🧩 Section 4 – Error Handling in Dart

# 🎯 Learning Goals

By the end of this section, students will be able to:

* Use **try**, **catch**, **on**, and **finally** blocks
* Create and throw **custom exceptions**
* Understand and apply **error propagation** (rethrowing and handling in higher scopes)

# 🔹 1. Basic Error Handling with try / catch / finally

## 💡 Example 1: Handling Runtime Errors

void main() {

try {

int result = 10 ~/ 0; // Integer division by zero

print(result);

} catch (e) {

print('An error occurred: $e');

} finally {

print('This block always runs, even if an error occurs.');

}

}

Explanation

* try: code that might throw an error
* catch: handles any exception
* finally: always executes (useful for closing files, releasing resources, etc.)

### 🧩 Exercise 1

Write a program that:

1. Takes two numbers from the user (you can hardcode for simplicity).
2. Divides the first number by the second.
3. Catches any exception (like division by zero) and prints a friendly message.
4. Prints a “Program finished.” message inside a finally block.

# 🔹 2. Using (on) and (catch) for Specific Exception Types

## 💡 Example 2: Handling Specific Exceptions

void main() {

try {

List<int> numbers = [1, 2, 3];

print(numbers[5]); // Out of range

} on RangeError catch (e) {

print('RangeError caught: $e');

} catch (e) {

print('Unknown error: $e');

} finally {

print('Done checking list.');

}

}

🧩 Explanation

* on lets you handle specific types of exceptions.
* You can use both on and catch together to access the exception object.

### 🧩 Exercise 2

1. Create a program that accesses an element of a list using an invalid index.
2. Catch the specific RangeError.
3. Add a general catch block for any unexpected error.
4. Print a closing message in the finally block.

# 🔹 3. Custom Exceptions

## 💡 Example 3: Defining and Throwing a Custom Exception

class InvalidAgeException implements Exception {

final String message;

InvalidAgeException(this.message);

@override

String toString() => 'InvalidAgeException: $message';

}

void checkAge(int age) {

if (age < 18) {

throw InvalidAgeException('Age must be at least 18.');

} else {

print('Access granted.');

}

}

void main() {

try {

checkAge(15);

} on InvalidAgeException catch (e) {

print(e);

} finally {

print('Age verification completed.');

}

}

Explanation

* Custom exceptions are created by implementing Exception.
* You can define meaningful messages and throw them when needed.

### 🧩 Exercise 3

1. Create a custom exception class called NegativeNumberException.
2. Write a function calculateSquare(int number) that throws this exception if the number is negative.
3. Call this function inside a try block and handle the exception gracefully.

# 🔹 4. Error Propagation (rethrow)

## 💡 Example 4: Passing Errors Up the Call Stack

void riskyOperation() {

try {

int result = 10 ~/ 0;

print(result);

} catch (e) {

print('Error caught in riskyOperation: $e');

rethrow; // Pass it up to higher-level function

}

}

void main() {

try {

riskyOperation();

} catch (e) {

print('Handled again in main: $e');

} finally {

print('Program completed.');

}

}

Explanation

* rethrow is used to pass the same exception to higher layers of the program.
* It’s helpful when you want to log or clean up before letting the main function handle the error.

### 🧩 Exercise 4

1. Write two functions: readFile() and main().
2. Inside readFile(), simulate a file read error using throw Exception('File not found').
3. Catch it in readFile() and print a message, then use rethrow.
4. In main(), catch the rethrown error and print “Error handled at top level”.

# 🏁 Summary Checklist

By the end of this section, you can now:  
✅ Use **try**, **catch**, **on**, and **finally**  
✅ Define and throw **custom exceptions**  
✅ Propagate errors using **rethrow**  
✅ Write safe and maintainable Dart code that gracefully handles failures