EXCEPTION HANDLING IN DART

Exception In Dart

An exception is an error that occurs at runtime during program execution. When the exception occurs, the flow of the program is interrupted, and the program terminates abnormally. There is a high chance of crashing or terminating the program when an exception occurs. Therefore, to save your program from crashing, you need to catch the exception.

Info

Note: If you are attempting a task that might result in an error, it's a good habit to use the try-catch statement.

Syntax

```
try {
// Your Code Here
  }
catch(ex){
// Exception here
}
```

Try & Catch In Dart

Try You can write the logical code that creates exceptions in the try block.

Catch When you are uncertain about what kind of exception a program produces, then a catch block is used. It is written with a try block to catch the general exception.

Example 1: Try Catch In Dart

In this example, you will see how to handle the exception using the try-catch block.

```
void main() {
   int a = 18;
   int b = 0;
   int res;

   try {
      res = a ~/ b;
      print("Result is $res");
   }
    // It returns the built-in exception related to the occurring
exception
   catch(ex) {
      print(ex);
    }
}
```

Show Output

Run Online

Finally In Dart Try Catch

The **finally** block is always executed whether the exceptions occur or not. It is optional to include the final block, but if it is included, it should be after the try and catch block is over.

On block is used when you know what types of exceptions are produced by the program.

Syntax

```
try {
.....
}
on Exception1 {
....
}
catch Exception2 {
....
}
finally {
// code that should always execute whether an exception or not.
```

```
}
```

Example 2: Finally In Dart Try Catch

In this example, you will see how to handle the exception using the try-catch block with the finally block.

```
void main() {
  int a = 12;
  int b = 0;
  int res;
  try {
    res = a ~/ b;
  } on UnsupportedError {
    print('Cannot divide by zero');
  } catch (ex) {
    print(ex);
  } finally {
    print('Finally block always executed');
  }
}
```

› Show Output

Run Online

Throwing An Exception

The throw keyword is used to raise an exception explicitly. A raised exception should be handled to prevent the program from exiting unexpectedly.

Syntax

```
throw new Exception_name()
```

Example 3: Throwing An Exception

In this example, you will see how to throw an exception using the throw keyword.

```
void main() {
  try {
    check_account(-10);
  } catch (e) {
    print('The account cannot be negative');
  }
}

void check_account(int amount) {
  if (amount < 0) {
    throw new FormatException(); // Raising explanation externally
  }
}
```

Show Output

Run Online

Why Is Exception Handling Needed?

Exceptions provide the means to separate the details of what to do when something out of the ordinary happens from the main logic of a program. Therefore, exceptions must be handled to prevent the application from unexpected termination. Here are some reasons why exception handling is necessary:

- To avoid abnormal termination of the program.
- To avoid an exception caused by logical error.
- To avoid the program from falling apart when an exception occurs.
- To reduce the vulnerability of the program.
- To maintain a good user experience.
- To try providing aid and some debugging in case of an exception.

How To Create Custom Exception In Dart

As you go advance, you need to create your exception; Dart enables you to create your exception.

Syntax

```
class YourExceptionClass implements Exception{
```

```
  // constructors, variables & methods
}
```

Example 4: How to Create & Handle Exception

This program throws an exception when a student's mark is negative. You will understand **implements** in the object-oriented programming section.

```
class MarkException implements Exception {
  String errorMessage() {
    return 'Marks cannot be negative value.';
  }
}

void main() {
  try {
    checkMarks(-20);
  } catch (ex) {
    print(ex.toString());
  }
}

void checkMarks(int marks) {
  if (marks < 0) throw MarkException().errorMessage();
}
```

› Show Output

Run Online

Example 5: How to Create & Handle Exception

This program throws an exception when you find the square root of a negative number.

```
import 'dart:math';

// custom exception class
class NegativeSquareRootException implements Exception {
  @override
  String toString() {
    return 'Sqauare root of negative number is not allowed here.';
  }
}

// get square root of a positive number
```

```
num squareRoot(int i) {
  if (i < 0) {
    // throw `NegativeSquareRootException` exception
    throw NegativeSquareRootException();
  } else {
    return sqrt(i);
  }
}

void main() {
  try {
    var result = squareRoot(-4);

    print("result: $result");
  } on NegativeSquareRootException catch (e) {
    print("Oops, Negative Number: $e");
  } catch (e) {
    print(e);
  } finally {
    print('Job Completed!');
  }
}
```

› Show Output

Run Online

Video

Watch our video on exception handling in Dart.