MODUL II
DART DASAR

List

List merupakan sebuah object yang menyimpan kelompok data yang disusun berdasarkan index. Untuk membuat list dapat menggunakan kurung sama seperti pada array.

Contoh

```
void main(List<String> args) {
  List <int> list = [22, 21, 20];
  print(list[0]);
}
```

Element list yang sudah dibuat dapat ditambah elemen ke dalam objeknya dengan cara menggunakan metode **add().**

Contoh

```
void main(List<String> args) {
  List <int> list = [22, 21, 20];
  list.add(10);
  print(list[2]);
  print(list[3]);
}
```

Map

Map merupakan sebuah object yang setiap elemennya berupa pasangan kunci (*key*) dan nilai (*value*). Perbedaan dari list, map memiliki kunci sedangkan list hanya memiliki value. Dalam object map kunci harus bersifat unik, tetapi nilai boleh sama. Untuk membuat map dapat menggunakan kurung kurawal { }.

Contoh

```
void main(List<String> args) {
  Map<String, String> bahasa = {
    'id' : 'Bahasa Indonesia',
    'en' : 'English',
    'ar' : 'Arabic',
  };
  print(bahasa['id']);
}
```

Menampilkan Berbagai Tipe Data

Beberapa tipe data dapat disebut dengan dynamic. Kita dapat membuat beberapa tipe data untuk dijadikan dalam satu variable, caranya dengan menggunakan dynamic / var dalam pendeklarasian. Untuk menampilkan beberapa data / aray / list dapat menggunakan looping for.

Contoh:

```
void main(List<String> args) {
  dynamic berbagai = ["String", 2, 2.0, true];
  var berbagaiVar = ["String", 2, 2.0, true];

  for (var item in berbagai) {
    print("Output : $item");
  }
  for (var item in berbagaiVar) {
    print("Output : $item");
  }
}
```

List Multidimensi

List multidimensi adalah list yang berisi list/array didalamnya.

Contoh.

```
void main(List<String> args) {
  var multidimensi = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
  ];

  List<List<dynamic>> berbagai = [
    [1, "b", 3],
    [4, "e", 6],
    [7, "h", 9]
  ];

  print(multidimensi[0][1]);
  print(berbagai[0][1]);
}
```

Update list pada dart 2.3.2

Method add and addAll

Contoh.

```
void main(List<String> args) {
  List<int> listku = [1];
  List<int> list = [6, 2, 3, 4];
  print(listku[0]);

  listku.addAll(list);
  listku.forEach((element) => {
    print(element)
   });
}
```

Tambahan – Method pada List and Maps

- **forEach()**

  Untuk menampilkan tiap-tiap element.

  Contoh.

```
void main(List<String> args) {
  List<int> listku = [6, 2, 3, 4];

  listku.forEach((element) => {
    print(element)
   });
}
```

- **contains()**

  Mengecek isi dalam list, mereturn boolean.

  Contoh.

```
void main(List<String> args) {
  List<int> list = [6, 2, 3, 4];
  print(list.contains(2));
}
```

- **sort()**

  Mengurutkan data

  Contoh.

```
void main(List<String> args) {
  List<int> list = [6, 2, 3, 4, 20, 11, 8];
  list.sort((a, b) => a.compareTo(b));
  print(list);
}
```

- **reduce(), fold()**

  Compress list menjadi single

  Contoh.

```
void main(List<String> args) {
  List<int> list = [6, 2, 3, 4, 20, 11, 8];
  var sumData = list.reduce((value, element) => value + element);
  print(sumData);
  var nextSum = list.fold(10, (previousValue, element) =>
print("$previousValue, $element"));
}
```

- **every()**

  Berfungsi untuk melakukan check pada tiap element.

  Contoh.

```
void main(List<String> args) {
  List<Map<String, dynamic>> listKota = [
    {'nama': 'Yogyakarya', 'penduduk': 200},
    {'nama': 'Surabaya', 'penduduk': 150},
    {'nama': 'Solo', 'penduduk': 100},
    {'nama': 'Jakarta', 'penduduk': 400},
  ];
  var contoh = listKota.every((element) => element['penduduk'] >=100
);
  print(contoh);
}
```

- **where(), firstWhere(), singleWhere()**

  Contoh.

```
void main(List<String> args) {
  List<Map<String, dynamic>> listKota = [
    {'nama': 'Yogyakarya', 'penduduk': 200},
    {'nama': 'Surabaya', 'penduduk': 150},
    {'nama': 'Solo', 'penduduk': 100},
    {'nama': 'Jakarta', 'penduduk': 400},
  ];

  var whereKota = listKota.where((element) => element['penduduk'] >
100);
  print(whereKota);

  var     firstKota     =     listKota.firstWhere((element)     =>
element['penduduk'] < 200);
  print(firstKota);

  //jika singleWhere menemukan lebih dari 1 data maka akan error
  var     sigleKota     =     listKota.singleWhere((element)     =>
element['penduduk'] == 100);
  print(sigleKota);
}
```

- **take(), skip()**

  Contoh.

```
void main(List<String> args) {
  var data = [1, 2, 3, 4, 5, 6];
  print(data.take(2));
  print(data.skip(2));
}
```

- **expand()**

  Contoh.

```
void main(List<String> args) {
  var pairs = [[1, 2], ["a", "b"], [3, 4]];
  var flatmaps = pairs.expand((pair)=> pair);
  print(flatmaps);
}
```

Object Oriented

Merupakan suatu metode pemrograman yang berorientasi pada objek. Program-program yang telah ada merupakan gabungan dari beberapa komponen-komponen kecil yang sudah ada sebelumnya.

Class

Class merupakan blueprint dari sebuah objek. Class merupakan desain dari suatu objek yang akan dibuat. Jika rancangannya baik maka hasilnaya baikm begitu juga sebaliknya. Untuk mendeklarasi class dapat menggunakan sintax.

```
// menggunakan pada main
nama_kelas nama_objek = new nama_kelas();

//deklarasi
class nama_kelas{

}
```

Contoh

```
void main(List<String> args) {
  Persegi kotak = new Persegi();
  kotak.sisi = 2;
  print(kotak.hitungLuas());
}

class Persegi{
  // dapat menggunakan null safety atau menggunakan late
  late double sisi;

  double hitungLuas(){
    return sisi*sisi;
  }
}
```

Enkapsulasi (Pembungkusan)

Merupakan sebuah metode untuk mengatur struktur class dengan menyembunyikan alur kerja dari class tersebut. Struktur class disini adalah property dan method. Dengan adanya enkapsulasi kita dapat membatasi akses dari struktur class. Pada pemrograman dart tidak ada

keyword untuk membuat attribute menjadi private atau pun public, cukup dengan menambahkan underscore "_" sebelum nama attribut atau metode.

Setter dan Getter

Merupakan metode kelas yang digunakan untuk memanipulasi data. Getter digunakan untuk membaca atau mendapatkan data sedangkan setter digunakan untuk mengatur data pada kelas.

Contoh Setter Getter

```
void main(List<String> args) {
  Persegi kotak = new Persegi();
  kotak.setSisi(4);
  print(kotak.getSisi());
}

class Persegi{
  double? _sisi;
  void setSisi(double? _sisi){
    this._sisi = _sisi;
  }
  double getSisi(){
    return _sisi!;
  }
}
```

Cara lain

```
void main(List<String> args) {
  Persegi kotak = new Persegi();
  kotak.setSisi = 4;
  print(kotak.getSisi);
}

class Persegi{
  double? _sisi;
  set setSisi(double? _sisi){
    this._sisi = _sisi;
  }
  double get getSisi => _sisi!;
}
```

Inheritance

Inheritance atau yang biasa kita kenal extends merupakan konsep OOP dimana sebuah class dapat menurunkan property dan method yang dimiliki kepada kelas lain. Class yang akan diturunkan biasa disebut dengan parent class, super class, atau base class. Sedangakan class yang menerima penurunan biasa disebut dengan child class, sub class, derived calss, atau heir class.

Contoh

main.dart

```
import 'anjing.dart';
import 'kucing.dart';

void main(List<String> args) {
  Anjing anjing = Anjing();
  Kucing kucing = Kucing();

  anjing.setNyawa = 2;
  kucing.setNyawa = 9;

  print("Nyawa anjing: ${anjing.getNyawa}");
  print("Nyawa kucing: ${kucing.getNyawa}");

  print("method anjing : " + anjing.suara());
  print("method kucing : " + kucing.mandi());
}
```

hewan.dart

```
class Hewan{
  int? _nyawa;

  int get getNyawa => _nyawa!;
  set setNyawa(int? nyawa){
    nyawa! < 0
      ? _nyawa=nyawa*-1
      : _nyawa=nyawa;
  }
}
```

anjing.dart

```
import 'hewan.dart';

class Anjing extends Hewan{
  String suara()=>"guk guk guk";
}
```

kucing.dart

```
import 'hewan.dart';

class Kucing extends Hewan{
  String mandi() => "byur byur byur";
}
```

Polymorism

Merupakan konsep oop dimana suatu objek yang berbeda-beda dapat diakses melalui interface yang sama. Metode pada objek polymoprphic dapat beradaptasi dengan metode objek yang lain.

Contoh

main.dart

```
import 'anjing.dart';
import 'kucing.dart';

void main(List<String> args) {
  Anjing anjing = Anjing();
  Kucing kucing = Kucing();

  print("suara anjing : " + anjing.suara());
  print("suara kucing : " + kucing.suara());
}
```

hewan.dart

```
class Hewan{
  String suara(){
    return "Suara-suara pada hewan";
  }
}
```

anjing.dart

```
import 'hewan.dart';

class Anjing extends Hewan{
  @override
  String suara()=>"guk guk guk";
}
```

kucing.dart

```
import 'hewan.dart';

class Kucing extends Hewan{
  @override
  String suara() => "meow meow";
}
```

Constructor

Merupakan metode khusus yang akan dijalankan secara otomatis pada saat sebuah objek dibuat atau pada saat perintah new. Constructor biasa digunakan untuk membuat proses awal dalam mempersiapkan objek, seperti memberikan nilai pada property.

Contoh 1

```
void main(List<String> args) {
  Persegi kotak = new Persegi(4);
  print(kotak.getSisi);
}

class Persegi{
  double? _sisi;
  Persegi(double sisi);
```

```
  double get getSisi => _sisi!;
}
```

Contoh 2

```
void main(List<String> args) {
  var data1 = new User.nama("Dimas");
  var data2 = new User.alamat("Yogya");

  print(data1.nama);
  print(data2.alamat);
}

class User{
  String? nama;
  String? alamat;

  User.nama(this.nama);
  User.alamat(this.alamat);
}
```

Async, Future, Await

Materi ini berkaitan dengan kasus ketika ingin menjalankan sintaks dengan syarat kode sebelumnya harus di jalankan terlebih dahulu, misalnya pada kasus untuk mengambil data dari rest api. Hal ini dikenal dengan istilah **blocking**. Sebaliknya **non-blocking** berarti program berjalan dengan mengeksekusi sintaks dari baris ke baris secara paralel.

Contoh

```
import 'dart:async';

void main(List<String> args) {
  print("pertama");
  Timer(Duration(seconds: 2), (() => print("ketiga - delay")));
  print("kedua");
}
```

Future

Merupakan salah satu keyword yang disediakan oleh dart. Keyword ini berguna untuk mengembalikkan nilai untuk waktu yang akan datang.

Contoh

```
import 'dart:async';

void main(List<String> args) {
  fetchData();
  print("Test mengambil data");
}

Future<void> fetchData(){
  return Future.delayed(Duration(seconds: 2), () => print("Akhmal Dimas
Pratama"),);
}
```

Async, Await

Keywoed async dan await untuk mendefinisikan fungsi asynchronous dan menggunakan valuenya. Dengan syarat yang perlu di perhatikan

1. Sebelum menggunakan tambahkan keybord **async** sebelum body.
2. Keyword **await** berfungsi jika berada dalam **async**.

Contoh

```
import 'dart:async';

void main(List<String> args) async {
  print(await createUser());
  print("Data terambil");
}

Future<String> createUser() async{
  String order = await fetchData();
  return "Nama : $order";
}

Future<String> fetchData(){
  return Future.delayed(Duration(seconds: 2), () => "Akhmal Dimas Pratama");
}
```

Try-Catch

Untuk menangani handling error pada fungsi async, jika aplikasi sukses maka akan masuk ke try, jika pada try error akan di lempar dan ditangkap pada catch.

Contoh error paksa

```
import 'dart:async';

void main(List<String> args) async {
  await createUser();
}

Future<void> createUser() async{
  try{
    String order = await fetchData();
  print("Nama : $order");
  print("Data terambil");
  }catch(e){
    print("Error: $e");
  }
}

Future<String> fetchData(){
  return Future.delayed(Duration(seconds: 2), () => throw "no data");
}
```

Mockup

Sebelum membuat aplikasi biasanya akan dibuat terlebih dahulu ui atau desain mockupnya. Secara umum prosesnya: sketsa -> wireframe -> mockup/prototype.

Tools yang umum digunakan adalah figma atau bisa juga adobe XD.

Link referensi materi: https://uxplanet.org/a-step-by-step-guide-to-creating-mobile-app-wireframes-ede2e5c53a75

Flutter

Sebelum memulai menggunakan flutter ada beberapa tahapan yang harus di lakukan.

System requirements

OS : windows min versi 7 SPI atau yang pasti 64 bit. Untuk linux dan mac hanya wajib mrnggunakan 64 bit.

Disk Space : Windows ruangkan sekitar 400 mb. Linux 600 mb. Mac 700 mb. Belum termasuk tools/IDE.

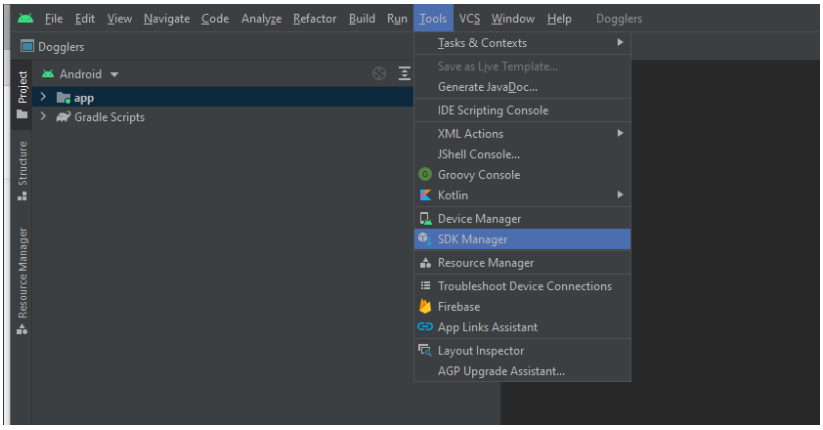
Tools : Pastikan telah menginstall git dan dapat digunakan melalui command line.

Install Android Studio

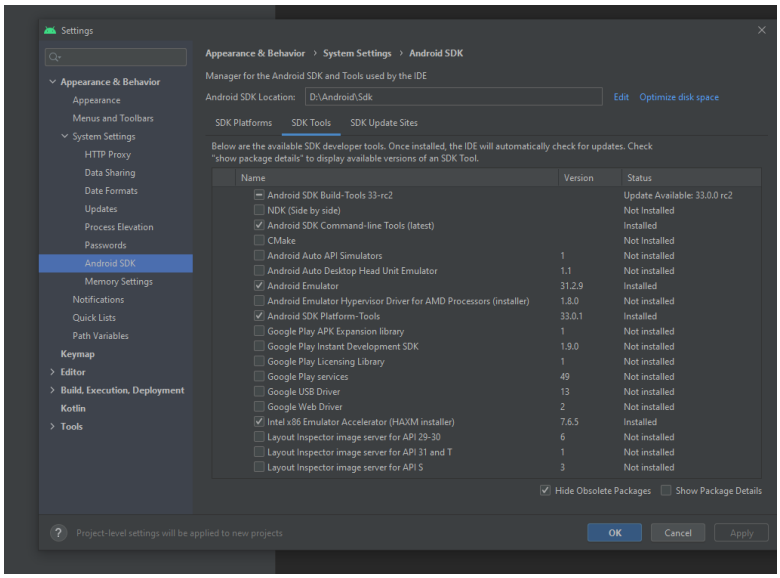
Ini wajib karena yang di ambil dari android studio adalah dependencies yang menyertainya, untuk menuliskan code dapat menggunakan editor lain. Beberapa hal yang dibutuhkan Flutter dari Android Studio untuk proses development Android:

1. Android SDK
2. Android SDK Platform-Tools
3. Android SDK Build-Tools

Silahkan download dan install Android Studio link https://developer.android.com/studio sesuai dengan sistem operasi yang kamu punya. Jika sudah buka menu **Tools > SDK Manager**



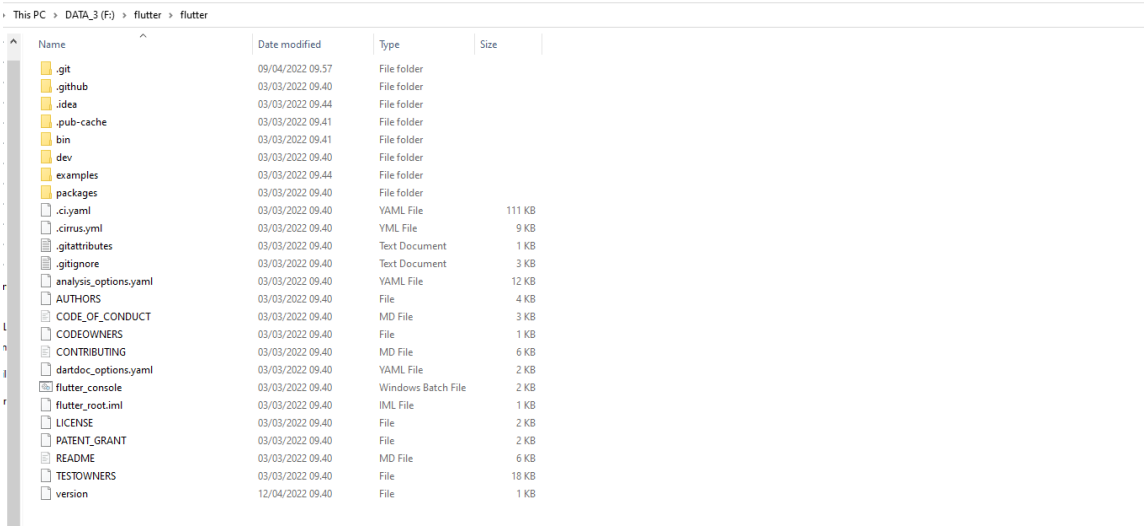
Kemudian pilih tab **SDK Tools** dan centang ketiga komponen diatas



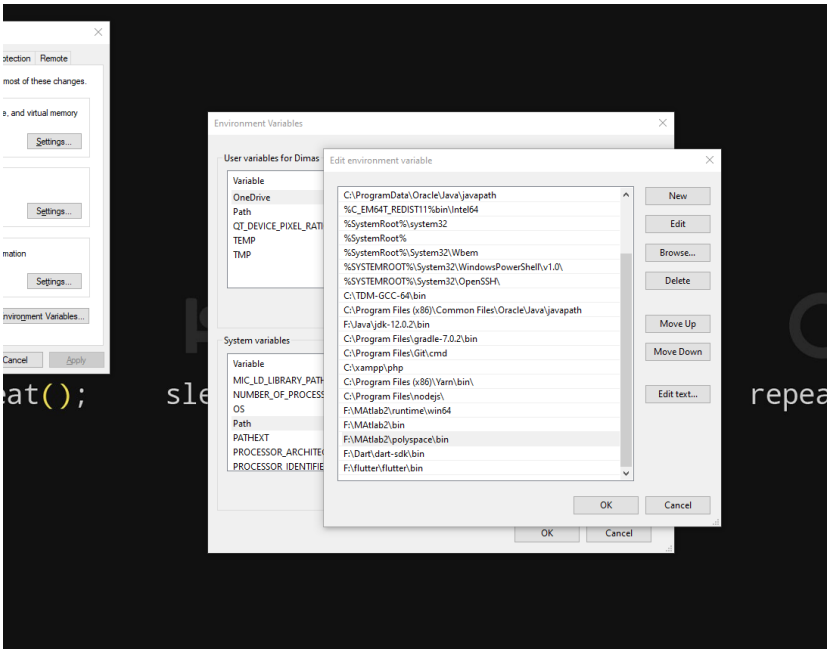
Install Flutter SDK

Flutter juga wajib untuk di install.

1. Link https://docs.flutter.dev/get-started/install Pilih os yang kamu gunakan. Downlaod
2. Ekstrak hasil download kedalam folder yang kamu inginkan (**Warning:** jika menggunakan windows jangan masukkan ke dalam folder **C:/Program Files**). Contoh saya meletakkan pada F:\flutter\flutter



3. Kemudian **path** dari flutter SDK yang telah di ekstrak. Contoh F:\flutter\flutter\bin



4. Terakhir, *running* command: **flutter doctor** untuk mengecek apakah masih ada *dependencies* yang belum ter-install.

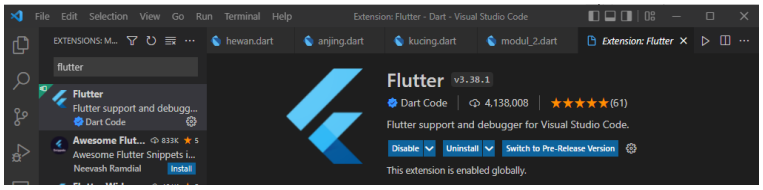
```
Doctor summary (to see all details, run flutter doctor -v):
[√] Flutter (Channel stable, 2.10.3, on Microsoft Windows [Version 10.0.19043.1586], locale id-ID)
[√] Android toolchain - develop for Android devices (Android SDK version 31.0.0)
[√] Chrome - develop for the web
[X] Visual Studio - develop for Windows
    X Visual Studio not installed; this is necessary for Windows development.
      Download at https://visualstudio.microsoft.com/downloads/.
      Please install the "Desktop development with C++" workload, including all of its default components
[√] Android Studio (version 2021.1)
[√] IntelliJ IDEA Community Edition (version 2020.3)
[√] VS Code (version 1.66.1)
[√] Connected device (3 available)
[√] HTTP Host Availability

! Doctor found issues in 1 category.
```

Konfigurasi Editor

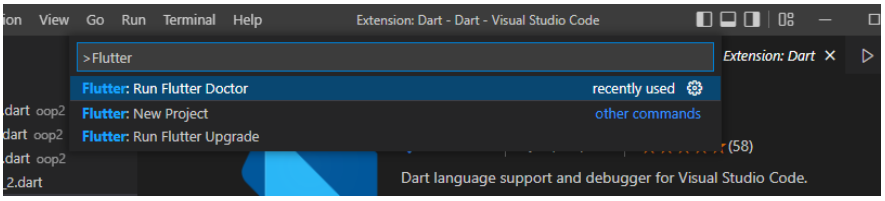
Jika memakai Android Studio sangat berat, dapat menggunakan VS Code. Tahapan konfigurasi VS Code

1. Install VS Code
2. Tekan **Crtl + Ship + P** atau **View > Command Palette**
3. Ketik **Extensions: Install Extension**, tekan enter
4. Pada kotak pencarian, cari **flutter**. Kemudian pilih dan install



Note: jangan lupa menginstall **Dart** plugin juga.

5. Reload VS Code
6. Untuk memastikan dapat membuka **View > Command palette**. Ketik **Flutter: Run Flutter Doctor**.



Mempersiapkan Smartphone / Emulator

Untuk emulator dapat menggunakan device asli / handphone android atau juga dapat menggunakan emulator bawaan android studio. Untuk aplikasi penyambungan jika menggunakan handphone sendiri dapat menggunakan **vysor** link https://www.vysor.io/ .

Installasi fluter via terminal

Dapat menggunakan command **flutter create nama_app**.

Untuk meng run dari terminal dapat menggunakan direktori dari folder yang sudah di create. Jika blm dapat menggunakan **cd nama_app**. Kemudian untuk meng-run dapat menggunakan **flutter run**.