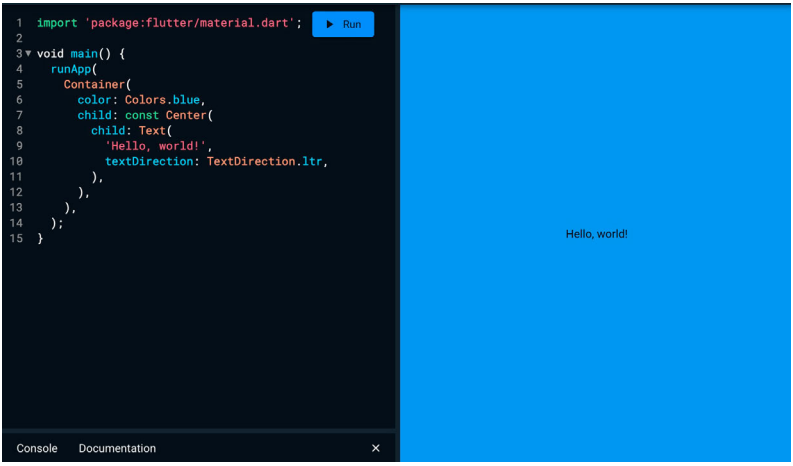
図3.1 DartPadでアプリを実行した様子



ここでサンプルコードについて解説します。最初のimportから始まる行はパッケージ(パッケージについては第4章で解説します)からライブラリに定義されたクラスや関数を利用可能にするものです。ここではFlutterのマテリアルデザインに準拠したアプリを開発するためのクラスや関数をインポートしています。

続いてDartコードのエントリポイントとなるmain関数があります。そして、runApp関数[注1]の引数はContainer(矩形の描画)、Center(中央配置のレイアウト)、Text(文字の描画)の3つが階層構造になっています(**図3.2**)。

図3.2 ウィジェットの階層構造

```
Container(
  color: Colors.blue,
  child:
    const Center(
      child:
        Text(
          'Hello, world!',
          textDirection: TextDirection.ltr,
        ),
    ),
),
```

この3つはすべてウィジェット(Widget)と呼ばれるオブジェクトです。FlutterアプリのUIはこのウィジェットの階層構造(ウィジェットツリー)をも

注1 runApp関数は引数に与えられたウィジェットを画面全体に適用する関数です。

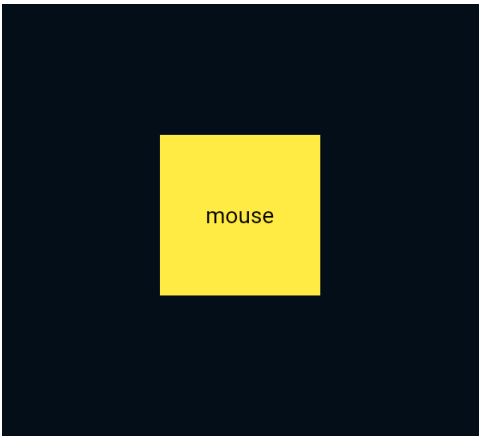
とに作られます。ほとんどのウィジェットは次の2つのクラスに分類できます。状態を持たないStatelessWidgetと、状態を持つStatefulWidgetです。

3.2 状態を持たないWidget —— StatelessWidget

状態を持たないウィジェットクラスを**StatelessWidget**と言います。まずは次のサンプルコードをDartPadで実行してみましょう。

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

void main() {
  runApp(
    Center(
      child: Container(
        color: Colors.yellow,
        width: 100,
        height: 100,
        child: const Center(
          child: Text(
            'mouse',
            textDirection: TextDirection.ltr,
          ),
        ),
      ),
    ),
  );
}
```

図3.3　**黄色いネズミを配置した実行結果**



画面中央に黄色の四角形、「mouse」というテキストを配置しました(**図3.3**)。

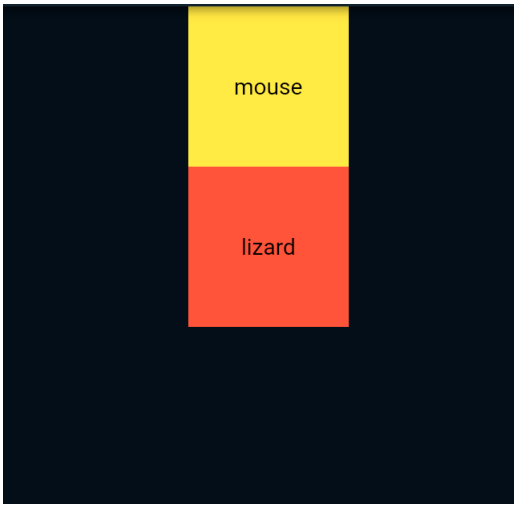
同じ要領で他の動物を追加してみたいと思います。**runApp**関数に直接引数で渡されている**Center**ウィジェットを**Column**という垂直なレイアウトをするウィジェットに変更します(❶)。続いて、**Column**の**children**配列に、赤いトカゲを追加します(❷)。

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

void main() {
  runApp(
    Column( ——❶
      children: [
        Container(
          color: Colors.yellow,
          width: 100,
          height: 100,
          child: const Center(
            child: Text(
              'mouse',
              textDirection: TextDirection.ltr,
            ),
          ),
        ),
        Container(                                    ┐
          color: Colors.red,                          ├❷
          width: 100,                                 │
          height: 100,                                ┘
```

```
          child: const Center(
            child: Text(
              'lizard',
              textDirection: TextDirection.ltr,  ❷
            ),
          ),
        ),
      ],
    ),
  );
}
```

図3.4のように黄色いネズミと赤いトカゲが並びました。

図3.4　**黄色いネズミと赤いトカゲが並んだ実行結果**



独自のStatelessWidgetを定義する

ここで、Columnの子ウィジェットがテキストと色以外が同じ冗長なものになりました。テキストと色をパラメータにウィジェットを構成する機能があると共通化できそうです。

StatelessWidgetを利用して実現してみましょう。

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

void main() {
```

```
  runApp(
    const Column(
      children: [
        AnimalView(
          text: 'mouse',
          color: Colors.yellow,
        ),
        AnimalView(
          text: 'lizard',
          color: Colors.red,
        ),
      ],
    ),
  );
}

class AnimalView extends StatelessWidget {
  const AnimalView({super.key, required this.text, required this.color});

  final String text;
  final Color color;

  @override
  Widget build(BuildContext context) {
    return Container(
        color: color,
        width: 100,
        height: 100,
        child: Center(
          child: Text(
            text,
            textDirection: TextDirection.ltr,
          ),
        ),
    );
  }
}
```

❶

❷

StatelessWidgetを継承したAnimalViewクラスを実装しました(❷)。Columnの子ウィジェットが再利用可能になったことによってrunAppの引数がシンプルになりました(❶)。

AnimalViewクラスはクラス変数にテキストと色を持ち、コンストラクタ引数で渡されます。

オーバーライドしたbuildメソッドでこのウィジェットのUIを構成します。

ちなみにコンストラクタの第一引数のKeyは、フレームワークがウィジェ

ットのライフサイクルを判断する際に用いられるオブジェクトです(第9章で詳しく解説します)。多くのケースでは省略(null)で問題ありません。ウィジェットのコンストラクタは名前付き引数、第一引数を**Key**型とすることが慣習とされています。

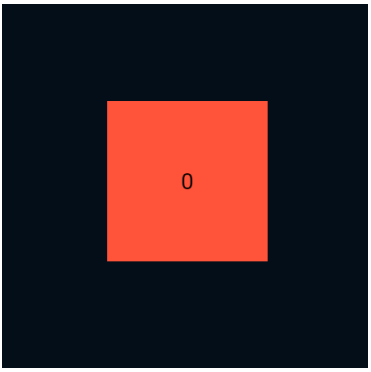
3.3 状態を持つウィジェット —— StatefulWidget

状態を持ち、自身で表示を更新ができるウィジェットクラスを**StatefulWidget**と言います。サンプルコードをDartPadで実行してみましょう。解説のためサンプルコードはStatelessWidgetからスタートします。

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

void main() {
  runApp(
    const Center(
      child: Counter(),
    ),
  );
}

class Counter extends StatelessWidget {
  const Counter({super.key});

  final _count = 0;

  @override
  Widget build(BuildContext context) {
    return Container(
      color: Colors.red,
      width: 100,
      height: 100,
      child: Center(
        child: Text(
          '$_count',
          textDirection: TextDirection.ltr,
        ),
      ),
    );
  }
}
```

赤い四角形の中にテキスト「0」が描画されました(**図3.5**)。

図3.5 **赤い四角形の中に0が描画された実行結果**



Widgetのタップ操作を検知する

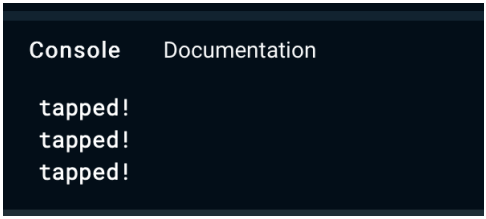
今回のサンプルは、赤い四角形をタップすると数字がカウントアップするアプリにしたいと思います。タップ操作を検知するために`Container`ウィジェットを`GestureDetector`ウィジェットで包みます(❶)。

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

void main() {
  runApp(
    const Center(
      child: Counter(),
    ),
  );
}

class Counter extends StatelessWidget {
  const Counter({super.key});

  final _count = 0;

  @override
  Widget build(BuildContext context) {
    return GestureDetector( ——❶
      onTap: () {
        print('tapped!');
      },
```

```
      child: Container(
        color: Colors.red,
        width: 100,
        height: 100,
        child: Center(
          child: Text(
            '$_count',
            textDirection: TextDirection.ltr,
          ),
        ),
      ),
    );
  }
}
```

赤い四角形をタップするとコンソールにメッセージが表示されます(**図3.6**)。

図3.6 **コンソールに出力されるメッセージ**



StatefulWidgetを継承する

続いて、タップした際に数字をカウントアップするように変更を加えたいと思います。

クラス変数`_count`は`final`で宣言されているため変更することはできません。仮に`_count`を`int _count = 0;`のように宣言したとしても、変化した`_count`の値に追従して画面が更新されることはありません。このようなケースではStatefulWidgetを採用します。

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

void main() {
  runApp(
    const Center(
      child: Counter(),
    ),
```