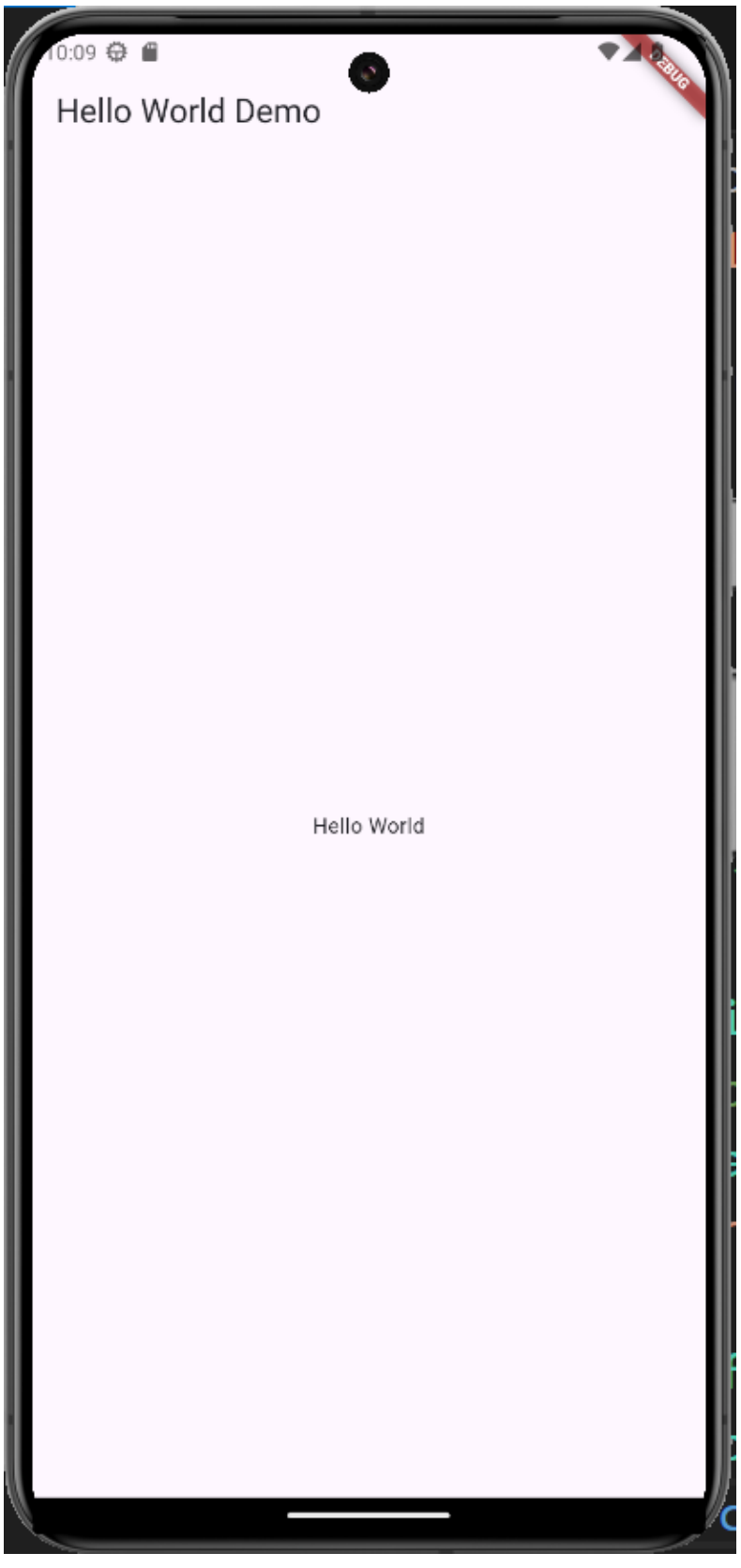
위젯의 기본개념

위젯의 기본개념

- 위젯
  - 앱을 구성하는 가장 기본적인 단위
  - 버튼, 텍스트와 같은 사용자와 상호작용하는 요소뿐만 아니라
    패딩이나 마진처럼 눈에 보이지 않는 요소도 모두 위젯으로 처리
  - 플러터는 위젯을 조합하는 형태로 앱을 구현

Hello World 앱 분석

- 실행 화면

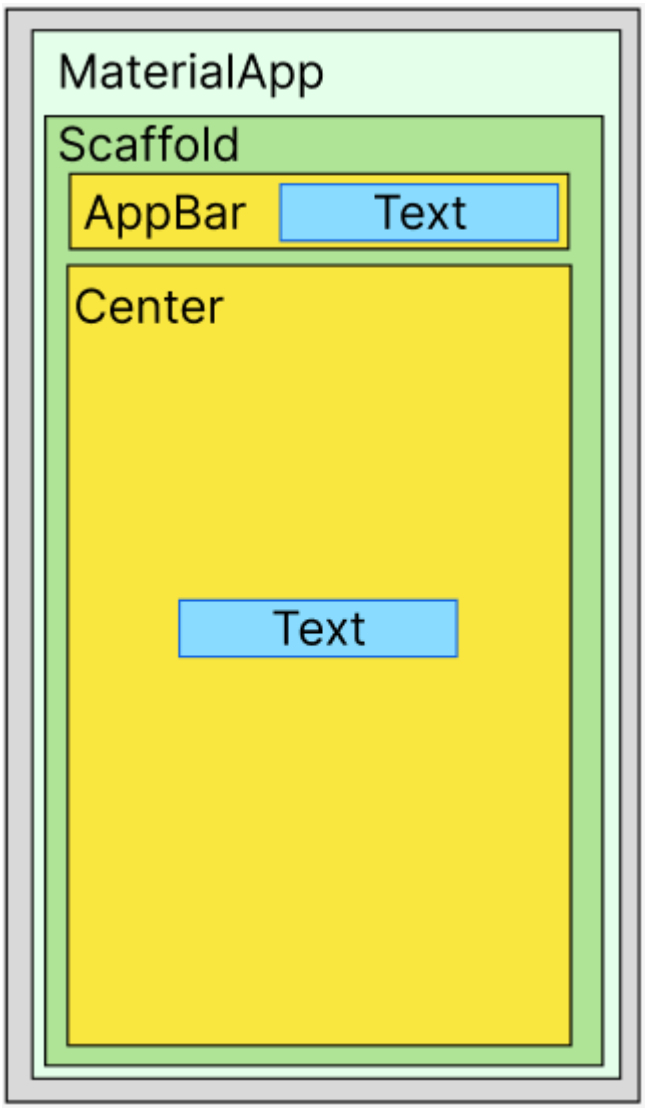


- 구현

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

// main() : 프로그램 실행 함수 => runApp() : 가장 먼저 화면에 나타날 위젯 실행
void main() => runApp(HelloWorld());

// 첫번째로 실행될 위젯 구현
// StatelessWidget 클래스 : 변화없이 화면 표시만을 위한 위젯
class HelloWorld extends StatelessWidget {
  // build가 실제 실행될 기본적인 위젯을 반환
  @override
  Widget build(BuildContext context) {
    // MaterialApp 클래스 : 안드로이드 머터리얼 디자인 적용 위젯
    return MaterialApp(
      title : 'First Flutter App',
      // Scaffold 클래스
      // - MaterialApp 내에서 실제적인 머티리얼 디자인의 기본적인 뼈대를 구성
      home : Scaffold(
        appBar: AppBar(
          title: const Text('Hello World Demo'),
        ), // AppBar
```

```
        body: const Center(
          child: const Text('Hello World'),
        ), // Center
      ) //Scaffold
    ); //MaterialApp
  }
}
```

- 플러터 앱의 화면 구성은 위젯으로 시작해서 위젯으로 끝
- 실행순서
  - main() ⇒ runApp(HelloWorld()) ⇒ build() ⇒ MaterialApp() ⇒ Scaffold()
- Hello World 앱 구조



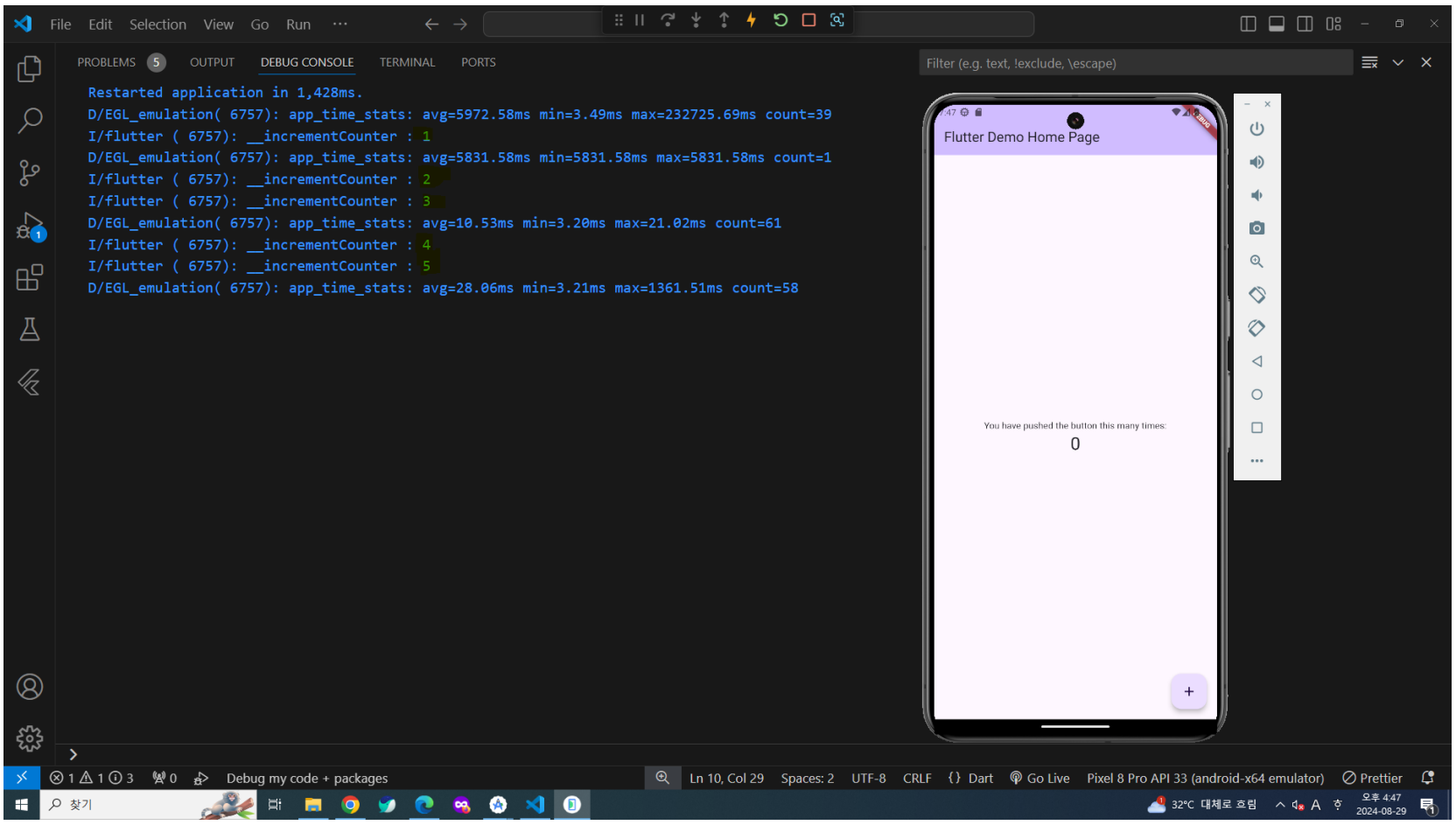
StatelessWidget

- 상태(State)를 가지지 않는 위젯으로 변화에 대해 반응하지 않음
- 상태 변화를 감지하지 않기 때문에 화면을 구성할 때 최초 한번만 build() 함수를 호출

> build() : 위젯을 렌더링하는 함수

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

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

class MyApp extends StatelessWidget {
  int _counter = 0;

  void _incrementCounter() {
    _counter++;
```

```
    print('__incrementCounter : $_counter');
  }

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        useMaterial3: true,
      ),
      //home: const MyHomePage(title: 'Flutter Demo Home Page'),
      home : Scaffold(
        appBar: AppBar(
          backgroundColor: Theme.of(context).colorScheme.inversePrimary,
          title: Text('Flutter Demo Home Page'),
        ),
        body: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: <Widget>[
              const Text(
                'You have pushed the button this many times:',
              ),
              Text(
                '$_counter',
                style: Theme.of(context).textTheme.headlineMedium,
              ),
            ],
          ),
        ),
        floatingActionButton: FloatingActionButton(
          onPressed: _incrementCounter,
          tooltip: 'Increment',
          child: const Icon(Icons.add),
        ),
      )
    );
  }
}
```

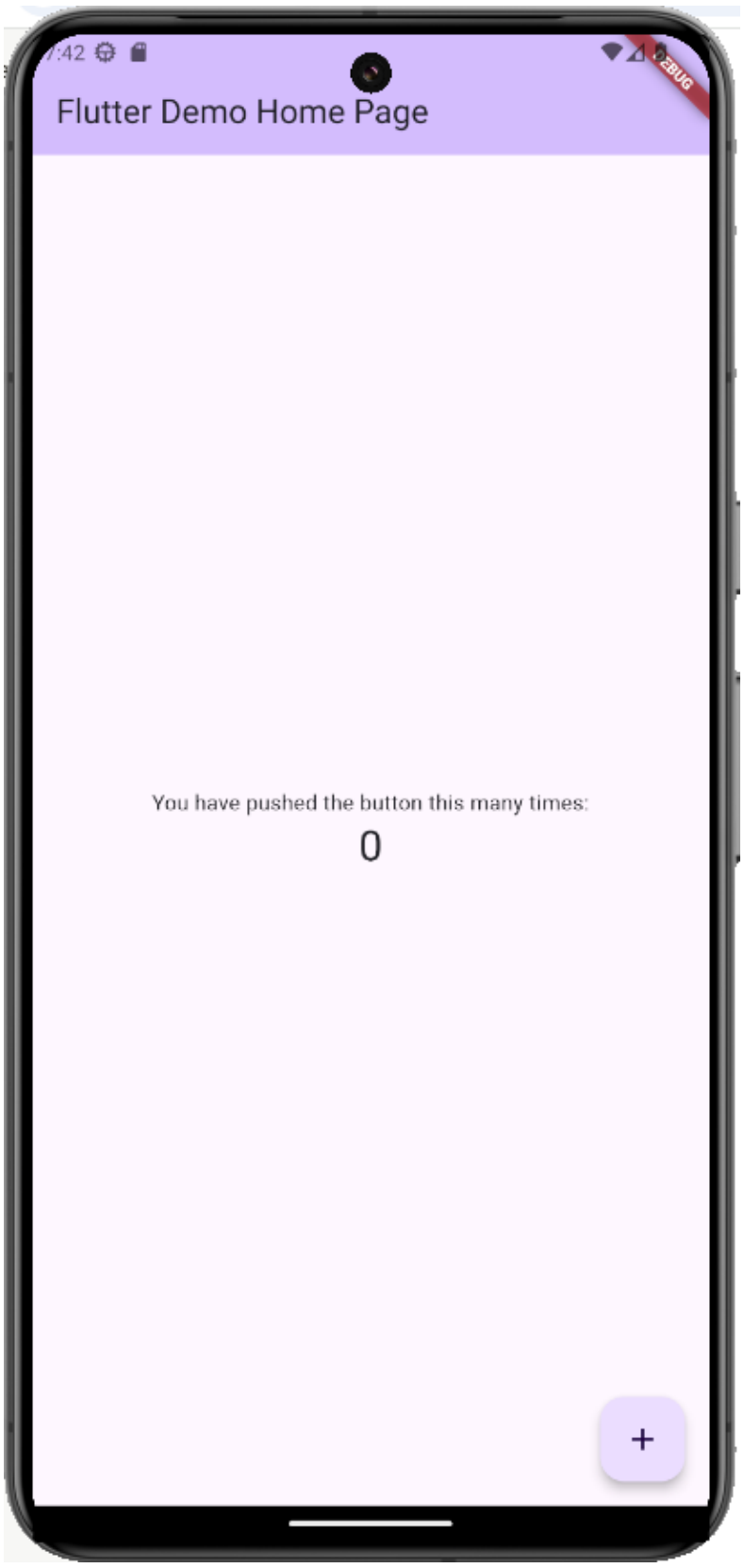


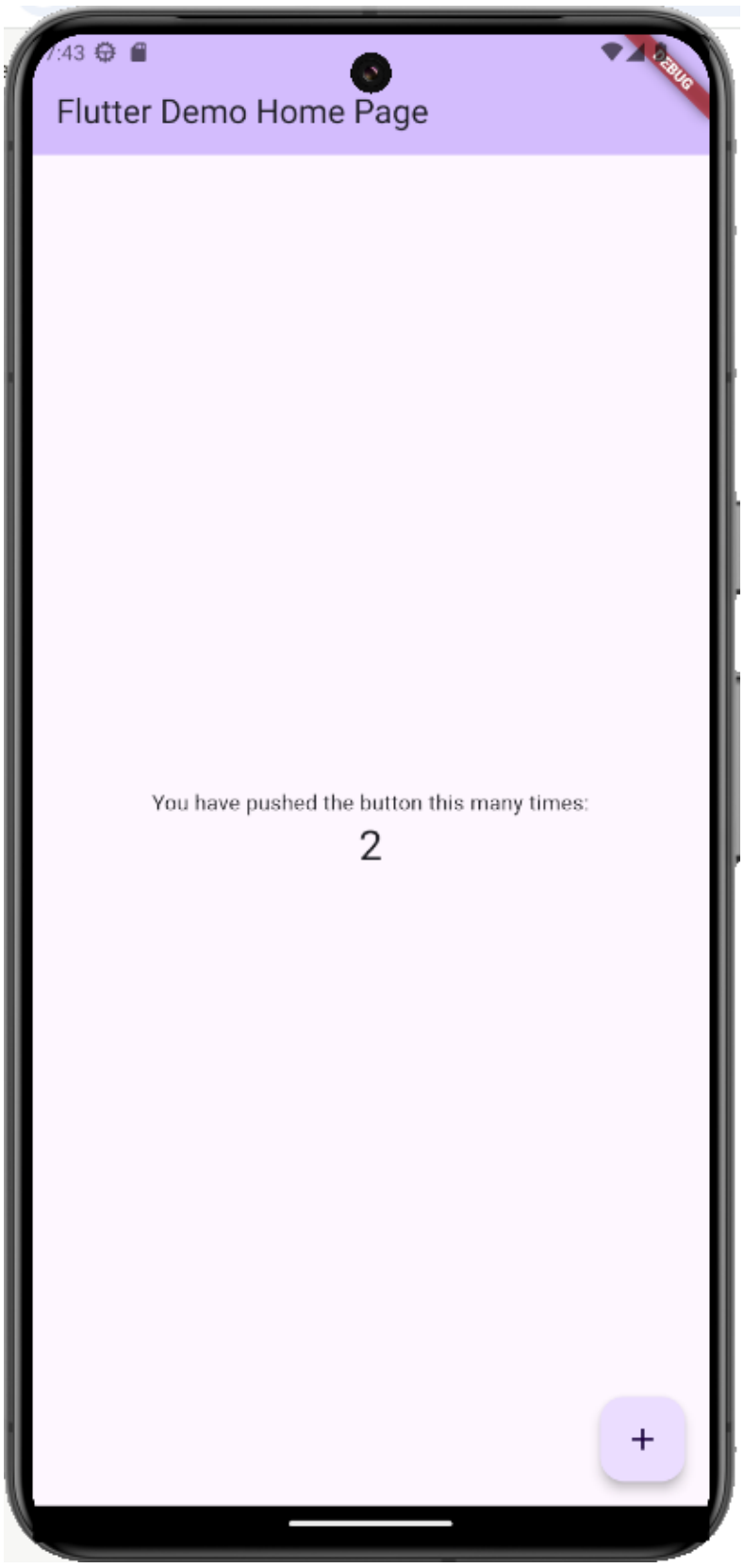
StatefulWidget

- 상태 변경을 감지하고 변경된 사항을 화면에 반영
- createState() 함수를 통해 상태 변경을 담당하는 State 객체를 생성

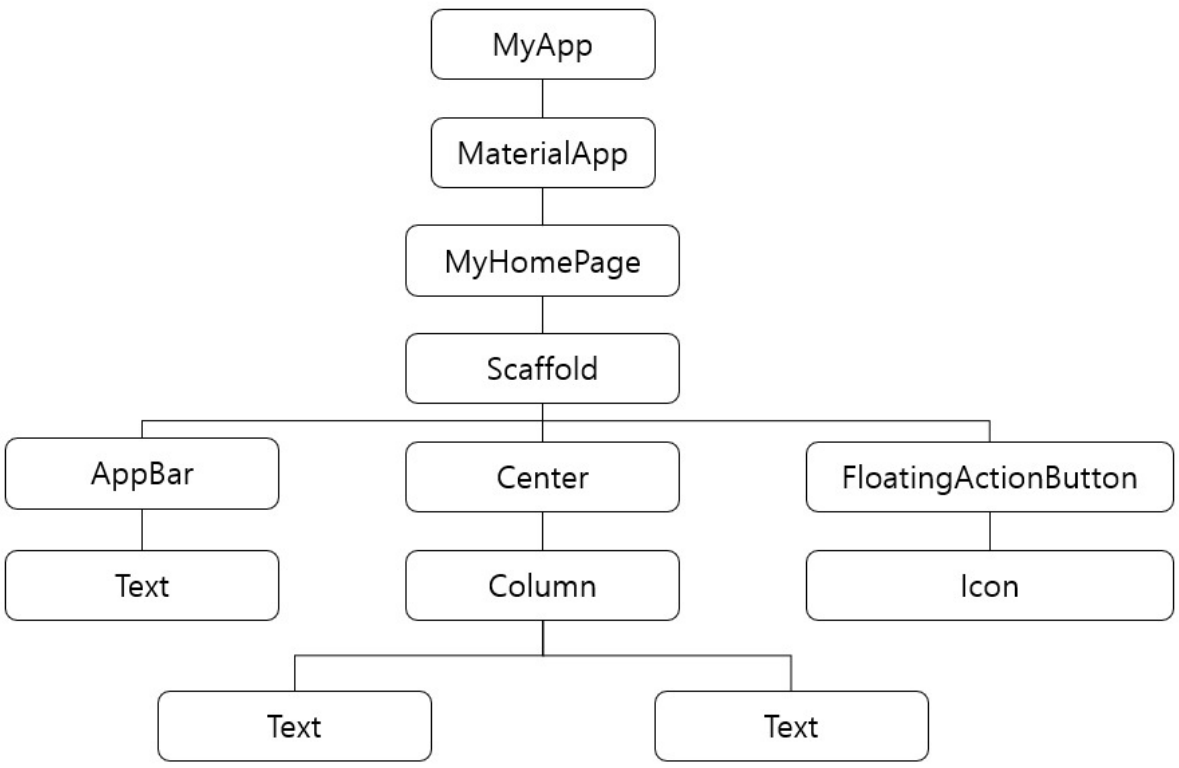
```
import 'package:flutter/material.dart';

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        useMaterial3: true,
      ),
      home: const MyHomePage(title: 'Flutter Demo Home Page'),
    );
  }
}

class MyHomePage extends StatefulWidget {
  const MyHomePage({super.key, required this.title});

  final String title;

  @override
  State<MyHomePage> createState() => _MyHomePageState();
}
```

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

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        backgroundColor: Theme.of(context).colorScheme.inversePrimary,
        title: Text(widget.title),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            const Text(
              'You have pushed the button this many times:',
            ),
            Text(
              '$_counter',
              style: Theme.of(context).textTheme.headlineMedium,
            ),
          ],
        ),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: _incrementCounter,
        tooltip: 'Increment',
        child: const Icon(Icons.add),
      ),
    );
  }
}
```


7:42
DEBUG
Flutter Demo Home Page
You have pushed the button this many times:
0
+

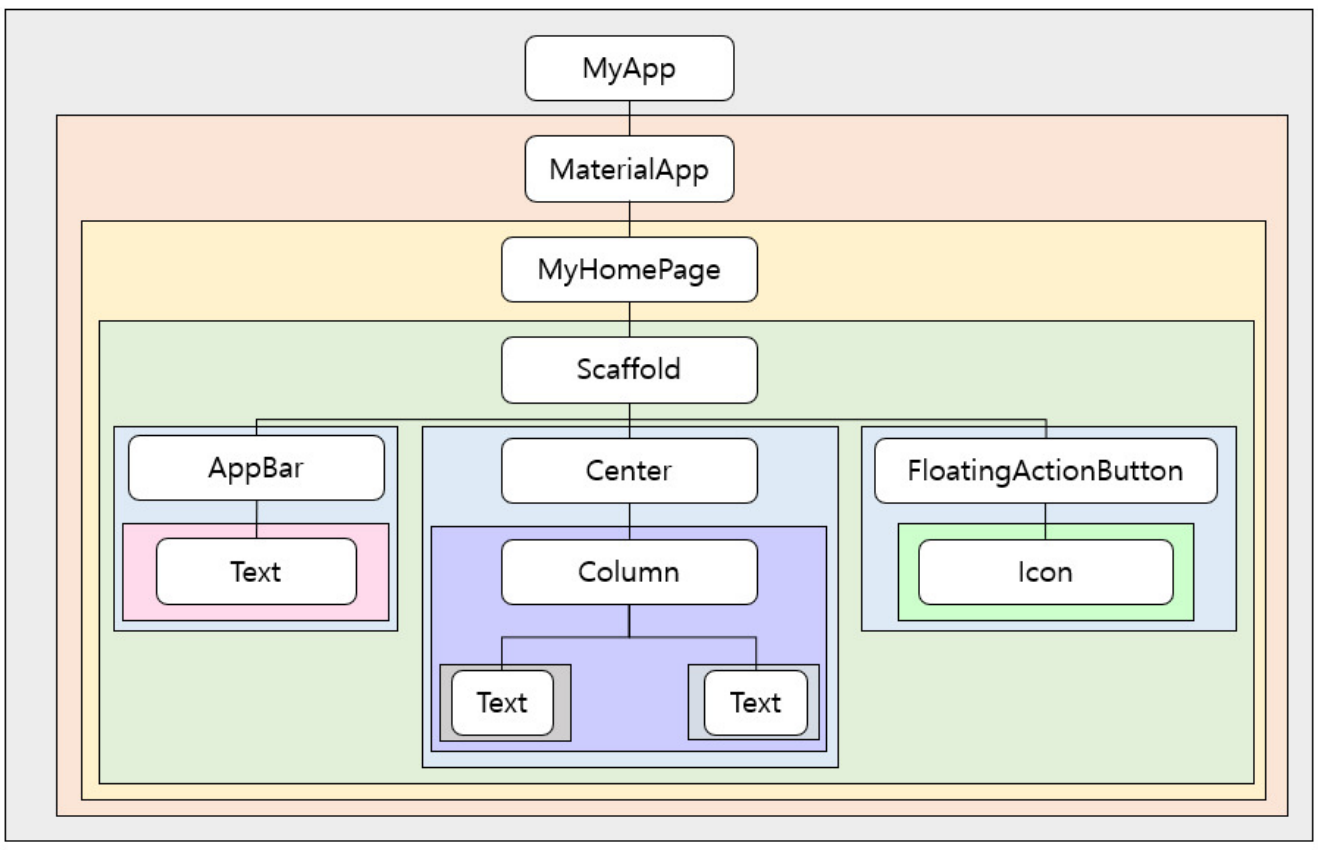


- State 객체
  - 변경 가능한 특징을 가지고 있으며 상태 변경에 대한 처리를 담당
  - 위젯이 빌드 완료된 후 읽을 수 있으며 위젯이 유효한 동안에 State는 변경될 수 있음
- BuildContext
  - 위젯 트리에서 위젯의 위치에 대한 참조
  - 하나의 BuildContext 는 하나의 위젯만 가짐
- Widget Tree
  - 위젯은 트리 구조로 구성되며 위젯 트리(Widget Tree)라고 함
  - 카운터 데모 앱의 위젯 트리



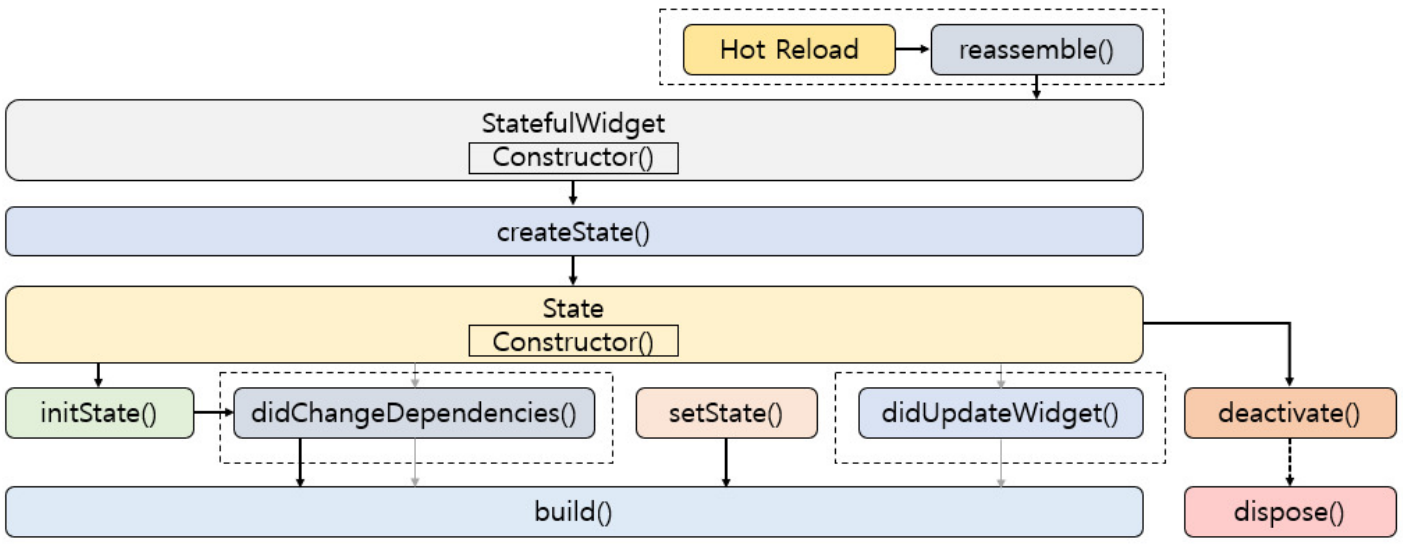
플러터는 세가지 트리를 가진다

- 위젯(Widget) : 요소의 구성(Configuration)을 기술하고 처리
- 요소(Element)
  - 트리의 특정 위치에서 위젯을 인스턴스화
  - BuildContext가 참조하는 대상
- 렌더 객체(RenderObject) : 크기, 레이아웃 등을 다루고 렌더링을 처리
- BuildContext를 위젯 트리에 도식화



  - State 생성 → 트리의 특정 위치를 참조하는 BuildContext 존재
    → 해당 BuildContext에 연결 → 연결된 각 BuildContext에 위젯 배치(인스턴스화)
    → 요소(Element) 생성

StatefulWidget 생명주기



- 주요 함수

1. createState()
    - StatefulWidget 객체의 생성자 호출 후 바로 실행
    - State 객체를 생성하는 역할
2. initState()
    - State 객체의 생성자 호출 후 실행
    - 위젯이 최초 생성되는 상황에서만 실행 ⇒ 처음 한번만 호출
3. didChangeDependencies()
    - State 객체의 initState() 호출 후 실행
    - 해당 위젯이 의존하는 위젯이 변경되면 호출
        - 위젯 A가 위젯 B를 상속받은 경우 위젯 B가 업데이트될 때
4. build()
    - 위젯을 렌더링하는 함수로 변경이 있을 때마다 호출
5. setState()
    - State 객체의 상태가 변경되었다는 것을 프레임워크에 알리는 용도
    - State 객체의 상태가 변경될 때마다 setState 함수를 호출해야 함
      ⇒ 프레임워크가 상태가 변경되었음을 알고 build 함수를 호춣
6. didUpdateWidget()
    - 부모 위젯이 재빌드되어 위젯이 갱실될 때 호출
      이후 build 함수가 호출되며 다시 렌더링 진행
7. deactivate()
    - 트리에서 State 객체가 제거될 때마다 호출
        - 경우에 따라 제거된 State 객체가 다른 buildContext에 연결 ⇒ build 호출
8. dispose()
    - 트리에서 State 객체가 영구적으로 제거될 때 호출 ⇒ build 호출되지 않음
9. reassemble()
    - Hot Reload를 실행 시 호출 ⇒ build 호출

- 예제 실습

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

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

class TestApp extends StatelessWidget {
```

```
  TestApp() {
    print('TestApp()');
  }

  @override
  Widget build(BuildContext context) {
    print('build 0');
    return MaterialApp(
      title: 'StatefulWidget Lifecycle App',
      home: _FirstStatefulWidget(),
    );
  }
}

class _FirstStatefulWidget extends StatefulWidget {
  _FirstStatefulWidget() {
    print('_FirstStatefulWidget()');
  }

  @override
  State<StatefulWidget> createState() => _FirstStatefulWidgetState();
}

class _FirstStatefulWidgetState extends State<_FirstStatefulWidget> {
  late int _counter;

  _FirstStatefulWidgetState() {
    print('_FirstStatefulWidgetState() ${this.mounted}');
  }

  @override
  Widget build(BuildContext context) {
    print('build() 1 ${this.mounted}');
    return Scaffold(
      appBar: AppBar(title: Text('(1) StatefulWidget Lifecycle')),
      body: Column(
        children: <Widget>[
          ElevatedButton(
              child: Text('Go Next'),
              onPressed: () {
                Navigator.of(context)
                    .push(MaterialPageRoute(builder: (context) {
                  return _SecondStatefulWidget();
                }));
              }),
          ElevatedButton(
              child: Text('Counter'),
              onPressed: () {
                _onClick();
              }),
          Row(
            children: <Widget>[
              Text('$_counter'),
            ],
            mainAxisAlignment: MainAxisAlignment.center,
          )
        ],
      ),
```

```
    );
  }

  @override
  void initState() {
    print('initState() 1');
    super.initState();
    _counter = 0;
  }

  @override
  void reassemble() {
    print('reassemble() 1');
    super.reassemble();
  }

  @override
  void didChangeDependencies() {
    print('didChangeDependencies() 1');
    super.didChangeDependencies();
  }

  @override
  void dispose() {
    print('dispose() 1');
    super.dispose();
  }

  @override
  void deactivate() {
    print('deactivate() 1');
    super.deactivate();
  }

  @override
  void didUpdateWidget(_FirstStatefulWidget oldWidget) {
    print('didUpdateWidget() 1');
    super.didUpdateWidget(oldWidget);
  }

  void _onClick() {
    print('onClick() 1');
    if (this.mounted) {
      setState(() {
        print('setState() 1');
        _counter++;
      });
    }
  }
}

class _SecondStatefulWidget extends StatefulWidget {
  _SecondStatefulWidget() {
    print('_SecondStatefulWidget()');
  }

  @override
  State<StatefulWidget> createState() => _SecondStatefulWidgetState();
```

```
}

class _SecondStatefulWidgetState extends State<_SecondStatefulWidget> {
  late int _counter;

  _SecondStatefulWidgetState() {
    print('_SecondStatefulWidgetState() ${this.mounted}');
  }

  @override
  Widget build(BuildContext context) {
    print('build() 2 ${this.mounted}');
    return Scaffold(
      appBar: AppBar(title: Text('(2) StatefulWidget Lifecycle')),
      body: Column(
        children: <Widget>[
          ElevatedButton(
              child: Text('Go Back'),
              onPressed: () {
                Navigator.of(context).pop();
              }),
          ElevatedButton(
              child: Text('Counter'),
              onPressed: () {
                _onClick();
              }),
          Row(
            children: <Widget>[
              Text('$_counter'),
            ],
            mainAxisAlignment: MainAxisAlignment.center,
          )
        ],
      ),
    );
  }

  @override
  void initState() {
    print('initState() 2');
    super.initState();
    _counter = 0;
  }

  @override
  void reassemble() {
    print('reassemble() 2');
    super.reassemble();
  }

  @override
  void didChangeDependencies() {
    print('didChangeDependencies() 2');
    super.didChangeDependencies();
  }

  @override
  void dispose() {
```

```
    print('dispose() 2');
    super.dispose();
  }

  @override
  void deactivate() {
    print('deactivate() 2');
    super.deactivate();
  }

  @override
  void didUpdateWidget(_SecondStatefulWidget oldWidget) {
    print('didUpdateWidget() 2');
    super.didUpdateWidget(oldWidget);
  }

  void _onClick() {
    print('onClick() 2');
    if (this.mounted) {
      setState(() {
        print('setState() 2');
        _counter++;
      });
    }
  }
}
```

- 실습 순서
  1. 프로젝트 실행


```
Connecting to VM Service at ws://127.0.0.1:53525/eUkHsDfFlS0=/ws
Connected to the VM Service.
I/flutter ( 8605): TestApp()
I/mple.couter_app( 8605): Compiler allocated 4533KB to compile void android.view.ViewRootImpl.performTraversals()
I/flutter ( 8605): build 0
I/flutter ( 8605): _FirstStatefulWidget()
I/flutter ( 8605): _FirstStatefulWidgetState() false
I/flutter ( 8605): initState() 1
I/flutter ( 8605): didChangeDependencies() 1
I/flutter ( 8605): build() 1 true
W/Parcel  ( 8605): Expecting binder but got null!
D/ProfileInstaller( 8605): Installing profile for com.example.couter_app
```

2. Conter 버튼 3번 클릭


```
D/ProfileInstaller( 8605): Installing profile for com.example.couter_app
I/flutter ( 8605): onClick() 1
I/flutter ( 8605): setState() 1
I/flutter ( 8605): build() 1 true
D/EGL_emulation( 8605): app_time_stats: avg=44947.36ms min=5.70ms max=89889.02ms count=2
I/flutter ( 8605): onClick() 1
I/flutter ( 8605): setState() 1
I/flutter ( 8605): build() 1 true
D/EGL_emulation( 8605): app_time_stats: avg=9.33ms min=2.73ms max=106.97ms count=55
D/EGL_emulation( 8605): app_time_stats: avg=55.61ms min=13.98ms max=720.31ms count=18
I/flutter ( 8605): onClick() 1
I/flutter ( 8605): setState() 1
I/flutter ( 8605): build() 1 true
```

3. Go Next 버튼 클릭


```
D/EGL_emulation( 8605): app_time_stats: avg=1606.84ms min=13.13ms max=57248.98ms count=36
I/flutter ( 8605): _SecondStatefulWidget()
I/flutter ( 8605): _SecondStatefulWidgetState() false
I/flutter ( 8605): initState() 2
I/flutter ( 8605): didChangeDependencies() 2
I/flutter ( 8605): build() 2 true
W/OnBackInvokedCallback( 8605): OnBackInvokedCallback is not enabled for the application.
W/OnBackInvokedCallback( 8605): Set 'android:enableOnBackInvokedCallback="true"' in the application manifest.
```

- Navigator.push() 함수를 사용
  - Stack에서 현재 위젯을 삭제하지 않고 호출하는 위젯을 쌓는 방식
    ⇒ 위젯이 제거되지 않으므로 deactivate() 호출되지 않음
    ⇒ AppBar에 뒤로가기를 이용하여 현재 위젯으로 돌아올 수 있음
  - 호출하는 위젯에서 현재 위젯으로 돌아올 것을 가정

> Navigator.pushReplacement() 함수를 사용하는 경우
>
> - 새롭게 호출하는 위젯이 신규화면으로 대체
>   ⇒ 위젯이 영구적으로 제거되므로 deactivate()와 dispose() 함수 호출
>   ⇒ AppBar에 뒤로가기가 사라짐

4. Conter 버튼 1번 클릭


```
D/EGL_emulation( 8605): app_time_stats: avg=6857.77ms min=2.26ms max=129931.01ms count=19
I/flutter ( 8605): onClick() 2
I/flutter ( 8605): setState() 2
I/flutter ( 8605): build() 2 true
```

5. Go Back 버튼 클릭


```
D/EGL_emulation( 8605): app_time_stats: avg=1500.22ms min=12.83ms max=54876.45ms count=37
I/flutter ( 8605): deactivate() 2
I/flutter ( 8605): dispose() 2
```

6. Hot reload 실행


```
I/flutter ( 8605): reassemble() 1
I/flutter ( 8605): build 0
I/flutter ( 8605): _FirstStatefulWidget()
I/flutter ( 8605): didUpdateWidget() 1
I/flutter ( 8605): build() 1 true
Reloaded 0 libraries in 241ms (compile: 10 ms, reload: 0 ms, reassemble: 61 ms).
D/EGL_emulation( 8605): app_time_stats: avg=9208.29ms min=3.54ms max=239185.06ms count=26
```

- reassemble() ⇒ TestApp의 Build() ⇒ _FirstStateWidget의 생성자 호출
  - _FirstStateWidgetState 객체를 새롭게 생성되지 않음
  - 부모 위젯인 TestApp이 재빌드 된 경우로
    _FirstStateWidgetState 객체의 didUpdateWidget() 호출 후 build()