

Dart - 클래스

☯ 상태

플러터

class에서 property를 선언할 때는 var가 아닌 타입을 이용한다.

```
class Player{
  String name = 'nico';
  int xp = 1500;
}

void main(){
  var player = Player();    // 클래스가 하나 생성된다.
  print(player.name);       // 기본값인 'nico' 출력
  player.name = 'lala';
  print(player.name);       // 이름이 lala가 되었으므로 'lala'출력
}
```

만약 player의 name을 바꾸지 못하게 하고 싶다면? final 쓰면 됨.

```
class Player{
  final String name = 'nico';
  int xp = 1500;
}
```

중요! dart에서는 클래스 안의 변수를 사용할 때 this를 사용하지 않아도 된다! `#{this.name}` 뭐 이렇게 써도 동작은 하는데, 클래스 메서드 내에서 this를 사용하는 것은 권장되지 않는다.

```
class Player{
  String name = 'nico';
  int xp = 1500;

  void sayHello(){
    print("hi I'm #{name}");
  }
}
```

물론 메서드 내에 같은 이름의 변수가 있다면 this를 써야겠지.

```
class Player{
  String name = 'nico';
  int xp = 1500;

  void sayHello(){
    var name = 'lala';
    print("hi I'm ${this.name}");
  }
}
```

근데 이런거 아니면 this 쓰지 않지~

Constructors

클래스에 인자를 전달함으로써 새로운 오브젝트를 만들어보자.

constructor method를 만들어 주어야 하는데, 이 이름은 class이름과 동일해야한다.

```
class Player{
  late final String name;    // 이름이 한 번 지정되면 변경되지 않게, 그러나 값을 나중에
  int xp;                    // 입력받을 거니까.

  Player(String name, int xp){
    this.name = name;        // 여기를 위해 late를 사용한다.
    this.xp = xp;
  }

  void sayHello(){
    var name = 'lala';
    print("hi I'm ${this.name}");
  }
}

void main(){
  var player = Player("nico", 1500);
  player.sayHello();
  var player2 = Player("lynn", 2500);
  player2.sayHello();
}
```

그런데 우리 오브젝트가 가질 변수들의 타입을 클래스에서 정의해줬는데, 꼭 name과 xp의 타입을 계속 정의해주어야하나?

그래서 이걸 줄일 수 있다.

사용자가 보낸 인자의 순서에 따라 오브젝트를 만드는 것~ late도 지울 수 있음

```
class Player{
  final String name;
  int xp;

  Player(this.name, this.xp);
}
```

와~ 깔끔해~

이렇게 할 때는 위치가 중요함!!!

Named Constructor Parameters

그런데 위치로 하면 클래스가 클 때 통제가 힘들어지니까,(타입만 알려줘서 사용할 때 뭐였는지 보고와야해...) 이름을 붙여서 보내보자.

단순하다. { } 를 사용하는 것...

```
class Player{
  final String name;
  int xp;

  Player({
    required this.name,
    required this.xp});
}
```

required를 붙인 이유는 이 클래스로 오브젝트를 만들기위해 필수인 값이라는 뜻임 변수 선언과 같아

함수를 사용할 때에 조금 달라진다.

```
void main(){
  var player = Player(
    name: "nico",
```

```

        xp: 1200,
    );
    player.sayHello();
}

```

플러터에는 위치기반이 많지만, 일단 좋아보이는 건 이름 붙인 것이지 않나?

```

class Player{
    final String name, team;
    int xp, age;
}

```

헐라리... 같은 타입이면 한 줄에 선언 가능, 근데 위 처럼 선언하면 name과 team 모두 final 이 되니까 유의할 것.

Named Constructors

기본적으로 클래스로 오브젝트를 만들 때 클래스명과 같은 constructor가 호출된다. 그러나 이렇게 새로 만들 때 경우에 따라 다르게 동작하길 원한다면??

우선 사용법은 아래와 같다.

예를 들어 한 constructor는 blue팀의 xp가 0인 플레이어를 만들고, 다른 것은 red팀의 xp가 0인 플레이어를 만든다고 가정하자.

```

void main(){
    var player = Player.createBluePlayer(
        name: "nico",
        age: 21,
    );

    var redplayer = Player.craeteRedPlayer(
        name: "lynn",
        age: 24,
    );
}

```

사용자가 이름과 나이를 보내면 자동으로 팀과 xp를 할당하는 것이다.

```

class Player{
  final String name, team;
  int xp, age;

  Player({
    required this.name,
    required this.xp,
    required this.team,
    required this.age,
  });

  Player.createBluePlayer({required String name, required int age}):
    this.age = age,
    this.name = name,
    this.team = 'blue',
    this.xp = 0;

  Player.createRedPlayer({required String name, required int age}):
    this.age = age,
    this.name = name,
    this.team = 'red',
    this.xp = 0;
}

```

▪

저기 새로운 constructor 뒤에 붙는 것은 세미콜론이 아니라 콜론임 조심~~~

: 뒤에서 하는 일은 Player 클래스를 초기화 시키는 것이다.

: 는 dart한테 player 오브젝트 초기화시키겠다고 말해주는 것~~~

뭐... 이것도 정의할 때 positional을 쓰든 named쓰든 상관없음 required랑 {} 떼어주면 되지.

```

Player.fromJson(Map<String, dynamic> playerJson)
  : name = playerJson['name'],
    xp = playerJson['xp'],
    team = playerJson['team'];

```

```
apiData.forEach((playerJson) {
  var player = Player.fromJson(playerJson);
  player.sayHello();
})
```

Cascade Notation

정의한 오브젝트의 값들을 바꾸고 싶다~~하면

```
void main(){
  var nico = Player(name: 'nico', xp:1200, team: 'red');
  nico.name = 'las';
  nico.xp = 1200000;
  nico.team = 'blue';
}
```

보통은 이렇게 했어야 했겠지... 그런데 dart에서는 좀 더 간단하게 할 수 있다.

```
void main(){
  var nico = Player(name: 'nico', xp:1200, team: 'red')
  ..name = 'las'
  ..xp = 1200000
  ..team = 'blue';
}
```

..으로 바꾸고, 마지막에만 ; 세미콜론을 붙여주는 것이다.

그런데~ nico를 바꾼다는 것은 어떻게 알 수 있는 거지... 했더니 바로 앞 클래스를 가리키나 봐.

바로 앞에서, 세미콜론을 붙이지 않은 저 클래스!!

```
void main(){
  var nico = Player(name: 'nico', xp:1200, team: 'red');
  var potato = nico
  ..name = 'las'
  ..xp = 1200000
  ..team = 'blue'
  ..sayHello();
}
```

이렇게도 가능함! 심지어 클래스 내의 함수도 사용할 수 있음.

와~ 진짜 편하다~~~~개발자들이 좋아한대. 그리고 많이 사용할거래!

Enums

변수에 값을 할당할 때 바보같이 오타내는 것을 막아준다. 그러니까, 선택의 폭을 좁혀주는 것임.

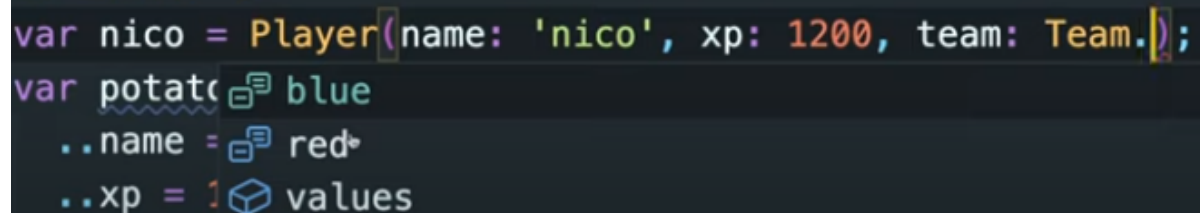
```
enum Team { red, blue }

class Player{
  String name;
  int xp, age;
  Team team;
}
```

따옴표 안 써도 된다 써도 되긴하는데 뭐.

그리고! 이제 team은 String형식이 아니다! enum으로 정의한 저걸 사용함.

사용할 때도말야.



```
var nico = Player(name: 'nico', xp: 1200, team: Team.);
var potato = Player(name: 'blue', xp: 100, team: Team.);
..name = 'red';
..xp = 100;
```

이렇게 Team. 을 하면 아래에 뜬다~~~!!

수정할 때도

```
..team = Team.blue
```

이렇게 쓰면 됨 우와 편하긴하당

플러터 색상들도 아마 enum일거라고 하심.

Abstract Classes

추상화 클래스는 오브젝트를 생성할 수 없고, 그저 다른 클래스들이 직접 구현해야하는 메소드들을 모아둔 청사진이다.

```
abstract class Human {  
    void walk();  
}
```

사용할 때도 단순히 extends를 붙이면 된다.

```
class Player extends Human{  
    void walk(){  
        print("I'm walking")  
    }  
}
```

물론 추상화 클래스에 정의한 함수도 작성해주어야 한다. 청사진에서 적어둔 것, 반환방식 같은 것만 지키며 방식을 바꾸면 된다.

들어가야 하는 값도 지켜야하는지 이따 실습해보자고~~

+

```
abstract class Human{  
    void walk(String day);  
}  
  
class Player extends Human{  
    String name;  
    int age;  
  
    void walk(String day){  
        print("$name is walking in ${day}");  
    }  
  
    Player(this.name, this.age);  
}  
  
void main(){  
    var player1 = Player("이름", 15);  
    print(player1.name);  
    player1.walk("monday");  
}
```

이것도 됐음~

아래 내용은 안됨! 그러니까 청사진 들어가는 것 나오는 것 다 지켜야 함!!

```
abstract class Human{
    void walk();
}

class Player extends Human{
    String name;
    int age;

    void walk(String day){
        print("$name is walking in ${day}");
    }

    Player(this.name, this.age);
}

void main(){
    var player1 = Player("이름", 15);
    print(player1.name);
    player1.walk("monday");
}
```

Inheritance

```
class Human{
    final String name;
    Human(this.name);

    void sayHello(){
        print("Hi my name is $name");
    }
}
```

위 클래스를 상속하고 싶다면~ Human을 쓰고는 싶은데 저 코드를 다시 쓰기 싫다면

```
class Player extends Human{
    Player({
        required this.team,
```

```

    required String name,
  }) : super(name);
}

void main(){
  String team;
  var player = Player(team: "red", name:"nico");

  player.sayHello();
}

```

super은 부모 클래스와 상호작용할 수 있게 한다.

super안에 있는거 named로 바꿔도 됨 부모클래스도 그렇게 받도록 바꿔야 하지만~

: 잊지마~

만약 부모 클래스에서 정의된 함수를 내가 직접 커스터마이징 하고싶다면?

뭔가 추가하고 싶다면?

```

class Player extends Human{
  Player({
    required this.team,
    required String name,
  }) : super(name);

  @override
  void sayHello(){
    super.sayHello(); // 부모 요소의 함수
    print("and I play for $team");
  }
}

```

Mixins

플러터에서 자주 사용하는 것 그건 Mixin

생성자가 없는 클래스를 의미한다. wow

클래스에 프로퍼티들을 추가하거나 할 때 사용한다.

```

class Strong {
  final double strengthLevel = 1500.99;
}

class QuickRunner{
  void runQuick(){
    print("ruuuuuuun!!!!");
  }
}

class Player with Strong, QuickRunner {
  Player({required this.team});
}

class with QuickRunner{}

```

extends가 아닌 with를 사용한다. 다른 클래스의 프로퍼티와 메소드를 그냥 긁어 오는 것 뿐! 오직 그것만을 위해 상속할 필요가 없다는 것이 장점이다.

플러터 플러그인들을 사용할 때 많이 접하게 될 것이라 한다.

여러 클래스에 재사용 할 수 있다는 것이 장점~~~

사용할 때는 그냥 main에서, 예로 위 같은 경우 Player.runQuick(); 이렇게 사용하면 돼 상속이 아님, 부모 자식 관계가 아님 그냥 재네가 가진 것을 가져오는 것~~

future stream isolate ← dart만의 특성~