**Paradigma Functional Programming**

Seperti yang sudah disampaikan di awal, Dart adalah bahasa yang mendukung *multiparadigm*. Artinya selain merupakan bahasa pemrograman berorientasi objek, penulisan *syntax* Dart juga menggunakan gaya *functional programming*.

*Functional programming* adalah paradigma pemrograman di mana proses komputasi didasarkan pada fungsi matematika murni. *Functional programming* (FP) ditulis dengan gaya deklaratif yang berfokus pada “*what to solve*” dibandingkan “*how to solve*” pada gaya imperatif.

Berikut ini beberapa konsep dan karakteristik *functional programming*:

**Pure functions**

*Pure functions* berarti sebuah fungsi bergantung pada argumen atau parameter yang dimasukkan ke dalamnya. Sehingga pemanggilan fungsi dengan nilai argumen yang sama akan selalu memberikan hasil yang sama pula. Contohnya pada fungsi sum() berikut nilai yang dikembalikan akan bergantung pada argumen yang diberikan.

1. int sum(int num1, int num2) {
2. return num1 + num2;
3. }

**Recursion**

Pada *functional programming* tidak ada konsep perulangan seperti for atau while. Iterasi pada *functional programming* dilakukan melalui rekursi atau pemanggilan fungsi dari fungsi itu sendiri, hingga mencapai kasus dasar.

1. int fibonacci(n) {
2. if (n <= 0) {
3. return 0;
4. } else if(n == 1) {
5. return 1;
6. } else {
7. return fibonacci(n - 1) + fibonacci(n - 2);
8. }
9. }

**Immutable variables**

Variabel pada *functional programming* bersifat *immutable*, artinya kita tidak bisa mengubah sebuah variabel ketika sudah diinisialisasi. Alih-alih mengubah nilai variabel, kita bisa membuat variabel baru untuk menyimpan data. Mekanisme ini bertujuan agar kode kita menjadi lebih aman karena *state* dari aplikasi tidak akan berubah sepanjang aplikasi berjalan.

1. var x = 5;
2. x = x + 1; // Contoh variable yang tidak immutable

**Functions are first-class citizen and can be higher-order**

Maksud dari *function* merupakan *first-class citizen* adalah bahwa *function* berlaku sama seperti komponen pemrograman yang lain. Sebuah fungsi bisa dimasukkan ke variabel menjadi parameter dalam suatu fungsi dan juga menjadi nilai kembalian pada fungsi. *Higher order functions* adalah fungsi yang mengambil fungsi lain sebagai argumen dan juga dapat mengembalikan fungsi.

Pada modul ini kita akan mempelajari bagaimana penulisan gaya *functional* dengan bahasa Dart.

## **Anonymous Functions**

Masih ingatkah Anda dengan materi function dan cara membuatnya? Seperti yang kita tahu, untuk mendeklarasikan sebuah fungsi kita perlu mendefinisikan nilai kembalian dan juga nama fungsinya.

1. int sum(int num1, int num2) {
2. return num1 + num2;
3. }

Kebanyakan fungsi pada Dart memiliki nama seperti sum(), main(), atau print(). Pada Dart kita bisa membuat fungsi yang tidak bernama alias nameless atau anonymous. Anonymous function ini juga dikenal dengan nama ***lambda***.

Untuk membuat lambda atau anonymous function kita cukup menuliskan tanda kurung untuk menerima parameter dan body function-nya.

1. void main() {
2. (int num1, int num2) {
3. return num1 + num2;
4. };
5. }

Lalu bagaimana kita bisa menggunakan fungsi tersebut? Seperti yang telah dijelaskan sebelumnya bahwa function adalah first-class citizen, maka fungsi juga merupakan sebuah objek yang bisa disimpan ke dalam variabel. Kita bisa menggunakan keyword var atau secara eksplisit menggunakan tipe data Function.

1. void main() {
2. var sum = (int num1, int num2) {
3. return num1 + num2;
4. };
6. Function printLambda = () {
7. print('This is lambda function');
8. };
9. }

Untuk memanggilnya kita bisa langsung memanggil nama variabelnya seperti berikut:

1. printLambda();
2. print(sum(3, 4));

Selain itu lambda juga mendukung function expression untuk membuat kode fungsi menjadi lebih ringkas dengan memanfaatkan fat arrow (=>).

## **Higher-Order Functions**

Setelah mempelajari modul sebelumnya, Anda mungkin bertanya apa yang bisa dilakukan dengan lambda atau anonymous function?

Kita bisa memanfaatkan lambda untuk membuat higher-order function. Higher order function adalah fungsi yang menggunakan fungsi lainnya sebagai parameter, menjadi tipe kembalian, atau keduanya.

Coba perhatikan fungsi berikut:

1. void myHigherOrderFunction(String message, Function myFunction) {
2. print(message);
3. print(myFunction(3, 4));
4. }

Fungsi di atas merupakan higher order function karena menerima parameter berupa fungsi lain. Untuk memanggil fungsi di atas, kita bisa langsung memasukkan lambda sebagai parameter maupun variabel yang berisi nilai berupa fungsi.

void **main**() {

  // Opsi 1

  int Function(int num1, int num2) sum = (int num1, int num2) => num1 + num2;

**myHigherOrderFunction**('Hello', sum);

  // Opsi 2

**myHigherOrderFunction**('Hello', (num1, num2) => num1 + num2);

}

void **myHigherOrderFunction**(

  String message,

  int Function(int num1, int num2) myFunction,

) {

**print**(message);

**print**(**myFunction**(3, 4));

}

Jika disimulasikan fungsi myHigherOrderFunction akan memanggil fungsi sum yang dijadikan parameter.

1. void myHigherOrderFunction(String message, Function myFunction) {
2. print(message);
3. print(myFunction(3, 4));  // sum(3, 4)    // return 3 + 4
4. }

Namun deklarasi higher order function ini bisa menjadi sedikit tricky. Misalnya kode di bawah ini tidak akan terdeteksi eror namun ketika dijalankan, aplikasi Anda akan mengalami crash. Tahukah kenapa?

1. void myHigherOrderFunction(String message, Function myFunction) {
2. print(message);
3. print(myFunction(4));
4. }

Karena kita tidak menentukan spesifikasi dari fungsi seperti jumlah parameter atau nilai kembaliannya, maka semua jenis fungsi akan bisa dijalankan termasuk pemanggilan myFunction seperti di atas. Untuk mengatasinya kita bisa lebih spesifik menentukan seperti apa fungsi yang valid untuk menjadi parameter.

1. void myHigherOrderFunction(String message, int Function(int num1, int num2) myFunction) { }

Pada fungsi di atas kita perlu memasukkan fungsi dengan dua parameter dan nilai kembali berupa int sebagai parameter.

Pada materi collection sebenarnya kita telah menggunakan satu fungsi yang merupakan higher order function yaitu fungsi forEach(). Sebagai contoh kita punya daftar bilangan fibonacci yang disimpan ke sebuah variabel.

1. var fibonacci = [0, 1, 1, 2, 3, 5, 8, 13];

IntelliJ IDEA akan menunjukkan suggestion apa saja yang perlu menjadi parameter. Kita bisa melihat bahwa forEach membutuhkan satu parameter berupa fungsi.

Sehingga ketika memanggil fungsi ini kita bisa melakukan operasi pada masing-masing item misalnya mencetak ke konsol.

1. fibonacci.forEach((item) {
2. print(item);
3. });

## **Closures**

Suatu fungsi dapat dibuat dalam lingkup global atau di dalam fungsi lain. Suatu fungsi yang dapat mengakses variabel di dalam lexical scope-nya disebut dengan ***closure***. Lexical scope berarti bahwa pada sebuah fungsi bersarang (nested functions), fungsi yang berada di dalam memiliki akses ke variabel di lingkup induknya.

Berikut ini adalah contoh kode implementasi closure:

void **main**() {

  var closureExample = **calculate**(2);

**closureExample**();

**closureExample**();

}

Function **calculate**(base) {

  var count = 1;

  return () => **print**("Value is ${base + count++}");

}

Ketika kode di atas dijalankan, konsol akan tampil seperti berikut:

1. Value is 3
2. Value is 4

Di dalam fungsi calculate() terdapat variabel count dan mengembalikan nilai berupa fungsi. Fungsi lambda di dalamnya memiliki akses ke variabel count karena berada pada lingkup yang sama. Karena variabel count berada pada scope calculate, maka umumnya variabel tersebut akan hilang atau dihapus ketika fungsinya selesai dijalankan. Namun pada kasus di atas fungsi lambda atau closureExample masih memiliki referensi atau akses ke variabel count sehingga bisa diubah. Variabel pada mekanisme di atas telah tertutup (close covered), yang berarti variabel tersebut berada di dalam closure.

## **Rangkuman Materi**

Anda berada di akhir dari modul Functional Programming. Mari kita uraikan materi yang sudah Anda pelajari untuk mempertajam pemahaman.

* Functional Programming (FP) merupakan paradigma pemrograman di mana proses komputasi didasarkan pada fungsi matematika murni.
* Functional programming ditulis dengan gaya deklaratif yang berfokus pada “what to solve” dibandingkan “how to solve” pada gaya imperatif.
* Terdapat beberapa konsep dan karakteristik functional programming, antara lain:
  + Pure functions, fungsi bergantung pada argumen dan parameter yang dimasukkan ke dalamnya.
  + Recursion, fungsi yang tidak memiliki konsep perulangan sehingga fungsi hanya berfokus pada perintah dasar.
  + Immutable variables, fungsi bersifat immutable, artinya tidak bisa mengubah sebuah variabel ketika sudah diinisialisasi.
  + First-class citizen, fungsi memiliki sifat yang setara dengan komponen pemrograman yang lain.
  + Higher-order function, fungsi yang menggunakan fungsi lain sebagai argumen dan juga dapat mengembalikan fungsi.
* Anonymous function atau lambda adalah fungsi yang tidak memiliki nama.
* Closure merupakan suatu fungsi yang dapat mengakses variabel di dalam lexical scope-nya.

Dengan ringkasan tersebut, diharapkan Anda dapat memahami semua materi yang telah disampaikan. Jika belum, Anda bisa ulas kembali materi yang diberikan pada modul ini. Untuk Anda yang sudah merasa mantap, yuk lanjut ke modul berikutnya!