

Excepciones

Pruebas



Yosafat Coronel
GDSC ESCOM IPN
GitHub: YosafatM



Developer Student Clubs
ESCOM - IPN

Itinerario

- Conceptos
 - Interfaz
 - Call Stack
 - Isolate
- Tipos de fallas
 - Error
 - Excepción
- Try-catch
 - Keywords



Conceptos



¿Qué es una interfaz?

¿Y las interfaces?

Sobre la pila de llamadas

Sobre el Isolate

Interfaz

Un interfaz sería el elemento que **permite** que dos (o más) partes se **comuniquen**. El **Command Line Interface (CLI)**, por ejemplo, permite comunicarnos con la computadora por comandos.

Notemos que las interfaces suelen ser **vagas en su definición**, esto es útil para que las partes a comunicar **implementen** cómo se hará la comunicación.

Interfaz

En Dart, TODAS las clases tienen una interfaz (llamada implícita), no podemos hacer interfaces sin definir una clase.

```
abstract class Exception {  
  factory Exception([var message]) => _Exception(message);  
}
```

Exception es usada como interfaz, comunicará al programador con todas las excepciones que existan. La parte de **abstract** y **factory** lo veremos la siguiente clase.

Interfaz

En Dart, TODAS las clases tienen una interfaz (llamada implícita), no podemos hacer interfaces sin definir una clase.

```
abstract class Exception {  
  factory Exception([var message]) => _Exception(message);  
}
```

Exception es usada como interfaz, comunicará al programador con todas las excepciones que existan. La parte de **abstract** y **factory** lo veremos la siguiente clase.

Interfaz

Para implementar una interfaz, se indica con **implements**, no cuenta como heredar, así que podemos hacer **ambas** a la vez.

```
/// Default implementation of [Exception] which carries a message.  
class _Exception implements Exception {  
  final dynamic message;  
  
  _Exception([this.message]);  
  
  String toString() {  
    Object? message = this.message;  
    if (message == null) return "Exception";  
    return "Exception: $message";  
  }  
}
```

Stack

Dart ordena las llamadas a funciones en una pila de llamadas, con nosotros, la primera función siempre es main.

```
function one(){  
  two();  
}  
  
function two() {  
  three();  
}  
  
function three() {  
  console.trace("Call  
  Stack");  
}
```

Call Stack

The diagram illustrates the state of the call stack at six different points (1 through 6) during the execution of the provided Dart code. Each point is represented by a vertical rectangle divided into sections representing function frames:

- Point 1:** Shows a single frame for the `one()` function.
- Point 2:** Shows frames for `two()` and `one()` stacked vertically.
- Point 3:** Shows frames for `three()`, `two()`, and `one()` stacked vertically.
- Point 4:** Shows frames for `two()`, `one()`, and `one()` stacked vertically. Note that the innermost frame is labeled `one()` again, likely a typo or a specific implementation detail.
- Point 5:** Shows frames for `one()` and `one()` stacked vertically.
- Point 6:** Shows an empty stack.

<https://images.app.goo.gl/1JrfoXYeX8W5hX79A>

Stack

Las excepciones y los errores (los que están más detallados), **guardan de qué parte se generaron**, por eso es que Dart muchas veces nos indica la línea de código.

El Stack **no es infinito**, y si llamamos demasiadas funciones, el Stack se desborda, lo que nos da el **error de Stack Overflow**.

https://youtu.be/vl_AaCgudcY

Isolate

Dart tiene el principio de mantener las cosas simples (KIS), decidieron hacer Dart con un hilo principal (que corre el event loop) y un espacio de memoria privado. A estas dos cosas juntas se le conoce como un Isolate.

La idea es reducir considerablemente la complejidad, sin embargo, podemos hacer Isolates, solo que es privado su espacio, por lo que, se deben comunicar por mensajes entre ellos.

Tipos



¿Qué es un error?
¿Qué es una excepción?
Ejemplos

Error

Un error es un fallo del programa que el desarrollador **debió evitar**, por ejemplo, intentar acceder un método que no existe.

```
void main() {  
    var nombre = "Ay, hola";  
    print(nombre[10]);  
}
```

*Salida: Uncaught Error:
RangeError (index): Index out of
range: index should be less than
8: 10*

Excepc ión

```
try {  
    breedMoreLlamas();  
} on OutOfLlamasException {  
    buyMoreLlamas();  
}
```

Una excepción también es un fallo, pero se espera que sea manejado, ya que pueden crearse por el usuario final:

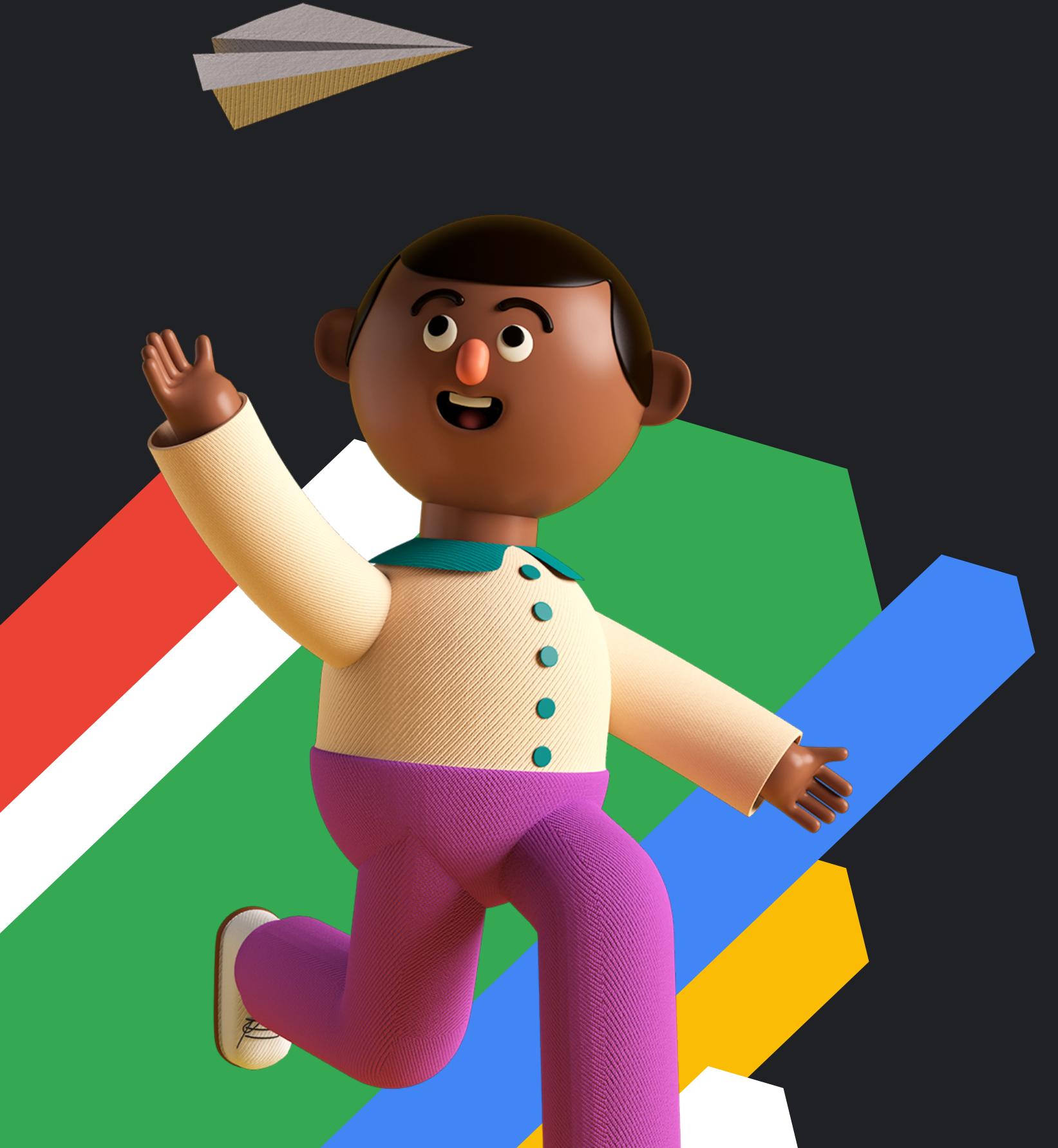
Cuando ya no tenemos llamas, se hace la excepción OutOfLlamas, pero la manejamos comprando más

Fallas

Cuando sucede un fallo, Dart (o nosotros mismos) 'lanzamos' una excepción/error. Dart se encarga de ver el **Call Stack** y ponerle origen, como la línea en la que ocurrió o la función.

Luego de eso, si no se maneja la excepción el Isolate es detenido, si da error, **automáticamente detiene el Isolate**, y con eso el programa, (eso quiere decir que la aplicación en Flutter se muere).

try-catch



Estructura del try-catch
Multiple catch
Finally keyword
Throw
Rethrow

try

```
try {  
    breedMoreLlamas();  
} on OutOfLlamasException {  
    buyMoreLlamas();  
}
```

El try-catch es la estructura para 'atrapar' las excepciones 'lanzadas'. Imaginando que la excepción es una pelota. Y nosotros ambos jugadores.

try contiene un bloque en el que esperamos o preveemos que lance una excepción

try-ca

tch

try <bloque>

[on <types>] [catch

<id>] <bloque>]*

*Estructura del
try-catch*

[finally <bloque>]

throw

Throw nos sirve para 'alzar' o 'lanzar' excepciones, aunque también podemos lanzar objetos arbitrarios (lo cual no es recomendable):

```
throw 'Out of llamas!';
```

```
throw FormatException('Expected at least 1 section');
```

Con esto, Dart se entera de que esto lo debe tratar como un error/excepción, para que lo manejen en las llamadas del Call Stack, si no se hace, el Isolate se detiene

on

On se refiere a los conjuntos, 'sobre la intersección', aunque solo está permitido poner un tipo por on, se pueden poner múltiples on.

```
void main() {  
  try {  
    throw FormatException('Houston');  
  } on FormatException {  
    print('Tenemos un problema');  
  }  
}
```

Imprime Tenemos un problema, pero no en rojo

catch

Catch sirve para ponerle un identificador a la excepción, con esto podemos acceder al contenido de la variable (porque también es un objeto):

```
try {  
    breedMoreLlamas();  
} on OutOfLlamasException {  
    // A specific exception  
    buyMoreLlamas();  
} on Exception catch (e) {  
    // Anything else that is an exception  
    print('Unknown exception: $e');  
} catch (e) {  
    // No specified type, handles all  
    print('Something really unknown: $e');  
}
```

Los catch se pueden usar con los on

catch

Catch también puede traer el **StackTrace**, que contiene información de la **Call Stack**, como la última llamada que se registró, solamente hay que poner otro identificador en el catch:

```
try {  
    // ...  
} on Exception catch (e) {  
    print('Exception details:\n $e');  
} catch (e, s) {  
    print('Exception details:\n $e');  
    print('Stack trace:\n $s');  
}
```

Los catch se pueden usar con los on

finally

Finally es un bloque opcional del try, su bloque se ejecuta se haya atrapado o no alguna excepción. Es útil para la 'limpieza' de lo que abrimos (como cerrar algo):

```
try {  
    breedMoreLlamas();  
} finally {  
    // Always clean up, even if an exception is thrown.  
    cleanLlamaStalls();  
}
```

*Siempre llama al
cleanLlamaStalls,
incluso si se lanzó una
excepción*

NOTA

El `catch` debe ir
acompañado de al menos
uno de los bloques
opcionales que vimos

rethrow

Cuando
excepción ya fue
atrapada, podemos
usar **rethrow** para
volverla a pasar en la
Call Stack

```
void misbehave() {  
    try {  
        dynamic foo = true;  
        print(foo++); // Runtime error  
    } catch (e) {  
        print('misbehave() partially handled ${e.runtimeType}.');  
        rethrow; // Allow callers to see the exception.  
    }  
}  
  
void main() {  
    try {  
        misbehave();  
    } catch (e) {  
        print('main() finished handling ${e.runtimeType}.');  
    }  
}
```

*La excepción se maneja parcialmente en
misbehave y se termina de manejar en main*

**¡Muchísimas
gracias!**

