Ringkasan Aturan Effective Dart

Setelah mempelajari *code convention* dan Effective Dart, pada modul ini kita akan membahas tentang beberapa contoh aturan yang ada di dalam Effective Dart.

DO name types using UpperCamelCase.

Class, enum, typedef, dan type parameter harus menggunakan huruf kapital pada huruf pertama dari setiap kata termasuk kata pertama.

```
1. abstract class Animal {}

2. abstract class Mammal extends Animal {}

3. mixin Flyable {}

4. class Cat extends Mammal with Walkable {}
```

DON’T use prefix letters.

Karena Dart dapat memberitahu Anda tipe, cakupan, dan properti lain pada sebuah deklarasi, maka tidak ada alasan untuk menambahkan prefix pada sebuah identifier.

```
1. var instance;  // good

2. var mInstance;  // bad
```

PREFER starting function or method comments with third-person verbs.

Sebuah komentar dokumentasi harus fokus menjelaskan apa yang dilakukan kode tersebut. Menambahkan kata kerja dari sudut pandang orang ketiga di awal komentar adalah salah satu cara melakukannya.

```
1. // Returns `true` if [username] and [password] inputs are valid.
```

```
2. bool isValid(String username, String password) { }
```

PREFER a noun phrase for a non-boolean property or variable.

Seorang developer yang membaca kode kita akan fokus pada *apa* yang ada pada property. Jika mereka lebih peduli tentang *bagaimana* suatu property ditentukan, maka lebih baik dibuat menjadi method dengan nama menggunakan kata kerja.

```
1. // Good
2. cat.furColor;
3. calculator.firstNumber;
4. list.length;
5.
6. // Bad
7. list.deleteItems;
```

Sementara untuk variabel atau property booleans PREFER gunakan kata kerja *non-imperative*, seperti:

```
1. list.isEmpty
2. dialog.isOpen
```

DO use ?? to convert null to a boolean value.

Aturan ini berlaku ketika sebuah expression dapat mengevaluasi nilai true, false, atau null dan Anda perlu meneruskan hasilnya ke sesuatu yang tidak menerima nilai null.

```
1. if(stock?.isEnough) {
2.   print('Making you a cup of coffee...');
```

```
3. }
```

Kode di atas akan menghasilkan exception ketika stock-nya null. Untuk mengatasinya kita perlu memberikan nilai default ketika nilai awalnya null. Sehingga kodenya akan menjadi seperti berikut:

```
1. stock?.isEnough ?? false;
```

AVOID using curly braces in interpolation when not needed.

Seperti yang kita tahu, Dart dilengkapi dengan fitur *string interpolation* untuk menggabungkan nilai string dengan nilai lain secara lebih mudah.

```
1. print('Hi, ${name}, You are ${thisYear - birthYear} years old.');
```

Namun jika Anda menginterpolasi identifier sederhana, maka curly braces ({}) tidak perlu ditulis.

```
1. print('Hi, $name, You are ${thisYear - birthYear} years old.');
```

PREFER async/await over using raw futures.

Kode *asynchronous* bisa jadi sangat sulit untuk dibaca dan di-*debug*, bahkan ketika menggunakan abstraksi yang bagus seperti Future. *Sintaks* async-await memungkinkan Anda menuliskan kode *asynchronous* dengan gaya *synchronous* sehingga memudahkan membaca kode.

```
1. // raw future
2. void main() {
3.   getOrder().then((value) {
4.     print('You order: $value');
5.   })
6.   .catchError((error) {
```

```
7.     print('Sorry. $error');
8.   });
9.   print('Getting your order...');
10.  }
11.
12.
13.  // async-await
14.  void main() async {
15.    print('Getting your order...');
16.    try {
17.      var order = await getOrder();
18.      print('You order: $order');
19.    } catch (error) {
20.      print('Sorry. $error');
21.    }
22.  }
```

CONSIDER making the code read like a sentence.

Penamaan dalam kode baik itu nama variabel, fungsi, maupun lainnya adalah hal yang sangat penting namun juga tidak mudah. Sebagai solusi kita bisa membayangkannya seolah sedang membuat kalimat.

```
1. // “If store is open ...”
2. If (store.isOpen)
3.
4. // “hey garfield, eat!”
```

```
5. garfield.eat();
6.
7. // Bad example
8. // Ambigu apakah memerintahkan toko untuk tutup atau mendapatkan statu
   s dari toko
9. If (store.close)
```

CONSIDER using function type syntax for parameters.

Dart memiliki *sintaks* khusus untuk mendefinisikan parameter yang tipenya adalah fungsi. Caranya yaitu dengan menuliskan tipe kembalian sebelum nama parameter kemudian parameter dari fungsi setelahnya.

```
1. List filter(bool predicate(item)) { }
```

Sejak Dart versi 2, terdapat notasi umum untuk tipe fungsi sehingga bisa digunakan untuk parameter berupa fungsi.

```
1. List filter(Function predicate) { }  // function type syntax
```

Sebenarnya beberapa aturan di atas hanyalah sebagian dari seluruh aturan yang ada dalam Effective Dart. Selengkapnya Anda dapat mempelajari panduan dan aturan Effective Dart ini pada tautan berikut:

- Effective Dart: Style