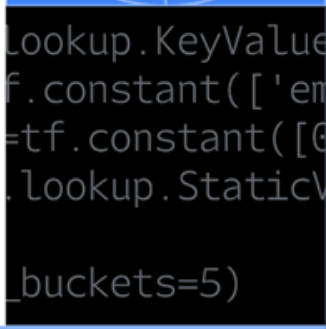
Google Developer Student Clubs
Hanyang University

App Front 2주차 강의 정리



3.3 Optional Positional Parameters

[ ] 로 감싸고, ? 로 null 허용해주고 null 일 경우 값(default)을 넣어준다

```
String sayHello(
  String name,
  int age,
  [String? country = 'cuba']
) {
  return "Hello $name, you are $age, you're from $country";
}


void main() {
  print(sayHello(
    age: 12,
    name: 'nico'
  ));
}
```

3.4 QQ Operator

?? Left ?? Right -> Left가 null 일 경우 right를 실행하고, null 이 아닐 경우 left를 실행하는 연산자

??= Left ??= Right -> Left가 null 일 경우, right 값을 left에 입력하고, left가 null 이 아닐 경우에는 아무 연산도 일어나지 않는 연산자

```dart
String capitalizeName(String? name) =>
  name != null ? name.toUpperCase() : 'ANON';

String capitalizeName(String? name) =>
  name.toUpperCase() ?? 'ANON';

void main() {
  String name = null;
  name??='nico'; //name에 nico가 들어감
  name??='anon'; //name에 nico가 있으니까 anon은 들어가지 않음
}
```

3.5 Typedef

자료형이 헷갈릴 때, alias을 만드는 방법
typedef 로 데이터타입을 정의할 수 있다!

```
typedef ListOfInts = List<int>

List<int> reverseListOfNumbers(List<int> list) {
  var reversed = list.reversed;
  return reversed.toList();
}

void main() {
  reverseListOfNumbers([1,2,3]);
}
```

4.0 Your First Dart Class

class [클래스명] { } → 의 형태로 선언할 수 있다.
클래스를 인스턴스로 생성할 수 있고, 인스턴스명.~ 로 내부 함수나 변수를 사용할 수 있다.

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

  void sayHello() {
    print("Hi my name is $name"); //this 를 사용할 순 있지만, class 내에서 사용하지 않는
  }
}

void main() {
  var player = Player();
  print(player.name);
  player.name = 'lalaala';
  print(player.name);

  player.sayHello();
}
```

4.1 Constructor

`constructor`는 `class`와 같은 이름으로 생성해준다.

Dart

Copy Caption

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

  Player(String name, int xp) {
    this.name = name;
    this.xp = xp;
  }

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

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

4.1 Constructor

`Player` class처럼 일일히 다 선언해주긴 귀찮고 손이 많이 간다.
`late` 를 지우고 `constructor` 에 `this` 를 통하여 바로 변수에 연결해주면 된다.

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

  Player(String name, int xp) {
    this.name = name;
    this.xp = xp;
  }

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

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

  Player(this.name, this.xp);

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

4.2 Named Constructors Parameters

많은 Positional argument 가 있으면 헷갈린다!!

`{ }` 를 추가하고 `required`를 추가하여 Constructor를 생성한다!

```dart
class Player{
  final String name;
  int xp;
  String team;
  int age;

  Player(this.name, this.xp, this.team, this.age); //<- Positional argument

  Player({
    required this.name,
    required this.xp,
    required this.team,
    required this.age,}); //<- Named argument

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

void main() {
  var player = Player(
    name:'nico',
    xp: 1500,
    team: 'blue',
    age: 21
  );
}
```

4.3 Named Constructors

많은 Positional argument 가 있으면 헷갈린다!!

`{ }` 를 추가하고 `required`를 추가하여 Constructor를 생성한다!

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

  // positional parameter 사용
  Player.createRedPlayer(String name, int age):
    this.age = age,
    this.name = name,
    this.team = 'red',
    this.xp = 0;

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

void main() {
  var bluePlayer = Player.createBluePlayer(name: 'nico',age: 21);
  var redPlayer = Player.createRedPlayer('lynn', 15);
}
```

4.5 Cascade Notation

`..` → Cascade
동일한 개체에 대해 일련의 작업을 수행할 수 있다.
함수 호출 외에도 동일한 개체의 필드에 액세스할 수도 있다.
이렇게 하면 임시 변수를 생성하는 단계를 줄일 수 있고 보다 유동적인 코드를 작성할 수 있다.

```
void main() {
  var nico = Player(name: ' nico', xp: 1000, team: 'red');
    var potato = nico
    ..name = 'las'
    ..team = 'blue'
    ..xp = 1500
    ..sayHello();
}
```

4.5 Cascade Notation

`..` → Cascade
동일한 개체에 대해 일련의 작업을 수행할 수 있다.
함수 호출 외에도 동일한 개체의 필드에 액세스할 수도 있다.
이렇게 하면 임시 변수를 생성하는 단계를 줄일 수 있고 보다 유동적인 코드를 작성할 수 있다.

```
void main() {
  var nico = Player(name: ' nico', xp: 1000, team: 'red');
    var potato = nico
    ..name = 'las'
    ..team = 'blue'
    ..xp = 1500
    ..sayHello();
}
```

```
void main() {
  var nico = Player(name: ' nico', xp: 1000, team: 'red');
  var potato = nico;
  potato.name = 'las';
  potato.team = 'blue';
  potato.xp = 1500;
  potato.sayHello();
}
```

4.6 Enum

개발자들이 실수하지
않도록 도와주는 것!!

선택의 폭을 좁혀주는 역할

```
enum Team { red, blue } // << 선언 방법!!
enum XPLevel {beginner, medium, pro}

class Player {
  String name;
  XPLevel xp;
  Team team;
  Player({required this.name, required this.xp, required this.team});

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

void main() {
  var nico = Player(name: 'nico', xp: XPLevel.beginner, team: Team.blue);
  nico.name = 'las';
  nico.xp = XPLevel.pro; // << 사용 방법!!
  nico.team = Team.red; // << 사용 방법!!
}
```

4.6 Enum

개발자들이 실수하지
않도록 도와주는 것!!

선택의 폭을 좁혀주는 역할

```
enum Team { red, blue } // << 선언 방법!!
enum XPLevel {beginner, medium, pro}

class Player {
  String name;
  XPLevel xp;
  Team team;
  Player({required this.name, required this.xp, required this.team});

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

void main() {
  var nico = Player(name: 'nico', xp: XPLevel.beginner, team: Team.blue);
  nico.name = 'las';
  nico.xp = XPLevel.pro; // << 사용 방법!!
  nico.team = Team.red; // << 사용 방법!!
}
```

4.7 Abstract Classes

추상화 클래스! → 객체를 생성할 수 없다.

다른 클래스들이 직접 구현해야 하는 메서드를 모아둔 청사진 느낌

메소드의 이름과 반환 타입만 정하여 정의할 수 있다.

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

enum Team { red, blue }
enum XPLevel {beginner, medium, pro}

class Player extends Human {
  String name;
  XPLevel xp;
  Team team;
  Player({required this.name, required this.xp, required this.team});

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

  void walk() {
    print('im walking');
  }
}

class Coach extends Human {
    void walk() {
    print('hes walking');
  }
}
```

4.8 Inheritance

extends 로 상속을 받고

super로 부모 class의 값을 호출할 수 있다.

`super` 확장(상속)한 부모 클래스의 property에 접근하게 하거나 method를 호출할 수 있게 해준다

확장한 부모 클래스가 생성자를 포함하고 있는데, 그 생성자를 사용하려면 그 부모 클래스의 생성자를 호출해야 한다.

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

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

enum Team {red, blue}

class Player extends Human {
  final Team team;

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

  @override
  void sayHello() {
    super.sayHello();
    print('and I play for ${team}');
  }
}

void main() {
  var player = Player(
    team: Team.red,
    name: 'nico'
  );
}
```

4.9 mixin

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

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

mixin Strong {
  final double strenghtLevel = 1500.99;
}
mixin QuickRunner {
  void runQuick() {
    print("ruuuuuuuun!");
  }
}
mixin Tall {
  final double height = 1.99;
}
```

```dart
enum Team {red, blue}

class Player extends Human with Strong, QuickRunner  {
  final Team team;

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

}

void main() {
  var player = Player(
    team: Team.red,
    name: 'nico'
  );

  print(player.strenghtLevel);
}
```