**Exceptions**

Sebuah aplikasi yang sudah berjalan mungkin mengalami eror dan *crash*. Kondisi eror pada saat aplikasi berjalan (*runtime*) ini dikenal dengan ***exception***. Ketika *exception* terjadi aplikasi akan dihentikan dan program setelahnya juga tidak akan dieksekusi.

Salah satu contoh *exception* yang mungkin terjadi adalah pada aplikasi kalkulator. Di mana menurut prinsip matematika kita tidak bisa membagi bilangan lain dengan bilangan nol (0).

1. var a = 7;
2. var b = 0;
3. print(a ~/ b);

Coba jalankan kode di atas. Anda akan mendapatkan eror seperti berikut:

Unhandled exception:  
IntegerDivisionByZeroException  
#0 int.~/ (dart:core-patch/integers.dart:24:7)  
#1 main (file:///home/dicoding/Playground/dart/Belajar%20Dart/bin/main.dart:24:11)  
#2 \_startIsolate.<anonymous closure> (dart:isolate-patch/isolate\_patch.dart:305:32)  
#3 \_RawReceivePortImpl.\_handleMessage (dart:isolate-patch/isolate\_patch.dart:174:12)

Sebagai *developer*, tentunya menjadi tugas kita untuk memastikan aplikasi tetap berjalan bagaimana pun kondisinya, termasuk mengatasi ketika terjadi *exception*. Lantas bagaimana?

Untuk menangani *exception*, mari gunakan ***try*** dan ***catch***. Pindahkan kode yang berpeluang menimbulkan *exception* ke dalam blok try.

1. try {
2. var a = 7;
3. var b = 0;
4. print(a ~/ b);
5. }

Kode di atas masih belum lengkap karena di dalam sebuah blok try ada kode yang diasumsikan berpeluang menjadi *exception* sehingga perlu ditangani. Pada eror sebelumnya telah diketahui bahwa *exception* yang terjadi adalah IntegerDivisionByZeroException. Sehingga kita bisa memanfaatkan *keyword* on untuk mengatasi ketika *exception* tersebut terjadi.

1. try {
2. var a = 7;
3. var b = 0;
4. print(a ~/ b);
5. } on IntegerDivisionByZeroException {
6. print('Can not divide by zero.');
7. }

Nah, kita telah berhasil membuat program kita aman dari *crash* ketika berusaha membagi bilangan dengan nol. Namun bagaimana dengan *exception* lainnya yang belum kita ketahui? Pada Dart sendiri telah tersedia beberapa *exception* seperti IOException, FormatException, dsb. Untuk menangani *exception* yang tidak diketahui secara spesifik, kita bisa menggunakan *keyword* catch setelah blok try.

1. try {
2. var a = 7;
3. var b = 0;
4. print(a ~/ b);
5. } catch(e) {
6. print('Exception happened: $e');
7. }

Kode *catch* membutuhkan satu parameter yang merupakan objek *exception*. Kita dapat mencetak *exception* tersebut ke layar untuk menampilkan *exception* apa yang terjadi. Output kode di atas adalah:

1. Exception happened: IntegerDivisionByZeroException

Selain itu, kita juga dapat menambahkan satu parameter lagi di dalam catch yang merupakan objek *stack trace*. Dari *stack trace* ini kita bisa melihat detail *exception* dan di baris mana *exception* tersebut terjadi.

1. try {
2. var a = 7;
3. var b = 0;
4. print(a ~/ b);
5. } catch(e, s) {
6. print('Exception happened: $e');
7. print('Stack trace: $s');
8. }
10. /\*
11. Output :
12. Exception happened: IntegerDivisionByZeroException
13. Stack trace: #0      int.~/ (dart:core-patch/integers.dart:24:7)
14. #1      main (file:///home/dicoding/Playground/dart/Belajar%20Dart/bin/main.dart:25:13)
15. #2      \_startIsolate.<anonymous closure> (dart:isolate-patch/isolate\_patch.dart:305:32)
16. #3      \_RawReceivePortImpl.\_handleMessage (dart:isolate-patch/isolate\_patch.dart:174:12)
17. \*/

Blok catch dapat digabungkan dengan on. Catch akan menangkap apabila tidak ada exception yang memenuhi blok on yang terpasang.

### Finally

Selain try, on, dan catch, ada satu blok lagi yang ada dalam mekanisme exception handling, yaitu finally. Blok finally akan tetap dijalankan tanpa peduli apa pun hasil yang terjadi pada blok try-catch.

1. try {
2. var a = 7;
3. var b = 0;
4. print(a ~/ b);
5. } catch(e, s) {
6. print('Exception happened: $e');
7. print('Stack trace: $s');
8. } finally {
9. print('This line still executed');
10. }