let error = NSError(

domain: "com.example.myapp",

code: 1001,

userInfo: [NSLocalizedDescriptionKey: "Unknown error occurred."])

enum NetworkError: Error {

case disconnected

case timeout(duration: Double)

case serverError(code: Int)

}

do {

let contents = try readFile(at: "/path/to/file")

print("File contents: \(contents)")

} catch FileError.notFound {

print("File not found.")

} catch {

print("An unknown error occurred.")

}

enum FileError: Error {

case fileNotFound(path: String)

case permissionDenied(path: String)

case unknown(IOError)

}

extension FileError: LocalizedError {

public var errorDescription: String? {

switch self {

case .fileNotFound(let path):

return "Unable to find the file at \(path)."

case .permissionDenied(let path):

return "No permission to access the file at \(path)."

case .unknown(let error):

return "Unknown file error occurred: \(error.localizedDescription)."

}

}

}

func fetchDataWithRetry(url: URL, attempts: Int) throws -> Data {

for i in 1...attempts {

do {

return try Data(contentsOf: url)

} catch {

if i == attempts {

throw error

}

sleep(2) // Wait before retrying

}

}

throw DataError.retryFailed

}

func loadProfileImage() -> UIImage {

do {

return try fetchImage(from: remoteURL)

} catch {

return UIImage(named: "defaultAvatar")!

}

}

func displayFeature() {

do {

let data = try featureData()

renderFeature(with: data)

} catch {

renderBasicFeature()

}

}

func processLargeBatch() {

let checkpoint = loadLastCheckpoint()

for i in checkpoint.index..<batch.count {

do {

try process(batch[i])

saveCheckpoint(i)

} catch {

logError(error)

break

}

}

}

if fileSavedWithErrors {

displayToast("Document saved with errors. Please review the document.")

}

func fetchData(from url: URL) async throws -> Data {

let (data, response) = try await URLSession.shared.data(from: url)

guard let httpResponse = response as? HTTPURLResponse,

httpResponse.statusCode == 200 else {

throw NetworkError.invalidResponse

}

return data

}

enum NetworkError: Error {

case connectionError(String)

case serverError(code: Int, message: String)

case authenticationError(String)

case invalidRequest(String)

case notFound

case timeout

}

func fetchData(from url: URL, completion: @escaping (Result<Data, NetworkError>) -> Void) {

let task = URLSession.shared.dataTask(with: url) { data, response, error in

guard let httpResponse = response as? HTTPURLResponse else {

completion(.failure(.connectionError("No valid response from server")))

return

}

if let error = error {

completion(.failure(.connectionError(error.localizedDescription)))

return

}

guard (200...299).contains(httpResponse.statusCode) else {

completion(.failure(.serverError(code: httpResponse.statusCode, message: HTTPURLResponse.localizedString(forStatusCode: httpResponse.statusCode))))

return

}

guard let data = data else {

completion(.failure(.invalidRequest("No data received")))

return

}

completion(.success(data))

}

task.resume()

}

intPublisher

.catch { error -> Just<Int> in

print("Error occurred: \(error)")

return Just(-1) // Emitting a fallback value

}

.sink(receiveCompletion: { print($0) },

receiveValue: { print($0) })

intPublisher

.replaceError(with: -1)

.sink(receiveValue: { print($0) })

intPublisher

.retry(3)

.sink(receiveCompletion: { print($0) },

receiveValue: { print($0) })

intPublisher

.tryMap { value -> Int in

if value < 0 { throw MyError.negativeNumber }

return value \* 2

}

.catch { error -> Just<Int> in

return Just(-1)

}

.sink(receiveValue: { print($0) })

**os\_log("This is a debug message", log: OSLog.default, type: .debug)**

func fetchDataUsingFallback() {

fetchData { result in

switch result {

case .success(let data):

process(data)

case .failure:

useCachedData()

}

}

}

class CircuitBreaker {

private var failureCount = 0

private var isOpen = false

func call(operation: @escaping () throws -> Void) {

guard !isOpen else {

print("Circuit is open, operation aborted.")

return

}

do {

try operation()

failureCount = 0

} catch {

failureCount += 1

if failureCount > 3 {

isOpen = true

// Optionally set a timer to try closing the circuit later

}

print("Operation failed:", error)

}

}

func reset() {

isOpen = false

failureCount = 0

}

}

// Anti-Pattern

func processFile(name: String) throws -> Data {

guard let file = files[name] else {

throw FileNotFoundError.missing

}

return file

}

// Best Practice

func processFile(name: String) -> Result<Data, Error> {

guard let file = files[name] else {

return .failure(FileNotFoundError.missing)

}

return .success(file)

}

// Anti-Pattern

**func** findUser(id: Int) -> (user: User?, error: NSError?)

// Best Practice

**enum** UserError: Error {

**case** notFound

**case** networkFailure(NetworkError)

}

**func** findUser(id: Int) -> Result<User, UserError>