상태관리

상태관리

- 여러 위젯에서 공통적으로 사용하는 데이터의 변경 사항을 관리하는 경우
  - StatefulWidget의 setState 함수를 통해 상태(Stste)를 갱신하는 방식일때
    A 위젯에서 데이터가 갱신되면 B 위젯으로 전달 후 데이터를 갱신해야하는 문제 발생

Provider

- 공식사이트 : https://pub.dev/packages/provider
- 참고사이트 : https://terry1213.com/flutter/flutter-provider/#21-contextwatcht-contextreadt

> 장점
>
> 1. 데이터 생성과 소비의 분리
> 2. 공유 데이터를 쉽게 구현
> 3. 코드 간결성

- 다양한 종류의 객체 및 데이터 스트림을 관리할 수 있음
- 각 데이터의 성격이나 목적에 따라 다양한 provider를 제공
  (ListenableProvider, StreamProvider, FutureProvider 등)

1. Provider 패키지 설치
   - pubspec.yaml

```
dependencies:
  flutter:
    sdk: flutter
  # provider 추가
  provider : ^6.1.2
```

2. 데이터 생성

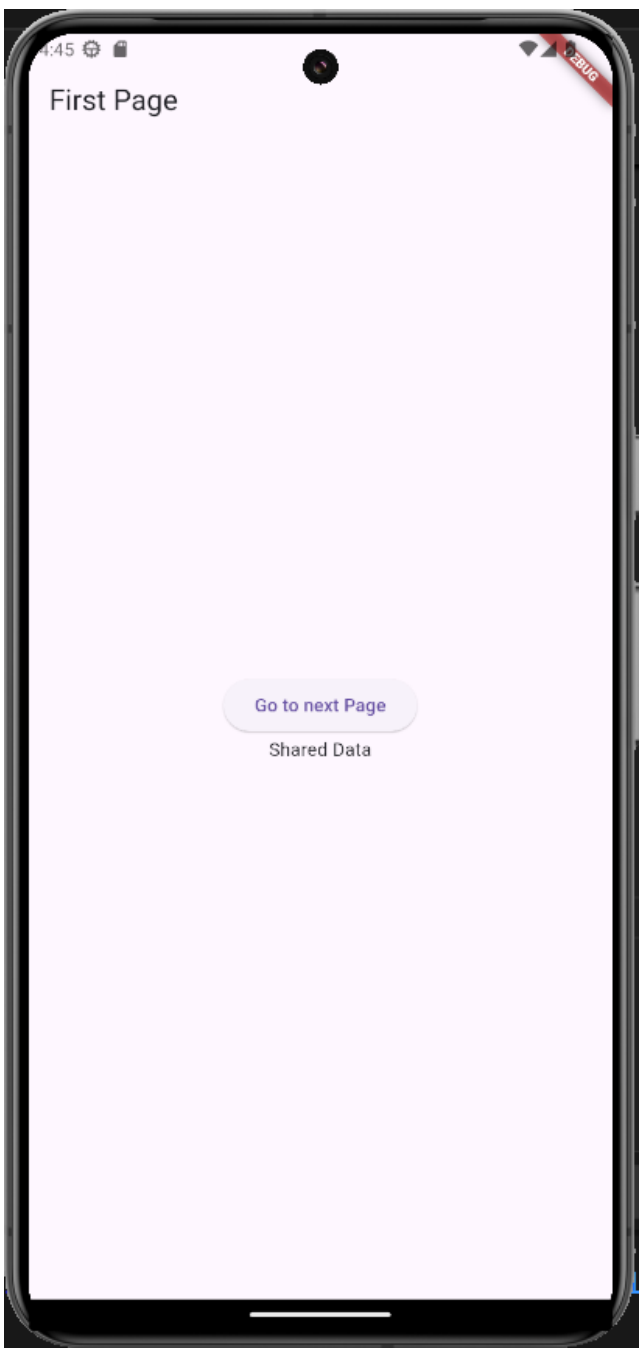
```
class WidgetDemo extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
      // Provider 생성
    return Provider<String>.value(
        // 여러 위젯과 공유할 데이터를 등록
      value : 'Shared Data',
      child : MaterialApp(
          title: 'Flutter Demo App',
          initialRoute: '/page1',
          routes: {
            '/page1' : (context) => FirstPage(),
            '/page2' : (context) => SecondPage(),
        }
      )
    );
  }
}
```

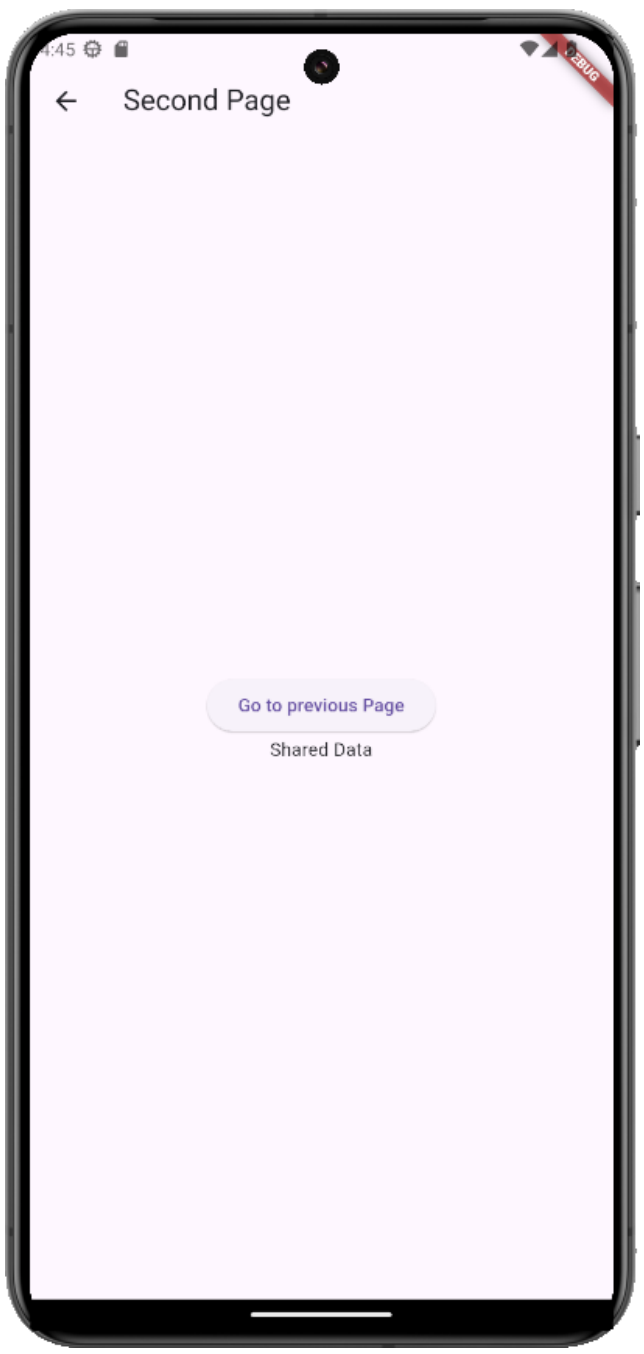
3. 데이터 소비

```
class FirstPage extends StatefulWidget {
  @override
  State<StatefulWidget> createState()=> _FirstPageState();

}

class _FirstPageState extends State<FirstPage>{
  var result;

  @override
  Widget build(BuildContext context) {
      // Provider에 등록된 데이터를 가져옴 => 변경될 경우 알림을 받도록 설정
    result = Provider.of<String>(context);
    return Scaffold(
      appBar: AppBar(
        title: Text('First Page'),
      ),
      body: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          Center(
            child: ElevatedButton(
                onPressed: () async {
                    // 버튼을 클릭할 경우 새로운 위젯을 호출 : routes에 등록된 이름
                  var data = await Navigator
                    .pushNamed(context , '/page2');
                  },
                child: Text('Go to next Page'),
              ),
          ),
          Text('$result')
        ],
      )
    );
  }
}

class SecondPage extends StatelessWidget{

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Second Page'),
      ),
    body: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: <Widget>[
          Center(
            child: ElevatedButton(
                    // 버튼을 클릭할 경우 현재 위젯을 제거
                onPressed: ()=> Navigator.pop(context),
                child: Text('Go to previous Page'),
            ),
          ),
          // Consumer을 통해 위젯을 감쌈
```

```
            Consumer<String>(
              builder : (context, value, child){
                return Text('$value');
              }
            )
          ]
        )
      );
  }
}
```

4. 실행결과





ChangeNotifierProvider

- ChangeNotifier 객체를 다루는 Provider

> ChangeNotifier : 위젯을 상태관리가 가능하도록 만들어주는 클래스

1. 데이터 생성

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

class Counter with ChangeNotifier {
  // 공통으로 사용할 데이터를 설정
  int _count = 0;

  int get count => _count;
```

```
  void incrementCount() {
    _count++;

    // 데이터가 변경되었음을 ChangeNotifierProvider에 알려주기 위해 호출
    notifyListeners();
  }
}
```

2. 데이터 소비

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'models/counter.dart';

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      home: ChangeNotifierProvider<Counter>(
          //provider가 제공된 MyHomePage하위의 모든 위젯은 provider에 접근이 가능하다
          create: (_) => Counter(), //bulider -> create
          child: const MyHomePage(title: 'Flutter Demo Home Page')),
    );
  }
}

class MyHomePage extends StatelessWidget {
  const MyHomePage({Key? key, required this.title}) : super(key: key);

  final String title;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(title),
      ),
      body: Center(
        child:Text('현재 숫자 : ${context.watch<Counter>().count}'),
           //Text( '현재 숫자: ${Provider.of<Counter>(context).count}'),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: ()
            => Provider.of<Counter>(context, listen: false).incrementCount(),
           //  => context.read<Counter>().incrementCount(),
        tooltip: 'Increment',
        child: const Icon(Icons.add),
      ),
    );
```

```
   }
 }
```

1. context.watch<T>() / context.read<T>()

	context.watch<T>()	context.read<T>()
차이점	T의 데이터 값이 변경되었을 때 위젯을 재빌드	T의 데이터 값이 변경되었을 때 위젯을 빌드하지 않음
용도	T의 데이터 값을 화면에 보여준다	T의 데이터 값을 변경하는 등의 이벤트 처리

2. Provider.of<T>(context)

	Provider.of<T>(context)	Provider.of<T>(context, listen : false)
차이점	T의 데이터 값이 변경되었을 때 위젯을 재빌드	T의 데이터 값이 변경되었을 때 위젯을 빌드하지 않음
용도	T의 데이터 값을 화면에 보여준다	T의 데이터 값을 변경하는 등의 이벤트 처리

> 변경에 대한 알림을 받지 않기 위해서 listen 매개변수의 값을 false로 설정

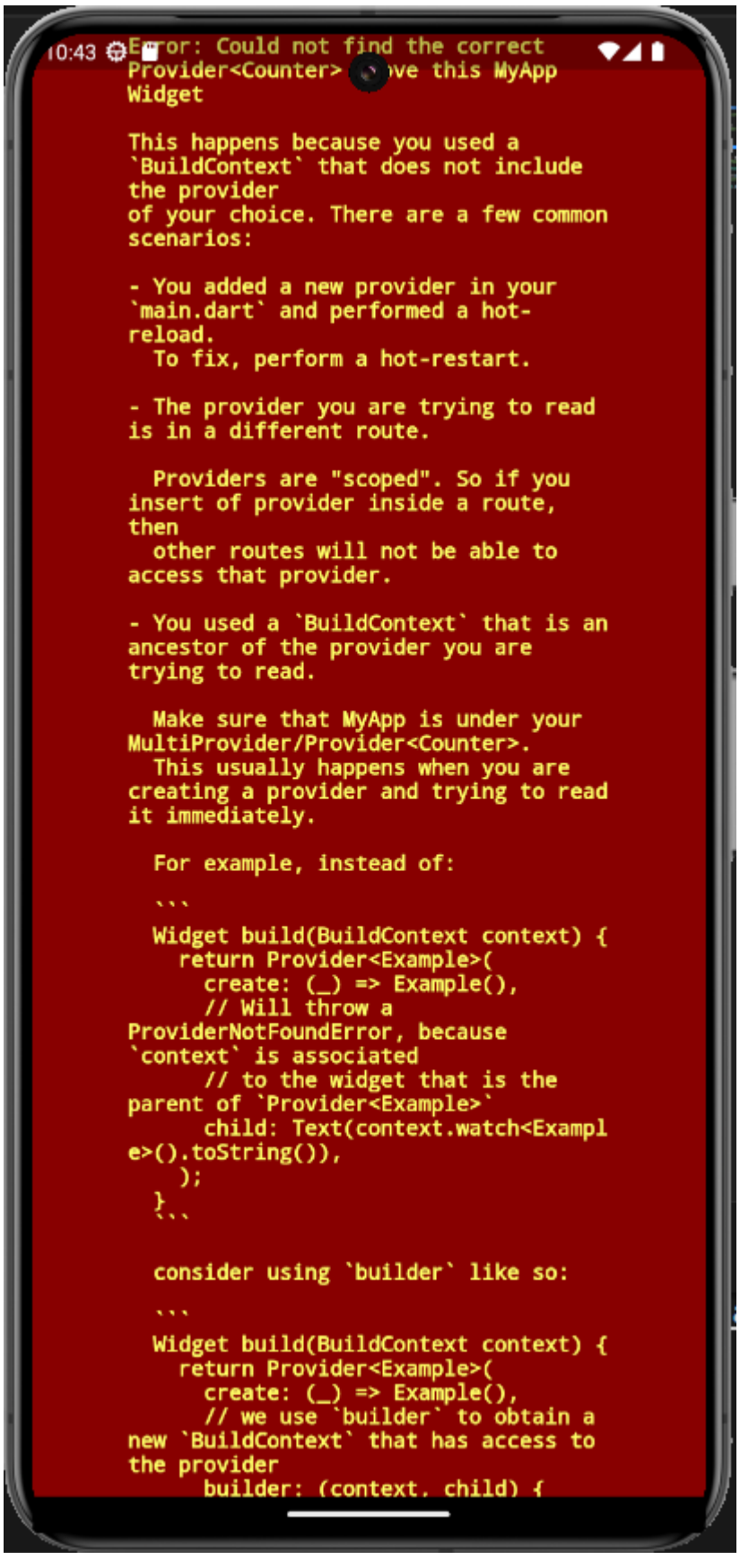
3. Consumer()

```
Consumer<CountModel>(
            //Consumer는 값을 사용할 위젯을 감싼후 빌더에서 값을 넘겨 준다.
            builder :(context, value, child) {
                    return Text('$value.count');
                }
    );
```

- 위에 설명된 context.watch<T>() , context.read<T>() 등을 사용할 수 없을 때 사용
  - 하나의 build 메소드에서 Provider를 생성하고 소비하는 경우로
    이 때는 Cousumer를 사용해야 Provider를 소비할 수 있음
- 에러가 나는 코드

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'models/counter.dart';

void main() {
  runApp(const MyApp());
}


class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return ChangeNotifierProvider<Counter>(
          create: (_) => Counter(), //bulider -> create
          child: MaterialApp(
            title: 'Flutter Demo',
            home: Scaffold(
                appBar: AppBar(
                    title: Text('Flutter Demo Home Page'),
                  ),
                body: Center(
                    child:Text(
                                '현재 숫자 : ${context.watch<Counter>().count}'
```

```
                                ),

                  ),
                floatingActionButton: FloatingActionButton(
                    onPressed: ()
                    => Provider.of<Counter>(context, listen: false)
                    .incrementCount(),
                    tooltip: 'Increment',
                    child: const Icon(Icons.add),
                  ),
              ),
          ),
      );
  }
}
```



- 정상적으로 실행되는 코드

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'models/counter.dart';
```

```
void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return ChangeNotifierProvider<Counter>(
      create: (_) => Counter(), //bulider -> create
      child : MaterialApp(
           title: 'Flutter Demo',
           home: Scaffold(
               appBar: AppBar(
                   title: Text('Flutte Demo Home Page'),
                 ),
             body: Center(
                 child:Consumer<Counter>(
                    //Consumer는 값을 사용할 위젯을 감싼 후
                    // 빌더에서 값을 넘겨 준다
                    builder: (_, counter, __) => Text(
                        // 두번째 매개변수 : Cunsumer가 사용할 오브젝트 타입
                        // (여기서는 Counter)
                        counter.count.toString(),//'$_count',
                        style: Theme.of(context).textTheme.headlineLarge,
                      ),
                  ),
               ),
             floatingActionButton: Consumer<Counter>(
               builder: (_, counter, __) => FloatingActionButton(
                 onPressed: () => counter.incrementCount(),
                 tooltip: 'Increment',
                 child: const Icon(Icons.add),
               ),
             ),
           )
       );
  }
}
```

3. 실행결과

MultiProvider

- 여러개의 Provider를 사용해야 하는 경우

1. 데이터 생성

```
// lib/models/counter.dart
import 'package:flutter/material.dart';

class Counter with ChangeNotifier {
  // 공통으로 사용할 데이터를 설정
  int _count = 0;

  int get count => _count;

  void incrementCount() {
    _count++;

    // 데이터가 변경되었음을 ChangeNotifierProvider에 알려주기 위해 호출
    notifyListeners();
  }
}
```

```
// lib/models/text.dart
import 'package:flutter/material.dart';

class Message with ChangeNotifier {
  // 공통으로 사용할 데이터를 설정
  String _msg = 'Hello!';

  String get msg => _msg;

  void changeMsg(msg) {
    this._msg = msg;

    // 데이터가 변경되었음을 ChangeNotifierProvider에 알려주기 위해 호출
    notifyListeners();
  }
}
```

2. 데이터 소비

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'models/counter.dart';
import 'models/text.dart';

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      home: MultiProvider(
          // 사용하고자 하는 여러 Provider를 등록
         providers: [
            ChangeNotifierProvider<Counter>(create: (_) => Counter()),
            ChangeNotifierProvider<Message>(create: (_) => Message()),
         ],
         child: const MyHomePage(title: 'Flutter Demo Home Page')),
    );
  }
}

class MyHomePage extends StatelessWidget {
  const MyHomePage({Key? key, required this.title}) : super(key: key);

  final String title;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(title),
      ),
```

```
      body: Column(
        children: [
          Center(
            child : Text('현재 메세지: ${Provider.of<Message>(context).msg}'),
          ),
          Text('현재 숫자 : ${context.watch<Counter>().count}'),
        ],
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: (){
          Provider.of<Counter>(context, listen: false).incrementCount();
          context.read<Message>().changeMsg('Hello, Flutter!');
        },
        tooltip: 'Increment',
        child: const Icon(Icons.add),
      ),
    );
  }
```

3. 실행결과