D4 Rekayasa Perangkat Lunak (RPL)
Jurusan Teknik Informatika
Politeknik Negeri Indramayu

Asynchronous Progamming (Dart)

Fachrul Pralienka Bani Muhamad, S.ST., M.Kom.
fachrul.pbm@polindra.ac.id

2025

Tujuan

Setelah materi ini disampaikan, mahasiswa diharapkan mampu

- Menjelaskan perbedaan fungsi dan operasi synchronous dan asynchronous
- Menentukan kapan untuk menggunakan asynchronous
- Mengimplementasikan asynchronous pada Bahasa pemrograman Dart

Outlines

- Definisi asynchronous
- Term pada asynchronous
- Asynchronous pada Dart
- Class `Future`
- Keyword `async` & `await`

Definisi asynchronous

- Suatu proses eksekusi kode program secara paralel (bersamaan) tanpa adanya proses antrian
- Operasi asynchronous memungkinkan program untuk menyelesaikan suatu pekerjaan ketika operasi yang lain belum selesai
- Umumnya asynchronous digunakan pada proses
    1. Fetching (perolehan) data melalui suatu jaringan
    2. Writing (pencatatan) data ke database
    3. Reading (pembacaan) data dari suatu file

Asynchronous operation & function

- Terdapat ‘term’ atau istilah yang digunakan pada asynchronous yaitu operation dan function
- Asynchronous operation → suatu operasi yang mengizinkan operasi lain dieksekusi sebelum operasi tersebut selesai
- Asynchronous function → suatu fungsi yang mengeksekusi **minimal satu** operasi **asynchronous** serta boleh juga ada/tidak ada operasi synchronous
  - Synchronous operation → suatu operasi yang menghalangi operasi lain selama operasi tersebut belum diselesaikan
  - Synchronous function → suatu fungsi yang hanya dapat mengeksekusi opersi-operasi synchronous

Asynchronous pada Dart

- Operasi asynchronous pada Bahasa pemrograman Dart dapat dilakukan dengan menggunakan hal berikut:
  - Class `Future`
  - Keyword `async`
  - Keyword `await`

Class Future

- `future` dengan awalan non-kapital adalah suatu instance dari class `Future`
- Class `Future` adalah class yang merepresentasikan hasil dari suatu operasi asynchronous
- Terdapat dua state pada hasil operasi tersebut, yaitu:
  - *Uncompleted* ➔ mengembalikan suatu instance `future` yang tidak selesai
  - *Completed* ➔ hanya didapatkan apabila suatu operasi asynchronous telah berhasil dengan suatu nilai tertentu atau error
    - Completed with value ➔ Misalnya `Future<String>` yang berhasil mengembalikan instance `future` dengan nilai `String`
    - Completed with error ➔ Operasi telah dilakukan (selesai) tetapi mengembalikan error

Class `Future`

- Pembuatan method dengan tipe `Future<T>` dapat mengembalikan instance `future` dengan nilai bertipe data `T`, dimana `T` adalah tipe data **primitive** maupun tipe data **object**
- Apabila tidak dibutuhkan kembalian data, maka kode dapat disajikan menjadi `Future<void>`
- Class `Future` memiliki method static (dapat dipanggil tanpa melalui instansiasi) yang seringkali digunakan sebagai contoh asyhnchronous, yaitu `delayed()`

Contoh 1 - asynchronous yang tidak tepat

- Berikut adalah contoh kode program asynchronous yang tidak tepat

```dart
// This example shows how *not* to write asynchronous Dart code.

String createOrderMessage() {
  var order = fetchUserOrder();
  return 'Your order is: $order';
}

Future<String> fetchUserOrder() =>
    // Imagine that this function is more complex and slow.
    Future.delayed(
      const Duration(seconds: 2),
      () => 'Large Latte',
    );

void main() {
  print(createOrderMessage());
}
```

Contoh 1 - asynchronous yang tidak tepat

- Berikut adalah contoh kode program asynchronous yang tidak tepat

```dart
// This example shows how *not* to write asynchronous Dart code.

String createOrderMessage() {
  var order = fetchUserOrder();
  return 'Your order is: $order';
}

Future<String> fetchUserOrder() =>
    // Imagine that this function is more complex and slow.
    Future.delayed(
      const Duration(seconds: 2),
      () => 'Large Latte',
    );

void main() {
  print(createOrderMessage());
}
```

Contoh 1 - asynchronous yang tidak tepat

- Berikut adalah contoh kode program asynchronous yang tidak tepat

```dart
// This example shows how *not* to write asynchronous Dart code.

String createOrderMessage() {
  var order = fetchUserOrder();
  return 'Your order is: $order';
}

Future<String> fetchUserOrder() =>
    // Imagine that this function is more complex and slow.
    Future.delayed(
      const Duration(seconds: 2),
      () => 'Large Latte',
    );

void main() {
  print(createOrderMessage());
}
```

```
Your order is: Instance of '_Future<String>'
```

Penjelasan contoh 1

- Pada intinya program asynchronous tersebut berhasil berjalan, tetapi hasil yang ditampilkan tidak sesuai yang diharapkan
- Hasil yang diharapkan adalah `'Your order is: Large Latte'`
- Hal tersebut terjadi karena `fetchUserOrder()` **dipanggil** dengan operasi **synchronous** melalui synchronous function `createOrderMessage()`
- Untuk menangani ketidaksesuaian hasil tersebut, maka kode program (contoh 1) dapat dimodifikasi menjadi berikut (contoh 2)

Contoh 2 - asynchronous yang mengembalikan nilai

- Berikut adalah contoh hasil modifikasi penggunaan class `Future` yang benar dan berhasil dieksekusi hingga mengembalikan nilai

```dart
Future<void> fetchUserOrder() {
  // Imagine that this function is fetching user info from another service or database.
  return Future.delayed(const Duration(seconds: 2), () => print('Large Latte'));
}

void main() {
  fetchUserOrder();
  print('Fetching user order...');
}
```

Contoh 2 - asynchronous yang mengembalikan nilai

- Berikut adalah contoh hasil modifikasi penggunaan class `Future` yang benar dan berhasil dieksekusi hingga mengembalikan nilai

```dart
Future<void> fetchUserOrder() {
  // Imagine that this function is fetching user info from another service or database.
  return Future.delayed(const Duration(seconds: 2), () => print('Large Latte'));
}

void main() {
  fetchUserOrder();
  print('Fetching user order...');
}
```

Contoh 2 - asynchronous yang mengembalikan nilai

- Berikut adalah contoh hasil modifikasi penggunaan class `Future` yang benar dan berhasil dieksekusi hingga mengembalikan nilai

```dart
Future<void> fetchUserOrder() {
  // Imagine that this function is fetching user info from another service or database.
  return Future.delayed(const Duration(seconds: 2), () => print('Large Latte'));
}

void main() {
  fetchUserOrder();
  print('Fetching user order...');
}
```

```
Fetching user order...
Large Latte
```

Penjelasan contoh 2

- Program berhasil dijalankan dengan kembalian nilai yang sesuai harapan
- Modifikasi yang dilakukan adalah dengan menyederhanakan pemanggilan asynchronous function (penggunaan class Future) secara langsung tanpa melalui perantara synchronous function

Contoh 3 - asynchronous yang mengembalikan error

- Berikut adalah contoh asynchronous function (penggunaan class `Future`) yang benar dan berhasil dieksekusi tetapi mengembalikan error

```dart
Future<void> fetchUserOrder() {
// Imagine that this function is fetching user info but encounters a bug
  return Future.delayed(const Duration(seconds: 2),
      () => throw Exception('Logout failed: user ID is invalid'));
}

void main() {
  fetchUserOrder();
  print('Fetching user order...');
}
```

Contoh 3 - asynchronous yang mengembalikan error

- Berikut adalah contoh asynchronous function (penggunaan class `Future`) yang benar dan berhasil dieksekusi tetapi mengembalikan error

```dart
Future<void> fetchUserOrder() {
// Imagine that this function is fetching user info but encounters a bug
  return Future.delayed(const Duration(seconds: 2),
      () => throw Exception('Logout failed: user ID is invalid'));
}

void main() {
  fetchUserOrder();
  print('Fetching user order...');
}
```

Contoh 3 - asynchronous yang mengembalikan error

- Berikut adalah contoh asynchronous function (penggunaan class `Future`) yang benar dan berhasil dieksekusi tetapi mengembalikan error

```dart
Future<void> fetchUserOrder() {
// Imagine that this function is fetching user info but encounters a bug
  return Future.delayed(const Duration(seconds: 2),
      () => throw Exception('Logout failed: user ID is invalid'));
}

void main() {
  fetchUserOrder();
  print('Fetching user order...');
}
```

```
Fetching user order...
Uncaught Error: Exception: Logout failed: user ID is invalid
```

Penjelasan contoh 3

- Program berhasil dijalankan dengan mengembalikan error
- Error tersebut dihasilkan melalui `throw Exception` pada operasi asynchronous yang ada di `fetchUserOrder()`

Keyword `async` & `await`

- Kedua keywords tersebut menyediakan cara yang lebih rinci untuk mendefinisikan fungsi asynchronous dan menggunakan hasilnya
- Keyword `async` diletakkan sebelum deklarasi suatu body function, contoh penulisan:

```
void main() async { ··· }
```

atau

```
Future<void> main() async { ··· }
```

- Keyword `await` hanya dapat bekerja di dalam function yang terdapat keyword `async`, contoh penulisan:

```
print(await createOrderMessage());
```

Keyword `async` & `await`

- Penggunaan keyword `async` pada suatu function menandakan adanya suatu fungsi asynchronous
- Keyword `await` dapat digunakan untuk mendapatkan hasil lengkap dari suatu operasi asynchronous
- Keyword `await` hanya bekerja di dalam suatu fungsi dengan label `async`

Contoh 4 - asynchronous menggunakan `async` dan `await`

- Berikut adalah contoh asynchronous function dengan memanfaatkan keyword `async` dan `await`

```
Future<String> createOrderMessage() async {
  var order = await fetchUserOrder();
  return 'Your order is: $order';
}

Future<String> fetchUserOrder() =>
    // Imagine that this function is
    // more complex and slow.
    Future.delayed(
      const Duration(seconds: 2),
      () => 'Large Latte',
    );

Future<void> main() async {
  print('Fetching user order...');
  print(await createOrderMessage());
}
```

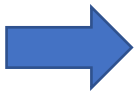
Contoh 4 - asynchronous menggunakan `async` dan `await`

- Berikut adalah contoh asynchronous function dengan memanfaatkan keyword `async` dan `await`

```dart
Future<String> createOrderMessage() async {
  var order = await fetchUserOrder();
  return 'Your order is: $order';
}

Future<String> fetchUserOrder() =>
    // Imagine that this function is
    // more complex and slow.
    Future.delayed(
      const Duration(seconds: 2),
      () => 'Large Latte',
    );

Future<void> main() async {
  print('Fetching user order...');
  print(await createOrderMessage());
}
```



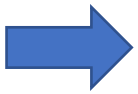
Contoh 4 - asynchronous menggunakan `async` dan `await`

- Berikut adalah contoh asynchronous function dengan memanfaatkan keyword `async` dan `await`

```dart
Future<String> createOrderMessage() async {
  var order = await fetchUserOrder();
  return 'Your order is: $order';
}

Future<String> fetchUserOrder() =>
    // Imagine that this function is
    // more complex and slow.
    Future.delayed(
      const Duration(seconds: 2),
      () => 'Large Latte',
    );

Future<void> main() async {
  print('Fetching user order...');
  print(await createOrderMessage());
}
```



```
Fetching user order...
Your order is: Large Latte
```

Penjelasan contoh 4

- Async function dieksekusi secara synchronous sampai dengan keyword await yang pertama
- Hal ini berarti di dalam body suatu async function, semua kode program yang dibuat sebelum keyword await akan dieksekusi secara synchronous
- Misalnya pada body method main, yaitu `print('Fetching user order')` → dieksekusi secara synchronous

```
Future<void> main() async {
  print('Fetching user order...');
  print(await createOrderMessage());
}
```

Latihan 1

Apa output yang dihasilkan kode berikut? Jelaskan!

```dart
Future<void> printOrderMessage() async {
  print('Awaiting user order...');
  var order = await fetchUserOrder();
  print('Your order is: $order');
}

Future<String> fetchUserOrder() {
  // Imagine that this function is more complex and slow.
  return Future.delayed(const Duration(seconds: 4), () => 'Large Latte');
}

void main() async {
  countSeconds(4);
  await printOrderMessage();
}

// You can ignore this function - it's here to visualize delay time in this example.
void countSeconds(int s) {
  for (var i = 1; i <= s; i++) {
    Future.delayed(Duration(seconds: i), () => print(i));
  }
}
```

Latihan 2

Modifikasikanlah kode program berikut yang berisi simulasi **perolehan data user** dan **data transaksi** yang masih dilakukan secara synchronous, agar dapat dieksekusi secara asynchronous

latihan_asynchronous - sequential.dart

```dart
Future<String> fetchUserData() {
  print("-> Memulai fetchUserData...");
  return Future.delayed(Duration(seconds: 2), () {
    print("<- fetchUserData selesai (2s).");
    return "Nama Pengguna: Joko Santoso, ID: 123";
  });
}

Future<String> fetchTransactionData() {
  print("-> Memulai fetchTransactionData...");
  return Future.delayed(Duration(seconds: 3), () {
    print("<- fetchTransactionData selesai (3s).");
    return "Total Transaksi Bulan Ini: Rp 500.000";
  });
}

```

```dart
void main() async {
  final startTime = DateTime.now();

  try {
    String userData = await fetchUserData();
    print("Data Pengguna berhasil diambil: $userData\n\n");

    String transactionData = await fetchTransactionData();
    print("Data Transaksi berhasil diambil: $transactionData\n\n");

    final endTime = DateTime.now();
    final duration = endTime.difference(startTime).inSeconds;

    print("Proses perolehan data pengguna dan transaksi telah selesai");
    print("Total waktu eksekusi: $duration detik");

  } catch (e) {
    print("Terjadi error: $e");
  }
}

```

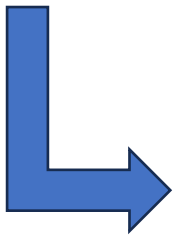
Latihan 2

Modifikasikanlah kode program berikut yang berisi simulasi **perolehan data user** dan **data transaksi** yang masih dilakukan secara synchronous, agar dapat dieksekusi secara asynchronous

```
-> Memulai fetchUserData...
<- fetchUserData selesai (2s).
Data Pengguna berhasil diambil: Nama Pengguna: Joko Santoso, ID: 123


-> Memulai fetchTransactionData...
<- fetchTransactionData selesai (3s).
Data Transaksi berhasil diambil: Total Transaksi Bulan Ini: Rp 500.000


Proses perolehan data pengguna dan transaksi telah selesai
Total waktu eksekusi: 5 detik
```



```
-> Memulai fetchUserData...
-> Memulai fetchTransactionData...


<- fetchUserData selesai (2s).
Data Pengguna berhasil diambil: Nama Pengguna: Joko Santoso, ID: 123


<- fetchTransactionData selesai (3s).
Data Transaksi berhasil diambil: Total Transaksi Bulan Ini: Rp 500.000


Semua perolehan data telah dilakukan secara paralel
Total waktu eksekusi: 3 detik.
```

Selesai

Terima kasih 🙏