**Pengenalan Dart Type System**

Dalam bahasa pemrograman, *type system* adalah sistem logis yang terdiri dari seperangkat aturan yang menetapkan properti atau tipe ke berbagai konstruksi program komputer, seperti variabel, *expression*, fungsi, atau modul. *Type system* ini memformalkan atau memberikan standar kategori tersirat yang digunakan *programmer* untuk tipe data, struktur data, atau komponen lainnya.

Dart menyebut *type system*-nya sebagai ***sound type system***. *Soundness* ini berarti program Anda tidak akan pernah bisa memasuki keadaan di mana sebuah ekspresi mengevaluasi nilai yang tidak cocok dengan jenis tipenya.

*Sound type system* pada Dart ini sama dengan *type system* pada Java atau C#. Di mana kondisi soundness ini dicapai dengan menggunakan kombinasi pemeriksaan statis (*compile-time error*) dan pemeriksaan saat *runtime*. Sebagai contoh, menetapkan String ke variabel int adalah kesalahan *compile-time*. Casting Object ke String dengan as String akan gagal ketika *runtime* jika objek tersebut bukan String.

Manfaat dari *sound type system* ini, antara lain:

* **Mengungkap *bug* terkait tipe pada saat *compile time*.**  
  *Sound type system* memaksa kode untuk tidak ambigu tentang tipenya, sehingga *bug* terkait tipe yang mungkin sulit ditemukan saat *runtime*, bisa ditemukan pada waktu kompilasi.
* **Kode lebih mudah dibaca.**  
  Kode menjadi lebih mudah dibaca karena Anda dapat mengandalkan nilai yang benar-benar memiliki tipe yang ditentukan. Tipe pada Dart tidak bisa berbohong.
* **Kode lebih mudah dikelola.**  
  Ketika Anda mengubah satu bagian kode, *type system* dapat memperingatkan Anda tentang bagian kode mana yang baru saja rusak.
* **Kompilasi *ahead of time* (AOT) yang lebih baik.**  
  Kode yang dihasilkan saat kompilasi AOT menjadi jauh lebih efisien.

## Generic

Jika Anda perhatikan pada dokumentasi collection seperti List, sebenarnya tipe dari List tersebut adalah List<E>. Tanda <...> ini menunjukkan bahwa List adalah tipe generic, tipe yang memiliki tipe parameter. Menurut coding convention dari Dart, tipe parameter dilambangkan dengan satu huruf kapital seperti E, T, K, atau V.

Secara umum generic merupakan konsep yang digunakan untuk menentukan tipe data yang akan kita gunakan. Kita bisa mengganti tipe parameter generic pada Dart dengan tipe yang lebih spesifik dengan menentukan instance dari tipe tersebut.

Sebagai contoh, perhatikan List yang menyimpan beberapa nilai berikut:

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

Tipe parameter yang digunakan pada variabel list di atas adalah int, maka nilai yang bisa kita masukkan adalah nilai dengan tipe int. Begitu juga jika kita menentukan tipe parameter String, maka tipe yang bisa kita masukkan ke dalam list hanya berupa String.

1. List<int> numberList = [1, 2, 3, 4, 5];
2. List<String> stringList = ['Dart', 'Flutter', 'Android', 'iOS'];
3. List dynamicList = [1, 2, 3, 'empat'];  // List<dynamic>

Berbeda jika kita tidak menentukan tipe parameter dari list. List tersebut tidak memiliki tipe yang menjadi acuan bagi kompiler sehingga semua tipe bisa disimpan ke dalam list. Variabel dynamicList di atas sebenarnya masih menerapkan generic dengan tipe dynamic sehingga tipenya menjadi List<dynamic>.

Dari kasus di atas kita bisa simpulkan bahwa Dart membantu kita menghasilkan kode yang type safe dengan membatasi tipe yang bisa digunakan ke dalam suatu objek dan menghindari bug. Selain itu generic juga bermanfaat mengurangi duplikasi kode. Misalnya ketika Anda perlu untuk menyimpan objek cache bertipe String dan int. Alih-alih membuat dua objek StringCache dan IntCache, Anda bisa membuat satu objek saja dengan memanfaatkan tipe parameter dari generic.

1. abstract class Cache<T> {
2. T getByKey(String key);
3. void setByKey(String key, T value);
4. }

Dengan Dart type system kita bisa mengganti tipe parameter yang digunakan sesuai dengan susunan hierarkinya. Perhatikan hierarki objek Animal berikut:

Dengan hierarki di atas, jika kita memiliki objek List<Bird> maka objek apa saja yang bisa kita masukkan ke list tersebut?

1. List<Bird> birdList = [Bird(), Dove(), Duck()];

Seluruh objek Bird atau objek turunannya bisa masuk ke dalam birdList. Namun, ketika menambahkan objek dari Animal, terjadi compile error karena objek Animal belum tentu merupakan objek Bird.

1. List<Bird> birdList = [Bird(), Dove(), Duck(), Animal()];  // Error

Berbeda jika kita mengisi List<Bird> dengan List<Animal> seperti berikut:

1. List<Bird> myBird = List<Animal>();

Kompiler tidak akan menunjukkan eror namun ketika kode dijalankan akan terjadi runtime error karena List<Animal> bukanlah subtype dari List<BIrd>.

1. Unhandled exception:
2. type 'List<Animal>' is not a subtype of type 'List<Bird>'

## Type Inference

Seperti yang kita tahu Dart mendukung type inference. Dart memiliki analyzer yang dapat menentukan menyimpulkan tipe untuk field, method, variabel lokal, dan beberapa tipe argumen generic. Ketika analyzer tidak memiliki informasi yang cukup untuk menyimpulkan tipe tertentu, maka tipe dynamic akan digunakan.

Misalnya berikut ini adalah contoh penulisan variabel map dengan tipe yang eksplisit:

1. Map<String, dynamic> company = {'name': 'Dicoding', 'yearsFounded': 2015};

Atau, Anda dapat menggunakan var dan Dart akan menentukan tipenya.

1. var company = {'name': 'Dicoding', 'yearsFounded': 2015}; // Map<String, Object>

Type inference menentukan tipe dari entri kemudian menentukan tipe dari variabelnya. Pada contoh di atas, kedua key dari map adalah String, namun nilainya memiliki tipe yang berbeda, yaitu String dan int, di mana keduanya merupakan turunan dari Object. Sehingga variabel company akan memiliki tipe Map<String, Object>.

Saat menetapkan nilai objek ke dalam objek lain, kita bisa mengganti tipenya dengan tipe yang berbeda tergantung pada apakah objek tersebut adalah consumer atau producer. Perhatikan assignment berikut:

1. Fish fish = Fish();

Fish fish adalah consumer dan Fish() adalah producer. Pada posisi consumer, aman untuk mengganti consumer bertipe yang spesifik dengan tipe yang lebih umum. Jadi, aman untuk mengganti Fish fish dengan Animal fish karena Animal adalah supertype dari Fish.

1. Animal fish = Fish();

Namun mengganti Fish fish dengan Shark fish melanggar type safety karena bisa saja Fish memiliki subtype lain dengan perilaku berbeda, misalnya FlyingFish.

1. Shark fish = Fish();  // Error

Pada posisi producer, aman untuk mengganti tipe yang umum (supertype) dengan tipe yang lebih spesifik (subtype).

1. Fish fish = Shark();

## Rangkuman Materi

Anda berada di akhir dari modul Dart Type System. Mari kita uraikan materi yang sudah Anda pelajari untuk mempertajam pemahaman.

* Type system merupakan seperangkat aturan yang menetapkan properti atau tipe ke berbagai konstruksi program komputer, seperti variable, expression, fungsi, atau modul.
* Type system ini memberikan standar kategori tersirat yang digunakan programmer untuk tipe data, struktur data, atau komponen lainnya.
* Pada Dart memiliki type system-nya dengan sound type system dengan maksud program akan selalu mengevaluasi nilai yang tidak cocok dengan jenis tipenya.
* Manfaat adanya sound type system pada Dart adalah:
  + Menemukan bug terkait tipe data pada saat compile time.
  + Kode lebih mudah dibaca.
  + Kode lebih mudah dikelola.
  + Kompilasi ahead of time (AOT) yang lebih baik.
* Pengertian Generic secara umum adalah konsep yang digunakan untuk menentukan tipe data yang akan digunakan.
* Manfaat Generic pada Dart adalah:
  + Type safety, maksudnya objek akan membatasi tipe yang bisa digunakan ke dalam objek tersebut.
  + Mengurangi duplikasi kode sehingga dapat mempermudah dalam membuat interface baru.
* Dart mendukung type inference, di mana memiliki analyzer yang dapat menyimpulkan tipe untuk field, method, variabel lokal, dan beberapa tipe argumen generic lainnya.

Mari kita lanjut ke modul berikutnya untuk mempelajari tentang proses asynchronous pada Dart. Let’s go!