./lib/app_notifier_provider.dart

```
import 'dart:convert';

import 'package:hiragana_converter/app_state.dart';
import 'package:hiragana_converter/data.dart';
import 'package:riverpod_annotation/riverpod_annotation.dart'; ——❶
import 'package:http/http.dart' as http;

part 'app_notifier_provider.g.dart';

@riverpod ——❷
class AppNotifier extends _$AppNotifier { ——❸
  @override
  AppState build() {
    return const Input(); ——❹
  }

  void reset() {
    state = const Input(); ——❺
  }

  Future<void> convert(String sentence) async {
    state = const Loading(); ——❻

    final url = Uri.parse('https://labs.goo.ne.jp/api/hiragana');
    final headers = {'Content-Type': 'application/json'};
    final request = Request(
      appId: const String.fromEnvironment('appId'),
      sentence: sentence,
    );
    final response = await http.post(
      url,
      headers: headers,
      body: jsonEncode(request.toJson()),
    );
    final result = Response.fromJson(
      jsonDecode(response.body) as Map<String, Object?>,
    );

    state = Data(result.converted); ——❼
  }
}
```

AppNotifierという名前のNotifierProviderを定義しました(❸)。Providerの宣言部分はriverpod_generatorパッケージに生成させるため、@riverpodアノテーションを付与しています(❷)。また、アノテーションを参照するため

riverpod_annotationパッケージをインポートしています(❶)。

初期状態を提供するbuildメソッドでは、Inputオブジェクトを返すようにしました(❹)。これはアプリ起動時は入力状態であるためです。

resetメソッドでは、Inputオブジェクトをstateに代入しています(❺)。これは変換結果を表示した後、再度入力状態に戻すときに呼び出すことを想定しています。

convertメソッドはInputFormウィジェットの「変換」ボタンをタップしたときに呼び出される処理を実装しています。❻で、stateにLoadingオブジェクトを代入し、アプリの状態をWeb APIのレスポンス待ちに変更しています。❼で、stateにDataオブジェクトを代入し、アプリの状態をWeb APIのレスポンスを受け取った状態に変更しています。それ以外は先ほどInputFormウィジェットに直接実装したコードと同じです。

これでアプリの状態管理のしくみは整いました。

8.8 状態に応じて表示を切り替える

最後に、アプリの状態に応じて表示を切り替える実装をします。

レスポンス待ち状態のウィジェットを実装する

Web APIのレスポンス待ちの表示レイアウトを作成します。libフォルダの配下にloading_indicator.dartという新しいファイルを追加し、以下のコードを記述します。

./lib/loading_indicator.dart

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

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

  @override
  Widget build(BuildContext context) {
    return const Center(
      /* ◆ CircularProgressIndicator
      回転アニメーションする円形のインジケータWidget */
      child: CircularProgressIndicator(),
```

```
    );
  }
}
```

StatelessWidgetを継承したLoadingIndicatorウィジェットを実装しました。中央にインジケータを配置したシンプルなウィジェットです。

変換完了状態のウィジェットを実装する

次にひらがなへの変換結果を表示するためのウィジェットを実装します。libフォルダの配下にconvert_result.dartという新しいファイルを追加し、以下のコードを記述します。

./lib/convert_result.dart

```
import 'package:flutter/material.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';
import 'package:hiragana_converter/app_notifier_provider.dart';

class ConvertResult extends ConsumerWidget {
  const ConvertResult({
    super.key,
    required this.sentence, ——❶
  });

  final String sentence;

  @override
  Widget build(BuildContext context, WidgetRef ref) {
    final notifier = ref.watch(appNotifierProvider.notifier);
    return Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Padding(
            padding: const EdgeInsets.symmetric(horizontal: 16),
            child: Text(sentence), ——❷
          ),
          const SizedBox(height: 20),
          ElevatedButton(
            onPressed: notifier.reset, ——❹
            child: const Text(
              '再入力',
            ),
          ),
```

❸

```
        ],
      ),
    );
  }
}
```

ConsumerWidgetを継承したConvertResultウィジェットを実装しました。変換結果の文字列をコンストラクタ引数で受け取ります(❶)。受け取った変換結果の文字列は❷で表示します。また、テキスト入力画面に戻るためのボタンを配置しました(❸)。ボタンタップ時のコールバックはAppNotifierのresetメソッドを呼び出しています(❹)。

画面の切り替えを行う

それでは、状態の変化にあわせて画面を切り替える挙動を実装していきましょう。

./lib/main.dart

```
import 'package:flutter/material.dart';
import 'package:hiragana_converter/app_state.dart';
import 'package:hiragana_converter/app_notifier_provider.dart';   ❶
import 'package:hiragana_converter/convert_result.dart';
import 'package:hiragana_converter/input_form.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';
import 'package:hiragana_converter/loading_indicator.dart';       ❶
void main() {
  runApp(
    const ProviderScope(
      child: MyApp(),                                             ❷
    ),
  );
}

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Hiragana Converter',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        useMaterial3: true,
      ),
```

```
      home: const HomeScreen(),
    );
  }
}

class HomeScreen extends ConsumerWidget { ——❸
  const HomeScreen({super.key});

  @override
  Widget build(BuildContext context, WidgetRef ref) {
    final appState = ref.watch(appNotifierProvider); ——❹
    return Scaffold(
      appBar: AppBar(
        backgroundColor: Theme.of(context).colorScheme.inversePrimary,
        title: const Text('Hiragana Converter'),
      ),
      body: switch (appState) {
        Loading() => const LoadingIndicator(),
        Input() => const InputForm(),
        Data(sentence: final sentence) => ConvertResult(sentence: sentence),
      },
    );
  }
}
```

❺

まず必要なコードをインポートし(❶)、Riverpodを利用するため、アプリのルートウィジェットを`ProviderScope`で包みました(❷)。

`HomeScreen`画面を`ConsumerWidget`を継承するように修正しました(❸)。`ConsumerWidget`を継承することで、`build`メソッドの第二引数に`WidgetRef`を受け取れるようになります。`WidgetRef`から`appNotifierProvider`を購読し(❹)、状態に応じて表示を切り替えるようにしました(❺)。表示の切り替えはswitch式で行っています。

ひらがな変換処理の呼び出しを修正する

最後に、`InputForm`ウィジェットの「変換」ボタンをタップしたときに`AppNotifier`の`convert`メソッドを呼び出すように修正します。

./lib/input_form.dart

```
// import 'dart:convert';
import 'package:flutter/material.dart';
// import 'package:hiragana_converter/data.dart';
// import 'package:http/http.dart' as http;
import 'package:flutter_riverpod/flutter_riverpod.dart';
import 'package:hiragana_converter/app_notifier_provider.dart';

class InputForm extends ConsumerStatefulWidget {
  const InputForm({super.key});

  @override
  ConsumerState<InputForm> createState() => _InputFormState();
}

class _InputFormState extends ConsumerState<InputForm> {

  final _formKey = GlobalKey<FormState>();
  final _textEditingController = TextEditingController();

  @override
  Widget build(BuildContext context) {
    return Form(
      key: _formKey,
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Padding(
            padding: const EdgeInsets.symmetric(horizontal: 16),
            child: TextFormField(
              controller: _textEditingController,
              maxLines: 5,
              decoration: const InputDecoration(
                hintText: '文章を入力してください',
              ),
              validator: (value) {
                if (value == null || value.isEmpty) {
                  return '文章が入力されていません';
                }
                return null;
              },
            ),
          ),
          const SizedBox(height: 20),
          ElevatedButton(
            onPressed: () async {
              final formState = _formKey.currentState!;
```

❶
❷
❸
❹

```
              if (!formState.validate()) {
                return;
              }
              final sentence = _textEditingController.text;
              await ref
                  .read(appNotifierProvider.notifier)
                  .convert(sentence);
            },
            child: const Text(
              '変換',
            ),
          ),
        ),
      ],
    ),
  );
 }

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

❺

InputFormウィジェットをConsumerStatefulWidgetを継承するように修正しました(❷)。これに伴って、_InputFormStateクラスをConsumerStateを継承するように修正しました(❹)。InputFormウィジェットのcreateStateメソッドの戻り値の型も変更します(❸)。必要なコードをインポートし、不要になったインポートの削除も行っておきましょう(❶)。

_InputFormStateクラスがConsumerStateを継承したため、WidgetRefにアクセスできるようになりました。「変換」ボタンのタップ時にWidgetRefからAppNotifierを取得し、convertメソッドを呼び出すように変更しました(❺)。

これでアプリの状態に応じて表示を切り替える挙動が実装できました。アプリを実行し、入力文字をひらがなに変換すると一覧の画面に遷移することを確認してください。

8.9 まとめ

入力したテキストをひらがなに変換するアプリを開発しました。文字入力とバリデーション、Web APIのリクエストやJSONの取り扱いといった実践的な機能を盛り込みました。

また、このハンズオンのポイントは次の2つです。

・アプリの状態をsealed classで実装し、簡潔にアプリの状態を表現した
・アプリの状態管理にRiverpodを利用し、ウィジェットとロジックを分離した

このハンズオンで採用したアプリの状態表現や状態管理は、筆者お勧めの設計パターンの一つです。