**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.

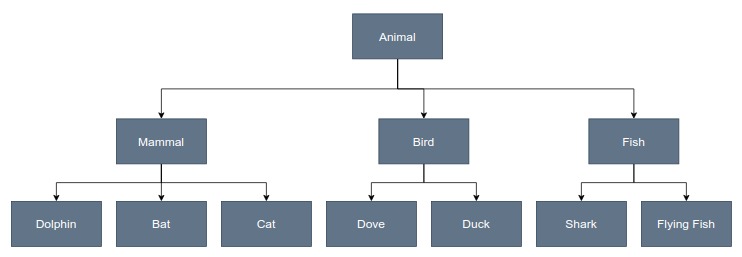
1. List<int> numberList = [1, 2, 3, 4, 5];
2. List<String> stringList = ['Dart', 'Flutter', 'Android', 'iOS'];
3. List dynamicList = [1, 2, 3, 'empat'];  // List<dynamic>

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:

[](https://www.dicoding.com/academies/191/tutorials/7667)

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>'