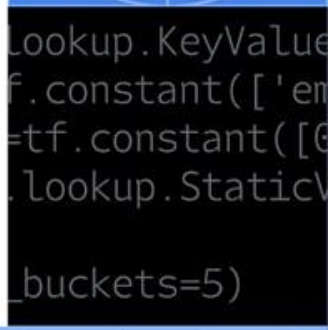
Google Developer Student Clubs
Hanyang University

App Front 3주차 발표



발표자: (App General) 박시형

1. State (상태)의 개념

① 위젯이 빌드될 때 동기적으로 읽을 수 있는 정보

② 위젯의 생명주기가 끝나기 전까지 변경될 수 있는 정보

예시 : color의 값은 Colors.white

```
Text(
    'Hey, Selena',
    style: TextStyle(
        color: Colors.white,
        fontSize: 28,
        fontWeight: FontWeight.w800,
    ), // TextStyle
), // Text
```

2. State (상태)의 구분

① App State

: 앱 전체에서 공유되고 사용자 인터페이스(UI)와 앱 동작에 영향을 미치는 데이터와 정보의 집합

〈 사용 목적 〉

: 앱의 데이터를 일관되게 유지하고 UI를 업데이트하는 동안 효율적으로 동작하는 것

〈 사용처 〉

- 로그인 정보
- 설정 옵션
- 데이터 소스
- UI 상태

2. State (상태)의 구분

② Widget State

: 위젯의 현재 상태를 나타내는 것

- 화면에 어떻게 표시되고 상호 작용하는지를 결정
- 사용자 입력, 데이터 변경 또는 앱의 다른 상황에 따라 동적으로 변할 수 있음

〈 사용 목적 〉

: 앱의 동적인 동작과 상호 작용을 구현 (앱의 상태를 관리하면서 화면을 동적으로 업데이트 가능)

〈 사용처 〉

- 사용자 입력 처리
- 동적 데이터 표시
- 앱의 다양한 화면 간 상태 공유:
- 애니메이션 및 트랜지션(화면 전환) 제어
- 다양한 화면 상태 관리
- 비즈니스 로직 처리

3. State (상태) 연결에 따른 위젯 구분

① Stateless

: 앱이 위젯의 상태를 감시하고 있을 필요가 없으므로 상태를 연결할 필요가 없는 위젯

예시: 앱을 처음 실행했을 때 나오는 도움말 페이지

〈 구현 방법 〉

: StatelessWidget 클래스를 상속받아서 만듦

3. State (상태)

① Stateless

: 앱이 위젯의 상태를 감시하고 ㅏ 없는 위젯

예시: 앱을 처음 실행했을 때 나

〈 구현 방법 〉

: StatelessWidget 클래스를 상

Example: StatelessWidget

3. State (상태) 연결에 따른 위젯

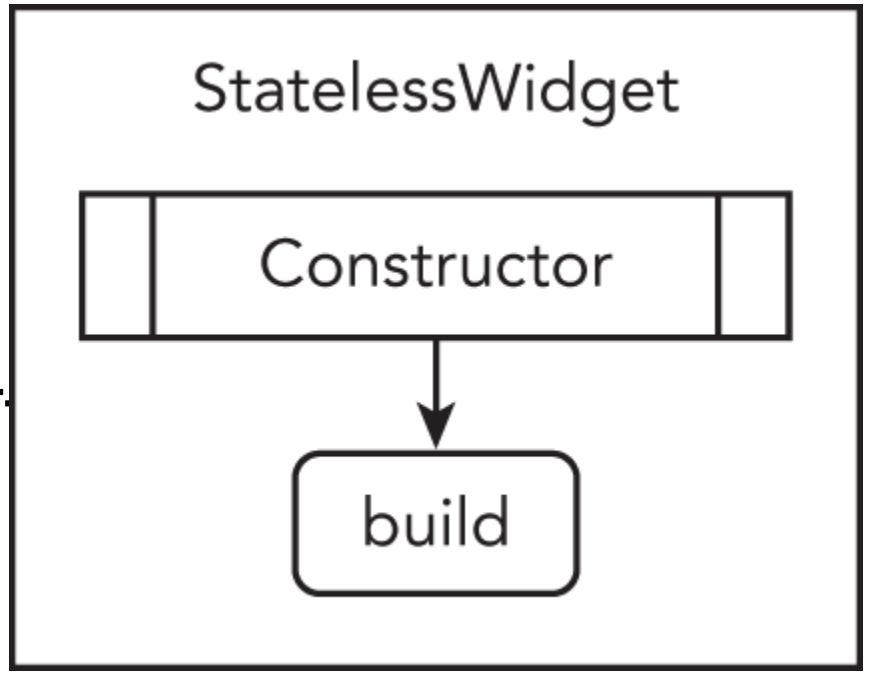
① Stateless(정적)

: 앱이 위젯의 상태를 감시하고 있을 필요가 없으므로 상태를 연결

예시: 앱을 처음 실행했을 때 나오는 도움말 페이지

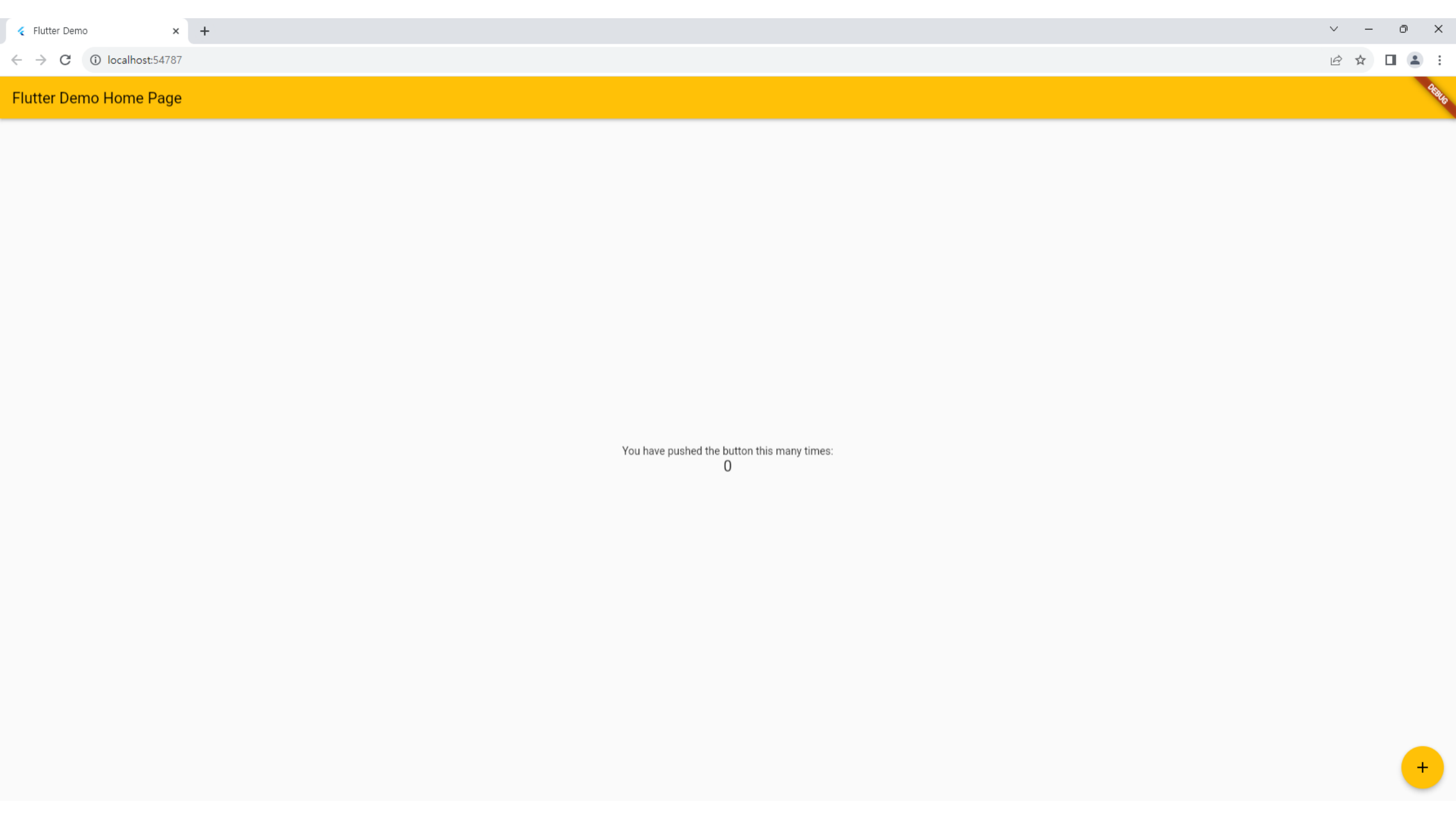
〈 구현 방법 〉

: StatelessWidget 클래스를 상

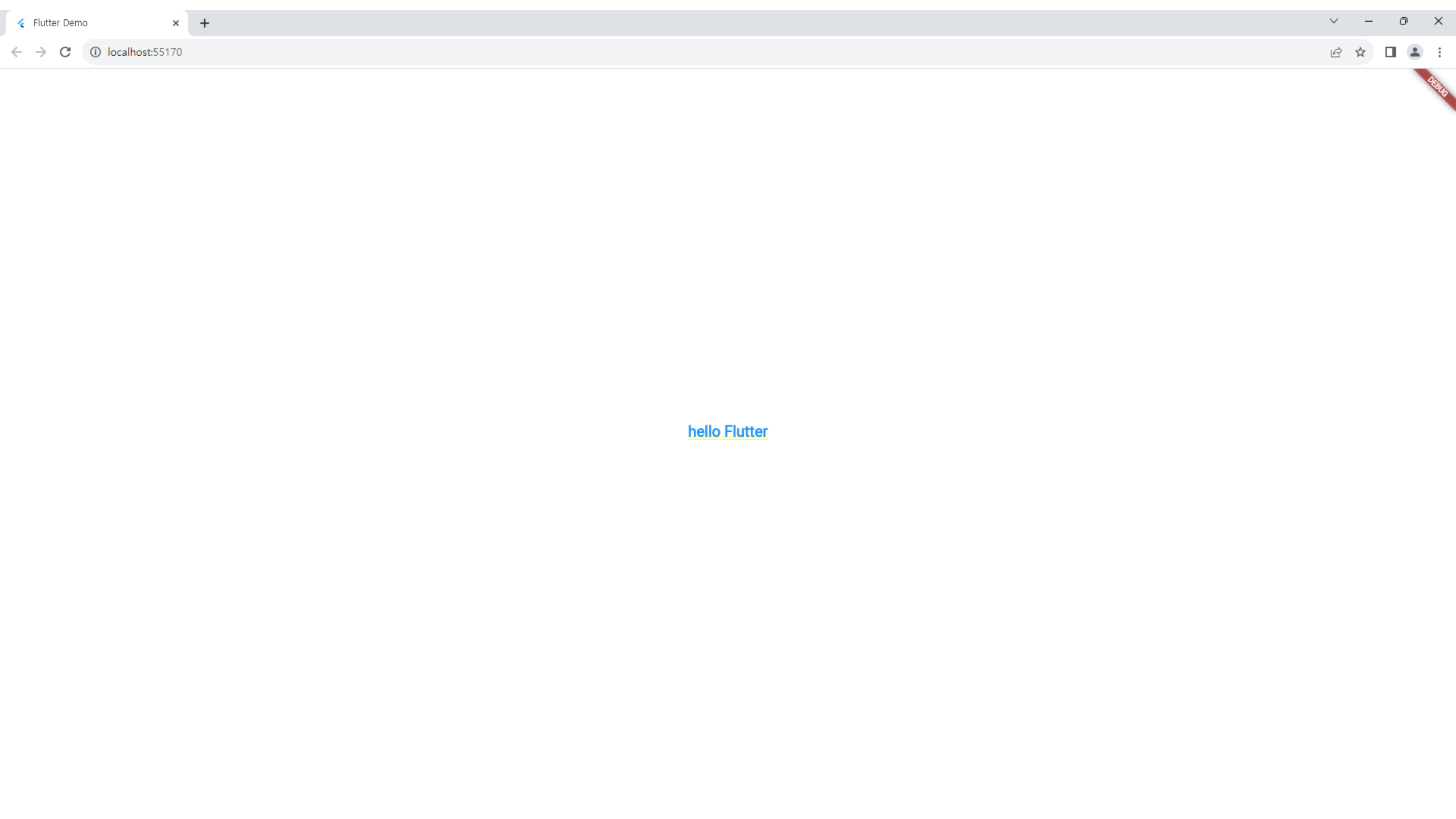


Example: StatelessWidget

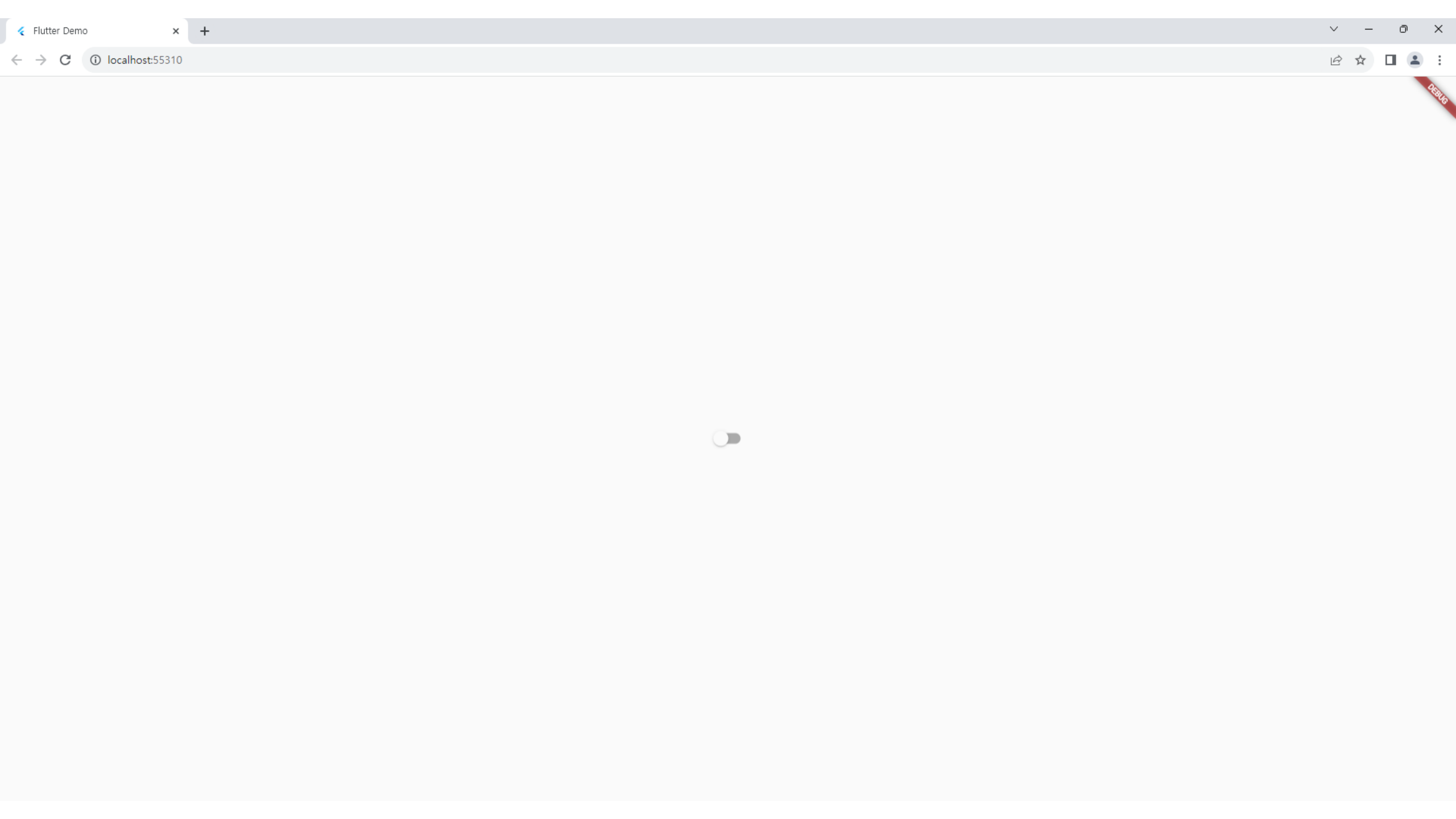
```dart
import 'package:flutter/material.dart';

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.amber,
      ),  // ThemeData
      home: MyHomePage(title: 'Flutter Demo Home Page'),
    );  // MaterialApp
  }
}

class MyHomePage extends StatefulWidget {
  MyHomePage({Key? key, required this.title}) : super(key: key);
  final String title;
  @override
  _MyHomePageState createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  int _counter = 0;

  void _incrementCounter() {
    setState(() {
      _counter++;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(widget.title),
      ),  // AppBar
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Text(
              'You have pushed the button this many times:',
            ),  // Text
            Text(
              '$_counter',
              style: Theme.of(context).textTheme.headline6,
            ),  // Text
          ],  // <Widget>[]
        ),  // Column
      ),  // Center
      floatingActionButton: FloatingActionButton(
        onPressed: _incrementCounter,
        tooltip: 'Increment',
        child: Icon(Icons.add),
      ), // This trailing comma makes auto-formatting nicer for build methods.  // FloatingActionButton
    );  // Scaffold
  }
}
```


Flutter Demo
localhost:54787
Flutter Demo Home Page
DEBUG
You have pushed the button this many times:
0
+

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.amber,
      ),  // ThemeData
      home: Container(
        color: Colors.white,
        child: Center(
          child: Text(
            'hello Flutter',
            textAlign: TextAlign.center,
            style: TextStyle(color: Colors.blue , fontSize: 20 ),
          ),  // Text
        ),  // Center
      ),  // Container
    );  // MaterialApp
  }
}
```


Flutter Demo
localhost:55170
DEBUG
hello Flutter

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

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

class MyApp extends StatelessWidget {
  var switchValue = false;
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        title: 'Flutter Demo',
        theme: ThemeData(
          primarySwatch: Colors.amber,
        ),  // ThemeData
        darkTheme: ThemeData.light(),
        home: Scaffold(
          body: Center(
            child: Switch(value : switchValue  , onChanged: (value) {
              switchValue = value;
            }),  // Switch
          ),  // Center
        )  // Scaffold
    );  // MaterialApp
  }
}
```


Flutter Demo
localhost:55310
DEBUG

3. State (상태) 연결에 따른 위젯 구분

② Stateful(동적)

: 앱이 위젯의 상태를 감시하다가 위젯이 특정 상태가 되면 알맞은 처리를 수행해야 할 때

예시: 계산기 앱에서 버튼을 누를 때마다 화면에 누른 숫자가 반영되어야 함

〈 구현 방법 〉

: StatefulWidget 클래스를 상속받아서 만듦

3. State (상태) 분

② Stateful(동적)

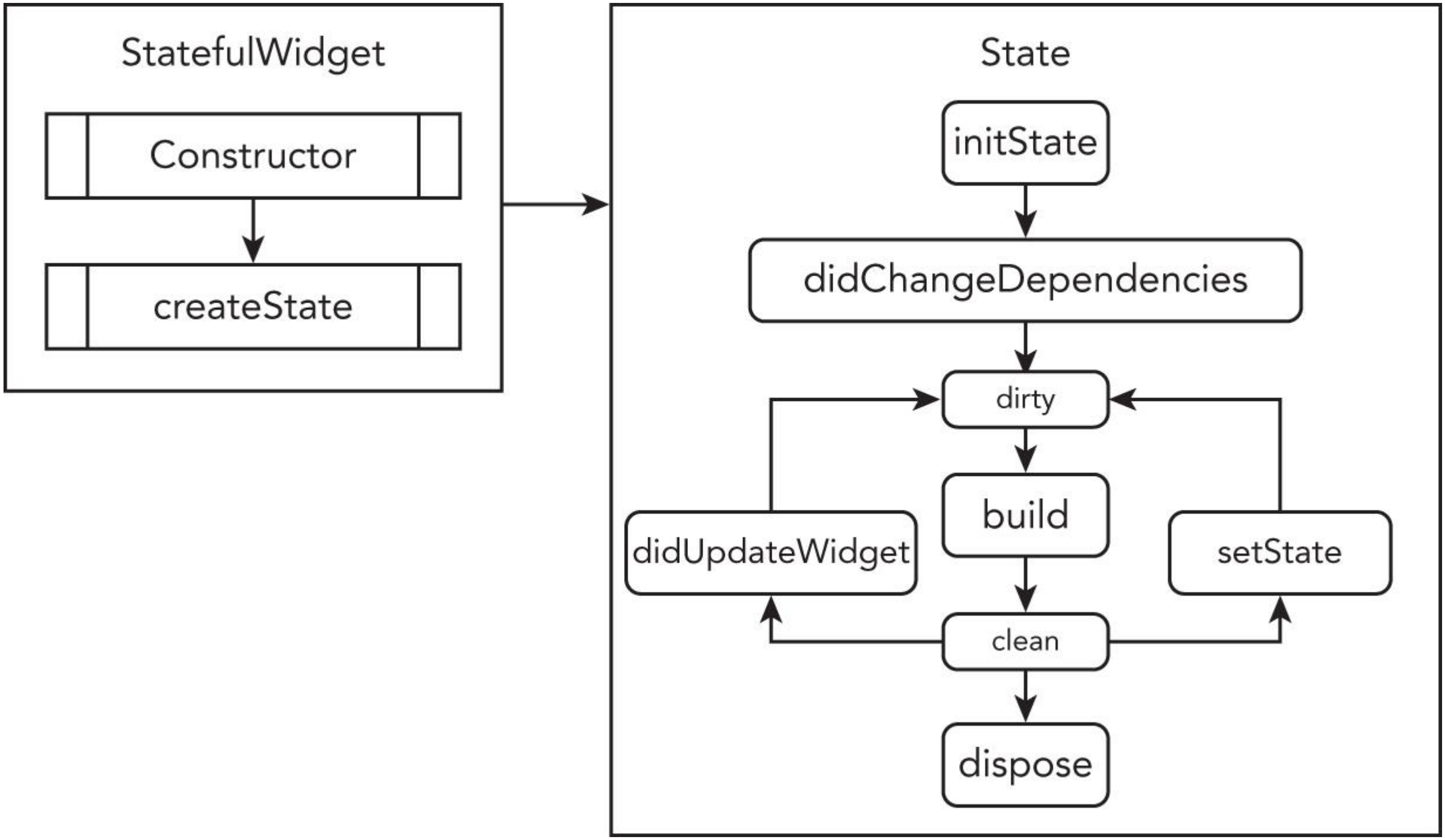
: 앱이 위젯의 상태를 감시하다 를 수행해야 할 때

예시: 계산기 앱에서 버튼을 누 야 함

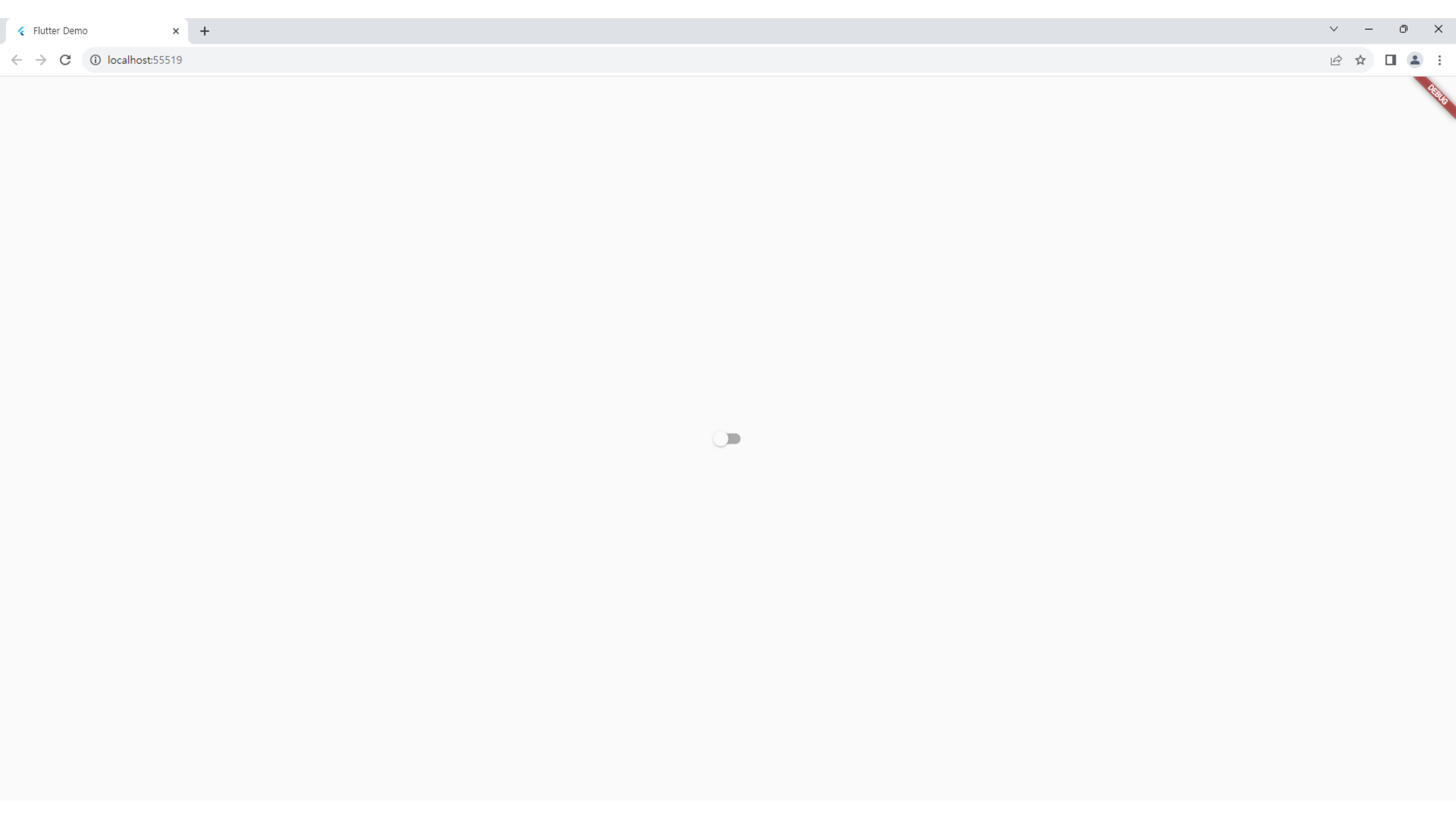
〈 구현 방법 〉

: StatefulWidget 클래스를 상

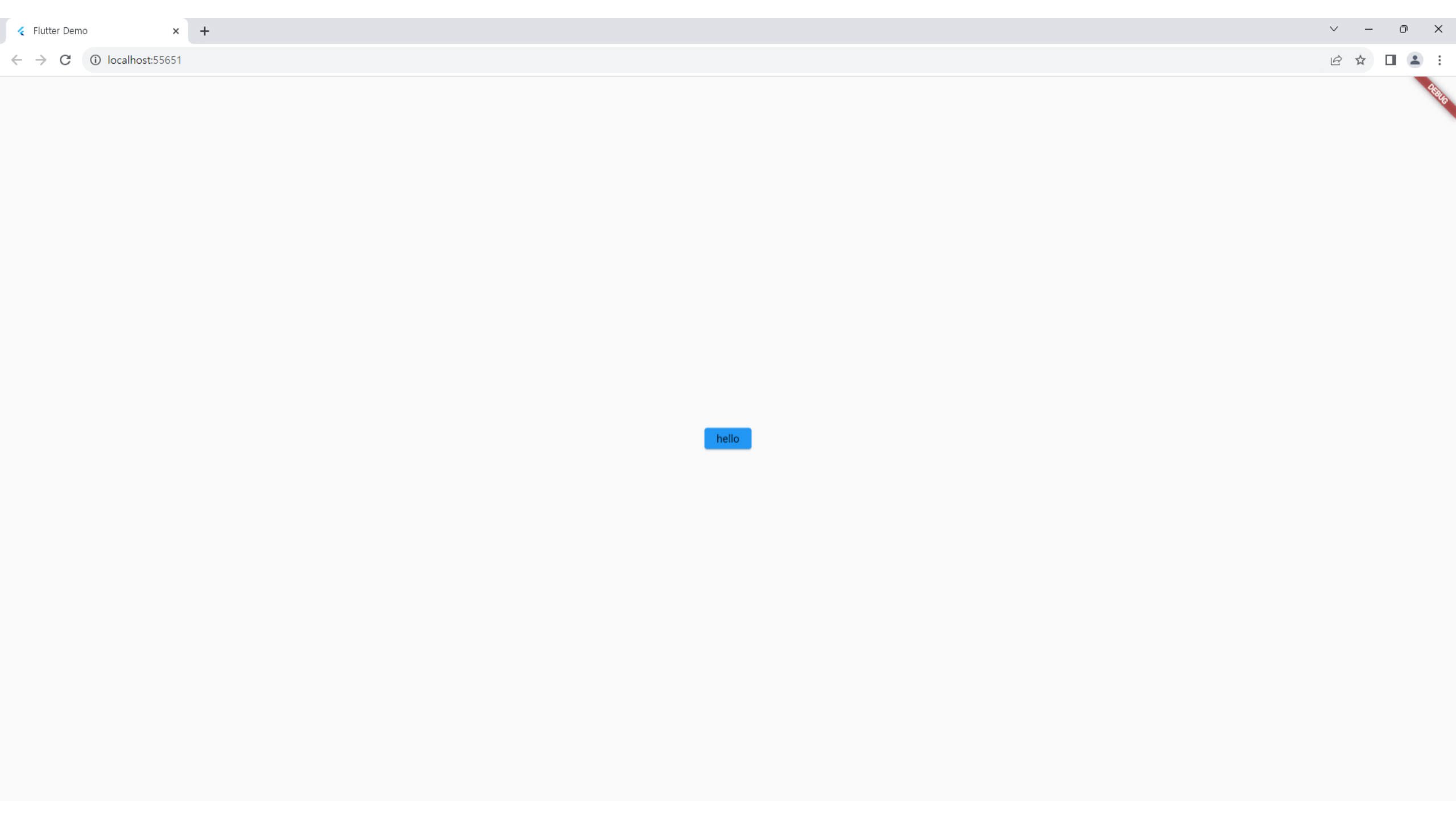
Example: StatefulWidget


StatefulWidget
Constructor
createState
State
initState
didChangeDependencies
dirty
build
didUpdateWidget
setState
clean
dispose


import 'package:flutter/material.dart';
void main() => runApp(MyApp());
class MyApp extends StatefulWidget {
@override
State<StatefulWidget> createState() {
return _MyApp();
}
}
class _MyApp extends State<MyApp>{
var switchValue = false;
@override
Widget build(BuildContext context) {
return MaterialApp(
title: 'Flutter Demo',
theme: ThemeData(
primarySwatch: Colors.amber,
), // ThemeData
darkTheme: ThemeData.light(),
home: Scaffold(
body: Center(
child: Switch(value: switchValue, onChanged: (value) {
print(value);
switchValue = value;
}), // Switch
), // Center
) // Scaffold
); // MaterialApp
}
}


Flutter Demo
localhost:55519
DEBUG

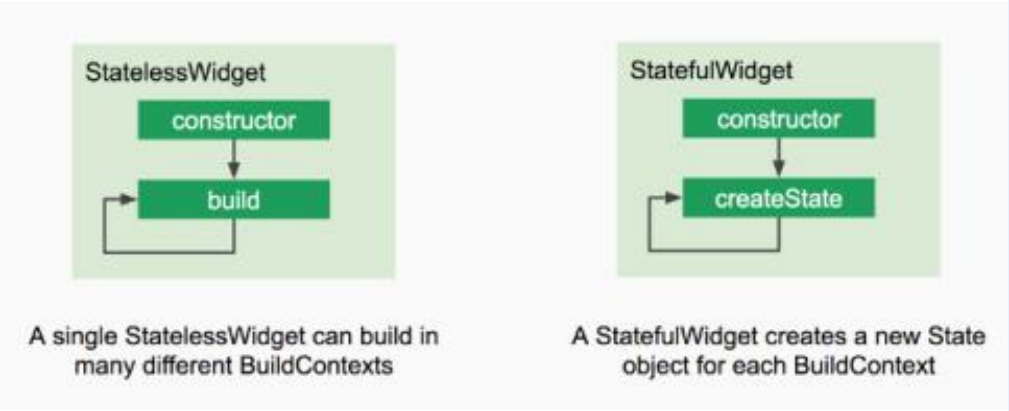
```dart
import 'package:flutter/material.dart';

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

class MyApp extends StatefulWidget {
  @override
  State<StatefulWidget> createState() {
    return _MyApp();
  }
}

class _MyApp extends State<MyApp> {
  var switchValue = false;
  String test = 'hello'; // 버튼에 들어갈 텍스트 입력
  Color _color = Colors.blue;

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        title: 'Flutter Demo',
        theme: ThemeData(
          primarySwatch: Colors.amber,
        ),  // ThemeData
        darkTheme: ThemeData.light(),
        home: Scaffold(
          body: Center(
            child: ElevatedButton(
                child: Text('$test'),
                style: ButtonStyle(backgroundColor: MaterialStateProperty.all(_color)),
                onPressed: () {
                  if (_color == Colors.blue) {
                    setState(() {
                      test = 'flutter';
                      _color = Colors.amber;
                    });
                  } else {
                    setState(() {
                      test = 'flutter';
                      _color = Colors.blue;
                    });
                  }
                }),  // ElevatedButton
          ),  // Center
        ));  // Scaffold, MaterialApp
  }
}
```

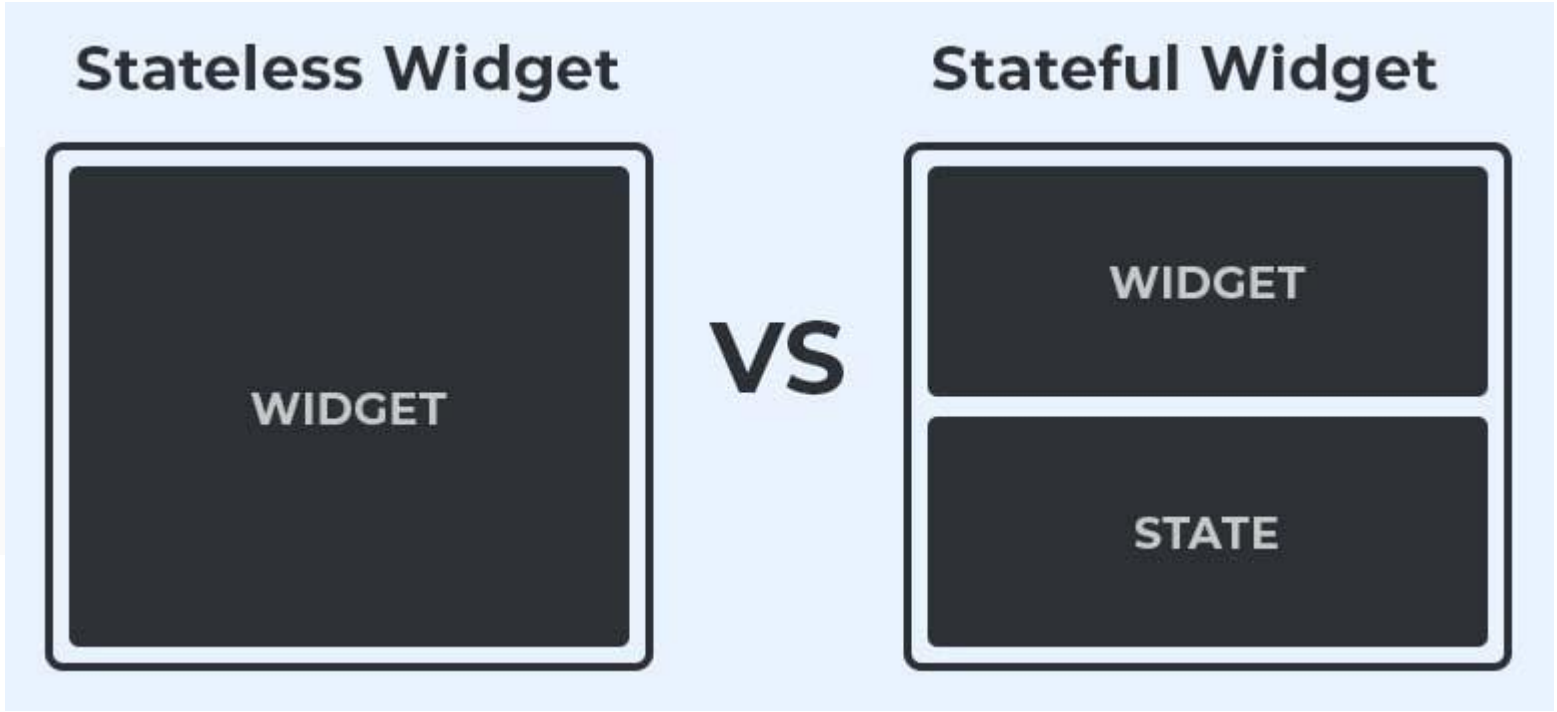

Flutter Demo
localhost:55651
DEBUG
hello

3. State (상태) 연결에 따른 위젯 구분

① StatelessWidget

: 갱신할 필요 X → 적은 자원으로 화면을 구성





② StatefulWidget

: 언제든 상태가 변경되면 특정한 처리를 수행해야 함 → 메모리, CPU 등 자원을 많이 소비

Q. 왜 State와 StatefulWidget 클래스를 나누어 놓았을까?

Q. 왜 State와 StatefulWidget 클래스를 나누어 놓았을까?

Answer: 성능 때문

: StatefulWidget보다 State 클래스가 상대적으로 더 무거움

→ StatefulWidget에서 감시하고 있다가 상태 변경 신호가 오면 State 클래스가 화면을 갱신

Q. 만약 StatefulWidget에서 바로 갱신하면?

Answer: 화면이 종료되어도 할당받은 메모리를 없앨 때까지 오랜 시간이 걸림

< 정리 >

- StatefulWidget 클래스: 상태 변경 감시 담당

- State 클래스: 실제 갱신 담당

4. 위젯의 생명주기

Q. 위젯의 생명주기를 알면?

: 언제 데이터를 주고받을지, 화면이 사라질 때 어떤 로직을 처리해야 할 지를 정리해서 넣기 가능

예시) 특정 화면에서 소리로 문서를 읽어주는데 화면을 종료해도 계속 소리가 나오면 안 됨
→ 이런 상황을 막기 위해 화면이 사라질 때 소리도 함께 멈추는 함수를 넣어야 함

〈 결론 〉
: 생명주기를 알면 앱의 동작이나 자원을 효율적으로 관리 가능하다

② StatefulWidget

: 언제든 상태가 변경되면 특정한 처리를 수행해야 함 → 메모리, CPU 등 자원을 많이 소비

4. 위젯의 생명주기

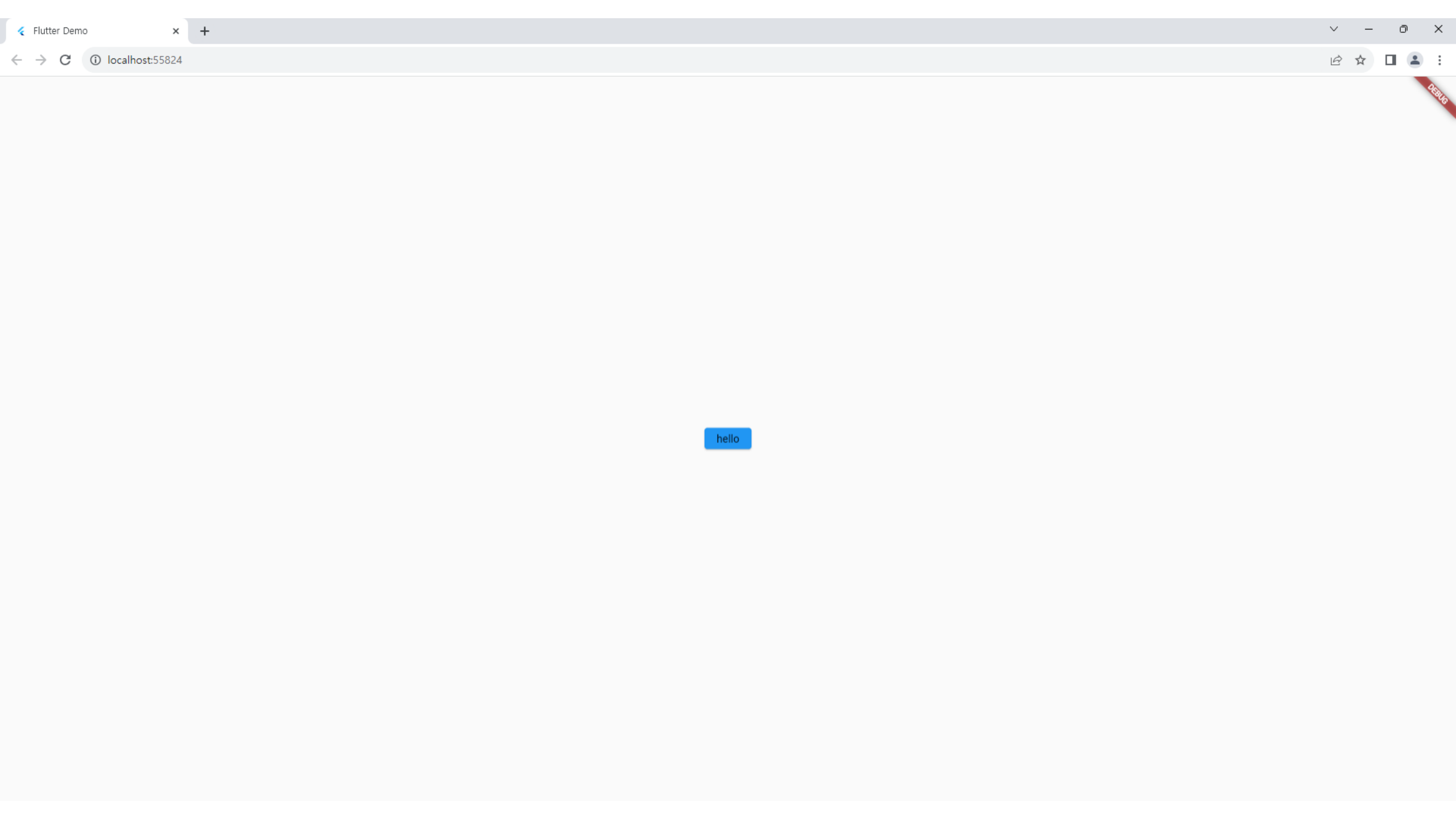
① StatelessWidget

: 한 번 만들어지면 갱신 불가능 → 생명주기가 없다 (다른 화면으로 넘어가면 모든 로직이 종료)

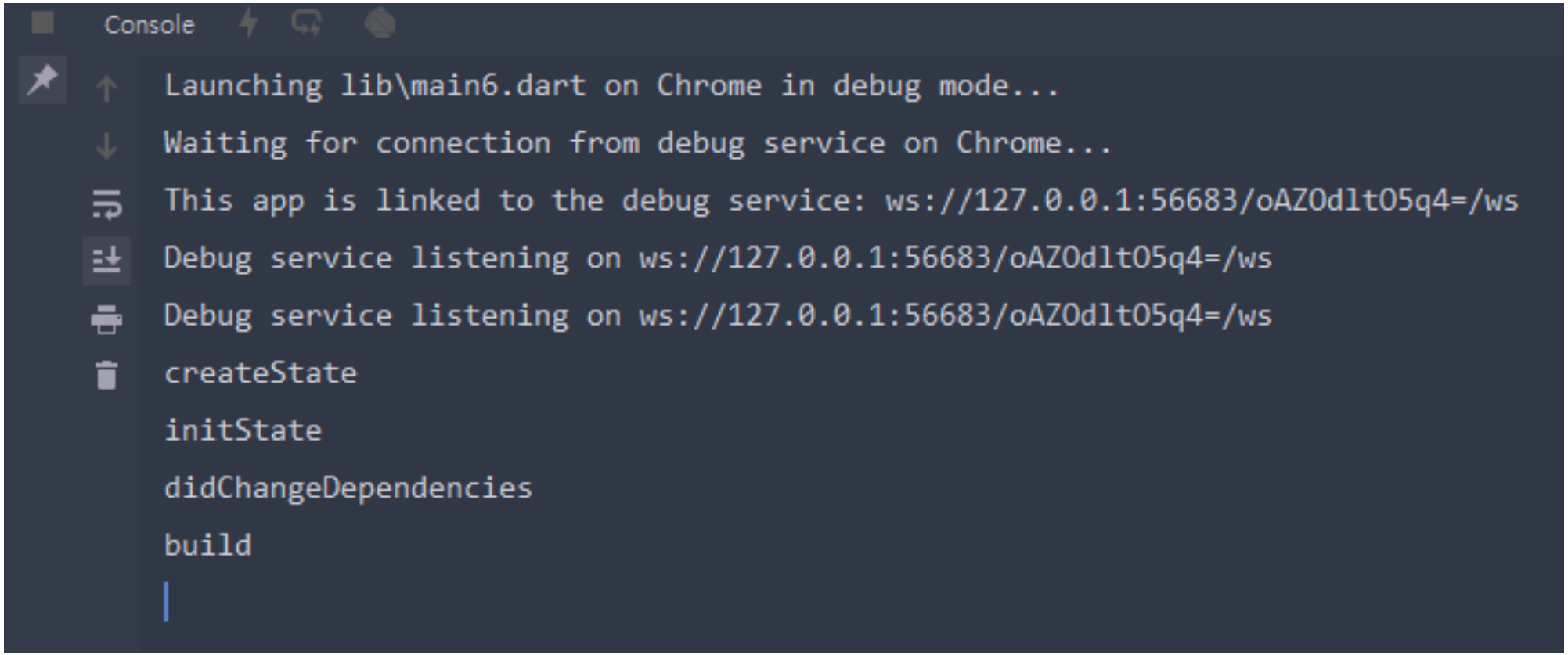
② StatefulWidget의 생명주기

: 10단계 생명주기

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

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

class MyApp extends StatefulWidget {
  @override
  State<StatefulWidget> createState() {
    print('createState');
    return _MyApp();
  }
}

class _MyApp extends State<MyApp> {
  var switchValue = false;
  String test = 'hello'; // 버튼에 들어갈 텍스트 입력
  Color _color = Colors.blue;

  @override
  void initState() {
    super.initState();
    print('initState');
  }

  @override
  void didChangeDependencies() {
    super.didChangeDependencies();
    print('didChangeDependencies');
  }

  @override
  Widget build(BuildContext context) {
    print('build');
    return MaterialApp(
        title: 'Flutter Demo',
        theme: ThemeData(
          primarySwatch: Colors.amber,
        ),  // ThemeData
        darkTheme: ThemeData.light(),
        home: Scaffold(
          body: Center(
            child: ElevatedButton(
                child: Text('$test'),
                style: ButtonStyle(backgroundColor: MaterialStateProperty.all(_color)),
                onPressed: () {
                  if (_color == Colors.blue) {
                    setState(() {
                      test = 'flutter';
                      _color = Colors.amber;
                    });
                  } else {
                    setState(() {
                      test = 'flutter';
                      _color = Colors.blue;
                    });
                  }
                }),  // ElevatedButton
          ),  // Center
        ));  // Scaffold, MaterialApp
  }
}
```


Flutter Demo
localhost:55824
DEBUG
hello

```
I/flutter (16949): createState
I/flutter (16949): initState
I/flutter (16949): didChangeDependencies
I/flutter (16949): build
```



4. 위젯의 생명주기

〈 StatefulWidget의 생명주기 〉

1. 상태를 생성하는 createState() 함수

: 다른 생명주기 함수들이 포함된 State 클래스를 반환한다 (위젯의 상태를 생성하는 함수)

```
class MyHomePage extends StatefulWidget {
  @override
  _MyHomePageState createState() => new _MyHomePageState();
}
```

4. 위젯의 생명주기

< StatefulWidget의 생명주기 >

2. 위젯을 화면에 장착하면 mounted == true

: createState() 함수가 호출됨 → mounted 속성이 true로 변경됨 → 위젯을 제어할 수 있는 buildContext 클래스에 접근 가능 → setState() 함수 이용 가능
(화면 구성도 안 되었는데 setState() 함수로 위젯을 건들 수 없기 때문)

따라서 아래처럼 setState() 함수를 호출하기 전에 mounted 속성을 점검 코드를 활용하여 좀 더 안전하게 작성할 수 있다

```
if (mounted) {
  setState()
}
```

4. 위젯의 생명주기

< StatefulWidget의 생명주기 >

3. 위젯을 초기화하는 initState() 함수

: initState() 함수는 위젯을 초기화할 때 한 번만 호출함

* 주로 데이터 목록을 만들거나 처음 필요한 데이터를 주고받을 때 호출함

```
@override
initState() {
  super.initState();
  _getJsonData();
}
```

→ initState() 함수를 호출할 때 내부에서 _getJsonData() 함수를 호출하여 서버에서 받은 데이터를 화면에 출력함

* 위젯을 초기화하는 initState() 함수에서 데이터를 준비해야 네트워크 통신이 안 되거나 데이터가 이상할 때 화면에 표시하기 전에 미리 적절하게 대응 가능

* InitState 메서드를 사용할 때 super.initState()를 호출해야 함
* initState는 항상 build 메서드 전에 실행된다

4. 위젯의 생명주기

< StatefulWidget의 생명주기 >

4. 의존성이 변경되면 호출되는 didChangeDependencies() 함수

: initState() 함수 호출 후 바로 호출되는 함수

- 데이터에 의존하는 위젯일 경우 화면에 표시하기 전에 꼭 호출해야 함

- 주로 상속받은 위젯을 사용할 때 피상속자가 변경되면 호출함

4. 위젯의 생명주기

〈 StatefulWidget의 생명주기 〉

5. 화면에 표시하는 build() 함수

: Widget을 반환하는 함수 (위젯에 화면을 렌더링함)

- build() 함수에서 위젯을 만들고 반환을 해야 화면에 표시가 됨

```
@override
Widget build(BuildContext context) {
  return MaterialApp(
    title: 'Flutter Demo',
    theme: ThemeData(
      primarySwatch: Colors.amber,
    ),
    home: MyHomePage(title: 'Flutter Demo Home Page'),
  );
}
```

4. 위젯의 생명주기

< StatefulWidget의 생명주기 >

6. 위젯을 갱신하는 didUpdateWidget() 함수

: 부모 위젯이나 데이터가 변경되어 위젯을 갱신해야 할 때 호출함 (화면 이동)

- initState()에서 특정 이벤트에 의해 위젯이 변경될 경우 didUpdateWidget() 함수를 호출해 위젯을 갱신 가능

- initState() 함수는 위젯을 초기화할 때 한 번만 호출되므로 위젯이 변경되었을 때 호출하는 didUpdateWidget() 함수를 필요로 함

```
@override
void didUpdateWidget(Widget oldWidget) {
  if (oldWidget.importantProperty != widget.importantProperty) {
    _init();
  }
}
```

4. 위젯의 생명주기

< StatefulWidget의 생명주기 >

7. 위젯의 상태를 갱신하는 setState() 함수

: 데이터가 변경되었다는 것을 알려주고 변경된 데이터를 이용해 화면의 UI를 변경 가능

- 플러터 앱을 만들어서 앱의 화면을 구성하므로 제일 많이 호출되는 함수

```
void updateProfile(String name) {
  setState(() => this.name = name);
}
```

4. 위젯의 생명주기

< StatefulWidget의 생명주기 >

8. 위젯의 상태 관리를 중지하는 deactivate() 함수

: State 객체가 플러터의 구성 트리로부터 제거될 때 호출됨

- 단, State 객체가 제거된다고 해당 메모리까지 지워지진 않음

- deactivate() 함수를 호출해도 dispose() 함수를 호출하기 전까지는 State 객체를 사용 가능함

4. 위젯의 생명주기

< StatefulWidget의 생명주기 >

9. 위젯의 상태 관리를 완전히 끝내는 dispose() 함수

: State 객체를 영구적으로 소멸할 때 호출함 (해당 위젯을 종료한다는 뜻)

예시) 네트워크 통신을 하다가 dispose() 함수를 호출하면 데이터 전송을 종료함

- 위젯을 소멸할 때 꼭 호출해야 하는 함수라면 dispose() 함수 안에서 호출해야 함

- deactivate() 함수 호출로 State 객체를 트리에서 제거 후 같은 State를 다른 트리에서 재사용할 경우 dispose() 함수를 호출되지 않음

4. 위젯의 생명주기

< StatefulWidget의 생명주기 >

10. 위젯을 화면에서 제거하면 mounted == false

: State 객체 소멸 후 마지막으로 mounted 속성이 false로 되면서 생명주기가 끝남

- mounted 속성이 false → State를 재사용 불가능 → setState() 함수 호출 불가능제거 후 같은 State를 다른 트리에서 재사용할 경우 dispose() 함수를 호출되지 않음

4. 위젯의 생명주기 표 정리

호출 순서	생명주기	내용
1	createState()	처음 스테이트풀을 시작할 때 호출
2	mounted == true	createState() 함수가 호출되면 mounted는 true
3	initState()	State에서 제일 먼저 실행되는 함수. State 생성 후 한 번만 호출
4	didChangeDependencies()	initState() 호출 후에 호출되는 함수
5	build()	위젯을 렌더링하는 함수. 위젯을 반환
6	didUpdateWidget()	위젯을 변경해야 할 때 호출하는 함수
7	setState()	데이터가 변경되었음을 알리는 함수. 변경된 데이터를 UI에 적용하기 위해 필요
8	deactivate()	State가 제거될 때 호출
9	dispose()	State가 완전히 제거되었을 때 호출
10	mounted == false	모든 프로세서가 종료된 후 mounted가 false로 됨

감사합니다

T h a n k y o u