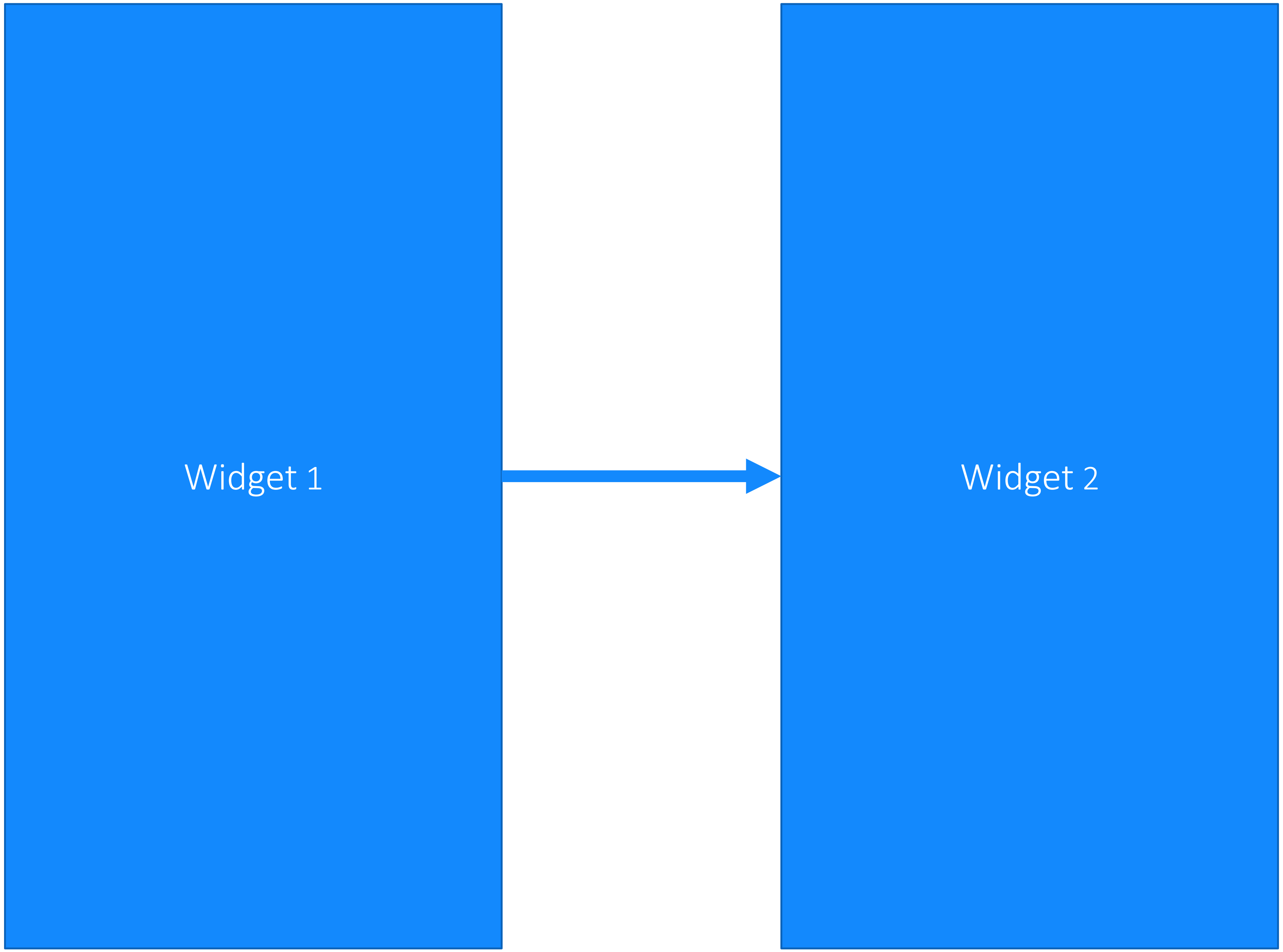
Navigation

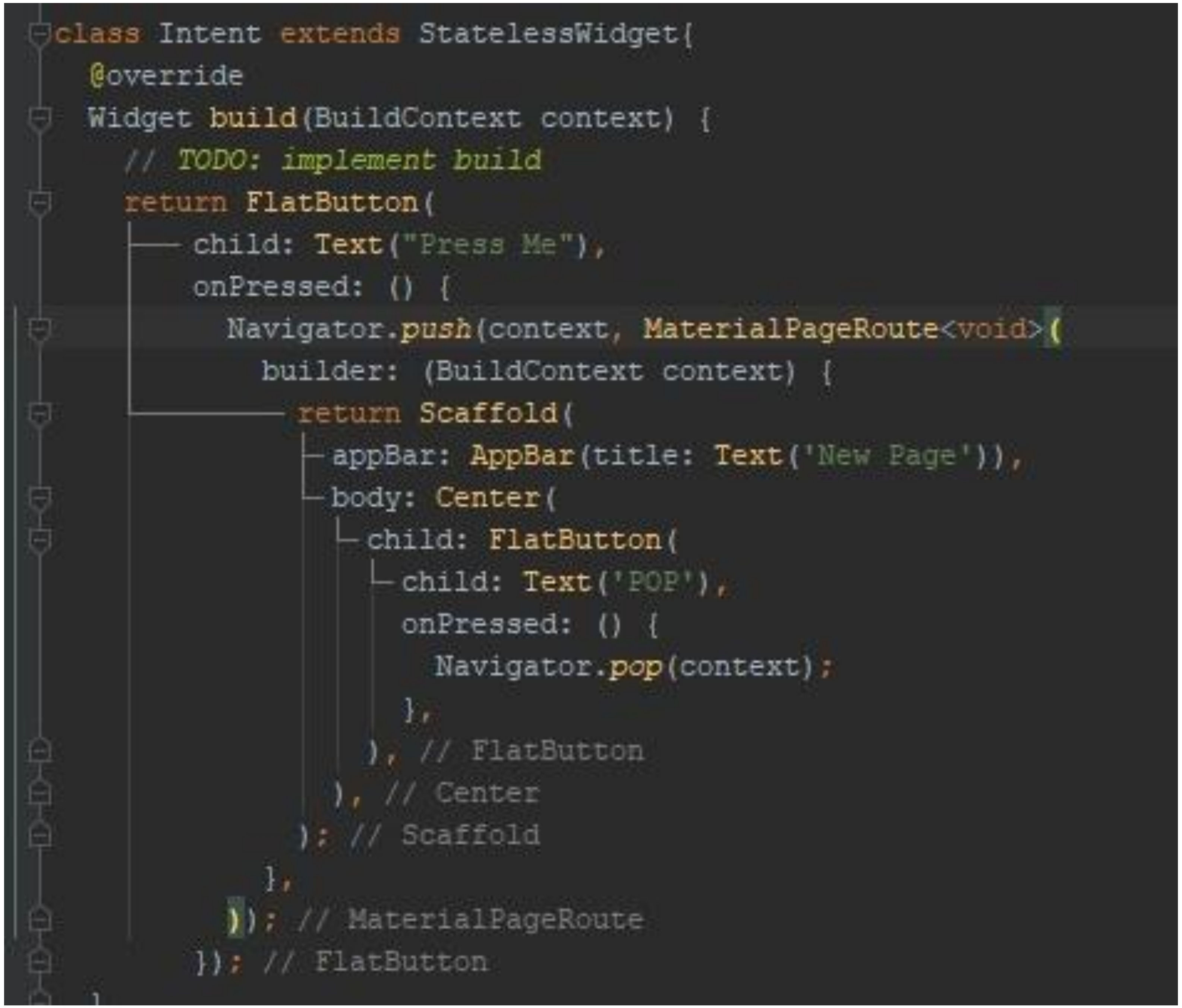

Widget 1
Widget 2

Navigator class

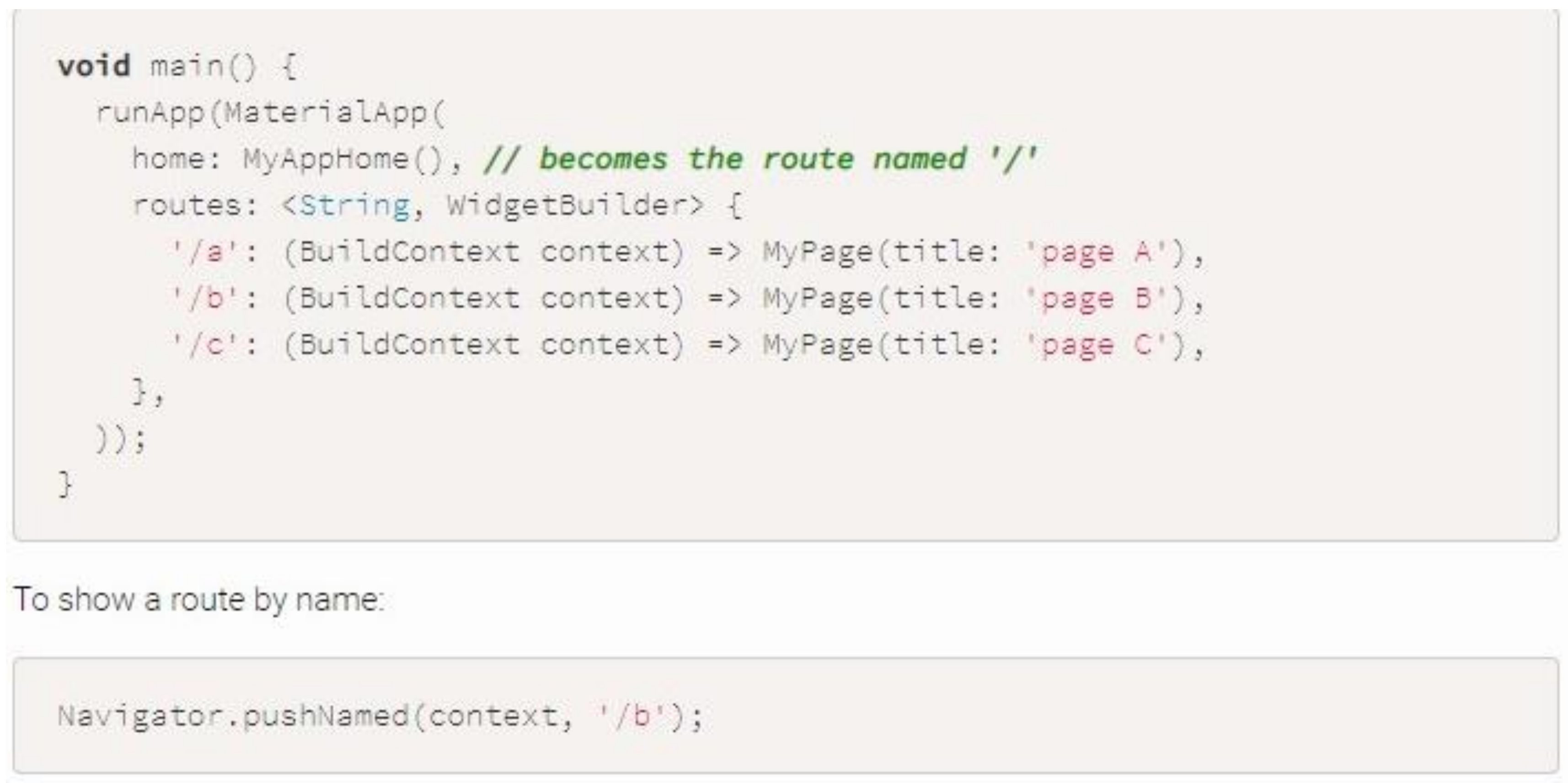
Navigator — это еще один Widget, управляющий страницами приложения в формате стека. Полноэкранные страницы называются маршрутами при использовании в Navigator. Navigator работает как реализация обычного стека.

Push: Метод push используется для добавления еще одного маршрута на вершину текущего стека. Новая страница отображается поверх предыдущей.

Pop: Поскольку Navigator работает как стек, он использует принцип LIFO (Last-In, First-Out). Метод pop удаляет верхний маршрут из стека, а пользователю отображается предыдущая страница.


```
class Intent extends StatelessWidget{
  @override
  Widget build(BuildContext context) {
    // TODO: implement build
    return FlatButton(
      child: Text("Press Me"),
      onPressed: () {
        Navigator.push(context, MaterialPageRoute<void>(
          builder: (BuildContext context) {
            return Scaffold(
              appBar: AppBar(title: Text('New Page')),
              body: Center(
                child: FlatButton(
                  child: Text('POP'),
                  onPressed: () {
                    Navigator.pop(context);
                  },
                ), // FlatButton
              ), // Center
            ); // Scaffold
          },
        )); // MaterialPageRoute
      }); // FlatButton
}
```

Using named navigator routes

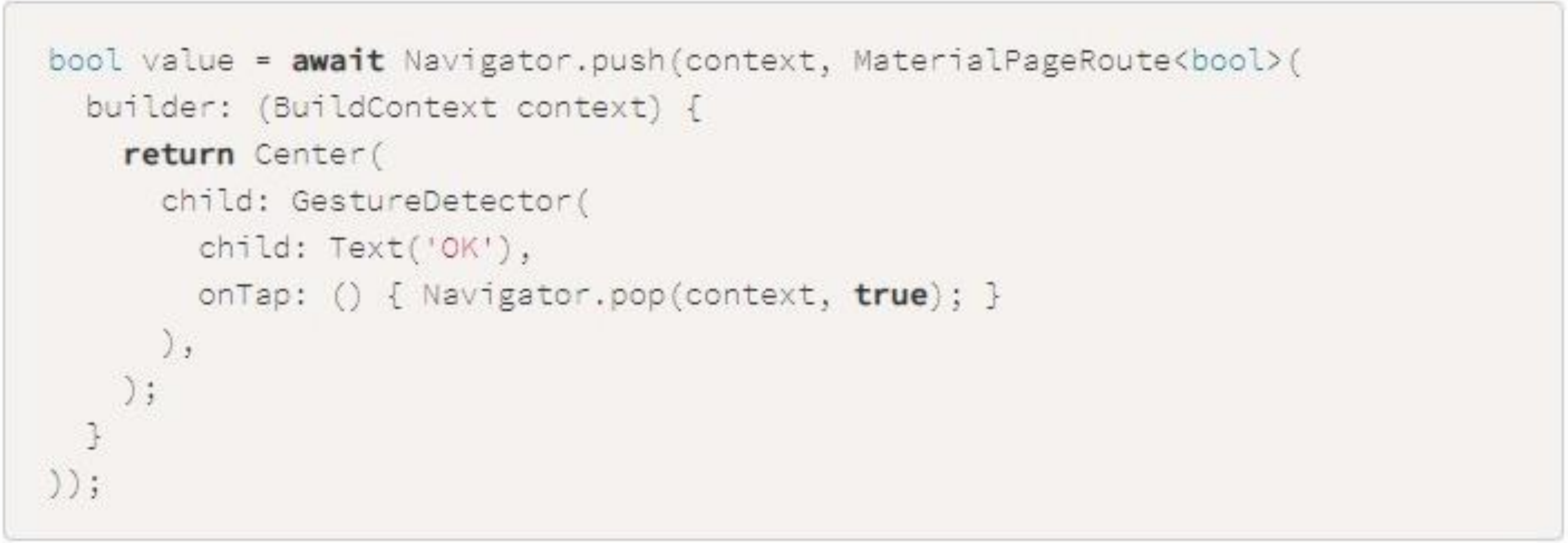


```
void main() {
  runApp(MaterialApp(
    home: MyAppHome(), // becomes the route named '/'
    routes: <String, WidgetBuilder> {
      '/a': (BuildContext context) => MyPage(title: 'page A'),
      '/b': (BuildContext context) => MyPage(title: 'page B'),
      '/c': (BuildContext context) => MyPage(title: 'page C'),
    },
  ));
}
```

To show a route by name:

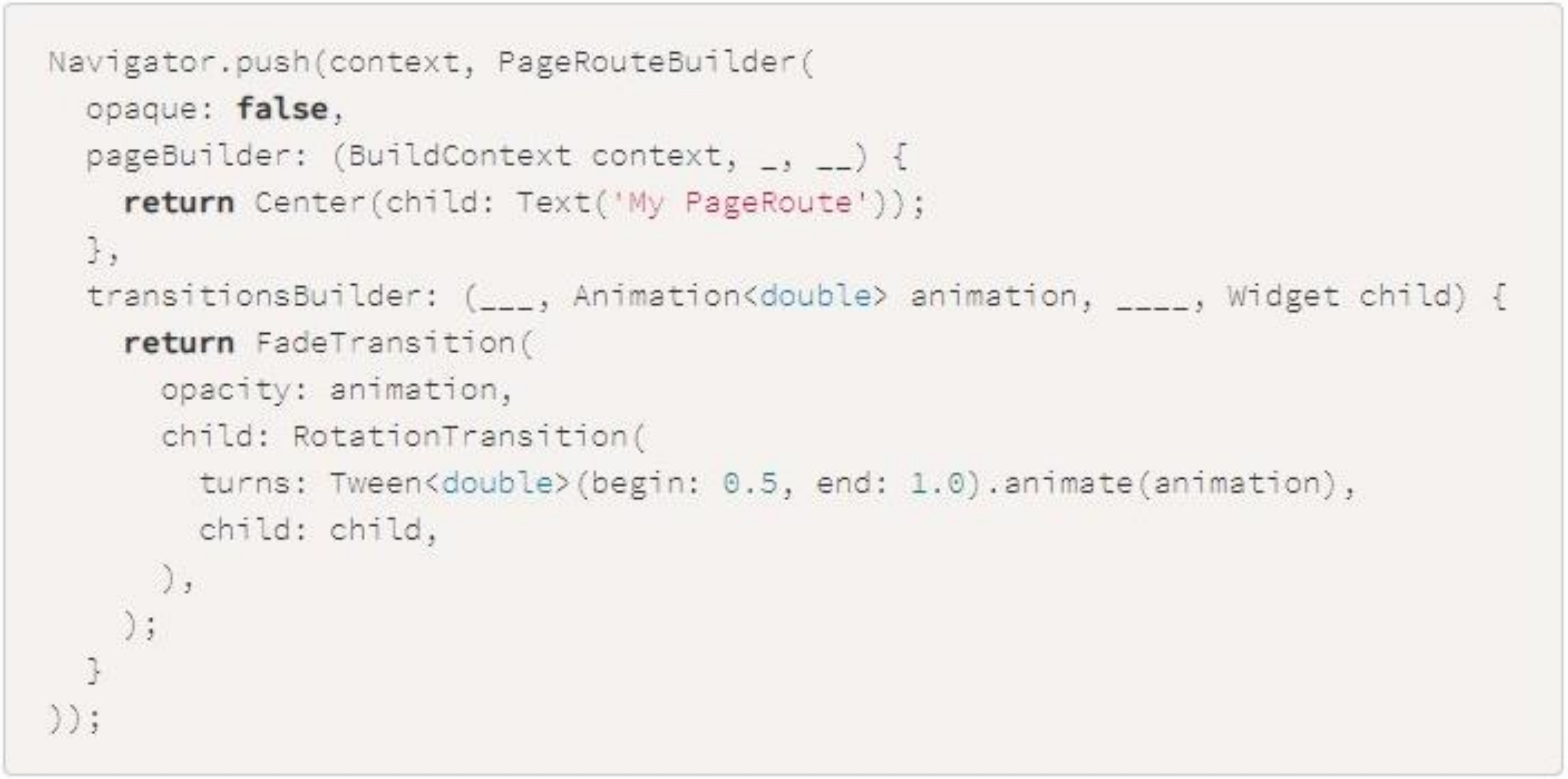
```
Navigator.pushNamed(context, '/b');
```

Routes can return a value



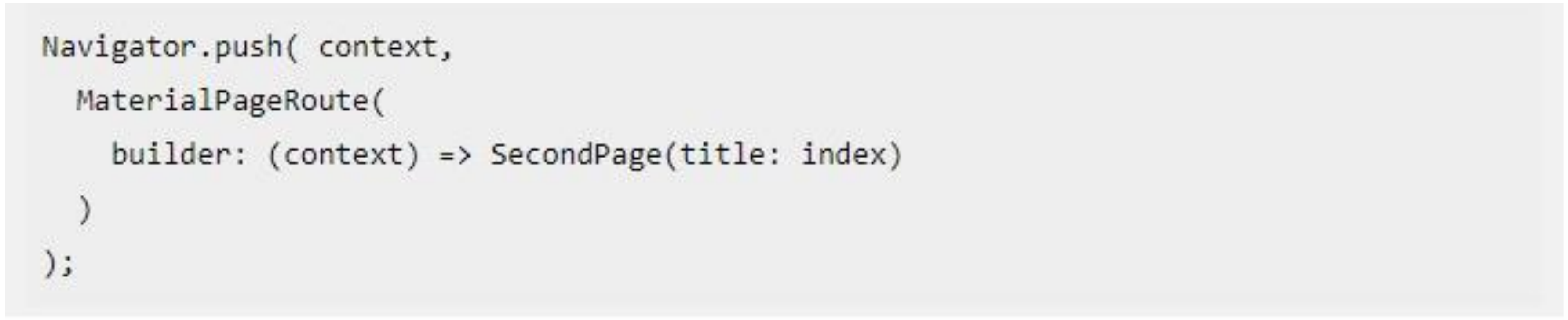
```
bool value = await Navigator.push(context, MaterialPageRoute<bool>(
  builder: (BuildContext context) {
    return Center(
      child: GestureDetector(
        child: Text('OK'),
        onTap: () { Navigator.pop(context, true); }
      ),
    );
  }
));
```

Custom routes



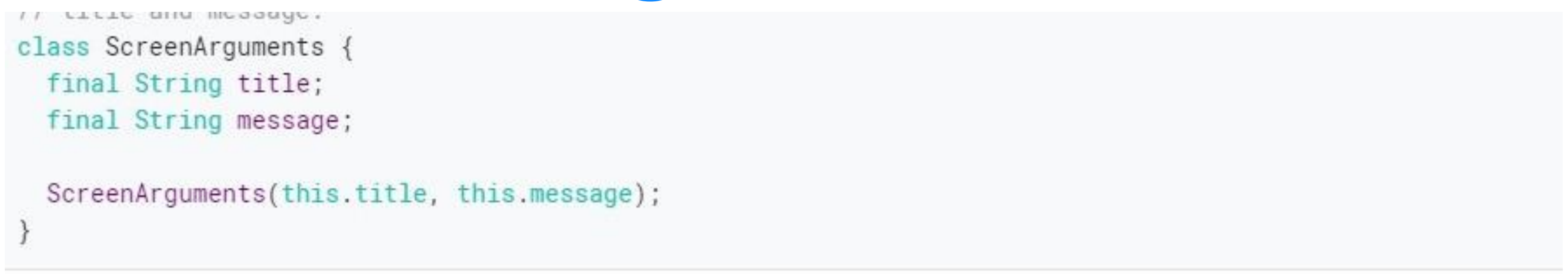
```dart
Navigator.push(context, PageRouteBuilder(
  opaque: false,
  pageBuilder: (BuildContext context, _, __) {
    return Center(child: Text('My PageRoute'));
  },
  transitionsBuilder: (___, Animation<double> animation, ____, Widget child) {
    return FadeTransition(
      opacity: animation,
      child: RotationTransition(
        turns: Tween<double>(begin: 0.5, end: 1.0).animate(animation),
        child: child,
      ),
    );
  }
));
```

Send value



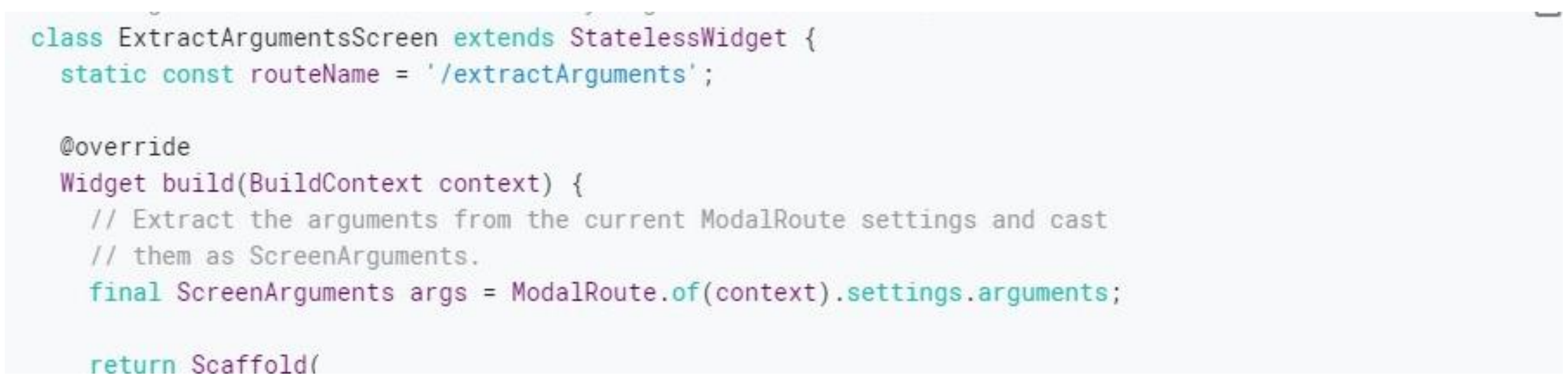
```
Navigator.push( context,
  MaterialPageRoute(
    builder: (context) => SecondPage(title: index)
  )
);
```

Define the arguments



```
// title and message.
class ScreenArguments {
  final String title;
  final String message;

  ScreenArguments(this.title, this.message);
}
```

Create a widget that extracts the arguments



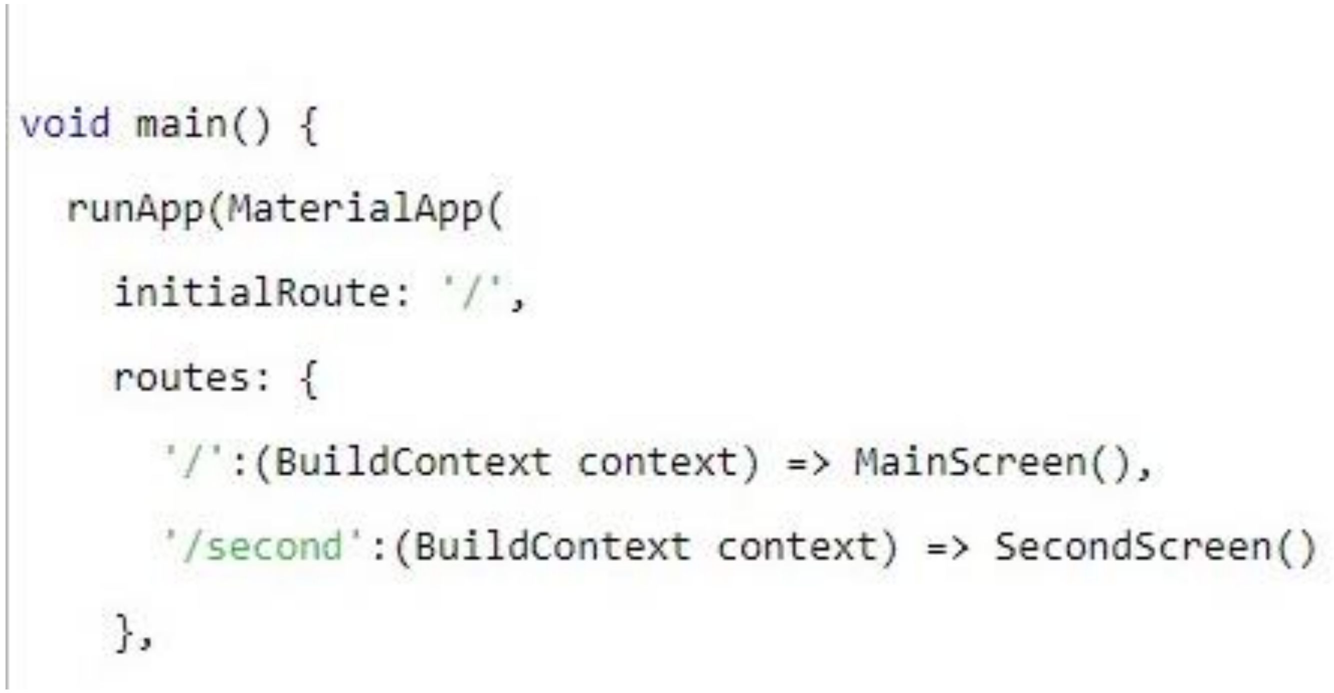
```
class ExtractArgumentsScreen extends StatelessWidget {
  static const routeName = '/extractArguments';

  @override
  Widget build(BuildContext context) {
    // Extract the arguments from the current ModalRoute settings and cast
    // them as ScreenArguments.
    final ScreenArguments args = ModalRoute.of(context).settings.arguments;

    return Scaffold(
```

Register the widget in the routes table

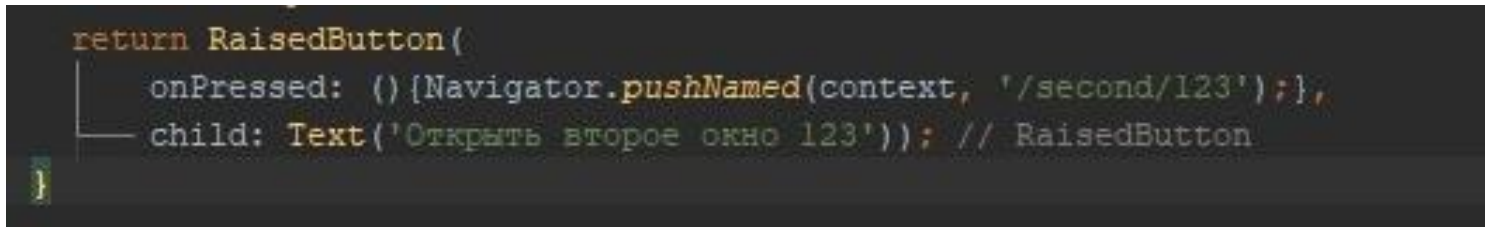
```
MaterialApp(
  routes: {
    ExtractArgumentsScreen.routeName: (context) => ExtractArgumentsScreen(),
  },
);
```

Navigate to the widget

```
RaisedButton(
  child: Text("Navigate to screen that extracts arguments"),
  onPressed: () {
    // When the user taps the button, navigate to a named route
    // and provide the arguments as an optional parameter.
    Navigator.pushNamed(
      context,
      ExtractArgumentsScreen.routeName,
      arguments: ScreenArguments(
        'Extract Arguments Screen',
        'This message is extracted in the build method.',
      ),
```



```
void main() {
  runApp(MaterialApp(
    initialRoute: '/',
    routes: {
      '/':(BuildContext context) => MainScreen(),
      '/second':(BuildContext context) => SecondScreen()
    },
```



```
  return RaisedButton(
      onPressed: (){Navigator.pushNamed(context, '/second/123');},
      child: Text('Открыть второе окно 123')); // RaisedButton
}
```

onGenerateRoute

```
routes: {
    '/':(BuildContext context) => MainScreen(),
    '/second':(BuildContext context) => SecondScreen()
},
onGenerateRoute: (routeSettings){
    var path = routeSettings.name.split('/');

    if (path[1] == "second") {
        return new MaterialPageRoute(
          builder: (context) => new SecondScreen(id:path[2]),
          settings: routeSettings,
        );
    }
}
```

PageViewer

A scrollable list that works page by page.

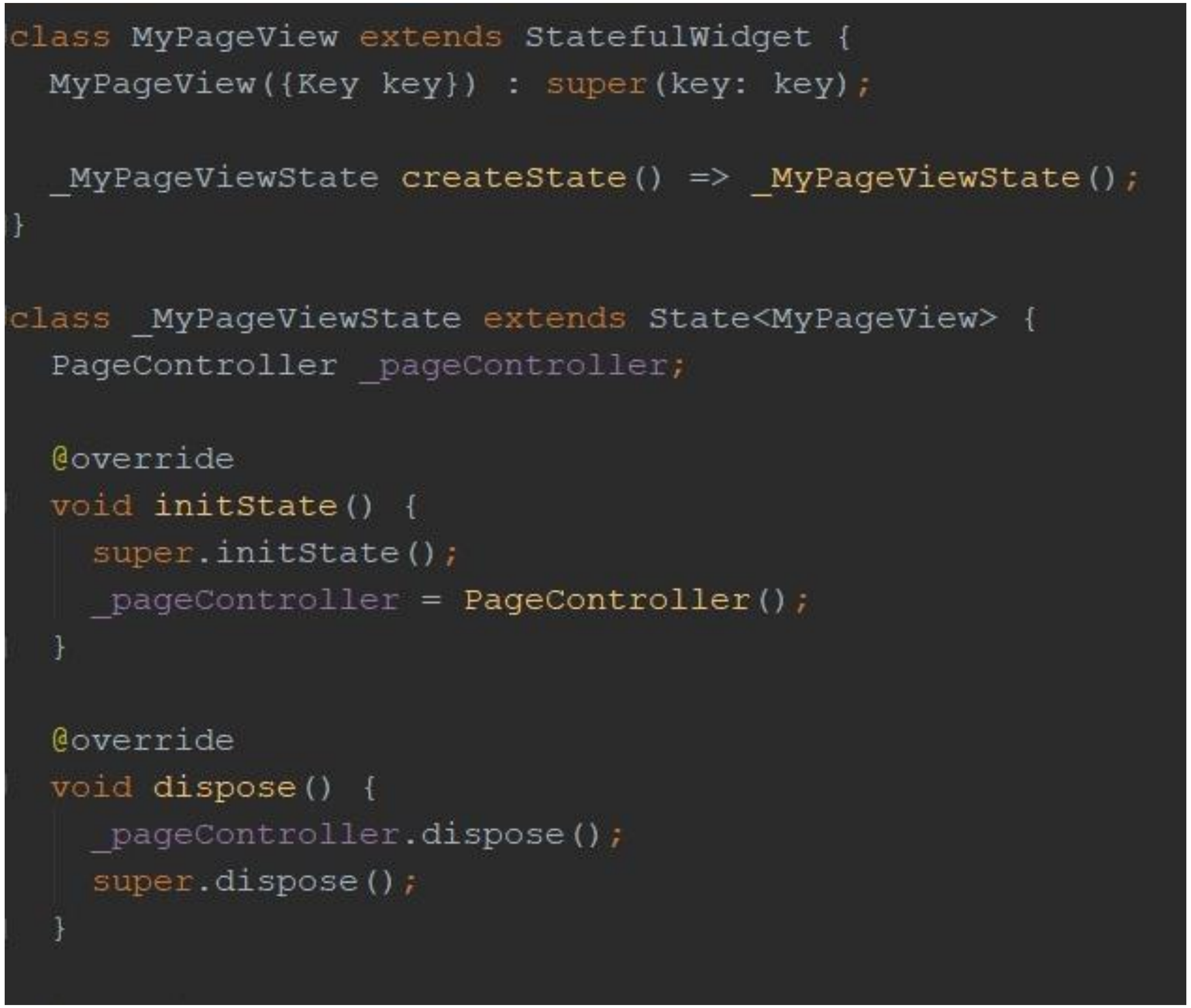
Flutter

Constructors

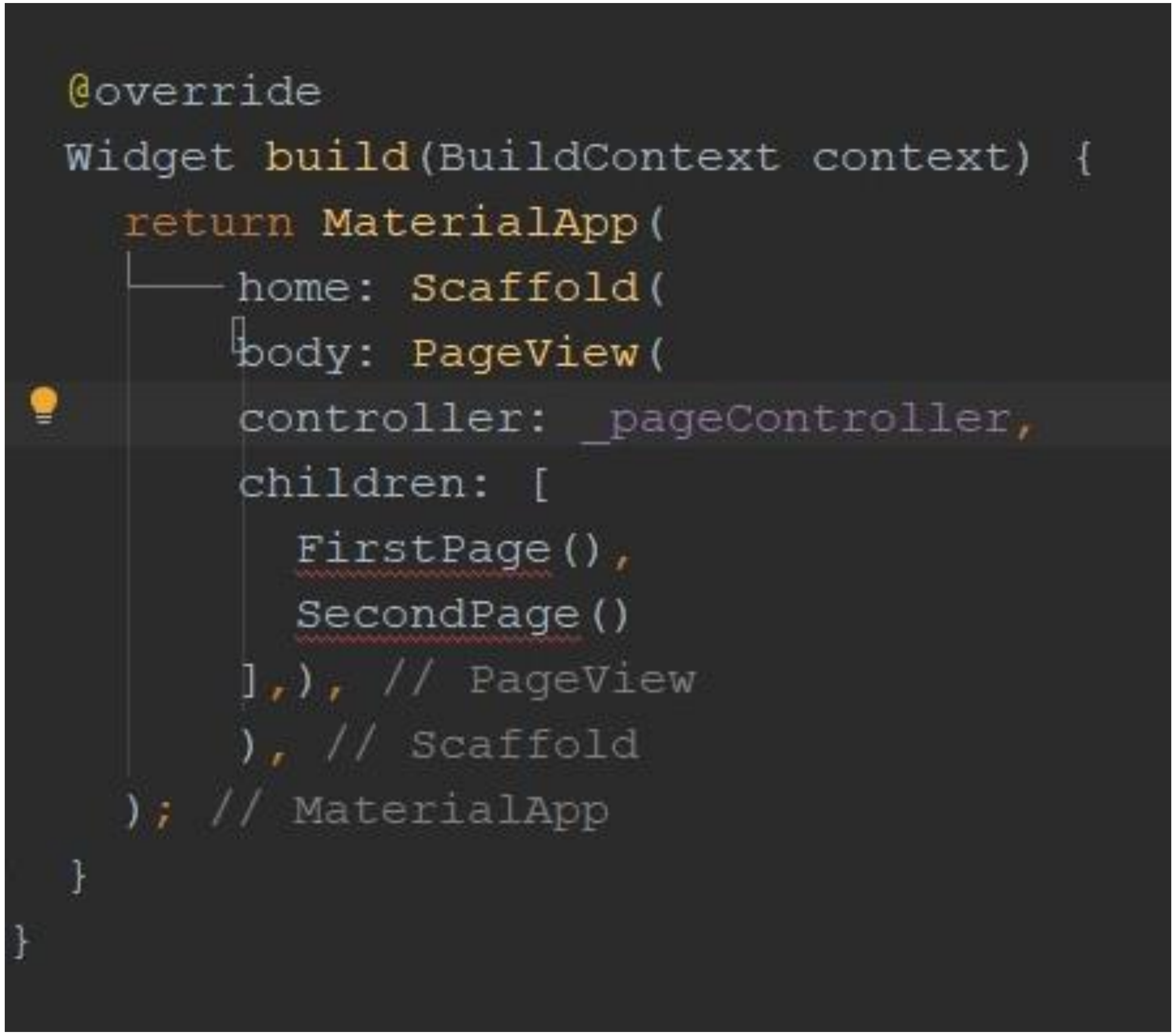
PageView({Key key, Axis scrollDirection: Axis.horizontal, bool reverse: false, PageController controller, ScrollPhysics physics, bool pageSnapping: t
rue,
 ValueChanged<int> onPageChanged, List<Widget> children: const [], DragStartBehavior dragStartBehavior: DragStartBehavior.start})
Creates a scrollable list that works page by page from an explicit List of widgets. [...]

PageView.builder({Key key, Axis scrollDirection: Axis.horizontal, bool reverse: false, PageController controller, ScrollPhysics physics, bool pageSna
pping: true,
 ValueChanged<int> onPageChanged, @required IndexedWidgetBuilder itemBuilder, int itemCount, DragStartBehavior dragStartBehavior: DragSt
artBehavior.start})
Creates a scrollable list that works page by page using widgets that are created on demand. [...]

PageView.custom({Key key, Axis scrollDirection: Axis.horizontal, bool reverse: false, PageController controller, ScrollPhysics physics, bool pageSn
apping: true,
ValueChanged<int> onPageChanged, @required SliverChildDelegate childrenDelegate, DragStartBehavior dragStartBehavior: DragStartBehavior.
start})
Creates a scrollable list that works page by page with a custom child model. [...]

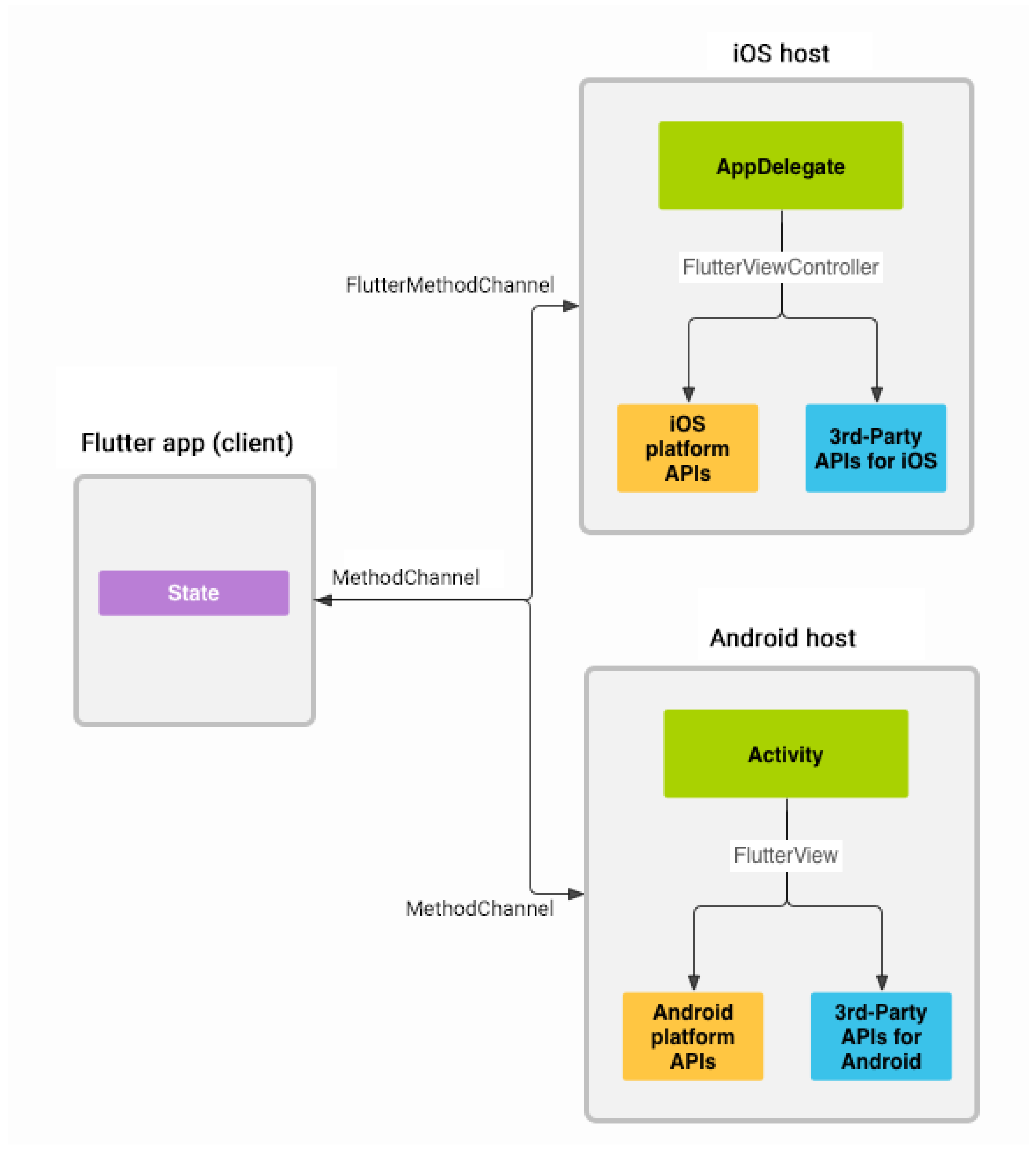


```
class MyPageView extends StatefulWidget {
  MyPageView({Key key}) : super(key: key);

  _MyPageViewState createState() => _MyPageViewState();
}

class _MyPageViewState extends State<MyPageView> {
  PageController _pageController;

  @override
  void initState() {
    super.initState();
    _pageController = PageController();
  }

  @override
  void dispose() {
    _pageController.dispose();
    super.dispose();
  }
```



```
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        body: PageView(
        controller: _pageController,
        children: [
          FirstPage(),
          SecondPage()
        ],), // PageView
        ), // Scaffold
    ); // MaterialApp
  }
}
```

Platform channel


iOS host
AppDelegate
FlutterViewController
iOS platform APIs
3rd-Party APIs for iOS
FlutterMethodChannel
Flutter app (client)
State
MethodChannel
Android host
Activity
FlutterView
Android platform APIs
3rd-Party APIs for Android
MethodChannel

Dart	Java	Kotlin	Obj-C	Swift
null	null	null	nil (NSNull when nested)	nil
bool	java.lang.Boolean	Boolean	NSNumber numberWithBool:	NSNumber(value: Bool)
int	java.lang.Integer	Int	NSNumber numberWithInt:	NSNumber(value: Int32)
int, if 32 bits not enough	java.lang.Long	Long	NSNumber numberWithLong:	NSNumber(value: Int)
double	java.lang.Double	Double	NSNumber numberWithDouble:	NSNumber(value: Double)
String	java.lang.String	String	NSString	String
Uint8List	byte[]	ByteArray	FlutterStandardTypedData typedDataWithBytes:	FlutterStandardTypedData(bytes: Data)
Int32List	int[]	IntArray	FlutterStandardTypedData typedDataWithInt32:	FlutterStandardTypedData(int32: Data)
Int64List	long[]	LongArray	FlutterStandardTypedData typedDataWithInt64:	FlutterStandardTypedData(int64: Data)
Float64List	double[]	DoubleArray	FlutterStandardTypedData typedDataWithFloat64:	FlutterStandardTypedData(float64: Data)
List	java.util.ArrayList	List	NSArray	Array
Map	java.util.HashMap	HashMap	NSDictionary	Dictionary

Dart side

```
static const platform = const MethodChannel('samples.flutter.dev/battery’);

String _batteryLevel = 'Unknown battery level.';

 Future<void> _getBatteryLevel() async {
  String batteryLevel;
  try {
   final int result = await platform.invokeMethod('getBatteryLevel');
   batteryLevel = 'Battery level at $result % .';
  } on PlatformException catch (e) {
   batteryLevel = "Failed to get battery level: '${e.message}'.";
  }

  setState(() {
   _batteryLevel = batteryLevel;
  });
 }
```

iOS side

```
@UIApplicationMain
@objc class AppDelegate: FlutterAppDelegate {
 override func application(
   _ application: UIApplication,
   didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

   let controller : FlutterViewController = window?.rootViewController as! FlutterViewController
   let batteryChannel = FlutterMethodChannel(name: "samples.flutter.dev/battery",
                                             binaryMessenger: controller.binaryMessenger)
   batteryChannel.setMethodCallHandler({
     (call: FlutterMethodCall, result: @escaping FlutterResult) -> Void in
     // Note: this method is invoked on the UI thread.
     // Handle battery messages.
   })

   GeneratedPluginRegistrant.register(with: self)
   return super.application(application, didFinishLaunchingWithOptions: launchOptions)
 }
}
```

```
private func receiveBatteryLevel(result: FlutterResult) {
  let device = UIDevice.current
  device.isBatteryMonitoringEnabled = true
  if device.batteryState == UIDevice.BatteryState.unknown {
    result(FlutterError(code: "UNAVAILABLE",
                        message: "Battery info unavailable",
                        details: nil))
  } else {
    result(Int(device.batteryLevel * 100))
  }
}


batteryChannel.setMethodCallHandler({
  [weak self] (call: FlutterMethodCall, result: FlutterResult) ->
Void in
  // Note: this method is invoked on the UI thread.
  guard call.method == "getBatteryLevel" else {
    result(FlutterMethodNotImplemented)
    return
  }
  self?.receiveBatteryLevel(result: result)
})
```

Android side

```
class MainActivity: FlutterActivity() {
  private val CHANNEL = "samples.flutter.dev/battery"

  override fun configureFlutterEngine(@NonNull flutterEngine: FlutterEngine) {
    super.configureFlutterEngine(flutterEngine)
    MethodChannel(flutterEngine.dartExecutor.binaryMessenger, CHANNEL).setMethodCallHandler {
      call, result ->
      // Note: this method is invoked on the main thread.
      // TODO
    }
  }
}
```

```
 private fun getBatteryLevel(): Int {
    val batteryLevel: Int
    if (VERSION.SDK_INT >= VERSION_CODES.LOLLIPOP) {
      val batteryManager = getSystemService(Context.BATTERY_SERVICE) as BatteryManager
      batteryLevel =
batteryManager.getIntProperty(BatteryManager.BATTERY_PROPERTY_CAPACITY)
    } else {
      val intent = ContextWrapper(applicationContext).registerReceiver(null,
IntentFilter(Intent.ACTION_BATTERY_CHANGED))
      batteryLevel = intent!!.getIntExtra(BatteryManager.EXTRA_LEVEL, -1) * 100 /
intent.getIntExtra(BatteryManager.EXTRA_SCALE, -1)
    }

    return batteryLevel
  }
```

```
MethodChannel(flutterEngine.dartExecutor.binaryMessenger, CHANNEL).setMethodCallHandler {
    // Note: this method is invoked on the main thread.
    call, result ->
    if (call.method == "getBatteryLevel") {
      val batteryLevel = getBatteryLevel()

      if (batteryLevel != -1) {
        result.success(batteryLevel)
      } else {
        result.error("UNAVAILABLE", "Battery level not available.", null)
      }
    } else {
      result.notImplemented()
    }
  }
```