



Tahun Ajar 2022/2023

PROGRAMMING PERANGKAT BERGERAK

Praktikum 2 :
Fundamental Dart

Pengembang Modul :
Novian Adi Prasetyo, S.Kom., M.Kom.
Akhmad Nur Alamsyah
Puspita Kartika Sari

Fundamental Dart

Dart merupakan *programming language* lintas platform atau platform independen yang artinya dapat dijalankan pada sistem operasi yang berbeda seperti Windows, Linux, Unix dan MacOS, dll yang awalnya dikembangkan oleh Google dan kemudian disetujui sebagai standar oleh Ecma, yang saat ini digunakan untuk membangun aplikasi web, server, desktop, dan seluler.

Membuat Program Hello World.

```
void main() {
    print('Hello, World!');
}
```

Untuk membuat variabel pada bahasa pemrograman dart, kita perlu mendefinisikan variabel tersebut dengan keyword **var**.

Contoh:

```
var name = 'Puspita';
var year = 2023;
var bulan = ['Januari', 'Februari', 'Maret', 'April'];
var image = {
    'buah': ['mangga'],
    'url': '//path/to/mangga.jpg'
};
```

Selain menggunakan var, kita bisa mendeklarasikan tipe data variabel secara eksplisit untuk menghindari kebingungan dan memudahkan proses debugging.

Contoh :

```
String greetings = 'Hello Dart!';
int year = 2023;
```

Beberapa tipe data yang didukung oleh dart, antara lain :

Tipe	Deskripsi	Contoh
int	Integer (bilangan bulat)	1, -2, 0
double	Bilangan desimal	3.14, 12.0, -23.23
num	Bilangan bulat dan bilangan desimal	7, 3.14, -99.00
bool	Boolean	true, false
String	Text yang terdiri dari 0 atau beberapa karakter	'Pemrograman', 'N'
List	Daftar nilai	[1,2,3], ['a', 'b', 'c']

Map	Pasangan key-value	{“a” : 3, “b” : 7}
dynamic	Tipe apa pun	

Control flow

Dart mendukung penggunaan control flow statements:

1. If and Else

```
if (angka > 0) {
    print('bilangan positif');
} else if (angka < 0) {
    print('bilangan negatif');
}
```

2. Switch case

```
switch (variable/expression) {
    case value1:
        // do something
        break;
    case value2:
        // do something
        break;
```

3. For Loops

```
for (int i = 1; i <= 100; i++ ) {
    print(i);
}
```

4. While and do-while

```
var i = 1;
while (i <= 100) {
    print(i);
    i++;
}
```

```
var i = 1;
do {
    print(i);
    i++;
} while (i <= 100);
```

List

Adalah tipe data yang digunakan untuk merepresentasikan koleksi nilai yang terurut. List dapat berisi elemen-elemen dengan tipe data yang sama atau berbeda. List dapat diinisialisasi dengan nilai-nilai yang diberikan, atau dapat dibuat kosong dan kemudian diisi kemudian.

Contoh :

```
List<int> numberList = [1, 2, 3, 4, 5];
```

Kode tersebut adalah contoh dari satu objek List yang berisi kumpulan data bertipe integer.

Contoh :

```
List dynamicList = [1, 'Informatika', true]; // List<dynamic>
```

Jika kita tidak mendefinisikan nilai secara eksplisit ke dalam List, maka List akan menyimpan tipe dynamic.

Spread Operator

Merupakan fitur dart yang berfungsi untuk menambahkan banyak nilai ke dalam List dengan cara yang singkat. Spread operator dituliskan dengan tiga titik (...)

Contoh tanpa spread operator:

```
var buah = ['Mangga', 'Apel', 'Jeruk', 'Manggis'];
var hewan = ['Ayam', 'Kelinci', 'Kucing'];
var allFavorites = [buah, hewan];
print(allFavorites);

/* output
[[Mangga, Apel, Jeruk, Manggis], [Ayam, Kelinci, Kucing]]
*/
```

Ket : Nilai List tidak tergabung, variabel allFavorites menjadi List yang menampung 2 List di dalamnya.

Contoh dengan spread operator:

```
var buah = ['Mangga', 'Apel', 'Jeruk', 'Manggis'];
var hewan = ['Ayam', 'Kelinci', 'Kucing'];
var allFavorites = [...buah,...hewan];
print(allFavorites);

/* output
[Mangga, Apel, Jeruk, Manggis, Ayam, Kelinci, Kucing]
*/
```

Ket : Dengan spread operator variabel hewan dan buah dapat menjadi 1 List.

Set

Set merupakan sebuah collection yang hanya dapat menyimpan nilai yang unik. Tidak boleh ada nilai duplikat.

Contoh :

```
var angkaSet = {1, 4, 6};  
Set<int> bilanganSet = new Set.from([1, 4, 6, 4, 1]);  
print(bilangan);  
  
// Output: {1, 4, 6}
```

Map

Map adalah sebuah collection yang dapat menyampaikan data dengan format key-value.

Contoh :

```
var kota = {  
  'Semarang': 'Jawa Tengah',  
  'Bandung': 'Jawa Barat',  
  'Malang': 'Jawa Timur'  
};
```

Ket : String yang berada disebelah kiri titik dua (:) adalah *key*, dan yang disebelah kanan adalah *value*.

OOP (Object Oriented Programming)

Adalah paradigma pemrograman yang berfokus pada penggunaan objek sebagai unit dasar dalam pembuatan program komputer. Objek di sini merujuk pada kumpulan data dan fungsi yang saling terkait dan terorganisir dengan baik.

Dalam OOP, program dikembangkan dengan cara mengidentifikasi objek-objek yang berbeda yang terkait dengan masalah yang ingin dipecahkan. Objek-objek ini kemudian dikelompokkan bersama dan diberi nama, atribut, dan metode.

Atribut adalah data atau informasi yang dimiliki oleh objek, sementara metode adalah fungsi atau perilaku yang dapat dilakukan oleh objek tersebut. Dengan demikian, OOP memungkinkan programmer untuk membuat program dengan struktur yang lebih terorganisir, modular, dan mudah dipelihara.

Contoh sederhana OOP adalah sebuah program untuk mengelola sebuah toko buku. Objek-objek dalam program tersebut bisa berupa buku, pelanggan, dan transaksi. Setiap objek memiliki atribut seperti judul buku, harga, nama pelanggan, dan tanggal transaksi, serta metode seperti menghitung harga total dan mengurangi stok buku.

Terdapat empat pilar dalam pemrograman berorientasi objek,

1. Encapsulation

Adalah suatu konsep dalam OOP yang memungkinkan data dan fungsi-fungsi yang berhubungan dengan data tersebut dikemas menjadi satu kesatuan yang disebut sebagai class. Class tersebut memiliki atribut (data) dan method (fungsi) yang hanya dapat diakses oleh class itu sendiri atau oleh class lain yang diberikan izin (dengan modifier akses). Dengan pengkapsulan, data dan fungsi-fungsi yang berhubungan dengan data tersebut terlindungi dari perubahan yang tidak diinginkan dan dapat diakses secara terkontrol.

2. Abstraction

Adalah konsep dalam OOP yang memungkinkan pengembang untuk fokus pada fitur penting dari suatu objek dan menyembunyikan detail implementasinya. Dalam abstraksi, kita hanya memperhatikan fitur dan fungsionalitas dari suatu objek tanpa perlu tahu detail bagaimana objek tersebut diimplementasikan. Contoh abstraksi adalah interface dan abstract class.

3. Inheritance

Adalah konsep dalam OOP yang memungkinkan suatu class untuk mewarisi atribut (data) dan method (fungsi) dari class lain yang lebih umum. Class yang mewarisi disebut sebagai *subclass* atau turunan (*child class*), sedangkan class yang menjadi sumber warisan disebut sebagai *superclass* atau induk (*parent class*). Dengan inheritance, *subclass* dapat memiliki atribut dan method yang sama dengan *superclass*, sehingga *subclass* dapat memperluas atau menambah fungsionalitas yang sudah ada di *superclass*.

4. Polymorphism

Adalah konsep dalam OOP yang memungkinkan suatu objek memiliki banyak bentuk atau tampilan yang berbeda. Dalam OOP, polymorphism dapat diterapkan dalam dua bentuk, yaitu method overloading dan method overriding. Method overloading adalah penulisan method dengan nama yang sama, tetapi dengan parameter yang berbeda-beda. Sedangkan method overriding adalah menimpa method yang sudah ada di superclass dengan method yang memiliki implementasi yang berbeda di subclass. Dengan polymorphism, kita dapat mengurangi penulisan kode yang berulang-ulang dan membuat kode lebih mudah dimengerti.

Class

Class merupakan sebuah blueprint untuk membuat objek. Di dalam class kita perlu mendefinisikan sifat (*attribute*) dan perilaku (*behaviour*) dari objek yang akan dibuat.

Contoh penggunaan class dengan membuat file hewan.dart :

```
class Hewan {  
    String nama;  
    int umur;  
    double berat;  
  
    Animal(this.nama, this.umur, this.berat);  
  
    void makan() {
```

```
    print('$nama makan.');
    berat = berat + 0.2;
}

void tidur() {
    print('$nama sedang tidur');
}
}
```

Ket : class Hewan memiliki attribute berupa nama, berat dan umur. Kemudian perlakunya adalah makan dan tidur.

Cara menggunakan class Hewan pada main.dart:

```
void main() {
    var kucing = Hewan('Ketty', 2, 3.2);
    kucing.makan();
    kucing.tidur();
    print(kucing.berat);
}
```

Properties & Methods

Properties atau properti adalah data yang dimiliki oleh suatu objek. Properti bisa berupa variabel, konstanta, atau objek lain. Properti juga dapat diakses dan dimodifikasi oleh objek itu sendiri atau objek lain yang memiliki akses ke properti tersebut. Contoh properti pada objek "mobil" bisa berupa warna, tahun produksi, nomor mesin, dan lain sebagainya. Sedangkan Methods atau metode adalah fungsi yang dimiliki oleh suatu objek. Method dapat menerima argumen dan mengembalikan nilai atau tidak mengembalikan nilai sama sekali. Method juga dapat diakses oleh objek itu sendiri atau objek lain yang memiliki akses ke method tersebut. Contoh method pada objek "mobil" bisa berupa method untuk menghidupkan mesin, method untuk mempercepat kendaraan, method untuk menghentikan kendaraan, dan lain sebagainya.

Sama seperti variabel, kita mendeklarasikan property di dalam class dengan menentukan tipe datanya atau menginisialisasikan nilainya secara eksplisit.

Contoh kode pada hewan.dart :

```
class Hewan {
    String _nama = '';
    int _umur = 0;
    double _berat = 0;
}
```

Ket : Setelah menambah **underscore** pada nama variabel, terdapat error di berkas main.dart ketika mengakses property umur. Karena saat ini umur bersifat private dan tidak bisa diakses dari luar berkasnya. Solusinya dengan menambahkan setter dan getter untuk mendapatkan nilai serta mengubahnya dari luar berkas.

```
// Setter
set nama (String value) {
    _name = value;
}

// Getter
double get berat => _berat;
```

Keseluruhan code pada hewan.dart

```
class Animal {
    String _name = '';
    int _age = 0;
    double _weight = 0;

    Animal(this._name, this._age, this._weight);

    // Setter
    set name(String value) {
        _name = value;
    }

    // Getter
    double get weight => _weight;

    void eat() {
        print('$_name is eating.');
        _weight = _weight + 0.2;
    }

    void sleep() {
        print('$_name is sleeping.');
    }
    void poop() {
        print('$_name is pooping.');
        _weight = _weight - 0.1;
    }
}
```

Inheritance

adalah kemampuan suatu program untuk membuat kelas baru dari kelas yang ada. Di dalam OOP kelas yang menurunkan sifat disebut sebagai kelas induk (parent class/superclass) sementara kelas yang mewarisi kelas induknya disebut sebagai kelas anak (child/subclass).

Untuk menerapkan inheritance gunakan keyword extends seperti contoh berikut :

```
class ChildClass extends ParentClass{  
}  
}
```

Setelah membuat class Hewan, kita dapat membuat kelas lainnya lalu melakukan extends ke kelas induknya.

Contoh: kita akan membuat class Meong yang mewarisi kelas Hewan

```
import 'Hewan.dart';  
class Meong extends Hewan {  
    String warnaBulu;  
  
    Meong(String nama, int umur, double berat, String warnaBulu);  
    super(nama, umur, berat) {  
        this.warnaBulu = warnaBulu;  
    }  
    void jalan() {  
        print('$nama berjalan');  
    }  
}
```

Kelas Meong adalah turunan dari kelas Hewan, maka kita dapat mengakses sifat dan perilaku dari Hewan melalui kelas Meong.

main.dart

```
import 'Meong.dart';  
  
void main() {  
    var kucing = Meong('Ketty', 2, 3.2, 'Putih');  
    kucing.walk();  
    kucing.makan();  
    print(kucing.berat);  
}
```

Abstract Class

Abstract merupakan gambaran umum dari sebuah kelas yang tidak dapat direalisasikan dalam sebuah objek. Kita telah membuat class Hewan sebelumnya, untuk menjadikannya sebuah kelas menjadi abstract hanya perlu menambahkan keyword abstract sebelum penulisan kelas :

main.dart

```
abstract class Hewan {  
    String nama;  
    int umur;  
    double berat;  
  
    // ...  
}
```

Sehingga kelas Hewan tidak lagi dapat diinisialisasi menjadi sebuah objek.

```
var kucing = Hewan('Ketty', 2, 3.2); //Error: The class 'Hewan' is  
abstract and can't be instantiated.
```

Implicit Interface

Dart tidak memiliki keyword atau syntax untuk mendeklarasikan interface seperti bahasa pemrograman OOP lainnya. Setiap class di dalam Dart dapat bertindak sebagai interface. Oleh karena itu interface pada dart dikenal sebagai *implicit interface*. Untuk mengimplementasikan interface, perlu menggunakan *keyword implements*. Beberapa interface dapat diimplementasikan sekaligus pada satu kelas.

Contoh :

```
class Flyable {  
    void fly() {}  
}
```

```
class Burung extends Hewan implements Flyable {  
    String warna;  
  
    Burung(String nama, int umur, double berat, this.warna) : super (nama,  
    umur, berat);  
  
    @override  
    void fly(){  
        print('$nama is flying');  
    }  
}
```

Ket : `@override` menunjukkan fungsi tersebut mengesampingkan fungsi yang ada di interface atau kelas induknya, lalu menggunakan fungsi yang ada dalam kelas itu sendiri sebagai gantinya.

Enumerated Types

Enum mewakili kumpulan konstan yang membuat kode kita lebih jelas dan mudah dibaca.

Contoh :

```
enum Pelangi {
    merah, jingga, kuning, hijau, biru, nila, ungu
}

enum Status {
    Todo, In_Progress, In_Review, Done
}
```

Enum memiliki beberapa property bawaan yang dapat digunakan untuk menampilkan seluruh nilai dalam bentuk list serta menampilkan item dan index.

Contoh :

```
print(Pelangi.values);
print(Pelangi.kuning);
print(Pelangi.biru.index);

// OUTPUT
[Pelangi.merah,     Pelangi.jingga,     Pelangi.kuning,     Pelangi.hijau,
Pelangi.biru, Pelangi.nila, Pelangi.ungu]
Pelangi.kuning
4
```

Paradigma Functional Programming

Fungsional programming adalah paradigma pemrograman di mana proses komputasi didasarkan pada fungsi matematika murni.

Beberapa konsep dan karakter functional programming :

1. Pure functions

Pure functions adalah sebuah fungsi yang bergantung dengan argumen atau parameter yang dimasukkan ke dalamnya.

Contoh :

```
int sum(int angka1, int angka2) {
    return angka1 + angka2
}
```

Ket : Pada fungsi **sum()** nilai yang akan dikembalikan bergantung pada argumen yang diberikan.

2. Recursion

Pada functional programming tidak ada konsep perulangan. Iterasi dilakukan melalui rekursi atau pemanggilan fungsi, hingga mencapai kasus dasar.

Contoh :

```
int fibonacci(n) {  
    if (n <= 0) {  
        return 0;  
    } else if(n == 1) {  
        return 1;  
    } else {  
        return fibonacci(n - 1) + fibonacci(n - 2);  
    }  
}
```

3. Immutable variables

artinya kita tidak bisa mengubah sebuah variabel ketika sudah diinisialisasi.

4. Functions are first-class citizen and can be higher-order

Function merupakan *first-class citizen* adalah bahwa function berlaku sama seperti komponen pemrograman yang lain. *Higher order functions* adalah fungsi yang mengambil fungsi lain sebagai argumen dan juga dapat mengembalikan fungsi.

Anonymous Functions

Anonymous function juga dikenal dengan nama lambda. Untuk membuatnya kita cukup menuliskan tanda kurung untuk menerima parameter dan body function-nya.

Contoh :

```
void main() {  
    var sum = (int angka1, int angka2) {  
        return angka1 + angka2;  
    };  
  
    Function printLambda = () {  
        print('Ini adalah fungsi lambda');  
    };  
  
    printLambda();  
    print(sum(3, 4));  
}
```

Selain itu juga dapat menggunakan expression untuk membuat kode fungsi menjadi lebih ringkas dengan fat arrow (\Rightarrow)

```
void main() {  
    var sum = (int angka1, int angka2) => angka1 + angka2;  
    Function printLambda = () => print('This is lambda function');  
  
    printLambda();  
    print(sum(3, 4));  
}
```

}

Higher-Order Functions

Higher order function adalah fungsi yang menggunakan fungsi lainnya sebagai parameter, menjadi tipe kembalian, atau keduanya.

Contoh :

```
void main() {  
    void contohHigherOrderFunction(String message, Function myFunction) {  
        print(message);  
        print(myFunction(3, 4));  
    }  
  
    // Opsi 1  
    Function sum = (int num1, int num2) => num1 + num2;  
    contohHigherOrderFunction('Hello', sum);  
  
    // Opsi 2  
    contohHigherOrderFunction('Hello', (num1, num2) => num1 + num2);  
}
```

Closures

adalah fungsi yang dapat mengakses variabel di dalam *lexical scope*-nya. *Lexical scope* berarti pada sebuah fungsi bersarang, fungsi yang berada di dalam memiliki akses ke variabel di lingkup induknya.

Contoh :

```
void main() {  
    var contohClosure = penjumlahan(2);  
    contohClosure();  
    contohClosure();  
}  
  
Function penjumlahan(base) {  
    var a = 1;  
    return () => print("Nilainya adalah ${base + a++}");  
}  
  
//Outputnya  
Nilainya adalah 3  
Nilainya adalah 4
```

Dart Type System

Type system adalah sistem logis yang terdiri dari seperangkat aturan yang menetapkan properti atau tipe ke berbagai konstruksi program komputer, seperti variabel, expression, fungsi, atau modul. Type system dalam dart disebut dengan sound type system.

Manfaat dari sound type system :

1. Mengungkap bug terkait tipe pada saat compile time.
2. Kode lebih mudah dibaca.
3. Kode lebih mudah dikelola
4. Kompilasi ahead of time (AOT) yang lebih baik.

Generic

Generic merupakan konsep yang digunakan untuk menentukan tipe data yang akan kita gunakan. Kita dapat mengganti tipe parameter generic pada Dart dengan lebih spesifik.

Contohnya dalam penggunaan List

```
List<int> bilangan = [1,2,3,4,5];
List<String> kata = ['Informatika', 'Flutter', 'Pemrograman'];
List dynamicList = [1, 2, 3, 'empat']; // List<dynamic>
```

Type Inference

Dart memiliki analyzer yang dapat menentukan tipe untuk field, method, variabel lokal, dan beberapa tipe argumen generic.

Contoh penulisan variabel map dengan tipe yang eksplisit

```
Map<String, dynamic> jurusan = {'prodi': 'Informatika', 'angkatan': 2020};
```

Ket : Kedua key dari map adalah String, namun nilainya memiliki tipe yang berbeda, yaitu String dan int, di mana keduanya merupakan turunan dari object. Sehingga variabel jurusan memiliki tipe **Map<String, Object>**.

Coding Convention

Adalah panduan untuk bahasa pemrograman tertentu yang merekomendasikan gaya pemrograman, praktik, dan metode untuk setiap aspek program yang ditulis dalam bahasa tersebut. Biasanya meliputi identitas, komentar, deklarasi, penamaan, arsitektur, dll.

Alasan mengapa code convention penting bagi programmer :

1. 40% - 80% biaya dari sebuah perangkat lunak digunakan untuk pemeliharaan (maintenance).
2. Sangat jarang suatu perangkat lunak dipelihara seterusnya oleh penulis aslinya.
3. Code convention meningkatkan keterbacaan kode, memungkinkan programmer untuk memahami kode baru dengan lebih cepat dan menyeluruh.
4. Source code lebih tertata rapi dan bersih sebagai sebuah produk.

Convention Dart dikenal dengan Effective Dart yang dibuat untuk mewujudkan dua hal berikut :

1. *Be consistent* (Konsisten).
2. *Be brief* (Ringkas)

Panduan Effective Dart

1. *Style guide*

Mendefinisikan aturan untuk mengatur kode, mengatur bagaimana format penamaan sebuah identifier, Dan menentukan penggunaan camelCase, underscore, atau komentar biasa untuk suatu kode.

2. *Documentation guide*.

Apa yang boleh ada dan tidak ada di dalam komentar dokumentasi maupun komentar biasa.

3. *Usage guide*

Mengatur bagaimana memanfaatkan fitur bahasa secara baik untuk menerapkan perilaku. Penggunaan statement atau expression dibahas di sini.

4. *Design guide*

Adalah panduan dengan cakupan terluas namun paling tidak mengikat. Mencakup bagaimana mendesain API library yang konsisten dan bisa digunakan.

5 kata kunci dalam Effective Dart

1. DO

Aturan yang diawali dengan DO maka praktik tersebut harus selalu diikuti.

Beberapa hal yang tercakup dalam aturan Do :

1. DO name type using UpperCamelCase

Class, enum, typedef, dan type parameter harus menggunakan huruf kapital pada huruf pertama dari setiap kata termasuk kata pertama.

Contoh :

```
abstract class Hewan {}  
abstract class Mammal extends Hewan {}  
mixin Flyable {}  
class Meong extends Mammal with Walkable {}
```

2. DO use ?? to convert null to a boolean value.

Berlaku ketika sebuah expression dapat mengevaluasi nilai true, false, atau null dan programer perlu meneruskan hasil ke suatu yang tidak menerima nilai null.

Contoh :

```
if(partnership?.isPartnership) {  
    print('Sudah bekerja sama');  
}
```

Ket : Kode tersebut akan menghasilkan exception ketika stock null. Untuk itu perlu adanya deklarasi nilai default ketika nilai awalnya null. Sehingga kodennya menjadi seperti berikut :

```
partnership?.isPartnership?? false;
```

2. DON'T

Aturan yang diawali dengan DON'T tidak baik untuk diterapkan.

Contohnya :

DON'T use prefix letters :

```
var instance; // good
var mInstance; // bad
```

3. PREFER

Adalah praktik yang harus diikuti. Namun, ada kemungkinan keadaan yang lebih masuk akal untuk melakukan sebaliknya.

Contohnya :

1. PREFER starting function or method comments with third-person verbs.

Sebuah komentar dokumentasi harus fokus menjelaskan apa yang dilakukan kode tersebut.

```
// Returns `true` if [username] and [password] inputs are
valid.

bool isValid(String username, String password) { }
```

4. AVOID

Avoid adalah kebalikan dari PREFER yang menjelaskan hal-hal yang tidak boleh dilakukan, namun kemungkinan ada alasan bagus untuk melakukannya pada beberapa kejadian.

Contohnya :

1. AVOID using curly braces in interpolation when not needed

Dart menggunakan fitur string interpolation untuk menggabungkan nilai string dengan nilai lain secara lebih mudah.

```
print('Hallo, ${nama}, berumur ${thisYear - birthYear}
tahun.');
```

Namun jika menginterpolasi identifier sederhana, maka curly braces ({{}}) tidak perlu ditulis.

```
print('Hallo, $nama, berumur ${thisYear - birthYear}
tahun.');
```

5. CONSIDER

Bisa diikuti atau tidak diikuti, tergantung pada keadaan dan preferensi.

Contohnya :

1. CONSIDER making the code read like a sentence

Penamaan dalam kode baik itu nama variabel, fungsi, maupun lainnya adalah hal yang sangat penting namun juga tidak mudah. Solusinya kita membuat seolah-olah sedang membuat kalimat.

```
// "If login is success ..."
If (login.isSuccess)

// "Hallo! Selamat datang kembali"
garfield.eat();

// Bad example
// Ambigu apakah memerintahkan toko untuk tutup atau
mendapatkan status dari toko
If (store.close)
```

Dart Future

Synchronous vs Asynchronous

Dalam Synchronous kode program dijalankan secara berurutan dari atas ke bawah, sedangkan program asynchronous memungkinkan suatu operasi berjalan sembari menunggu operasi lainnya selesai. Umumnya asynchronous digunakan pada operasi besar yang membutuhkan waktu lama, seperti mengambil data dari internet atau API, menyimpan data ke database, dan membaca data dari sebuah berkas.

Future

Future adalah sebuah objek yang mewakili nilai potensial atau kesalahan yang akan tersedia pada suatu waktu di masa mendatang. Penerima dari future dapat menentukan **callbacks** yang akan menangani nilai atau kesalahan. Future digunakan untuk melakukan pemrograman secara asynchronous.

Contoh :

```
Future<String> getProduct() {
  return Future.delayed(Duration(seconds: 3), () {
    return 'Matcha Latte';
  });
}
```

Ket : Pada Future bisa menambahkan method delayed untuk menunda eksekusi kode di dalam future.

Completed with data

Setelah Future dijalankan, perlu adanya handler untuk menangani status completed with data. Caranya dengan menggunakan method **.then()** dari objek Future.

Contoh :

```
void main() {  
    getProduct().then((value) {  
        print('You product: $value');  
    });  
    print('Getting your product...');  
}
```

Completed with error

Untuk mengatasi eror atau exception di dalam Future, kita dapat menambahkan method `.catchError()` setelah `then`.

```
void main() {  
    getProduct().then((value) {  
        print('You product: $value');  
    })  
    .catchError((error) {  
        print('Sorry. $error');  
    });  
    print('Getting your product...');  
}  
  
Future<String> getProduct() {  
    return Future.delayed(Duration(seconds: 3), () {  
        var isProductAvailable = false;  
        if (isProductAvailable) {  
            return 'Coffee Boba';  
        } else {  
            throw 'Our stock is not enough.';  
        }  
    });  
}
```

Future dengan `async-await`

Dart mempunyai keyword `async` dan `await` yang merupakan alternatif untuk dapat menuliskan proses asynchronous layaknya proses synchronous.

Contoh :

```
void main() async {  
    print('Getting your product...');  
    try {  
        var order = await getProduct();  
        print('You order: $order');
```

```
    } catch (error) {
        print('Sorry. $error');
    } finally {
        print('Thank you');
    }
}

Future<String> getProduct() {
    return Future.delayed(Duration(seconds: 3), () {
        var isProductAvailable = false;
        if (isProductAvailable) {
            return 'Matcha Latte';
        } else {
            throw 'Our stock is not enough.';
        }
    });
}
```

Ket : Untuk mengubahnya menjadi fungsi asynchronous tambahkan keyword `async` sebelum function body. Kemudian tambahkan keyword `await` sebelum fungsi yang mengembalikan nilai Future. Untuk menangani error cukup memanfaatkan **try-catch**.