Exception in Dart:

An **exception** is an error occurred at runtime because Dart runtime could not execute a statement successfully or any other thousand reasons. When an exception is thrown, there is a good chance that our program will crash.
Some of the exception in dart are FormatException, IntegerDivisionbyZeroException, IOException, Timeout etc.

We can handle exception in dart by using:

- Try
- On
- Catch
- Finally

To save our program from crashing, we need to catch an exception. This is done using `try/catch/finally` block you might already be familiar with. Let's take a classical example of dividing a number by zero.

```dart
void main() {
  int result = 12 ~/ 0;
  print('The result is $result');
}
```

This program will throw an exception because we cannot divide a number by zero. It will abnormally terminate your program by showing error:

```
shyamkhatiwada@Shyams-MacBook-Pro dart % dart "/Users/shyamkhatiwada/workspace/dart/bin/test.dart"
Unhandled exception:
IntegerDivisionByZeroException
#0      int.~/ (dart:core-patch/integers.dart:30:7)
#1      main (file:///Users/shyamkhatiwada/workspace/dart/bin/test.dart:2:19)
#2      _delayEntrypointInvocation.<anonymous closure> (dart:isolate-patch/isolate_patch.dart:283:19)
#3      _RawReceivePortImpl._handleMessage (dart:isolate-patch/isolate_patch.dart:184:12)
```

In order to avoid such problem, we must handle the exception by using above keywords:

We can handle the above exception by different ways which is shown below:

Case 1: using on keyword

- On keyword is used, if you know the exception that is going to be generated.

```dart
Run | Debug
void main() {
  // case 1: using on keyword
  try {
    int result = 12 ~/ 0;
    print('The result is $result');
  } on IntegerDivisionByZeroException {
    print("cannot divide by Zero");
  }
}
```

Case 2: using catch keyword

- Catch is used, if you don't know the exception that is going to be generated.

```dart
Run | Debug
void main() {
  // case 2: using catch keyword
  try {
    int result = 12 ~/ 0;
    print('The result is $result');
  } catch(e) {
    print("Eception occured: $e");
  }
}
```

Case 3: catch clause with exception object and Stack Trace object.

- Stack trace is used to know the events occurred before exception was thrown.

```dart
Run | Debug
void main() {
  // case 3: catch keyword with exception and stacktrace object
  try {
    int result = 12 ~/ 0;
    print('The result is $result');
  } catch (e, s) {
    print("Eception occured: $e");
    print("StackTrace \n $s");
  }
}
```

Here we will get the events occurred before exception in the output:

```
cannot divide by Zero
shyamkhatiwada@Shyams-MacBook-Pro dart % dart "/Users/shyamkhatiwada/workspace/dart/bin/test.dart"
Eception occured: IntegerDivisionByZeroException
StackTrace
 #0      int.~/ (dart:core-patch/integers.dart:30:7)
#1      main (file:///Users/shyamkhatiwada/workspace/dart/bin/test.dart:4:21)
#2      _delayEntrypointInvocation.<anonymous closure> (dart:isolate-patch/isolate_patch.dart:283:19)
#3      _RawReceivePortImpl._handleMessage (dart:isolate-patch/isolate_patch.dart:184:12)
```

Case 4: finally, keyword

- finally, is used for clean-up code.
- i.e., to close the resources- to close the file, close the database connection.
- finally, block is always executed whether the exception occurred or not.

```dart
Run | Debug
void main() {
  // case 3: catch keyword with exception and stacktrace object
  try {
    int result = 12 ~/ 0;
    print('The result is $result');
  } catch (e) {
    print("Eception occured: $e");
  } finally {
    print("This is finally block and is always executed");
  }
}
```