**Pengenalan Collections**

Kita telah mempelajari tentang tipe data dan *control flow*. Lanjut, kini kita akan menghadapi masalah yang lebih kompleks. Untuk bisa memecahkannya secara efisien, kita membutuhkan struktur data yang lebih canggih.

Selain *string*, *number*, dan *boolean*, Dart masih memiliki tipe data lain yang dapat menyimpan banyak data sekaligus yang dalam istilah pemrograman dikenal sebagai ***collections***. *Collections* merupakan sebuah objek yang bisa menyimpan kumpulan objek lain. Contoh *collections* pada Dart antara lain *List*, *Set*, dan *Map*.

## List

List sesuai namanya dapat menampung banyak data ke dalam satu objek. Dalam kehidupan sehari-hari kita menggunakan list untuk menyimpan daftar belanja, nomor telepon, dsb. Begitu pula dengan Dart kita bisa menyimpan bermacam-macam tipe data seperti string, number, dan boolean. Cara penulisannya pun sangat mudah. Perhatikan saja contoh berikut:

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

Kode di atas adalah contoh dari satu objek List yang berisi kumpulan data dengan tipe integer. Karena kompiler bisa mengetahui tipe data yang ada dalam sebuah objek List, maka tak perlu kita menuliskannya secara eksplisit.

1. var numberList = [1, 2, 3, 4, 5];
2. var stringList = ['Hello', 'Dicoding', 'Dart'];

Sesuai contoh di atas, kita mendefinisikan tipe data yang bisa dimasukkan ke dalam List di dalam tanda kurung siku ([ ]). Sama seperti variabel, jika kita tidak mendefinisikan nilai secara eksplisit ke dalam List, maka List akan menyimpan tipe dynamic atau bisa menyimpan semua tipe data.

1. List dynamicList = [1, 'Dicoding', true]; // List<dynamic>

Ketika bermain dengan sebuah List, tentunya ada saat di mana kita ingin mengakses posisi tertentu dari List tersebut. Untuk melakukannya, kita bisa menggunakan fungsi indexing seperti berikut:

1. print(dynamicList[1]);

Perhatikan kode di atas. Fungsi indexing ditandai dengan tanda [ ]. Jika Anda mengira bahwa konsol akan menampilkan angka 1, maka tebakan Anda kurang tepat. Karena dalam sebuah List, indeks dimulai dari 0. Maka ketika kita akan mengakses data pada dynamicList yang berada pada indeks ke-1, artinya data tersebut merupakan data pada posisi ke-2. Jadi data yang akan ditampilkan pada konsol adalah ***Dicoding***.

Lalu apa yang akan terjadi jika kita berusaha menampilkan item dari List yang berada di luar dari ukuran List tersebut? Sebagai contoh, Anda ingin mengakses indeks ke-3 dari dynamicList:

1. print(dynamicList[3]);

Hasilnya adalah eror! Kompiler akan memberitahukan bahwa perintah itu tidak bisa dijalankan. Berikut pesan eror yang akan muncul:

Unhandled exception:  
RangeError (index): Invalid value: Not in range 0..2, inclusive: 3

Pesan di atas memberitahu kita bahwa List telah diakses dengan indeks ilegal. Ini akan terjadi jika indeks yang kita inginkan negatif atau lebih besar dari atau sama dengan ukuran List tersebut.

Masih ingat looping? Untuk menampilkan seluruh item dari list kita bisa memanfaatkan looping. Contohnya perhatikan kode berikut:

1. for(int i = 0; i < stringList.length; i++) {
2. print(stringList[i]);
3. }

Pada kode di atas kita memanfaatkan perulangan sebanyak jumlah data di dalam list untuk mencetak data yang ada di dalam list. Banyaknya data di dalam list bisa kita panggil melalui properti .length.

Selain itu kita juga bisa menggunakan fungsi foreach untuk menampilkan data di dalam list.

1. stringList.forEach((s) => print(s));

Mekanisme di atas dikenal sebagai ***lambda*** atau **anonymous function**. Kita akan mempelajarinya lebih dalam pada modul yang akan datang.

Sejauh ini kita baru belajar menginisialisasikan dan mengakses data dari sebuah List. Pastinya Anda bertanya, “Bagaimana kita memanipulasi data pada List tersebut?” Nah, untuk menambahkan data ke dalam list, kita bisa menggunakan fungsi add().

1. stringList.add('Flutter');

Fungsi add ini akan menambahkan data di akhir list. Sehingga ketika dicetak, konsol akan menampilkan data berikut:

1. [Hello, Dicoding, Dart, Flutter]

Lalu bagaimana jika kita ingin menambahkan data namun tidak di akhir List? Jawabannya adalah dengan fungsi ***insert***. Di dalam fungsi insert kita perlu memasukkan 2 parameter, yaitu indeks list dan data yang akan dimasukkan.

1. stringList.insert(0, 'Programming');
2. // stringList = [Programming, Hello, Dicoding, Dart, Flutter]

Untuk mengubah nilai di dalam list, kita bisa langsung menginisialisasikan nilai baru sesuai indeks yang diinginkan.

1. stringList[1] = 'Application';

Sedangkan untuk menghapus data terdapat beberapa fungsi ***remove*** yang bisa kita gunakan, antara lain:

1. stringList.remove('Programming'); // Menghapus list dengan nilai Programming
2. stringList.removeAt(1);           // Menghapus list pada index ke-1
3. stringList.removeLast();          // Menghapus data list terakhir
4. stringList.removeRange(0, 2);     // Menghapus list mulai index ke-0 sampai ke-1 (indeks 2 masih dipertahankan)

### Spread Operator

Dart memiliki fitur menarik untuk menambahkan banyak nilai ke dalam List dengan cara yang singkat, yaitu spread operator.

Sesuai namanya “spread”, fitur ini digunakan untuk menyebarkan nilai di dalam collections menjadi beberapa elemen. Spread operator dituliskan dengan tiga titik (...). Mari kita lihat contoh kode berikut:

1. var favorites = ['Seafood', 'Salad', 'Nugget', 'Soup'];
2. var others = ['Cake', 'Pie', 'Donut'];
3. var allFavorites = [favorites, others];
4. print(allFavorites);
6. /\* output
7. [[Seafood, Salad, Nugget, Soup], [Cake, Pie, Donut]]
8. \*/

Nilai List tidak akan tergabung, Alih-alih menggabungkan nilainya, variabel allFavorites menjadi List yang menampung dua List di dalamnya. Nah, dengan spread operator kita dapat menggabungkan kedua List di atas:

1. var favorites = ['Seafood', 'Salad', 'Nugget', 'Soup'];
2. var others = ['Cake', 'Pie', 'Donut'];
3. var allFavorites = [...favorites, ...others];
4. print(allFavorites);
6. /\* output
7. [Seafood, Salad, Nugget, Soup, Cake, Pie, Donut]
8. \*/

Yup, dengan spread operator kita dapat memasukkan masing-masing elemen di dalam List ke dalam List lainnya.

Untuk mengatasi List yang bisa bernilai null, kita dapat menggunakan null-aware spread operator (...?) seperti berikut:

1. var list;
2. var list2 = [0, ...?list];
3. print(list2);
5. /\* output
6. [0]
7. \*/

## Set

Selanjutnya kita akan membahas jenis collection yang kedua, yaitu ***Set***. Set merupakan sebuah collection yang hanya dapat menyimpan nilai yang unik. Ini akan berguna ketika Anda tidak ingin ada data yang sama alias duplikasi dalam sebuah collection. Kita bisa mendeklarasikan Set dengan beberapa cara berikut:

1. var numberSet = {1, 4, 6};
2. Set<int> anotherSet = new Set.from([1, 4, 6, 4, 1]);

Perhatikan kode di atas. Di sana terdapat beberapa angka yang duplikat, yaitu angka 1 dan 4. Silakan tampilkan pada konsol dan lihat hasilnya.

1. print(anotherSet);
3. // Output: {1, 4, 6}

Secara otomatis Set akan membuang angka yang sama, sehingga hasilnya adalah {1, 4, 6}.

Untuk menambahkan data ke dalam Set kita dapat memanfaatkan fungsi add() atau addAll().

1. numberSet.add(6);
2. numberSet.addAll({2, 2, 3});

Fungsi add akan menambah satu item ke dalam Set, sementara addAll digunakan untuk menambahkan beberapa item sekaligus. Nilai yang duplikat akan diabaikan.

Lalu gunakan fungsi remove() untuk menghapus objek di dalam set yang diinginkan.

1. numberSet.remove(3);

Kode di atas akan menghapus nilai 3 di dalam Set, bukan indeks ke-3.

Kemudian untuk menampilkan data pada indeks tertentu, gunakanlah fungsi elementAt().

1. var numberSet = {1, 4, 6};
3. numberSet.add(6);
4. numberSet.addAll({2, 2, 3});
5. numberSet.remove(3);
7. print(numberSet.elementAt(2));
9. // Output: 6

Selain itu, Dart juga memiliki fungsi ***union*** dan ***intersection*** untuk mengetahui gabungan dan irisan dari 2 (dua) buah Set. Sebagai contoh:

1. var setA = {1, 2, 4, 5};
2. var setB = {1, 5, 7};
4. var union = setA.union(setB);
5. var intersection = setA.intersection(setB);
7. print(union);
8. print(intersection);
10. // union: {1, 2, 4, 5, 7}
11. // intersection: {1, 5}

## Map

Collection ketiga adalah Map, yakni sebuah collection yang dapat menyimpan data dengan format key-value. Perhatikan contoh berikut:

1. var capital = {
2. 'Jakarta': 'Indonesia',
3. 'London': 'England',
4. 'Tokyo': 'Japan'
5. };

String yang berada pada sebelah kiri titik dua (:) adalah sebuah key, sedangkan yang di sebelah kanan merupakan value-nya. Lalu untuk mengakses nilai dari Map tersebut, kita bisa menggunakan key yang sudah dimasukkan. Misalnya, kita bisa menggunakan key “Jakarta” untuk mendapatkan value “Indonesia”:

1. print(capital['Jakarta']);
3. // Output: Indonesia

Kita dapat menampilkan key apa saja yang ada di dalam Map dengan menggunakan property keys.

1. var mapKeys = capital.keys;
3. // mapKeys = (Jakarta, London, Tokyo)

Sedangkan untuk mengetahui nilai apa saja yang ada di dalam Map kita bisa menggunakan property values.

1. var mapValues = capital.values;
3. // mapValues = (Indonesia, England, Japan)

Untuk menambahkan key-value baru ke dalam Map, kita bisa melakukannya dengan cara berikut:

1. capital['New Delhi'] = 'India';
2. print(capital);
4. // Output: {Jakarta: Indonesia, London: England, Tokyo: Japan, New Delhi: India}

**Rangkuman Materi**

Untuk menyelesaikan masalah yang lebih kompleks secara efisien, kita membutuhkan struktur data yang lebih canggih dibandingkan tipe data biasa. Dart memiliki tipe data lain yang dapat menyimpan banyak data sekaligus, contohnya seperti List, Set, dan Map.

Beberapa hal yang telah kita bahas pada modul ini, antara lain:

* List digunakan untuk menyimpan banyak data. Data pada List disusun secara berurutan dan diakses menggunakan index.
  1. var numberList = [1, 2, 3, 4, 5];
  2. var stringList = ['Hello', 'Dicoding', 'Dart'];
* List menerapkan zero-based indexing, di mana 0 adalah indeks dari nilai pertama dan list.length - 1 adalah indeks dari nilai terakhir.
* Set digunakan untuk menyimpan banyak data secara unik, tidak ada duplikasi, tidak berurutan, dan tidak diindeks.
  1. var numberSet = {1, 4, 6};
* Pada Set memiliki beberapa fungsi yang bisa digunakan, antara lain:
  1. add(value), untuk menambahkan nilai value pada Set.
  2. remove(value), untuk menghapus nilai value pada Set.
  3. union(other), membuat set baru yang berisi gabungan Set ini dan Set other.
  4. intersection(other), membuat set baru yang berisi irisan Set ini dan Set other.
* Map menyimpan banyak data dengan format pasangan key-value
  1. var capital = {
  2. 'Jakarta': 'Indonesia',
  3. 'London': 'England',
  4. 'Tokyo': 'Japan'
  5. };
* Terdapat beberapa properti yang tersedia pada Map, antara lain:
  1. keys, untuk menampilkan seluruh key yang ada di dalam Map.
  2. values, untuk mengetahui seluruh nilai yang ada di dalam Map.
  3. clear(), untuk menghapus seluruh key-value yang ada di dalam Map.
* Saat mengakses key yang tidak ada pada map, nilai variabel akan bernilai null.
  1. print(capital[‘New Delhi’]); // null