

Chapter

# 10

**Project**

**만난 지 며칠 U&I**  
상태 관리, CupertinoDatePicker,  
Dialog, DateTime

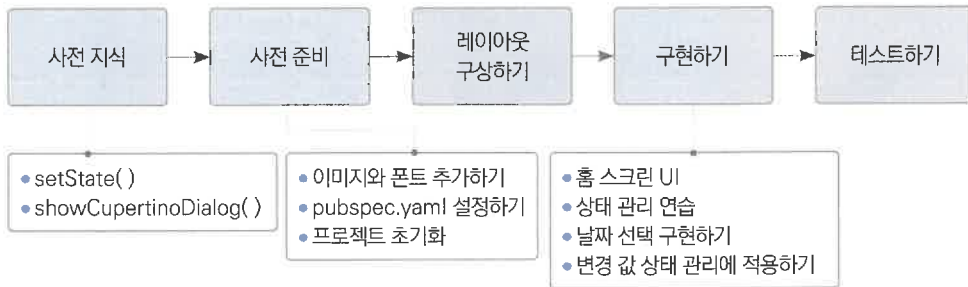


예제 위치	<a href="https://github.com/codefactory-co/flutter-golden-rabbit-novice-v3/tree/main/ch10/u_and_i">https://github.com/codefactory-co/flutter-golden-rabbit-novice-v3/tree/main/ch10/u_and_i</a>
프로젝트명	u_and_i
개발 환경	플러터 SDK : 3.24.4
미션	날짜를 지정하고 해당 날짜로부터 며칠이 지났는지 알려주는 앱을 만들어봐요.
기능	<ul style="list-style-type: none"> <li>• 사용자가 직접 원하는 날짜 선택</li> <li>• 날짜 선택 시 실시간으로 화면의 D-Day 및 만난 날 업데이트</li> </ul>
조작법	<ol style="list-style-type: none"> <li>1. 가운데 하트 클릭해서 날짜 선택 기능 실행</li> <li>2. 연도, 월, 일을 스크롤해서 원하는 날짜 선택</li> <li>3. 배경을 눌러서 날짜 저장하기 및 되돌아오기</li> </ol>
핵심 구성요소	<ul style="list-style-type: none"> <li>• Cupertino Widget</li> <li>• CupertinoDatePicker</li> <li>• Dialog</li> <li>• StatefulWidget 상태 관리</li> </ul>
플러그인	없음

## □ 학습 목표

D-Day 앱을 만듭니다. 사귀기 시작한 날짜를 선택하면 사귀기 시작한 날짜로부터 며칠이 지났는지 알려주는 앱입니다.

## □ 학습 순서



## □ 프로젝트 구상하기

이번 프로젝트에서 집중할 요소는 두 가지입니다. 첫 번째는 `StatefulWidget`을 이용한 상태 관리입니다. 9장에서 전자책자 앱을 만들며 `StatefulWidget`의 생명주기를 사용했지만 `setState()` 함수를 사용한 상태 관리는 아직 안 해봤습니다. 이번 프로젝트는 `setState()` 함수 사용법을 알아보겠습니다. 두 번째는 쿠퍼티노 `Cupertino` 위젯입니다.

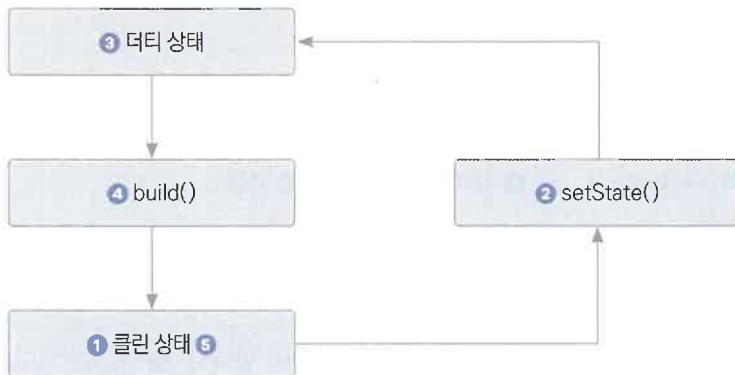
플러터는 두 가지 디자인 시스템을 지원합니다. 구글의 머터리얼 디자인을 기반으로 하는 Material 위젯과 iOS 스타일의 디자인인 Cupertino 위젯입니다. 이번 프로젝트를 진행하면서 다이얼로그 `dialog` 및 데이터픽커 `datepicker`를 Cupertino 위젯을 사용해서 구현하겠습니다.

## 10.1 사전 지식

### 10.1.1 setState( ) 함수

setState( ) 함수가 생명주기에서 어떤 역할을 하고 어떤 상황에 사용되는지 9장에서 이미 배웠습니다. 이번에는 setState( ) 함수를 어떻게 사용하는지를 알아보겠습니다.

State를 상속하는 모든 클래스는 setState( ) 함수를 사용할 수 있습니다. setState( ) 함수가 실행되는 과정은 다음과 같이 5단계입니다.



① StatefulWidget의 렌더링이 끝나고 클린<sup>clean</sup> 상태입니다. 플러터에서는 그 어떤 상태 변경 틀을 사용하든 클린 상태에서 상태를 변경해줘야 합니다. ② setState( )를 실행해서 원하는 속성들을 변경합니다. ③ 속성이 변경되고 위젯의 상태가 더티<sup>dirty</sup>로 설정됩니다. ④ build( ) 함수가 재실행됩니다. ⑤ State가 클린 상태로 다시 되 돌아옵니다.

setState( ) 함수를 실행하는 방법은 매우 간단합니다. setState( ) 함수는 매개변수 하나를 입력 받습니다. 이 매개변수는 콜백 함수이고 이 콜백 함수에 변경하고 싶은 속성들을 입력해주면 해당 코드가 반영된 뒤 build( ) 함수가 실행됩니다. 콜백 함수가 비동기로 작성되면 안 된다는 점에 주의하세요.

```
setState(() { // ① 실행
  number ++;
});
```

❶ `setState()` 함수의 첫 번째 매개변수에 상태(변수) 값을 변경하는 로직을 작성합니다. 예제에 서는 `number`라는 값을 1만큼 증가시키고 다시 `build()` 함수를 실행합니다.

### 10.1.2 `showCupertinoDialog()` 함수

`showCupertinoDialog()`는 다이얼로그를 실행하는 함수입니다. Cupertino라고 정의된 만큼 iOS 스타일로 다이얼로그가 실행되며 실행 시 모든 애니메이션과 작동이 iOS 스타일로 적용 됩니다.

```
import 'package:flutter/cupertino.dart'; // ❶ Cupertino 패키지 импорт

showCupertinoDialog( // ❷ Cupertino 다이얼로그 실행
  context: context, // ❸ BuildContext 입력
  barrierDismissible: true, // ❹ 외부 탭해서 다이얼로그 닫을 수 있게 하기
  builder: (BuildContext context) { // ❺ 다이얼로그에 들어갈 위젯
    return Text('Dialog');
  },
);
```

❶ Cupertino 위젯을 사용하려면 Cupertino 패키지를 꼭 불러와야 합니다. ❷ `showCupertinoDialog()` 함수를 실행하면 Cupertino 스타일의 다이얼로그를 실행할 수 있습니다. ❸ 모든 `showDialog()` 형태의 함수들은 `BuildContext`를 반드시 입력해야 합니다. ❹ 플러터에서 다이얼로그 위젯 외에 흐림 처리가 된 부분을 배리어(barrier)라고 부릅니다. 예를 들어 다이얼로그 위젯의 높이가 300인데 화면의 전체 높이가 1000이라면 나머지 700만큼의 부분이 배리어가 됩니다. `barrierDismissible`에 `true`를 입력하면 배리어를 눌렀을 때 다이얼로그가 닫히고 `false`를 입력하면 닫히지 않습니다. ❺ `builder()` 함수에 다이얼로그로 띄우고 싶은 위젯을 반환해주면 됩니다.

## 10.2 사전 준비

이번 강의에서는 에셋과 폰트를 추가해야 합니다. 에셋 추가 방법은 9.2.1 '이미지 추가하기'에서 배운 대로 진행하면 됩니다. 폰트는 프로젝트에서 사용할 글꼴을 의미합니다. 폰트 추가 방법은

아직 배우지 않았으니 곧이어 알아보겠습니다.

**ToDo 01** 먼저 실습에 사용할 프로젝트를 생성해주세요.

- 프로젝트 이름 : u\_and\_i
- 네이티브 언어 : 코틀린

### 10.2.1 이미지와 폰트 추가하기

**ToDo 01** 이미지와 폰트를 프로젝트에 추가하겠습니다. [asset] 폴더를 만들고 그 아래 [font]와 [img] 폴더를 만듭니다. 내려받은 예제에서 [asset/font]에 있는 폰트 파일들을 방금 만든 [font] 폴더로 드래그 앤드 드롭합니다. [asset/img]에 있는 그림 파일들을 방금 만든 [img] 폴더로 드래그 앤드 드롭합니다.

### 10.2.2 pubspec.yaml 설정하기

**ToDo 01** 이미지와 마찬가지로 폰트도 pubspec.yaml 파일에 추가합니다. 예셋 파일은 flutter 키의 assets 키에, 폰트 파일은 flutter 키의 fonts 키에 입력하면 됩니다. 이미지와 폰트를 pubspec.yaml에 적용해주세요.

```
pubspec.yaml

flutter:
  uses-material-design: true

  assets:
    - asset/img/    # 이미지를 프로젝트에 포함시키기

  fonts:
    - family: parisienne # family 키에 폰트 이름을 지정할 수 있습니다.
      fonts:
        - asset: asset/font/Parisienne-Regular.ttf # 등록할 폰트 파일의 위치

    - family: sunflower
      fonts:
        - asset: asset/font/Sunflower-Light.ttf
        - asset: asset/font/Sunflower-Medium.ttf
          weight: 500 # ❶ 폰트 두께. FontWeight 클래스의 값과 같습니다.
        - asset: asset/font/Sunflower-Bold.ttf
          weight: 700
```

❶ Weight는 폰트 두께를 의미합니다. 폰트의 두께별로 파일이 따로 제공되기 때문에 같은 폰트라도 다른 두께를 표현하는 파일은 weight값을 따로 표현해줘야 합니다. 두께값은 100부터 900까지 100 단위로 사용할 수 있으며 숫자가 높을수록 두꺼운 값을 의미합니다. 추후 플러터에서 사용할 때 FontWeight 클래스에 표현되는 두께값과 같습니다(예 : weight: 500 = FontWeight.w500).

❷ [pub get]을 실행해서 변경 사항을 반영합니다.

### 10.2.3 프로젝트 초기화하기

**ToDo** ❶ [lib] 폴더에 [screen] 폴더를 생성하고 앱의 기본 홈 화면으로 사용할 HomeScreen 위젯을 생성할 home\_screen.dart를 생성합니다. 다음과 같이 HomeScreen이라는 StatelessWidget을 생성해주세요.

```
import 'package:flutter/material.dart';                                     lib/screen/home_screen.dart

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Text('Home Screen'),
    );
  }
}
```

❷ lib/main.dart 파일에서도 마찬가지로 HomeScreen을 홈 위젯으로 등록해줘야 합니다.

```
import 'package:u_and_i/screen/home_screen.dart';                         lib/main.dart
import 'package:flutter/material.dart';

void main() {
  runApp(
    MaterialApp(
      home: HomeScreen(),
    ),
  );
}
```

```
);  
}
```

## 10.3 레이아웃 구상하기

지금까지 프로젝트에서는 Scaffold 위젯의 body 매개변수에 위젯 하나만 입력했습니다. 이번에는 ❶ \_DDay 위젯과 ❷ \_CoupleImage 위젯 두 가지를 위아래로 나누어서 구현하겠습니다.



홈 스크린 말고도 ❶ CupertinoDialog 를 추가로 구현해야 합니다. 중앙의 하트 아이콘을 클릭하면 CupertinoDialog 가 실행되는 구조를 만들겠습니다.



## 10.4 구현하기

UI 구현, 상태 관리 구현, 날짜 선택 기능 구현 순서로 진행하겠습니다. UI를 먼저 작업해서 앱 전체의 틀을 잡고 상태 관리를 설정해서 날짜 데이터를 관리할 기반을 잡습니다. 마지막으로 날짜 선택 기능을 추가해서 선택한 날짜에 따라 D-Day를 계산하는 기능을 구현하겠습니다.



## 10.4.1 홈 스크린 UI 구현하기

**ToDo 01** 지금까지는 HomeScreen 위젯 하나로 모든 화면을 구현했습니다. 하지만 이번에는 코드가 조금 더 많아지는 만큼 위젯을 두 위젯으로 나눠서 화면을 구성하겠습니다. 우선 HomeScreen의 위쪽 반을 구현할 \_DDay 위젯을 HomeScreen 위젯 아래에 생성하겠습니다.

**NOTE** \_DDay 위젯처럼 이름 첫 글자가 언더스코어이면 다른 파일에서 접근할 수 없습니다. 그래서 파일을 불러오기 했을 때 불필요한 값들이 한 번에 불러와지는 걸 방지할 수 있습니다.

```
import 'package:flutter/material.dart';lib/screen/home_screen.dart  
  
class HomeScreen extends StatelessWidget {  
  const HomeScreen({Key? key}) : super(key: key);  
  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      body: Text('Home Screen'),  
    );  
  }  
}  
  
class _DDay extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return Text('DDay Widget');  
  }  
}
```

**02** 다음은 HomeScreen의 아래쪽을 구현할 \_CoupleImage 위젯을 \_DDay 위젯 아래에 생성하겠습니다.

```
... 생략 ...lib/screen/home_screen.dart  
class _CoupleImage extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return Text('Couple Image Widget');  
  }  
}
```

**03** 이 두 위젯을 위아래로 서로 반씩 차지하게 배치해야 합니다. HomeScreen 위젯에 Column 위젯을 사용해서 \_DDay 위젯과 \_CoupleImage 위젯이 위아래에 놓이게 하겠습니다.

lib/screen/home\_screen.dart

```

... 생략 ...


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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: SafeArea( // ❶ 시스템 UI 피해서 UI 그리기
        top: true,
        bottom: false,
        child: Column(
          // ❷ 위아래 끝에 위젯 배치
          mainAxisAlignment: MainAxisAlignment.spaceBetween,

          // 반대쪽 최대 크기로 늘리기
          crossAxisAlignment: CrossAxisAlignment.stretch,
          children: [
            _DDay(),
            _CoupleImage(),
          ],
        ),
      ),
    );
  }
}
... 생략 ...

```

▼ 실행 결과



❶ 아이폰의 노치에 대비해서 위에는 SafeArea를 적용해주고, 이미지를 자연스럽게 구현하기 위해 아래에는 미적용합니다. ❷ MainAxisAlignment.spaceBetween을 사용해서 위아래 각각 끝에 \_DDay와 \_CoupleImage 위젯을 위치시킵니다.

**04** 배경색을 핑크색으로 변경하겠습니다.

lib/screen/home\_screen.dart

```

... 생략 ...

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

```

```

@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Colors.pink[100], // 핑크 배경색 적용
    ... 생략 ...
  );
}
}
... 생략 ...

```

**05** 기본값으로 적용해둔 글자들을 제거하고 목표하는 UI를 구현할 차례입니다. `_CoupleImage` 위젯에 커플 이미지를 적용하고 이미지의 높이를 화면 높이의 반만큼으로 조절하겠습니다.

```

... 생략 ...
class _CoupleImage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Center( // ❶ 이미지 중앙 정렬
      child: Image.asset(
        'asset/img/middle_image.png',

        // ❷ 화면의 반만큼 높이 구현
        height: MediaQuery.of(context).size.height / 2,
      ),
    );
  }
}

```

lib/screen/home\_screen.dart

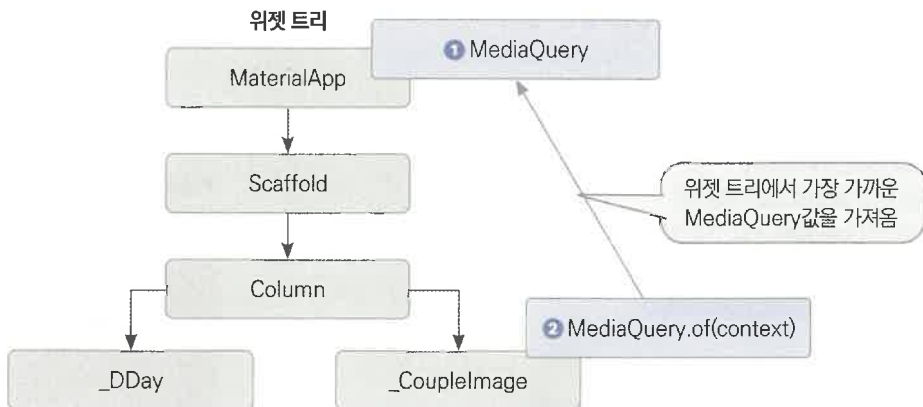
▼ 실행 결과



❶ `Center` 위젯을 사용해서 이미지를 중앙에 배치합니다. ❷ `MediaQuery.of(context)`를 사용하면 화면 크기와 관련된 각종 기능을 사용할 수 있습니다. 특히 여기서 사용된 `size` 게터를 불러오면 화면 전체의 너비 `width`와 높이 `height`를 쉽게 가져올 수 있습니다. 화면의 전체 높이를 2로 나눠서 화면 높이의 반만큼 이미지가 차지하도록 설정했습니다.

## .of 생성자

`.of(context)` 생성자를 초보자들이 많이 헛갈려합니다. `.of(context)`로 정의된 모든 생성자는 일반적으로 `BuildContext`를 매개변수로 받고 위젯 트리(widget tree)에서 가장 가까이에 있는 객체의 값을 찾아냅니다. 결과적으로 `MediaQuery.of(context)`는 현재 위젯 트리에서 가장 가까이에 있는 `MediaQuery`값을 찾아냅니다.



1 앱이 실행되면 `MaterialApp`이 빌드됨과 동시에 `MediaQuery`가 생성됩니다. 2 위젯 트리 아래에서 `MediaQuery.of(context)`를 실행하면 위젯 트리를 올라가며 가장 가까운 곳에 위치한 `MediaQuery`값을 가져옵니다. 비슷한 예제로는 `Theme.of(context)`나 `Navigator.of(context)` 등이 있습니다.

**06** 이제 `_DDay` 위젯을 구현할 차례입니다. `_DDay` 위젯은 여러 `Text` 위젯과 하트 아이콘(`IconButton`)으로 이루어졌습니다. 사귀기 시작한 날짜와 며칠이 지났는지 표시하는 글자는 날짜를 변경할 때마다 자동으로 바뀌게 코딩해야 하지만 일단은 임의의 글자들을 넣어두겠습니다.

... 생략 ...

lib/screen/home\_screen.dart

```
class _DDay extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Column(
      children: [
```

```

const SizedBox(height: 16.0),
Text( // 최상단 U&I 글자
  'U&I',
),
const SizedBox(height: 16.0),
Text( // 두 번째 글자
  '우리 처음 만난 날',
),
Text( // 임시로 지정한 만난 날짜
  '2021.11.23',
),
const SizedBox(height: 16.0),
IconButton( // 하트 아이콘 버튼
  iconSize: 60.0,
  onPressed: () {},
  icon: Icon(
    Icons.favorite,
  ),
),
const SizedBox(height: 16.0),
Text( // 만난 후 DDay
  'D+365',
),
],
);
}
}

```

... 생략 ...

▼ 실행 결과



**07** Text 위젯을 스타일링할 때는 style 매개변수를 사용하면 됩니다(6.3 '텍스트 관련 위젯'). 하지만 각 Text 위젯의 스타일이 아니라 Text 위젯의 기본 스타일을 변경하고 싶으면 어떻게 할까요? 공용으로 재사용 가능한 StatelessWidget을 만들어도 되지만 이런 상황에는 테마<sup>theme</sup>를 사용하면 편합니다. 테마를 사용하면 13가지 Text 스타일을 따로 저장하여 프로젝트로 불러와서 사용할 수 있습니다.

프로젝트 목표를 보면서 각 문장들을 스타일별로 나눕니다. 스타일명으로 display, headline, title, body, label 등의 명칭을 사용할 수 있으며 각각 large, medium small 사이즈가 존재합니다. 예를 들어 displayLarge, titleMedium, labelSmall 등의 이름으로 스타일을 미리 지정해둘 수 있습니다. 스타일명은 임의적으로 지정할 수 있습니다. 어떤 상황에 어떤 스타일명을 사용해야 한다는 절대적인 법칙은 없으니 자유롭게 본인만의 기준을 잡아서 스타일명을 적용하면 됩니다.



**08** main.dart 파일에 텍스트와 IconButton 테마를 정의하겠습니다.

```
... 생략 ...
void main() {
  runApp(
    MaterialApp(
      theme: ThemeData( // ① 테마를 지정할 수 있는 클래스
        fontFamily: 'sunflower', // 기본 글씨체
        textTheme: TextTheme( // ② 글씨 테마를 적용할 수 있는 클래스
          displayLarge: TextStyle( // headline1 스타일 정의
            color: Colors.white, // 글 색상
            fontSize: 80.0, // 크기
            fontWeight: FontWeight.w700, // 글 두께
            fontFamily: 'parisienne', // 글씨체
          ),
          displayMedium: TextStyle(
            color: Colors.white,
            fontSize: 50.0,
            fontWeight: FontWeight.w700,
          ),
          bodyLarge: TextStyle(
            color: Colors.white,
            fontSize: 30.0,
          ),
        ),
      ),
    ),
  );
}
```

lib/main.dart

```

        bodyMedium: TextStyle(
          color: Colors.white,
          fontSize: 20.0,
        ),
      ),
    ),
    home: HomeScreen(),
  ),
);
}

```

❶ MaterialApp에는 theme이라는 매개변수가 있습니다. 여기에는 ThemeData 클래스를 입력할 수 있습니다. ThemeData에서는 플러터가 기본으로 제공하는 대부분의 위젯의 기본 스타일을 지정할 수 있습니다. ❷ 글자의 테마를 정할 수 있는 매개변수입니다. 13가지 글자 스타일들을 모두 여기서 정의할 수 있습니다.

흔히 사용되는 ThemeData의 매개변수를 다음 표에 정리해두었습니다.

▼ 흔히 사용되는 ThemeData의 매개변수

매개변수	설명
fontFamily	기본 글씨체를 지정합니다.
textTheme	Text 위젯 테마를 지정합니다.
tabBarTheme	TabBar 위젯의 테마를 지정합니다.
cardTheme	Card 위젯의 테마를 지정합니다.
appBarTheme	AppBar 위젯의 테마를 지정합니다.
floatingActionButtonTheme	FloatingActionButton 위젯의 테마를 지정합니다.
elevatedButtonTheme	ElevatedButton 위젯의 테마를 지정합니다.
checkboxTheme	Checkbox 위젯의 테마를 지정합니다.

어느 정도 규칙이 보이나요? 예제로 작성한 매개변수들은 수많은 매개변수 중 극히 일부입니다. 이 매개변수들을 다 알고 있지 않아도 **위젯이름Theme**의 규칙을 이용해서 특정 위젯의 테마를 작업할 수 있습니다.

09 스타일 지정을 완료했으니 이제 Text 위젯에 스타일을 적용하겠습니다.

lib/screen/home\_screen.dart

... 생략 ...

```
class _DDay extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    // ❶ 테마 불러오기  
    final textTheme = Theme.of(context).textTheme;  
  
    return Column(  
      children: [  
        const SizedBox(height: 16.0),  
        Text(  
          'U&I',  
          style: textTheme.displayLarge, // headline1 스타일 적용  
        ),  
        const SizedBox(height: 16.0),  
        Text(  
          '우리 처음 만난 날',  
          style: textTheme.bodyLarge, // bodyText1 스타일 적용  
        ),  
        Text(  
          '2021.11.23',  
          style: textTheme.bodyMedium, // bodyText2 스타일 적용  
        ),  
        const SizedBox(height: 16.0),  
        IconButton(  
          iconSize: 60.0,  
          onPressed: () {},  
          icon: Icon(  
            Icons.favorite,  
            color: Colors.red, // ❷ 색상 빨강으로 변경  
          ),  
        ),  
        const SizedBox(height: 16.0),  
        Text(  
          'D+365',  
          // headline2 스타일 적용  
          style: textTheme.displayMedium,  
        ),  
      ],  
    );  
  }  
}
```

▼ 실행 결과





```

    ],
    );
}
}

```

① `MediaQuery.of(context)`와 똑같이 이해하면 됩니다. 위젯 트리 위 가장 가까운 Theme값을 가져옵니다. ② Icon의 색상도 Theme에서 지정할 수 있지만 Icon의 경우 색상이 각각 다른 경우가 많아서 직접 지정했습니다.

**10** 스타일을 모두 적용했으니 [Run] 버튼을 눌러서 프로젝트를 재시작하겠습니다. 변경한 theme는 MaterialApp의 매개변수에 입력했고 build() 함수에 입력되지 않은 값들은 핫 리로드로 반영이 안 되는 점을 꼭 기억하세요!

### 다양한 화면의 비율과 해상도에 따른 오버플로 해결하기

핸드폰은 화면의 비율과 해상도가 모두 다릅니다. 그렇기 때문에 하나의 화면을 기준으로 UI를 작업하면 다른 크기의 핸드폰에서 같은 UI 배치가 나오지 않을 수 있습니다. 현재 아래쪽 이미지를 화면의 반만큼 크기를 차지하도록 지정했습니다. 만약에 핸드폰의 크기가 작아서 상단의 글자들이 화면의 반 이상을 차지하면 아래쪽 이미지는 남은 공간보다 더 많은 높이를 차지합니다. 남은 공간은 화면의 반이 안 되는 데 이미지는 화면의 반을 차지하도록 코드를 작성했기 때문입니다. 플러터에서는 이런 상황을 오버플로(overflow)라고 합니다. 다음 캡처 화면은 작은 화면의 시뮬레이터 또는 에뮬레이터에서 같은 코드를 실행했을 때 오버플로가 나는 상황입니다.



이런 문제가 있을 때 해결 방법은 두 가지입니다. 글자나 이미지의 크기를 임의로 조절하거나 이미지가 남는 공간만큼만 차지하도록 코드를 작성하는 겁니다. 일반적으로 이미지가 남는 공간만큼 차지하도록 많이 작업하는데, Expanded 위젯을 사용해주면 됩니다 (6.6.4 'Expanded 위젯').

... 생략 ...

```
class _CoupleImage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Expanded( // Expanded 추가
      child: Center(
        child: Image.asset(
          'asset/img/middle_image.png',

          // Expanded가 우선순위를 갖게 되어 무시됩니다.
          height: MediaQuery.of(context).size.height / 2,
        ),
      ),
    );
  }
}
```

▼ 실행 결과



코드를 변경한 후 재실행하면 다음과 같이 오버플로 영역이 사라질 겁니다.

## 10.4.2 상태 관리 연습해보기

UI는 목표대로 완성했습니다. 이제 순차적으로 기능을 추가해볼 건데 StatefulWidget에서 setState() 함수를 사용해서 상태 관리를 하는 방법을 알아보겠습니다.

**ToDo 01** 먼저 HomeScreen을 StatefulWidget으로 변경하겠습니다.

... 생략 ...

```
class HomeScreen extends StatefulWidget {
  const HomeScreen({Key? key}) : super(key: key);

  @override
  State<HomeScreen> createState() => _HomeScreenState();
}

class _HomeScreenState extends State<HomeScreen> {
  @override
```

```
Widget build(BuildContext context) {
... 생략 ...
```

**02** 이번 프로젝트에서 상태 관리를 할 값은 ‘처음 만난 날’입니다. 이 날짜를 변수값으로 저장하고 변경하면서 사용하겠습니다. 오늘을 기준으로 이 변수값을 선언하겠습니다.

```
class _HomeScreenState extends State<HomeScreen> {
  DateTime firstDay = DateTime.now();
... 생략 ...
}
lib/screen/home_screen.dart
```

**03** 목표하는 기능은 하트 버튼을 누르면 날짜를 고를 수 있는 UI가 나오며, 날짜가 변경될 때마다 firstDay 변수를 맞춰서 변경시키는 겁니다. 하지만 현재 하트 버튼의 onPressed 매개변수가 \_DDay 위젯에 위치해 있어서 \_HomeScreenState에서 버튼이 눌렸을 때 콜백을 받을 수 없습니다. \_DDay 위젯에 하트 아이콘을 눌렀을 때 실행되는 콜백 함수를 매개변수로 노출해서 \_HomeScreenState에서 상태 관리를 하도록 코딩하겠습니다.

```
... 생략 ...
class _HomeScreenState extends State<HomeScreen> {
  DateTime firstDay = DateTime.now();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.pink[100],
      body: SafeArea(
        top: true,
        bottom: false,
        child: Column(
          mainAxisAlignment: MainAxisAlignment.spaceBetween,
          crossAxisAlignment: CrossAxisAlignment.stretch,
          children: [
            _DDay(
              // ⑤ 하트 눌렀을 때 실행할 함수 전달하기
              onPressed: onHeartPressed,
            ),
            _CoupleImage(),
          ],
        ),
      ),
    );
  }
}
lib/screen/home_screen.dart
```

```

        ],
    ),
),
);
}

void onHeartPressed(){ // ❷ 하트 눌렀을 때 실행할 함수
    print('클릭');
}

class _DDay extends StatelessWidget {
    // ❶ 하트 눌렀을 때 실행할 함수
    final GestureTapCallback onHeartPressed;

    _DDay({
        required this.onHeartPressed, // ❸ 상위에서 함수 입력받기
    });

    @override
    Widget build(BuildContext context) {
        final textTheme = Theme.of(context).textTheme;

        return Column(
            children: [
                ... 생략 ...
                IconButton(
                    iconSize: 60.0,
                    onPressed: onHeartPressed, // ❹ 아이콘 눌렀을 때 실행할 함수
                ),
                ... 생략 ...
            ],
        );
    }
}

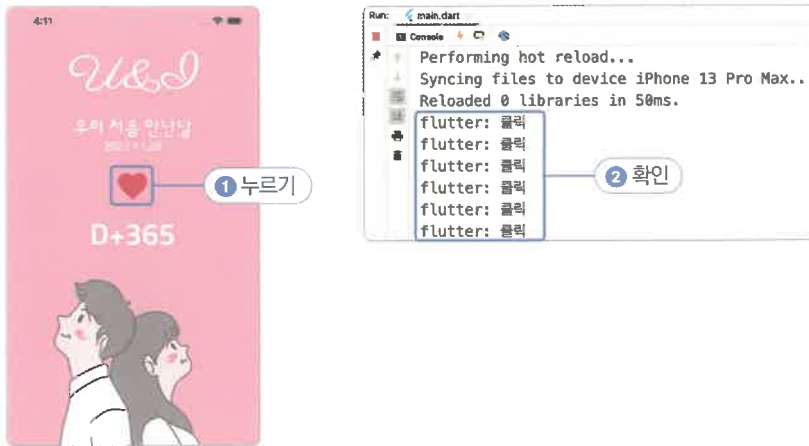
```

❶ IconButton의 onPressed 매개변수에 입력할 GestureTapCallback 타입의 변수를 정의합니다. GestureTapCallback은 Material 패키지에서 기본으로 제공하는 Typedef로, 버튼의 onPressed 또는 onTap 콜백 함수들이 GestureTapCallback 타입으로 정의돼 있습니다. 아무것도 반환하지 않고 아무것도 입력받지 않는 기본 형태의 함수로 정의돼 있습니다.

❷ onHeartPressed의 값을 생성자 매개변수를 통해 외부에서 정의받습니다. ❸ 기존에 정의했던 비어 있는 함수 대신에 onHeartPressed값을 넣어줍니다. ❹ 하트 아이콘을 눌렀을 때

실행할 함수를 정의합니다. ⑤ \_DDay 위젯 생성자에 추가된 매개변수 onHeartPressed에 \_HomeScreenState에 정의한 onHeartPressed 함수를 입력해줍니다.

04 ① 핫 리로드를 하고 하트 아이콘을 클릭합니다. ② 콘솔에 '클릭'이 잘 출력되는지 확인합니다.



05 하트 아이콘의 콜백 함수가 잘 실행되지만 아직 글자들이 firstDay 변수와 연동이 안 되어 있습니다. \_DDay 생성자에 매개변수로 firstDay값을 입력해주고 firstDay 변수를 기반으로 날짜와 D-Day가 렌더링되게 하겠습니다.

```
lib/screen/home_screen.dart

class _HomeScreenState extends State<HomeScreen> {
  DateTime firstDay = DateTime.now();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.pink[100],
      body: SafeArea(
        top: true,
        bottom: false,
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.stretch,
          mainAxisAlignment: MainAxisAlignment.spaceBetween,
          children: [
            _DDay(
              onHeartPressed: onHeartPressed,
              firstDay: firstDay, // ⑥
            ),
          ],
        ),
      ),
    );
  }
}
```

```

    ),
    _CoupleImage(),
  ],
),
),
);
}
... 생략 ...
}

class _DDay extends StatelessWidget {
  final GestureTapCallback onHeartPressed;
  final DateTime firstDay; // ❶ 사귀기 시작한 날

  _DDay({
    required this.onHeartPressed,
    required this.firstDay, // ❷ 날짜 변수로 입력받기
  });

  @override
  Widget build(BuildContext context) {
    final textTheme = Theme.of(context).textTheme;
    final now = DateTime.now(); // ❸ 현재 날짜시간

    return Column(
      children: [
        const SizedBox(height: 16.0),
        Text(
          'U&I',
          style: textTheme.displayLarge,
        ),
        const SizedBox(height: 16.0),
        Text(
          '우리 처음 만난 날',
          style: textTheme.bodyLarge,
        ),
        Text(
          '2021.11.23',
          // ❹ DateTime을 년.월.일 형태로 변경

```

```

        '${firstDay.year}.${firstDay.month}.${firstDay.day}',
        style: textTheme.bodyMedium,
      ),
      ... 생략 ...
      Text(
        'D+365',
        // ❸ DDay 계산하기
        'D+${DateTime(now.year, now.month,
          now.day).difference(firstDay).inDays + 1}',
        style: textTheme.displayMedium,
      ),
    ],
  );
}
}

```

❶ 위젯에서 사용할 DateTime값을 변수로 선언합니다. ❷ firstDay 변수의 값을 생성자의 매개변수로 외부에서 입력받도록 정의합니다. ❸ 현재 날짜시간 값을 now 변수에 저장합니다. ❹ DateTime 타입에는 year, month, day, hour, minute, second, millisecond, microsecond, weekday 등의 게터가 있습니다. 순서대로 DateTime값의 년, 월, 일, 시, 분, 초, 밀리초, 마이크로초 값을 가져올 수 있습니다.

❺ DateTime의 생성자에는 year, month, day, hour, minute, second, millisecond, microsecond 매개변수를 사용해서 원하는 날짜시간을 DateTime값으로 만들 수 있습니다. 그리고 DateTime은 difference() 함수를 사용해서 하나의 DateTime값을 또 다른 DateTime값과 비교할 수 있습니다. 추가적으로 difference() 함수는 Duration값을 반환하는데 Duration값에는 기간을 날짜로 반환하는 inDays 게터가 있습니다. 오늘 날짜와 firstDay 변수의 기간 차이를 일수로 계산하고 사귀는 첫 날은 '오늘부터 1일'로 보통 정의하기 때문에 1을 더해줬습니다.

❻ \_HomeScreenState의 firstDay 변수값을 매개변수로 입력해줬습니다.

**06** 저장하고 핫 리로드를 하면 ❶ 오늘 날짜가 처음 만난 날로 정의되고 ❷ 계산된 D-Day가 렌더링됩니다.



**07** 처음으로 `setState()` 함수를 사용하겠습니다. 상태 관리 테스트로 하트 아이콘을 누르면 `firstDay`가 하루씩 늘어나는 기능을 추가하겠습니다.

lib/screen/home\_screen.dart

```
... 생략 ...
class _HomeScreenState extends State<HomeScreen> {
  DateTime firstDay = DateTime.now();
  ... 생략 ...
  void onHeartPressed(){
    // ❶ 상태 변경 시 setState() 함수 실행
    setState(() {

      // ❷ firstDay 변수에서 하루 빼기
      firstDay = firstDay.subtract(Duration(days: 1));
    });
  }
}
```

❶ `setState()` 함수를 사용하는 방법입니다. 매개변수에 함수를 입력하고 함수에 변경하고 싶은 변수값을 지정해주면 됩니다. 원하는 만큼 기간을 뺄 수 있는 `subtract()` 함수를 사용해서 하트 버튼이 눌릴 때마다 `firstDay`값이 하루씩 줄어드는 기능을 추가했습니다.

❷ `DateTime`은 날짜와 시간을 저장할 수 있는 변수 타입이고 `Duration`은 기간을 정할 수 있는 변수 타입입니다. 매개변수에 `days`, `hours`, `minutes`, `seconds`, `milliseconds`, `microseconds`값들을 사용해서 원하는 기간을 정의할 수 있습니다.



08 ① 아이콘 버튼을 누르면 ② 처음 만난 날 날짜와 D-Day가 변경되는 걸 확인하겠습니다.



이로써 setState() 함수 사용법을 익혀봤습니다. 어떨까요? 복잡한 상태 관리 그림에 비해 생각보다 간단하지 않아요? firstDay값을 새로 지정하는 코드를 setState() 함수 없이 지정하면 아무리 하트 버튼을 눌러도 UI가 반영되지 않습니다. 값은 바뀌는데 build() 함수를 재실행하라는 신호를 받지 못하는 거죠. 꼭 한 번 테스트를 해보시길 바랍니다!

### 10.4.3 CupertinoDatePicker로 날짜 선택 구현하기

**ToDo 01** 하트 아이콘을 누르면 날짜가 변경되는 것까진 했는데 아직 직접 날짜를 고를 수 있는 기능을 만들지 못했습니다. showDialog() 함수와 CupertinoDatePicker 위젯을 사용해서 아이콘을 클릭하면 날짜를 선택할 수 있는 CupertinoDatePicker가 화면에 생성되도록 구현하겠습니다.

```
import 'package:flutter/material.dart';  
  
// ① 쿠퍼티노 (iOS) 위젯 사용하기 위해 필요  
import 'package:flutter/cupertino.dart';  
  
class _HomeScreenState extends State<HomeScreen> {  
  DateTime firstDay = DateTime.now();  
}
```

```

... 생략 ...
void onHeartPressed() {
    showCupertinoDialog( // ❷ 쿠퍼티노 다이얼로그 실행
        context: context, // ❸ 보여줄 다이얼로그 빌드
        builder: (BuildContext context) {
            // ❹ 날짜 선택하는 다이얼로그
            return CupertinoDatePicker(
                // ❺ 시간 제외하고 날짜만 선택하기
                mode: CupertinoDatePickerMode.date,
                onDateTimeChanged: (DateTime date) {},
            );
        },
    );
}
... 생략 ...

```

▼ 실행 결과



❶ Cupertino 패키지를 불러옵니다. ❷ showCupertinoDialog를 실행해서 하트 아이콘을 누르면 다이얼로그를 열어줍니다. ❸ builder 매개변수에 입력되는 함수에 다이얼로그에 보여주고 싶은 위젯을 반환해주면 해당 위젯을 다이얼로그에서 보여줄 수 있습니다. ❹ CupertinoDatePicker는 Cupertino 패키지에서 기본으로 제공하는 위젯입니다. 스크롤을 통해서 날짜를 정할 수 있고 정해진 값을 onDateTimeChanged 콜백 함수의 매개변수로 전달해줍니다.

❺ mode 매개변수는 날짜를 고르는 모드를 지정할 수 있습니다.

- CupertinoDatePickerMode.date : 날짜
- CupertinoDatePickerMode.time : 시간
- CupertinoDatePickerMode.dateAndTime : 날짜와 시간

**02** 하트 아이콘을 눌러서 CupertinoDatePicker가 다이얼로그에 제대로 실행되는지 확인하겠습니다.

현재는 다이얼로그를 닫을 방법이 없습니다. 다이얼로그를 없애고 싶으면 [Run] 버튼을 눌러서 앱을 재실행해주세요.

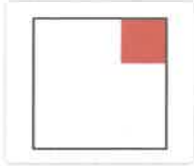
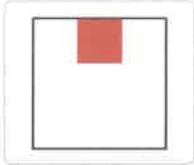
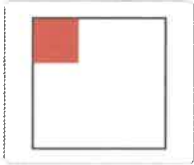
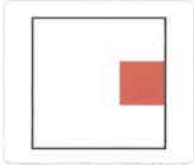

**03** 하트 아이콘을 누르면 다이얼로그가 열리면서 배경이 어두워지고 CupertinoDatePicker가 보이지만 계획과 달리 아래 공간만 조금 차지하는 흰색 배경의 형태가 아닙니다. 화면 아래에서 300픽셀만 CupertinoDatePicker가 차지하게 하고 CupertinoDatePicker의 배경을 흰색으로 변경하겠습니다.

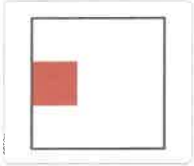
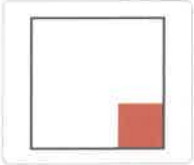
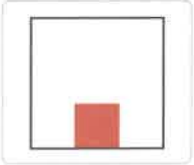
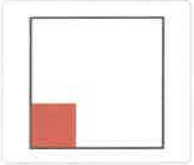
lib/screen/home\_screen.dart

```
... 생략 ...
class _HomeScreenState extends State<HomeScreen> {
  ... 생략 ...
  void onHeartPressed() {
    showCupertinoDialog(
      context: context,
      builder: (BuildContext context) {
        return Align( // ❶ 정렬을 지정하는 위젯
          alignment: Alignment.bottomCenter, // ❷ 아래 중간으로 정렬
          child: Container(
            color: Colors.white, // 배경색 흰색 지정
            height: 300, // 높이 300 지정
            child: CupertinoDatePicker(
              mode: CupertinoDatePickerMode.date,
              onDateTimeChanged: (DateTime date) {},
            ),
          ),
        );
      },
      barrierDismissible: true, // ❸ 외부 탭할 경우 다이얼로그 닫기
    );
  }
}
... 생략 ...
```

❶ Align 위젯은 자식 위젯<sup>child widget</sup>을 어떻게 위치시킬지 정할 수 있습니다. ❷ Align 위젯의 alignment 매개변수에는 Alignment값을 입력할 수 있습니다. 여기서는 Alignment.bottomCenter를 입력해서 아래 중간에 자식 위젯을 배치했습니다. ❸ showCupertinoDialog의 barrierDismissible 매개변수는 배경을 눌렀을 때의 행동을 지정합니다. false값을 입력하면 배경을 눌러도 다이얼로그가 닫히지 않고 true값을 누르면 배경을 눌렀을 때 다이얼로그가 닫힙니다. Alignment의 정렬값을 표로 정리해뒀으니 참고하기 바랍니다.

▼ Alignment의 정렬값

속성	예제
Alignment.topRight 위 오른쪽	
Alignment.topCenter 위 중앙	
Alignment.topLeft 위 왼쪽	
Alignment.centerRight 중앙 오른쪽	
Alignment.center 중앙	

속성	예제
Alignment.centerLeft 중앙 왼쪽	
Alignment.bottomRight 아래 오른쪽	
Alignment. bottomCenter 아래 중앙	
Alignment.bottomLeft 아래 왼쪽	

**04** [Run] 버튼을 눌러서 앱을 재시작하고 다시 하트 아이콘을 누르면 CupertinoDatePicker 위젯이 아래에 흰색 배경으로 이쁘게 배치되는 걸 볼 수 있습니다. ❶ 이제 배경을 누르면 ❷ 다이얼로그가 닫힙니다.



#### 10.4.4 CupertinoDatePicker 변경 값 상태 관리에 적용하기

**ToDo 01** 이제 마지막 단계가 남았습니다. CupertinoDatePicker를 보여주는데 성공했고 setState() 함수를 이용한 상태 관리도 맞보았습니다. 이제 둘을 연결해서 CupertinoDatePicker의 날짜 값이 변경될 때마다 firstDay값을 변경하겠습니다.

```
... 생략 ...
lib/screen/home_screen.dart

void onHeartPressed() {
  showCupertinoDialog(
    context: context,
    builder: (BuildContext context) {
      return Align(
        alignment: Alignment.bottomCenter,
        child: Container(
          color: Colors.white,
          height: 300,
          child: CupertinoDatePicker(
```

```

        mode: CupertinoDatePickerMode.date,
        // ❶ 날짜가 변경되면 실행되는 함수
        onDateTimeChanged: (DateTime date) {
            setState(() {
                firstDay = date;
            });
        },
    ),
),
);
},
barrierDismissible: true,
);
}
... 생략 ...

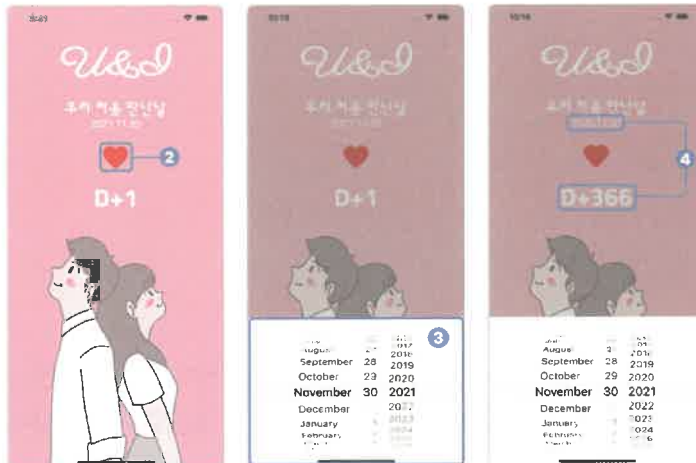
```

❶ onDateTimeChanged의 콜백 함수는 CupertinoDatePicker 위젯에서 날짜가 변경될 때마다 실행됩니다. 결과적으로 콜백 함수가 실행될 때마다 매개변수로 제공되는 date값을 firstDay 변수에 저장해주기만 하면 됩니다.

02 이제 CupertinoDatePicker에서 날짜를 변경해주면 '우리 처음 만난 날' 및 D-Day에 선택한 날짜값이 반영됩니다.

## 10.5 테스트하기

- 1 안드로이드 스튜디오에서 [Run] 버튼을 눌러서 시뮬레이터, 에뮬레이터 또는 본인 기기에서 앱을 실행해보세요.
- 2 빨간 하트 아이콘 버튼을 눌러서 CupertinoDatePicker를 다이얼로그로 실행해보세요.
- 3 다이얼로그가 열리고 CupertinoDatePicker가 렌더링되는 걸 확인해주세요.
- 4 CupertinoDatePicker 위젯을 조작해서 원하는 날짜를 선택합니다.



- 5 선택한 날짜가 '우리 처음 만난 날' 및 D-Day에 반영되는 걸 확인합니다.
- 6 배경을 눌러서 다이얼로그를 닫아주세요.

