



Exception Handling in Dart

Module 5 , Chapter 1

Introduction

Exceptions are unexpected events (like wrong input, missing files, or divide-by-zero) that interrupt program flow.

Exception Handling ensures your program doesn't crash — it helps handle errors gracefully.

Benefits:

- Prevents crashes
- Easier debugging
- Better user experience
- More reliable and maintainable code

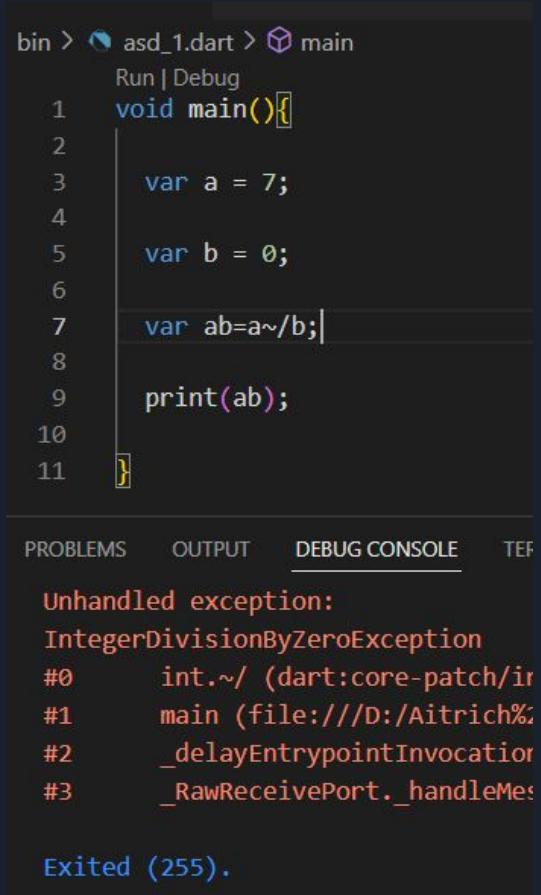


What is an Exception?

Example: dividing $a = 7$ by $b = 0 \rightarrow$ causes an error (“Division by zero”).

Such runtime problems are called exceptions.

Handling these is called exception handling.



The screenshot shows a code editor with a Dart file named 'asd_1.dart'. The code attempts to divide variable 'a' (which is 7) by variable 'b' (which is 0). The code is as follows:

```
bin > asd_1.dart > main
Run | Debug
void main(){
  var a = 7;
  var b = 0;
  var ab=a~/b;
  print(ab);
}
```

Below the code editor is a terminal window showing the output of the program. It displays an unhandled exception: IntegerDivisionByZeroException, followed by a stack trace with three frames, and finally the message 'Exited (255)'.

PROBLEMS	OUTPUT	DEBUG CONSOLE	TER

```
Unhandled exception:
IntegerDivisionByZeroException
#0      int.~/ (dart:core-patch/interned_string_patch.dart:10:7)
#1      main (file:///D:/Aitrich%20-%20OneDrive/Desktop/asd_1.dart:7:5)
#2      _delayEntrypointInvocation (dart:async/zone.dart:148:13)
#3      _RawReceivePort._handleMessage (dart:io/polling_io_backend.dart:44:5)

Exited (255).
```

Handling Exceptions in Dart

(try, catch, on, finally)

try: contains code that might cause an exception.

on: handles a specific type of exception.

```
Run | Debug
1 void main() [
2   // --- Case 1: Using 'on' when you know the exception type ---
3   print("CASE 1:");
4   try {
5     int result = 10 ~/ 0;
6     print("The result is $result");
7   } on IntegerDivisionByZeroException {
8     print("Cannot divided by zero");
9   }
10  print("---");
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS

CASE 1:
Cannot divided by zero

catch: handles unknown exceptions.

finally: runs always, whether an exception occurs or not.

Example cases:

Case 1: You know the exception → use on.

Case 2: You don't know the exception → use catch.

Case 3: Always want code to run → use finally.

```
--  
12 // --- Case 2: Using 'catch' when you don't know  
13 // the exception or for any exception ---  
14 print("CASE 2:");  
15 try {  
16     int result = 10 ~/ 0;  
17     print("The result is $result");  
18 } catch (e) {  
19     print("The exception throw is $e");  
20 }  
21 print("----");  
22  
23 // --- Case 3: Using 'finally' for cleanup code  
24 // (runs whether an exception is thrown or not) ---  
25 print("CASE 3:");  
26 try {  
27     int result = 10 ~/ 0;  
28     print("The result is $result");  
29 } catch (e) {  
30     print("The exception throw is $e");  
31 } finally {  
32     print("This is finally clause");  
33 }  
34 }
```

```
CASE 2:  
The exception throw is IntegerDivisionByZeroException  
----  
CASE 3:  
The exception throw is IntegerDivisionByZeroException  
This is finally clause
```

Exited.

No.	Exception	Description
1	FormatException	Data cannot be parsed due to incorrect format (e.g., text instead of number).
2	IntegerDivisionByZeroException	Occurs during integer division (~) by zero.
3	RangeError	Index or value is outside a valid, specific range (e.g., list index out of bounds).
4	TimeoutException	An asynchronous operation (Future) failed to complete before its deadline.
5	DeferredLoadException	A lazily loaded library failed to load at runtime.
6	IOException	Base class for exceptions related to file or network Input/Output operations.

Custom Exceptions

- Created when built-in exceptions don't cover your specific case.
Example: detecting device overheating.
- Uses these keywords:
 - class: defines a new exception type.
 - constructor: allocates memory for the object.
 - implements: defines how a class fulfills an interface.
 - throw: used to manually raise an exception.

```
Run | Debug
1 void main(){
2
3     try{
4         | throEx();
5     }
6     on CustomEx catch(e){
7         | print(e.a);
8     }
9 }
10
11 void throEx(){
12     | throw CustomEx('Custom Exception');
13 }
14
15 class CustomEx implements Exception{
16     | String a;
17     | CustomEx(this.a);
18 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Custom Exception

Exited.

Exercise

PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL

format is incorrect
Exited.

```
String x='ai';  
| int y=6;
```

PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL

IntegerDivisionByZeroException occurred!
Exited.

```
String x='5';  
| int y=0;
```

PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL

2
result less than 5
Exited.

```
String x='6';  
| int y=3;
```

```
Run | Debug  
void main(){  
    String x='123';  
    {  
        int y=5;  
        try{  
            int xx = int.parse(x);  
            var result=xx~/y;  
            print(result);  
            if(result<5){  
                throwExcep();  
            }  
        }  
        on CustomExcep catch (e){  
            print(e.a);  
        }  
        on FormatException{  
            print("format is incorrect");  
        }  
        on IntegerDivisionByZeroException{  
            print("IntegerDivisionByZeroException occurred!");  
        }  
    }  
    void throwExcep(){  
    {  
        throw CustomExcep('result less than 5');  
    }  
    }  
    class CustomExcep implements Exception{  
        String a;  
        CustomExcep(this.a);  
    }  
}
```

PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL PORTS

24

Exited.