
```



2.4 O mixin SingleTickerProviderStateMixin Agora que sabemos o que são os mixins, podemos fazer uso de um que o Flutter oferece: **SingleTickerProviderStateMixin**.

Um Ticker é uma espécie de relógio que faz “tick” a cada frame de uma animação. Você se lembra das expressões 30fps ou 60fps? Elas indicam o número de frames per second de uma aplicação. Ou seja, quantas trocas de quadros ou frames acontecem por segundo. Um Ticker é um objeto que faz tick a cada troca de quadro. Neste caso, fazer tick significa notificar a quem estiver interessado que aquele intervalo de tempo passou.

Neste exemplo, vamos utilizar este mixin para fazer a animação que ocorre na transição de abas. Para construir as abas, vamos precisar de

- **TabController**: gerencia a seleção de abas e as animações entre elas. Quando a instanciamos, informamos o número de abas e um TickerProvider, que poderá ser a nossa própria classe, por conta do uso do SingleTickerProviderStateMixin.

- **TabBar**: Widget que exibe uma barra de abas.

- **TabBarView**: Widget que exibe o conteúdo de uma aba selecionada.

Como este vai ser um Widget stateful, escrever duas classes. Aquele que define o Widget e aquela que define seu estado.

```dart
//Widget para telas "pequenas". Vamos exibir uma tela com duas abas.
class MobileLayout extends StatefulWidget {
  @override
  State<StatefulWidget> createState() {
    //vai ser definida a seguir
    return MobileLayoutState();
  }
}

//classe que representa o estado do Widget para telas pequenas
class MobileLayoutState extends State<MobileLayout>
                    with SingleTickerProviderStateMixin {
  //para lidar com as abas
  TabController? tabController;

  @override
  void initState() {
    super.initState();
    //vsync deve ser um tickerprovider
    //pode ser this por conta do uso do mixin
    tabController = TabController(length: 2, vsync: this);
  }

  @override
  Widget build(BuildContext context) {
    //gerenciador de layout que empilha seus filhos (coluna)
    return Column(
      children: [
        //A barra que exibe as abas
        TabBar(
          labelColor: Colors.black,
          controller: tabController,
          tabs: [Tab(text: "Aba 1"), Tab(text: 'Aba 2')],
        ),
        //gerenciador de layout para expandir seu conteúdo no eixo
        principal
        Expanded(
          child: TabBarView(
          controller: tabController,
          children: [
            Center(child: Text("Conteúdo da primeira aba")),
            Center(child: Text("Conteúdo da segunda aba"))
          ],
        ))
      ],
    );
  }
}
```

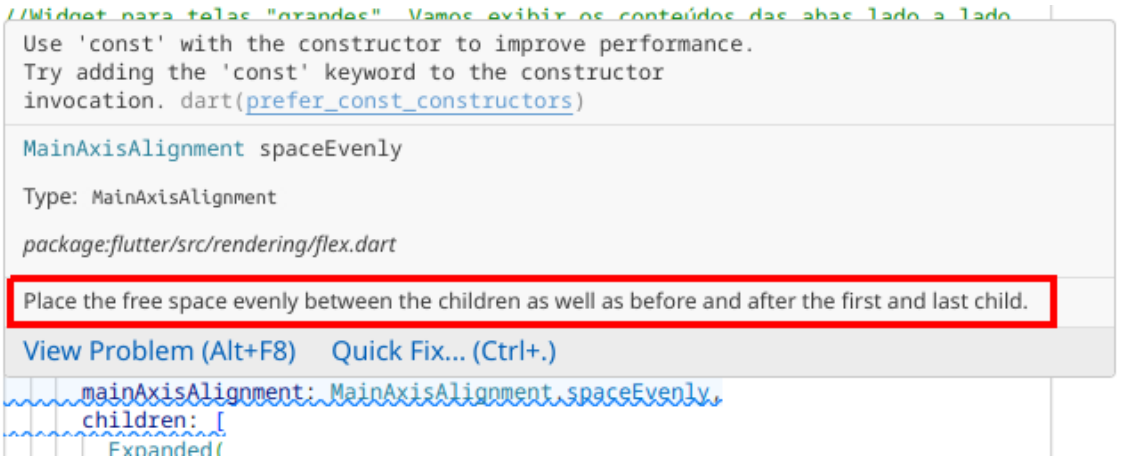
Já podemos utilizá-lo. Veja.

```
...
//essa classe decide qual Widget usar, de acordo com o tamanho da
tela
class MeuLayoutResponsivo extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text("Aplicação responsiva
        simples")),
      body: LayoutBuilder(builder: (context, constraints) {
      //breakpoint que definimos para dizer que a tela é pequena
        return constraints.maxWidth <= 578 ? MobileLayout() :
                                              Text("Grande");
      }),
    );
  }
}
...
```

A seguir, veja o resultado no emulador. (Lembre-se de fazer hot restart (SHIFT+R)).

Resta fazer a definição do Widget (e de seu estado) para telas maiores.

Nota. Lembre-se sempre de passar o mouse em cima do nome do recurso que você não conhecer para ler a sua documentação. VEja um exemplo de **MainAxisAlignment.spaceEvenly**.



O Widget e seu estado ficam assim.

```dart
//Widget para telas "grandes". Vamos exibir os conteúdos das abas lado a lado.
class WebLayout extends StatefulWidget {
  @override
  State<StatefulWidget> createState() {
    return WebLayoutState();
  }
}

//classe que representa o estado do Widget para telas grandes
class WebLayoutState extends State<WebLayout> {
  @override
  Widget build(BuildContext context) {
    return Row(
      mainAxisAlignment: MainAxisAlignment.spaceEvenly,
      children: [
        Expanded(
          child: Card(
            child: Center(child: Text("Conteúdo da primeira aba"))
          )
        ),
        Expanded(
          child: Card(
            child: Center(
              child: Text('Conteúdo da segunda aba'),
            )
          )
        )
      ],
    );
  }
}
```

Faça hot restart e novos testes.

Referências

Dart programming language | Dart. Google, 2023. Disponível em <https://dart.dev/>. Acesso em setembro de 2023.

Flutter - Build apps for any screen. Google, 2023. Disponível em <https://flutter.dev/>. Acesso em setembro de 2023.