Dart

Pemrograman Berbasis Mobile

Sejarah Dart

- Dikembangkan google pada tahun 2011 awalnya untuk Web Development
- Base untuk flutter
- Struktur sederhana Dart

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

Tipe Data

- Number (int, double)
- String
- Bool
- List
- Set
- Map

Deklarasi Variable

```dart
var name = 'Alice'; // Tipe String

var age = 20;       // Tipe int

String city = 'Jakarta';

int year = 2025;

final currentYear = DateTime.now().year;

const pi = 3.14159;
```

Final & Const

- **Final**: Nilai hanya boleh diinisialisasi sekali, tetapi **bisa** diinisialisasi saat *runtime*.
- Cocok jika nilai tergantung pada kondisi saat program berjalan.

- **Const**: Nilai dikompilasi pada waktu compile-time, benar-benar konstan.
- Harus diinisialisasi dengan ekspresi konstan (angka, string literal, atau ekspresi yang dinyatakan konstan).

Dart ⇒ Null safety

Secara default (dengan fitur null safety aktif), Dart mengharuskan semua variabel bertipe non-nullable artinya harus memiliki nilai/value saat dideklarasikan.

Jika tidak ada nilainya maka kompiler akan menolak dan menghasilkan pesan error

```
String name; // Error: Non-nullable variable 'name' must be assigned.
```

Late

```dart
late String name;


void main() {
  // ... melakukan sesuatu ...
  name = 'Alice'; // Baru di-assign
  print(name);      // Aman digunakan karena sudah diinisialisasi
}
```

Nullable (?)

```dart
String? name; // bisa bernilai null

void main() {
  print(name);   // null (by default)
  name = 'Alice';
  print(name);   // Alice
}
```

Inisialisasi Langsung di Deklarasi

```
String name = 'Belum diisi'; // default value
```

Percabangan

```
if (age < 18) {
  print('Minor');
} else if (age < 60) {
  print('Adult');
} else {
  print('Elder');
}
```

Perulangan

```
for (var i = 0; i < 5; i++) {
  print('Count: $i');
}
```

Collection If/For

```
var evenNumbers = [
  for (var i = 0; i < 10; i++)
    if (i % 2 == 0) i
];
// evenNumbers = [0, 2, 4, 6, 8]
```

Fungsi Dasar

```
// Tanpa tipe pengembalian eksplisit
sayHello(String name) {
  print('Hello, $name');
}

// Dengan tipe pengembalian eksplisit
int sum(int a, int b) {
  return a + b;
}
```

Fungsi dengan Named Parameters

```dart
// Named parameter bersifat optional, dapat diberi default value
void createUser({String name = 'Guest', int age = 0}) {
  print('Name: $name, Age: $age');
}

// Pemanggilan
createUser(name: 'Alice', age: 21);
createUser(); // Name: Guest, Age: 0
```

Arrow Function

```
int multiply(int x, int y) => x * y;
```

Class, Constructor, Properties, dan Methods

```dart
class Person {
  String name;
  int age;
  // Constructor
  Person(this.name, this.age);
  // Method
  void introduce() {
    print('My name is $name, I am $age years old.');
  }
}
```

```dart
void main() {
  var p = Person('Bob', 25);
  p.introduce();
}
```

Inheritance & Mixins

```
class Employee extends Person {
  double salary;
  Employee(String name, int age, this.salary) : super(name, age);

  void showSalary() {
    print('Salary: $salary');
  }
}
```

Mixins

Mixins adalah cara menambahkan perilaku/ functionality ke dalam kelas tanpa menggunakan pewarisan berlapis. Ini solusi Dart untuk multiple inheritance terkontrol.

```
mixin Logger {
  void log(String message) {
    print('LOG: $message');
  }
}
```

Mixins

```dart
class DataManager with Logger {
  void saveData() {
    // Kode simpan data
    log('Data saved successfully.');
  }
}
```

```dart
void main() {
  var manager = DataManager();
  manager.saveData();  // LOG: Data saved successfully.
}
```

Konsep Asinkron dalam Pemrograman Mobile

- Pemrograman asinkron adalah gaya pemrograman di mana sebuah tugas tidak harus menjalankan proses secara urut dan blok (menunggu hingga selesai) sebelum melanjutkan ke tugas berikutnya.
- Dalam konteks aplikasi mobile, artinya kita bisa menjalankan proses di “latar belakang” sambil tetap membiarkan antarmuka pengguna (UI) tetap responsif.

Mengapa Asinkron Penting untuk Mobile?

- **Responsivitas**: Di aplikasi mobile, jika thread utama (UI thread) diblok oleh proses panjang (misalnya menunggu data dari server), pengguna akan merasa aplikasi “macet” (freeze). Dengan asinkron, proses mengambil data dapat berjalan di latar belakang, sementara UI tetap aktif.
- **Efisiensi**: Mengurangi waktu tunggu (blocking time). Beberapa proses bisa berjalan bersamaan dan dikelola dengan sistem event loop di Dart.
- **User Experience (UX) Lebih Baik**: Pengguna bisa terus berinteraksi dengan aplikasi, sedangkan operasi berat berlangsung secara non-blocking.

Mekanisme Asinkron di Dart/Flutter

- Future: Objek yang merepresentasikan hasil yang akan tersedia di masa depan (entah sukses atau gagal).
- async/await: Sintaks sugar untuk menulis kode asinkron agar lebih mirip kode sinkron, memudahkan pembacaan dan error handling dengan try/catch.
- Isolates (Lanjutan): Dart juga mendukung isolates, yakni thread terpisah yang tidak berbagi memori secara langsung, tetapi berkomunikasi lewat pesan.

Mobile Asinkron

```dart
Future<String> fetchUserData()
async {

  // simulasi delay

  await
Future.delayed(Duration(seconds:
2));

  // misal data JSON di-decode di
sini

  return 'User Data from server';

}
```

```dart
void main() async {

  print('Mulai ambil data...');

  var result = await
fetchUserData();

  print(result);

  print('Selesai');

}
```

Streams di Dart

Apa itu Stream?

- Stream adalah aliran data yang datang secara bertahap/berurutan sepanjang waktu.
- Cocok untuk skenario di mana kita mendapat kumpulan data atau event secara bertahap, bukan sekaligus.
- Contohnya: real-time updates, sensor data, chat messages, file upload progress, dsb.

Streams di Dart

Tipe Stream

- Single-subscription Stream: Hanya boleh di-listen oleh satu pendengar (listener) pada satu waktu. Biasanya untuk proses data sekuensial sekali jalan (misalnya membaca file).
- Broadcast Stream: Dapat di-listen oleh banyak pendengar sekaligus. Cocok untuk event bus atau notifikasi yang perlu dikonsumsi banyak bagian aplikasi.

Kegunaan Streams

- Data Sensor (GPS)
- Realtime Chat
- Download/Upload Progress