week2

flutter는 여러 위젯의 조합으로 웹페이지 쉽게 만들수 있음.

캐릭터 카드 페이지 만들기

```
# 기본

import 'package:flutter/material.dart';

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Charactor card',
      home: Mycard(),
    );
  }
}

class Mycard extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('BRANTO'),
      ),
    );
  }
}
```

- title을 정중앙으로 옮기기: `centerTitle: true`
- `backgroundColor: Colors.RedAccent` → app바 색상 지정
- `elevation: 0.0` → app바가 떠있는걸 없애줌, 수치로 조절가능.

padding widget

padding: 특정 지점으로부터 사용하고자하는 위젯이 떨어져야할 거리 지정

appbar 밖으로 나와서 한줄 밑으로 내려오고

- body: Padding() 불러옴

- 정확한 패딩 속성을 지정하기 위해 한번더 불러옴 padding: EdgeInsets.fromLTRB → 원하는 숫자로 패딩값 지정해줌
- child: Column(→ padding 위젯의 child 위젯으로 불려왔기 때문에, 앱 화면에서는 왼쪽으로 30 위로부터 40 떨어진 곳에 위젯 새로로(column 위젯 새로로 배치시킴) 배치
- mainAxisAlignment 앱스크린 내에서 위젯들 세로로 정렬할때 쓰임. column은 위젯을 새로로만 배치시키지만, axis는 위젯을 앱스크린 내에서 새로축으로 상단 하단으로 정렬할 때 쓰임.

(참고! 위젯 지울때는 뒤에 클로징 레이블의 괄호부터 지워나가면 됨)

```
/* padding 위젯 삭제 전
import 'package:flutter/material.dart';

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Charactor card',
      home: Mycard(),
    );
  }
}

class Mycard extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('BRANTO'),
        centerTitle: true,
        backgroundColor: Colors.redAccent,
        elevation: 0.0,
      ),
      body: Padding(
          padding: EdgeInsets.fromLTRB(30.0, 40.0, 0.0, 0.0),
          child: Column(
              mainAxisAlignment: MainAxisAlignment.center,
              children: <Widget>[
                Text('Hello'),
                Text('Hello'),
                Text('Hello'),
              ])),
    );
  }
}
*/
```

```
// 패딩 위젯 삭제
```

- 이제 column 위젯을 center 위젯으로 감싸보겠음. column 위젯 클릭해보면 노란 정보가 생기는데, 거기에 center 위젯 클릭하면 됨. → 텍스트가 화면 정중앙에 보이게 됨.

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Charactor card',
      home: Mycard(),
    );
  }
}

class Mycard extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('BRANTO'),
        centerTitle: true,
        backgroundColor: Colors.redAccent,
        elevation: 0.0,
      ),
      body: Center(
        child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: <Widget>[
              Text('Hello'),
              Text('Hello'),
              Text('Hello'),
            ]),
      ),
    );
  }
}
```

일반적으로 center 위젯은 child 위젯을 단순히 정중앙에 배치한다고 여기고 넘어갈때가 많음. 그러나 center 위젯이 있다고 해서 항상 화면 위젯이 정중앙에 위치되는건 아님. → column위젯과 center 위젯이 만나면, center 위젯은 column위젯의 자식들에 대한 세로축 위치에 대해서는 관여하지 않고. 현재 column 위젯의 자식 위젯 세로축 높이에 자동으로 적용됨. 그래서 세로

축 정렬해주는 mainaxis 지우면 가로축으로만 정중앙에 위치함. 그래서 center위젯과 column 위치가 결합되었을때 세로축상에서 정중앙에 column 위젯의 자식들을 위치시키려면 mainAxisAlignment가 필요함. ⇒ **정리하자면, column 위젯을 가로축상으로 정중앙에 위치하려면 center 위젯을 사용하고, 세로축으로 정중앙에 위치시키려면 column 위젯 내에서 mainAxisAlignment 속성을 사용하면 됨.**

새로운 프로젝트 시작

11강→ 이제 이미지만 넣으면 됨. 우클릭 해서 새 폴더 만들기. 폴더명은 assets. 이 폴더로 사용할 이미지 두개를 끌어다놓으면 됨.

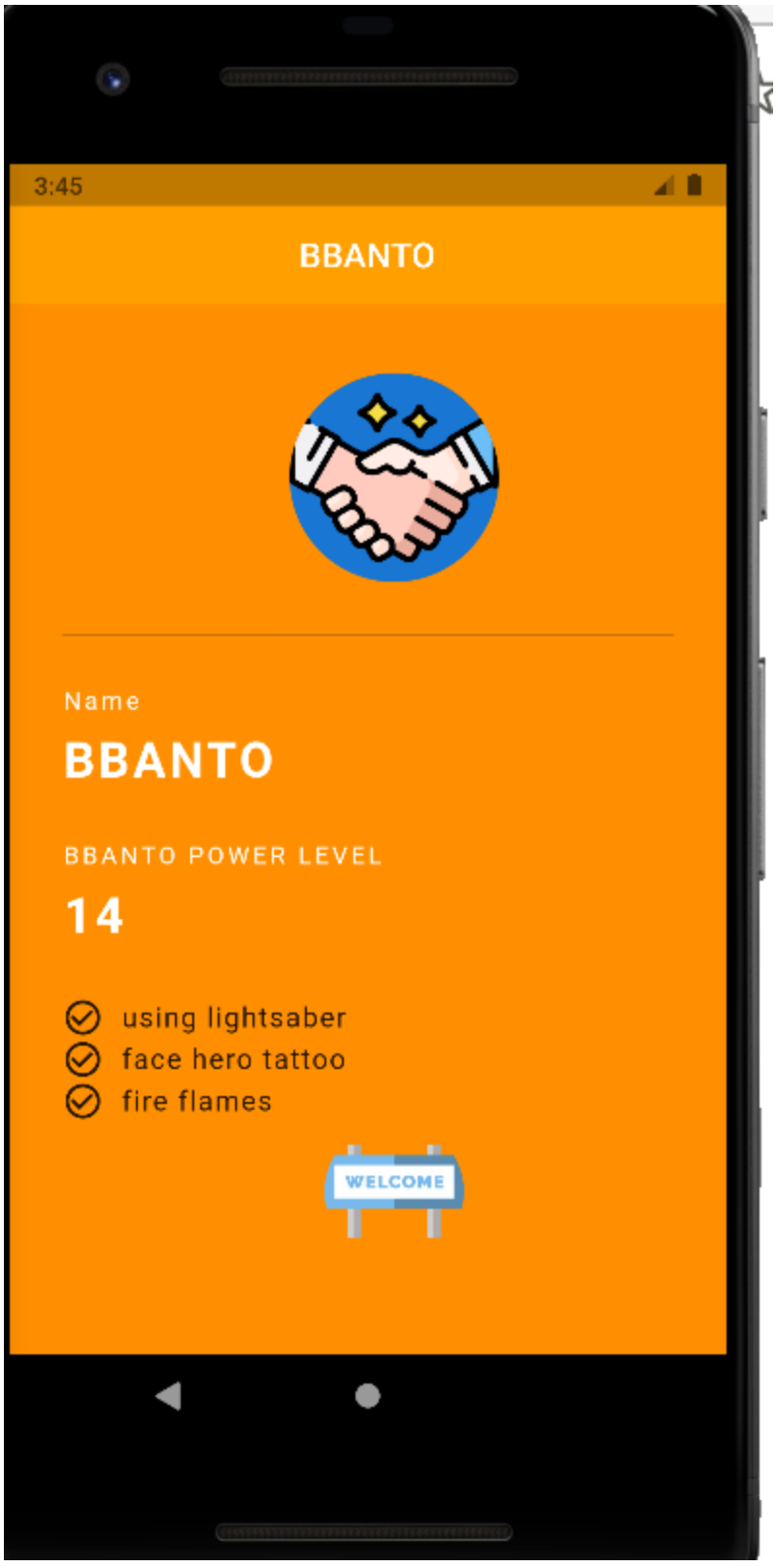
pubspec.yaml 페이지 클릭 → 대부분의 내용들이 커멘트 처리되어 있음. 여기에 assets에 이미지 경로가 보임. 이 부분을 드래그 후 crtl + / 눌러서 활성화 시킴. 다시 커멘트 시키려면 다시 crtl + / 처리해주면 됨 → 그리고 경로를 우리가 만든 폴더명과 이미지 파일명으로 입력해줌. (assets/image.png) 이런식으로 (들여쓰기에 상당히 민감하니 주의하기!) -밑에 exit code 0이 뜨면 이미지가 잘 등록된 거임.

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false, // 빨간띠로 앱 위에 debug라고 뜨는게 없어짐
      title: 'BBANTO',
      home: Grade(),
    );
  }
}

class Grade extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.amber[800], // background색상 설정
      appBar: AppBar(
        title: Text('BBANTO'),
        backgroundColor: Colors.amber[700],
        centerTitle: true,
        elevation: 0.0,
      ),
      body: Padding(
        padding: EdgeInsets.fromLTRB(30.0, 40.0, 0.0, 0.0),
```

```
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start, // 두 텍스트의 시작점 정렬 맞추기
          children: <Widget>[
            Center(
              child: CircleAvatar(
                // 이 위젯 클릭하면 나오는 좌측 전구 클릭해서 wrap with center 클릭하면 center 위젯이
 서클을 감싸고 이미지가 중앙으로 이동함!
                // 캐릭터처럼 동그리마 안에 하라고
                backgroundImage: AssetImage('assets/cooperate.png'),
                radius: 60.0, // circle의 크기 조절,
              ),
            ),
            Divider(
                // 구분선 만들어줌
                height:
                    60.0, // divider 자체의 높이가 아니라 위와 아래 사이의 간격이 합쳐서 60.0이라는 의
미. 즉 위로부터 30, 네임으로부터 30 떨어져있따.
                color: Colors.grey[850],
                thickness: 0.5, // 선두께
                endIndent: 30.0 // 선이 끝에서 어느정도 떨어져있을지 설정,
                ),
            Text(
              'Name',
              style: TextStyle(
                color: Colors.white,
                letterSpacing: 2.0, // 철자 간격 넓게 조정
              ),
            ),
            SizedBox(
              height: 10.0,
            ),
            Text(
              'BBANTO',
              style: TextStyle(
                  color: Colors.white,
                  letterSpacing: 2.0,
                  fontSize: 28.0,
                  fontWeight: FontWeight.bold),
              // 텍스트간의 길이가 달라서 시작점 정렬이 안되어있음 -> sizebox 사용해서 두 텍스트 사이의 가
로세로 크기 맘대로 조절하는 박스 넣어서 조절
            ),
            SizedBox(
              height: 30.0,
            ),
            Text(
              'BBANTO POWER LEVEL',
              style: TextStyle(
                color: Colors.white,
                letterSpacing: 2.0, // 철자 간격 넓게 조정
              ),
            ),
            SizedBox(
              height: 10.0,
            ),
```

```
            Text(
              '14',
              style: TextStyle(
                  color: Colors.white,
                  letterSpacing: 2.0,
                  fontSize: 28.0,
                  fontWeight: FontWeight.bold),
              // 텍스트간의 길이가 달라서 시작점 정렬이 안되어있음 -> sizebox 사용해서 두 텍스트 사이의 가
로세로 크기 맘대로 조절하는 박스 넣어서 조절
            ),
            SizedBox(
              height: 30.0,
            ),
            Row(
              children: <Widget>[
                //widget 대문자 주의
                Icon(Icons.check_circle_outline),
                SizedBox(
                  width: 10.0,
                ), // 문자와 아이콘 사이의 간격을 적당히 해주기 위함임. hegith가 아닌 width인 이유는 가
로로 늘어져있기 때문
                Text(
                  'using lightsaber',
                  style: TextStyle(
                    fontSize: 16.0,
                    letterSpacing: 1.0,
                  ),
                ), // .은 위젯이 가지고 있는 여러가지 속성이나 기능 관련 아이템중 하나를 선택하고 싶을 때
 사용함.
              ],
            ),
            Row(
              children: <Widget>[
                //widget 대문자 주의
                Icon(Icons.check_circle_outline),
                SizedBox(
                  width: 10.0,
                ), // 문자와 아이콘 사이의 간격을 적당히 해주기 위함임. hegith가 아닌 width인 이유는 가
로로 늘어져있기 때문
                Text(
                  'face hero tattoo',
                  style: TextStyle(
                    fontSize: 16.0,
                    letterSpacing: 1.0,
                  ),
                ), // .은 위젯이 가지고 있는 여러가지 속성이나 기능 관련 아이템중 하나를 선택하고 싶을 때
 사용함.
              ],
            ),
            Row(
              children: <Widget>[
                //widget 대문자 주의
                Icon(Icons.check_circle_outline),
                SizedBox(
```

```
                  width: 10.0,
                ), // 문자와 아이콘 사이의 간격을 적당히 해주기 위함임. hegith가 아닌 width인 이유는 가
로로 늘어져있기 때문
                Text(
                  'fire flames',
                  style: TextStyle(
                    fontSize: 16.0,
                    letterSpacing: 1.0,
                  ),
                ), // .은 위젯이 가지고 있는 여러가지 속성이나 기능 관련 아이템중 하나를 선택하고 싶을 때
 사용함.
              ],
            ),
            Center(
              child: CircleAvatar(
                backgroundImage: AssetImage('assets/traffic-sign.png'),
                radius: 40.0, // 뒤에 파란색 배경이 별로고 위아래 둘다 동그라미 이미지가 별로임
                backgroundColor: Colors.amber[800],
              ),
            )
          ],
        ),
      ),
    );
  }
}
```


3:45
BBANTO
Name
BBANTO
BBANTO POWER LEVEL
14
using lightsaber
face hero tattoo
fire flames
WELCOME

Class and Widget

예) 스마트폰 - 설계도, 어느곳에 어느 부품 들어가는지 명시 + 각 부품의 기능 명시해야 함. 설계도에 근거해서 스마트폰 만들어냄. 똑같은 설계도와 부품에 근거해서 만들어진 각각의 스마트폰이 완벽하게 동일한 것일까? 각각의 시리얼 넘버와 부품도 각각의 고유 시리얼 넘버를 갖고 있을것이고, 사용자도 각자의 유심을 넣어서 사용함. 하나의 같은 틀에서 찍어낸 스마트폰이더라도 각각 구별 가능한 스마트폰으로 존재함.

객체지향? 현실 세상에서 객체는 보거나 만지고 이해할 수 있는 모든 것을 의미함. 객체를 속성이나 기능으로 정의할 수 있음. 다시 스마트폰 설계도로 돌아와, 부품과 기능을 정의한 설계도면의 역할을 class가 함.

- class: 어떤 객체에 대한 속성과 기능에 대한 정의. class에 정의된 속성과 기능대로 만들어진 사물을 인스턴스라고 함. 설계도면 대로 만들어진 폰 하나하나는 인스턴스. 모양과 기능은 다 동일하지만 시리얼 넘버 등으로 각 인스턴스로서 스마트폰은 구별이 가능함. 코딩에서 객체와 인스턴스를 혼용해서 쓰곤 함. 하지만 엄밀히는 다름. dart로 class라고 선언하고 사람의 속성과 기능을 정의하는 순간 이 정보는 컴퓨터의 메모리상에 할당됨. 프로그래밍상에서 객체란 메모리에 할당되어진 순간의 클래스를 의미함. 클래스를 기반으로 한 사람이 코드상으로 새롭게 만들어질 때 인스턴스라고 함.
- class{

  속성: 액정, 카메라 모듈, 홈버튼, 스크린

  기능: 전화 걸기, 사진찍기, 인터넷 검색

  }

  (그렇다고 class에서 늘 속성, 기능 모두 정의해야하는 건 아님.)

⇒ 정리하면,

- 프로그래밍 상에서의 클래스란?

  객체가 가져야 하는 속성과 기능을 정의한 내용을 담고 있는 설계도 역할

- 프로그래밍 상에서의 객체란?

  클래스가 정의된 후 메모리상에 할당되었을 때 이를 객체라고 함

- 프로그래밍 상에서의 인스턴스란?

  클래스를 기반으로 생성 됨

  클래스의 속성과 기능을 똑같이 가지고 있고 프로그래밍 상에서 사용되는 대상

검색창에 dartpad 입력 →

변수 타입

- a = 2;
- a = 3;
- 프로그래밍 언어에서 같다라는 의미는 == 두번 써줌. 하나의 =는 오른쪽의 숫자나 문자를 왼쪽에 대입했다고 이해해야함.

```
class Person{
  String name = 'John';
  /* datatype에 대한 학습.
   * 굳이 변수 타입 지정 안해도 var를 앞에 쓰면
   * dart가 알아서 변수 타입을 추론한다는 뜻임 */
  int age;
  String sex;
}
// 이제 하나의 완벽한 클래스가 만들어짐. 메모리에 할당된 클래스를 객체라고 하고. 지금 객체가 등록됨.
// 이제 인스턴스를 만들어보겠음.

// main 함수 선언
void main(){

  Person p1 = new Person();
  p1.age = 30;
  print(p1.age);
}

// 나의 경우 null 오류가 났음. age랑 sex에 아무것도 지정안해줘서. 근데 이게 뭐 다트 업데이트때매 어쩔수없는듯
```

13강

class and widget

- 생성자와 관련된 함수의 구조와 기능
- 생성자의 구조와 역할
- 클래스와 위젯의 관계

dart는 typelanguage. 즉, 다트 내의 모든 것은 타입을 가지고 있음. addNumber 앞에 void 타입을 넣으면 에러가 뜸. void 타입은 아무것도 반환하지 않는 함수 타입임. addnumber 함수는 정수 입력받아 합 반환하기 때문에 데이터타입은 int형임. addnumber 앞에 int 쓰면 오류 없어짐.

→ 왜 매번 type 지정해줘야할까? 타입 추정가능하다고 했는데..

정수를 원하는데 소수가 나올수도 있어서

다시 생성자로 돌아와서 생성자 만들어보면.

생성자는 함수의 모양을 가지고 있기 때문에, argument 전달받을 괄호가 함수명 뒤에 필요함. 중괄호로 바디를 만들어줌 ⇒ 생성자의 기본 구조.

생성자는 인스턴스가 생성될 때 딱 한번만 호출됨. addNumber를 호출할때는, main 함수 내에서 함수명 뒤에 괄호를 붙이고 세미콜론을 입력했음 → addNumber();

생성자도 마찬가지이다. 일반함수처럼 class의 속성으로 만들었떤 변수들을 argument에 넣어줌. argument의 변수명은 똑같이 지정하지 않고 마음대로 정해도 되고 ,로 구분해주면 됨.

```
class Person{
  String name = 'John';
  /* datatype에 대한 학습.
   * 굳이 변수 타입 지정 안해도 var를 앞에 쓰면
   * dart가 알아서 변수 타입을 추론한다는 뜻임 */
  int age;
  String sex;

  Person(String name, int age, String sex){
    this.name = name;
    this.age = age;
    this.sex = sex;
  }
}
// 이제 하나의 완벽한 클래스가 만들어짐. 메모리에 할당된 클래스를 객체라고 하고. 지금 객체가 등록됨.
// 이제 인스턴스를 만들어보겠음.

addNumber(int num1, int num2){
  return num1 + num2;
} // 함수명 시작은 소문자고, 두번째단어는 대문자로 시작함. 함수인 이상 괄호값이 나옴. 중괄호로 바디 부분 만들어줌. 바디부분에는 함수가 실행할 기능들 정의해줌
// main 함수 선언
void main(){

  Person p1 = new Person('Tom', 30, 'male');
  Person p2 = new Person('Jane', 27, 'female');// 변수값 할당해줘야 인스턴스 오류 없이 만들어짐
  /* 사람에 대한 속성을 정의하면서 변수명만 만들고 변수값은 지정해주지 않아서.
```

```
   * 메인 함수에서 person class의 모든 속성을 전달해주기 위해
   * person class 타입으로 p1이라는 인스턴스를 생성하고
   * p1.age로 age 변수를 불러오고 30이라는 숫자 할당했음.
   * 근데 매번 class를 통해서 인스턴스를 생성한 후에 일일이 모든 클래스 내에 변수들에 변수값을 할당하는 일 번거로움
   * 그렇다고 미리 변수값 할당하면, 모든 인스턴스 같은 값을 갖게 됨
   * 이런 불편함과 오류 해결하기 위해
   * 클래스 네임에 생성자를 가지고 있음.
   * 우리가 만든 class에는 속성 말고 다른 요소는 전혀 없음.
   * 클래스를 만들때 따로 생성자를 만들지 않으면, 다트가 알아서 만들어줌 - 투명한 생성자
   * 지금부터 클래스 내에 생성자 만들어주겠음*/

  print(p1.age);
  print(p2.age);

}
```

만약 argument가 너무 많다면..? → named arguent → 생성자가 가지는 argument를 중괄호로 묶어서 다 선택사항으로 바꿔버림. 인스턴스의 argument에 입력하지 않아도 됨. 순서도 상관없고 쓰고 싶은거만 쓰면됨.

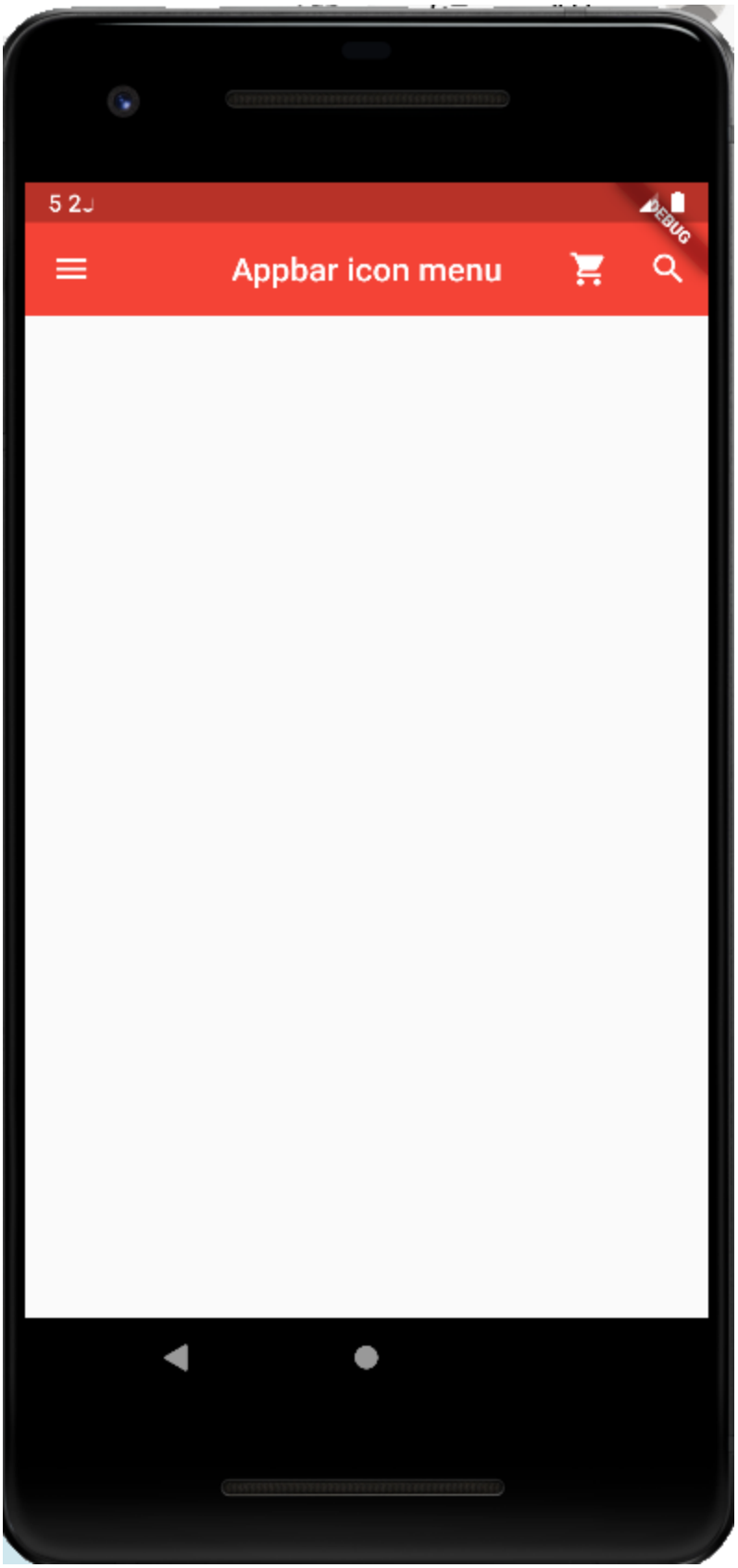
```
class Person{
  String name = 'John';
  int age;
  String sex;

  Person({String name, int age, String sex}){
    this.name = name;
    this.age = age;
    this.sex = sex;
  }
}
// 이제 하나의 완벽한 클래스가 만들어짐. 메모리에 할당된 클래스를 객체라고 하고. 지금 객체가 등록됨.
// 이제 인스턴스를 만들어보겠음.

addNumber(int num1, int num2){
  return num1 + num2;
} // 함수명 시작은 소문자고, 두번째단어는 대문자로 시작함. 함수인 이상 괄호값이 나옴. 중괄호로 바디 부분 만들어줌. 바디부분에는 함수가 실행할 기능들 정의해줌
// main 함수 선언
void main(){

  Person p1 = new Person(age:30);
  Person p2 = new Person(sex:'male');

  print(p1.age);
  print(p2.age);

}
```

⇒ 지금까지 모든 위젯들은 class로 생성된거였음... 일반적으로 위젯을 class라고 표현하기도 함.

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Appbar',
      theme: ThemeData(primarySwatch: Colors.red),
      home: MyPage(),
    );
  }
}

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Appbar icon menu'),
        centerTitle: true,
        elevation: 0.0,
        leading: IconButton(
          icon: Icon(Icons.menu),
          onPressed: () {
            print('menu button is clicked');
          }, // 기본적인 icon 메뉴만 만들어서
        ), // 타이틀 왼쪽에 위치시키는 햄버거 버튼
        actions: <Widget>[
          IconButton(
            icon: Icon(Icons.shopping_cart),
            onPressed: () {
              print('Shopping cart button is clicked');
            }, // 기본적인 icon 메뉴만 만들어서
          ),
          IconButton(
            icon: Icon(Icons.search),
            onPressed: () {
              print('search button is clicked');
            }, // 기본적인 icon 메뉴만 만들어서
          ),
        ],
      ),
    );
```

```
  }
}
```

```
// app bar icon 부분만 보면
leading: IconButton(
          icon: Icon(Icons.menu),
          onPressed: () {
            print('menu button is clicked');
          }, // 기본적인 icon 메뉴만 만들어서
        ), // 타이틀 왼쪽에 위치시키는 햄버거 버튼
        actions: <Widget>[
          IconButton(
            icon: Icon(Icons.shopping_cart),
            onPressed: () {
              print('Shopping cart button is clicked');
            }, // 기본적인 icon 메뉴만 만들어서
          ),
          IconButton(
            icon: Icon(Icons.search),
            onPressed: () {
              print('search button is clicked');
            }, // 기본적인 icon 메뉴만 만들어서
```


Appbar icon menu
DEBUG

App bar icon button

- leading: 아이콘 버튼이나 간단한 위젯을 왼쪽에 배치할 때
- actions: 복수의 아이콘 버튼 등을 오른쪽에 배치할 때
- onPressed: 함수의 형태로 일반 버튼이나 아이콘 버튼을 터치했을 때 일어나는 이벤트를 정의한 곳