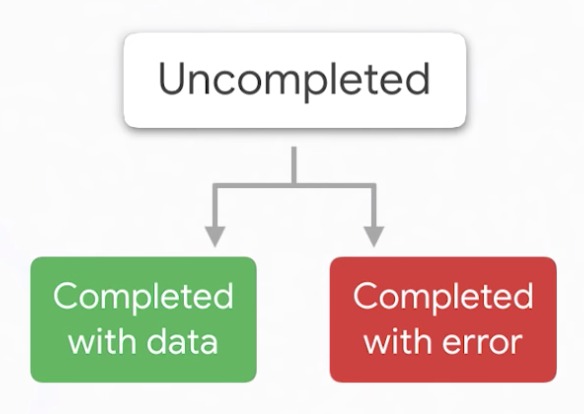
**Apa itu Future**

Untuk melakukan pemrograman secara *asynchronous* dengan Dart, kita menggunakan ***Future***. *Future* adalah sebuah objek yang mewakili nilai potensial atau kesalahan yang akan tersedia pada suatu waktu di masa mendatang. Anda dapat membayangkan *future* sebagai sebuah kotak paket. Saat Anda menerima paket tersebut, akan ada tiga kondisi yang mungkin terjadi, antara lain paket masih tertutup (***uncompleted***), paket dibuka lalu berisi barang sesuai pesanan (***completed with data***), dan paket dibuka namun terjadi kesalahan atau tidak sesuai (***completed with error***). Penerima dari *future* dapat menentukan *callbacks* yang akan menangani nilai atau kesalahan tersebut.

[](https://www.dicoding.com/academies/191/tutorials/7678?from=7675)

Sekarang coba perhatikan kode berikut:

1. void main() {
2. print('Creating the future');
3. print('main() done');
4. }

Tentunya Anda sudah tahu urutan program dan apa yang akan ditampilkan pada konsol. Lalu bagaimana jika perintah print yang pertama kita pindahkan ke dalam objek *future*.

1. void main() {
2. final myFuture = Future(() {
3. print('Creating the future');
4. return 12;
5. });
6. print('main() done');
7. }

Jika kode di atas dijalankan, seluruh fungsi main akan dieksekusi sebelum fungsi yang ada di dalam Future(). Ini disebabkan karena *future* masih berstatus *uncompleted*. Sehingga ketika program dijalankan, konsol akan tampil seperti berikut:

1. main() done
2. Creating the future

Lantas bagaimana caranya kita membuat dan memanfaatkan *future* ini?

**Uncompleted**

Mari kita buat sebuah fungsi yang mengembalikan nilai Future.

1. Future<String> getOrder() {
2. return Future.delayed(Duration(seconds: 3), () {
3. return 'Coffee Boba';
4. });
5. }

Pada Future kita bisa menambahkan *method* delayed untuk menunda eksekusi kode di dalam Future. Di mana parameter pertama berisi durasi penundaan dan parameter kedua adalah blok kode atau fungsi yang akan dijalankan. Pada kasus ini kita menggunakan *delayed* untuk menunda waktu eksekusi seolah kita sedang mengambil data dari internet. Karena nilai yang dikembalikan pada Future berupa String, kita bisa secara eksplisit menentukan tipe parameter *generic* Future<String>.

**Completed with data**

Kemudian setelah Future dijalankan, kita memerlukan *handler* untuk menangani status *completed with data*. Caranya gunakan method .then() dari objek Future.

1. void main() {
2. getOrder()**.then((value) {**
3. **print('You order: $value');**
4. **});**
5. print('Getting your order...');
6. }

Fungsi getOrder() akan dijalankan secara *asynchronous* hingga setelah 3 detik kode Future akan dijalankan dan mengembalikan nilai.

1. Getting your order...
2. You order: Coffee Boba  // Muncul setelah 3 detik

Bagaimana jika objek Future menghasilkan kondisi “*completed with error*?” Bagaimana menanganinya?

**Completed with error**

Kita dapat menambahkan *method* .catchError() setelah then. Sehingga ketika terjadi eror atau *exception* di dalam Future, blok kode ini akan dijalankan. Mari kita ubah sedikit kode di atas untuk mendukung skenario *completed with error*.

1. void main() {
2. getOrder().then((value) {
3. print('You order: $value');
4. })
5. **.catchError((error) {**
6. **print('Sorry. $error');**
7. **});**
8. print('Getting your order...');
9. }
11. Future<String> getOrder() {
12. return Future.delayed(Duration(seconds: 3), () {
13. **var isStockAvailable = false;**
14. **if (isStockAvailable) {**
15. **return 'Coffee Boba';**
16. **} else {**
17. **throw 'Our stock is not enough.';**
18. **}**
19. });
20. }

Menurut Anda apa yang akan ditampilkan di konsol? Coba jalankan aplikasinya untuk membuktikan.

Sampai sini harusnya Anda sudah paham dengan ketiga *state* yang ada pada *Future* serta bagaimana menuliskan kode untuk menanganinya. Seperti pada fungsi main() ada tiga blok kode yang mewakili *state Future*:

1. Fungsi getOrder() yang berisi Future yang masih ***uncompleted***.
2. Method then() yang menangani kondisi ***completed with data***.
3. Method catchError() yang menangani kondisi ***completed with error***.

Ada satu *method* lagi yang bisa kita gunakan yaitu whenComplete(). Method ini akan dijalankan ketika suatu fungsi Future selesai dijalankan, tak peduli apakah menghasilkan nilai atau eror. Ini seperti blok finally pada *try-catch-finally*.

1. void main() {
2. getOrder().then((value) {
3. print('You order: $value');
4. })
5. .catchError((error) {
6. print('Sorry. $error');
7. })
8. .whenComplete(() {
9. print('Thank you');
10. });
11. print('Getting your order...');
12. }