Do it! 플러터 앱 프로그래밍

3장 플러터 내부 구조 살펴보기

목차

03-1 플러터 프로젝트 구조 알아보기

- 프로젝트 생성시 자동으로 파일 생성
- 폴더안에 각각 내용별로 파일이 잘 배치되도록 하는것이 중요

폴더	내용	비고
android	안드로이드 프로젝트 관련 파일	안드로이드 스튜디오로 실행 가능
ios	iOS 프로젝트 관련 파일	엑스코드로 실행 가능(맥 전용)
lib	플러터 앱 개발을 위한 다트 파일	플러터 SDK 설치 필요
test	플러터 앱 개발 중 테스트 파일	테스트 편의성 제공

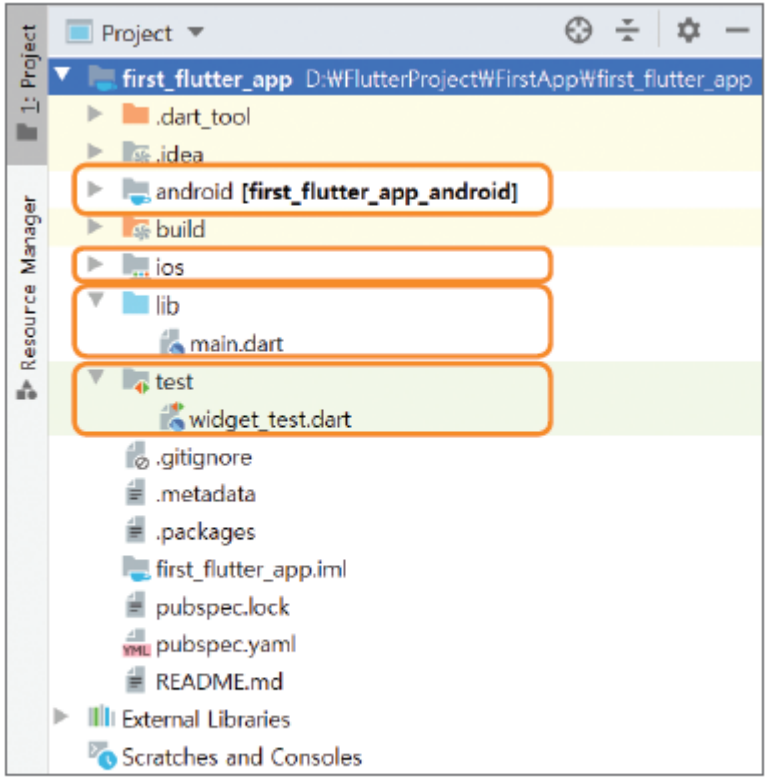


그림 3-1 플러터 프로젝트 구조

03-1 플러터 프로젝트 구조 알아보기

표 3-2 루트 안 파일 소개

파일	내용	비고
pubspec.yaml	패키지, 이미지, 폰트 설정	직접 관리
README.md	프로젝트 소개	
.gitignore	깃(git)에 커밋, 푸시 등 소스 코드를 업로드할 때 필요 없는 파일 기록	
.metadada	Flutter SDK 정보	자동 관리
.packages	Flutter SDK에 사용하는 기본 패키지 경로	
first_flutter_app.iml	파일이 자동으로 생성될 때 만들어지는 폴더 위치	
pubspec.lock	pubspec.yaml 파일에 적용된 패키지 위치	

03-1 플러터 프로젝트 구조 알아보기

- 플러터는 main() 에서부터 시작
- runApp()을 통해 앱의 시작하는 함수 호출
- Import 를 이용한 패키지 파일 호출

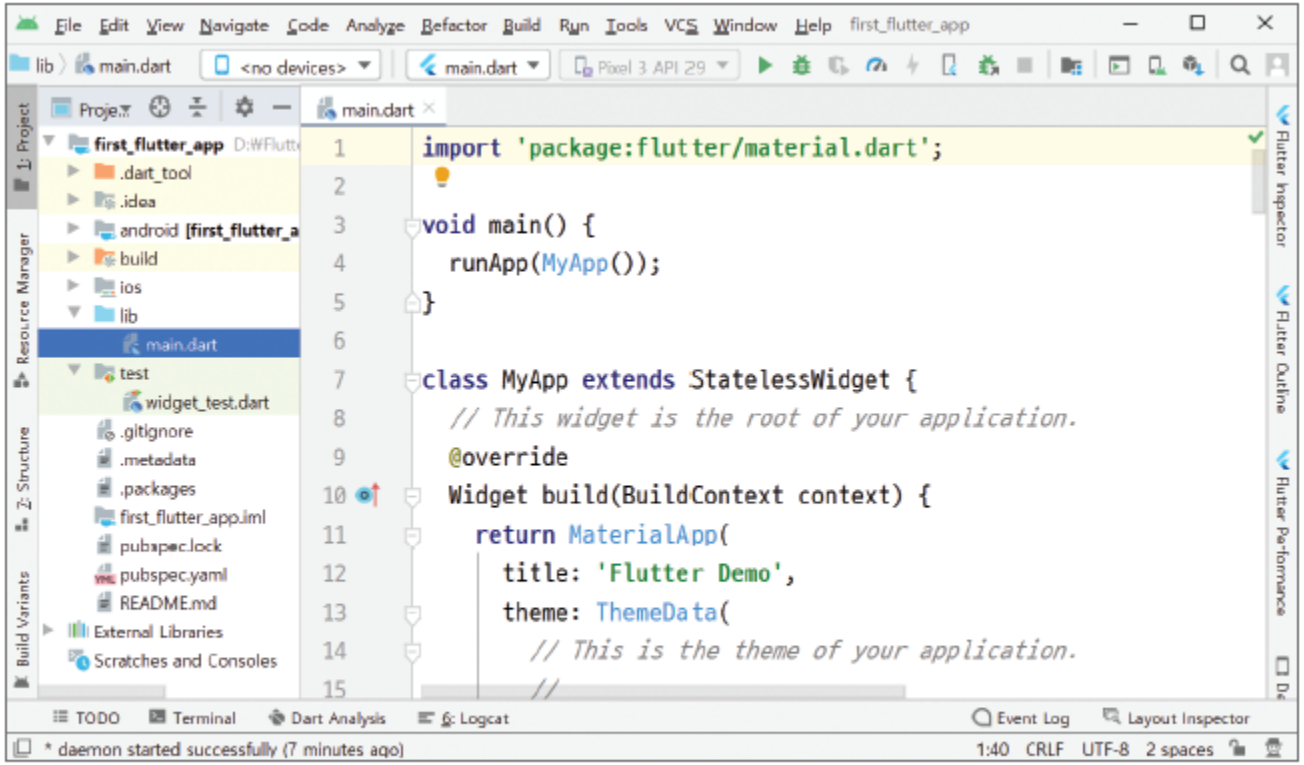


그림 3-2 플러터 메인 구조

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

03-1 플러터 프로젝트 구조 알아보기

- Extends는 상속을 의미
- Stateless Widget이라는 클래스를 상속받음
- @override 애너테이션을 이용해서 build()라는 함수를 재정의
- title은 앱의 이름을 정의
- runApp()을 이용해 클래스를 실행할 때는 MaterialApp() 함수를 반환 해야함
- theme는 앱의 색이나 설정을 정의
- home에는 앱을 실행할 때 첫 화면에 어떤 내용을 표시할지 정의

MyApp 클래스

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

03-1 모든 것은 위젯

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

Entry point

Build() 메소드

모든 위젯은 다른 위젯을 반환하는 build() 메소드를 반드시 포함해야 한다.

인수는 BuildContext 한 개를 갖는다.

new, const 생성자

많은 내장 위젯은 일반 생성자와 const 생성자를 모두 제공. 변경할 수 없는 위젯 인스턴스는 성능이 좋으므로 가능하면 const를 사용하는 것이 좋음. New, const 키워들 사용하지 않으면, 프레임워크가 const로 위젯을 추론하므로 크게 신경쓰지 않아도 됨.

플러터에서 클래스 인스턴스를 만들 때 new 키워드를 사용할 필요가 없음.

```
Widget build(BuildContext context){
  return Button(                    ← new 키워드 사용하지 않음
    child: Text("Submit"),
  );
}

Widget build(BuildContext context){
  return new Button(                ← new 키워드 사용
    child: Text("Submit"),
  );
}
```

03-1 플러터 프로젝트 구조 알아보기

- StatefulWidget 과 StatelessWidget의 차이

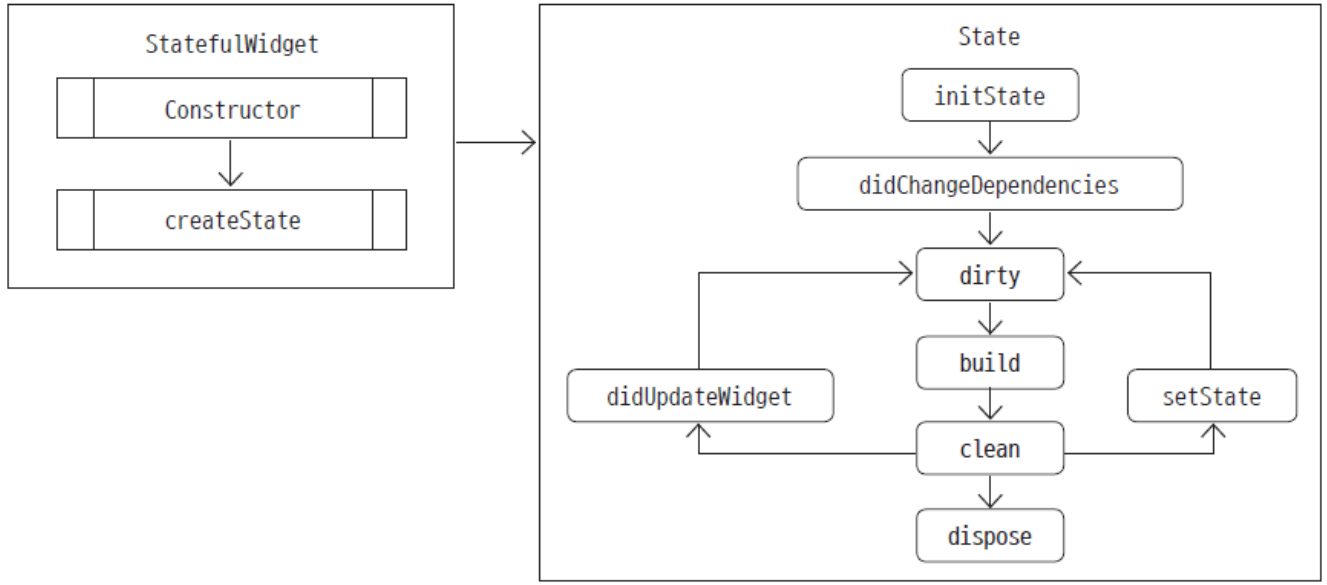


그림 3-4 스테이트풀 위젯

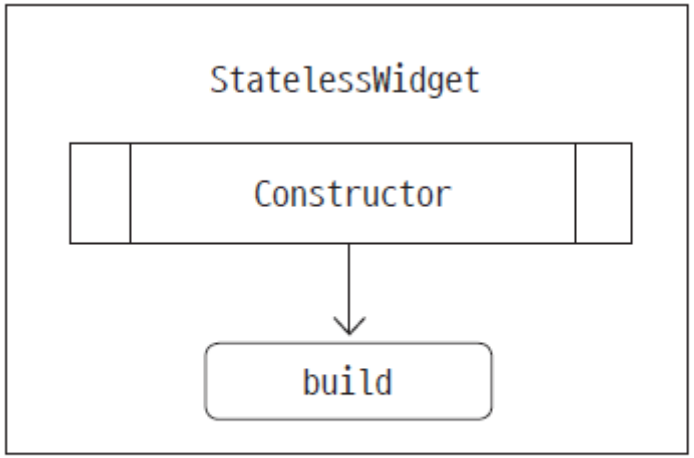


그림 3-3 스테이트리스 위젯

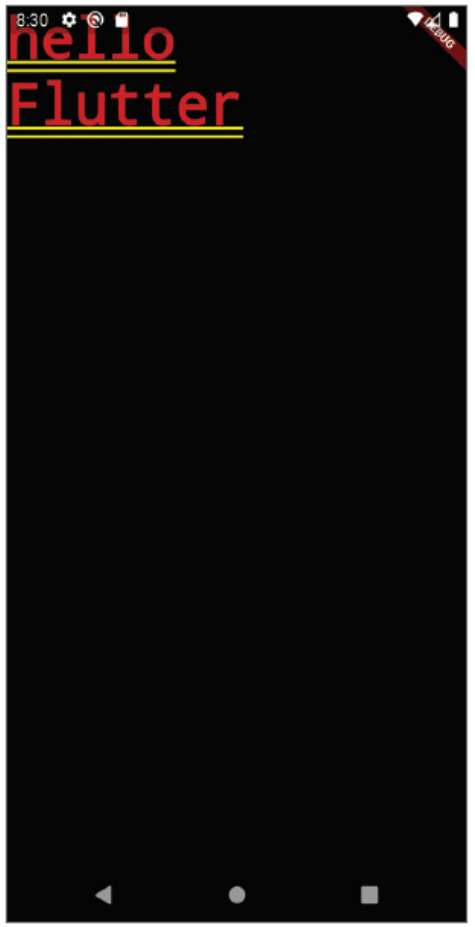
03-1 플러터 프로젝트 구조 알아보기

- 1번 실습하기 home 변경 후 빌드하기

• lib/main.dart

```
(...생략...)
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        (...생략...)
        home: Text('hello\nFlutter'),
    );
  }
}
```

▶ 실행 결과

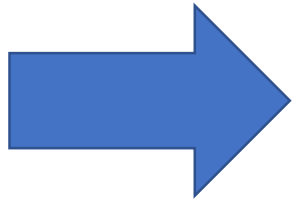


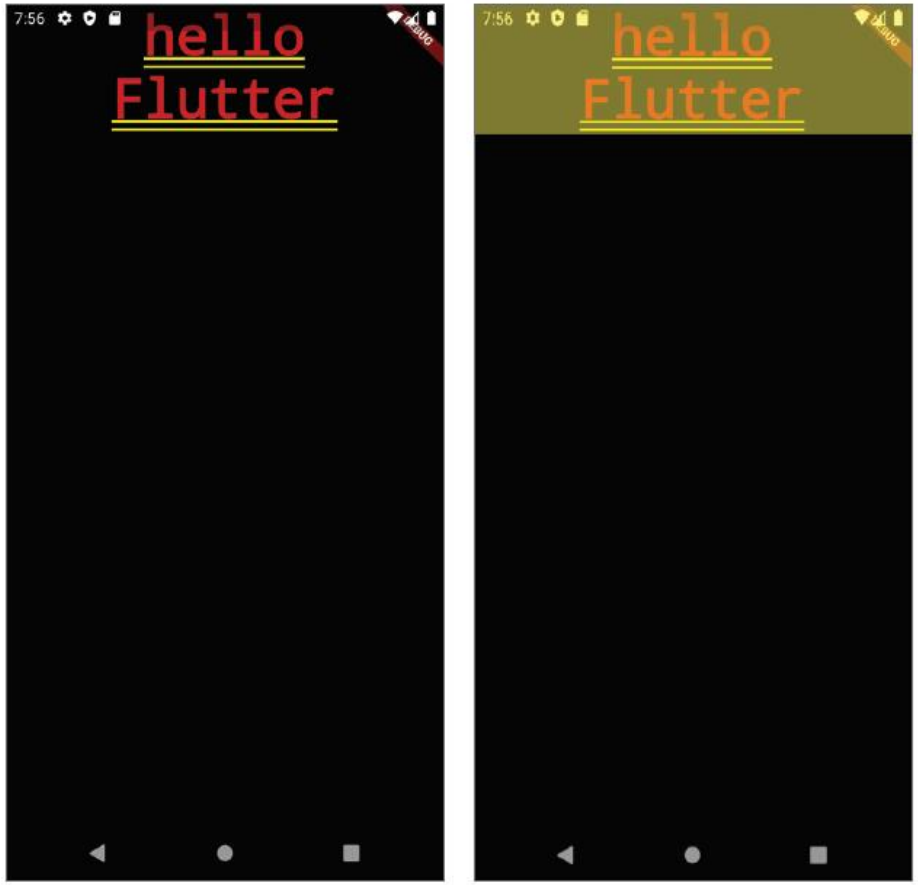
03-1 플러터 프로젝트 구조 알아보기

- 2번 실습하기 Text 설정 추가

• lib/main.dart

```
(...생략...)
home: Text('hello\nFlutter', textAlign: TextAlign.center)
(...생략...)
```



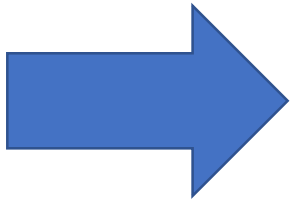


03-1 플러터 프로젝트 구조 알아보기

- 3번 실습하기 Center위젯 추가하기

• lib/main.dart

```
(...생략...)
home: Center(
  child: Text('hello\nFlutter', textAlign: TextAlign.center),
));
(...생략...)
```

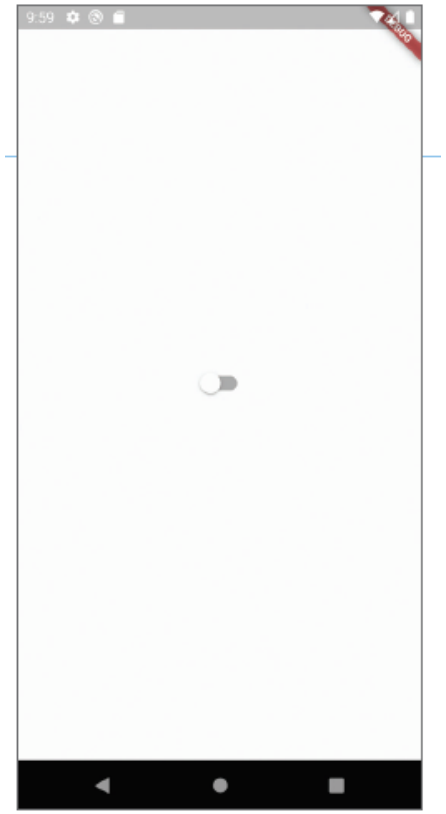


▶ 실행 결과

03-1 플러터 프로젝트 구조 알아보기

- 4번 실습하기 Switch 위젯 추가하기

▶ 실행 결과



• lib/main.dart

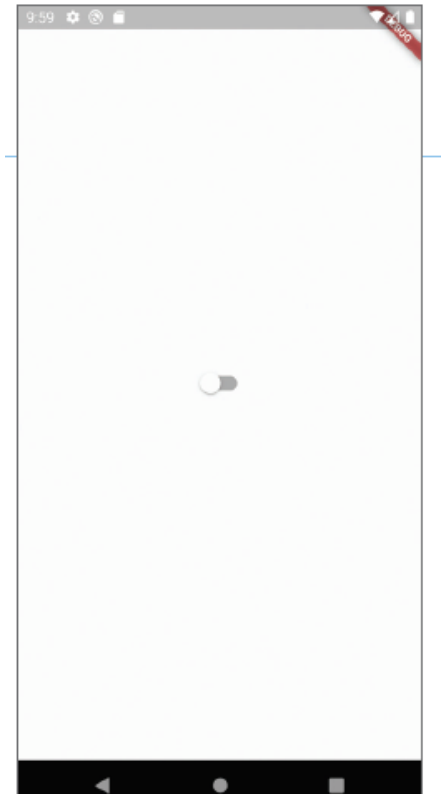
```
(...생략...)
class MyApp extends StatelessWidget {
  var switchValue = false;
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        title: 'Flutter Demo',
        theme: ThemeData(
          primarySwatch: Colors.blue,
          visualDensity: VisualDensity.adaptivePlatformDensity,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

▶ 실행 결과

03-1 플러터 프로젝트 구조 알아보기

- 4번 실습하기 Switch 위젯 추가하기
- StatelessWidget이라 변경이 안됨

▶ 실행 결과



• lib/main.dart

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

▶ 실행 결과

03-1 플러터 프로젝트 구조 알아보기

- StatefulWidget으로 변경하기
- 변경 후 Switch UI 변경되는 것을 확인

▶ 실행 결과

• lib/main.dart

```
(...생략...)
void main() {
  runApp(MyApp());
}

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

class _MyApp extends State<MyApp> {
  (...생략...)
        home: Scaffold(
          body: Center(
            child: Switch(
                value: switchValue,
                onChanged: (value) {
                  print(value);
                  switchValue = value;
                }),
(...생략...)
```

03-1 플러터 프로젝트 구조 알아보기

- 5번 실습하기 Button 위젯 추가하기

• lib/main.dart

```
(...생략...)
child: RaisedButton(
    child: Text('$test'),
    onPressed: () {
     if (test == 'hello') {
        setState(() {
          test = 'flutter';
        });
      } else {
        setState(() {
          test = 'hello';
        });
      }
    })),
(...생략...)
```

▶실행 결과

03-2 위젯의 생명주기 이해하기

- Stateful 과 Stateless 위젯

StatefulWidget보다 State 클래스가 상대적으로 더 무겁기 때 문에 StatefulWidget에서 감시하고 있다가 상태 변경 신호가 오면 State 클래스가 화 면을 갱신하도록 구현

StatefulWidget에서 바로 갱신하면 나중에 화면이 종료되어도 할당받은 메모리를 없앨 때까지 오랜 시간이 걸릴 수 있기때문에 상태 변경 감시는 StatefulWiget 클래스가 담당하고, 실제 갱신 등은 State 클래스가 담당하도 록 분리해 두도록 설계할 필요가 있음

03-2 위젯의 생명주기 이해하기

- **스테이트풀 위젯의 생명주기**

1.상태를 생성하는 createState() 함수

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

03-2 위젯의 생명주기 이해하기

- **스테이트풀 위젯의 생명주기**

2. Mounted 체크

3. initState() 함수 호출

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

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

03-2 위젯의 생명주기 이해하기

- **스테이트풀 위젯의 생명주기**

4. didChangeDependencies()
5. build() 함수 호출

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

03-2 위젯의 생명주기 이해하기

- **스테이트풀 위젯의 생명주기**

6. 앱 갱신시 didUpdateWidget() 호출
7. 위젯 상태를 갱신하는 setState() 함수 호출

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

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

03-2 위젯의 생명주기 이해하기

- **스테이트풀 위젯의 생명주기**

8. 위젯의 상태 관리를 중지하는 deactivate() 함수
9. 위젯의 상태 관리를 완전히 끝내는 dispose() 함수
10. 위젯을 화면에서 제거하면 mounted == false

03-2 위젯의 생명주기 이해하기

표 3-3 위젯의 생명주기 요약

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