

# Error Handling

# What is Error

---



There are four main categories of errors:

- Logical errors
- Generated errors
- Compile-time errors
- Runtime errors



Swift 에서 정의하는 심각도에 따른 4가지 유형의 오류

- simple domain error (단순 도메인 오류)
- recoverable (복구 가능한 오류)
- universal error (범용 오류)
- logic failure (논리적 오류)

# Error

---

## 1. Simple Domain Error (단순 도메인 오류)

- 명백하게 실패하도록 되어 있는 연산 또는 추측에 의한 실행 등으로 발생  
예) 1. 숫자가 아닌 문자로부터 정수를 파싱, 2. 빈 배열에서 어떤 요소를 꺼내는 동작 등
- 오류에 대한 자세한 설명이 필요하지 않으며 대개 쉽게 또 즉시 에러를 처리할 수 있음.
- Swift에는 Optional 등을 통해 잘 모델링되어 있음

## 2. Recoverable (복구 가능한 오류)

- 복잡한 연산을 수행하는 도중 실패가 발생할 수 있지만 사전에 미리 오류를 합리적으로 예측할 수 있는 작업  
예) 파일을 읽고 쓰는 작업, 네트워크 연결을 통해 데이터 읽기 등
- iOS에서는 NSError 또는 Error 를 이용하여 처리
- 일반적으로 이런 오류의 무시는 좋지 않으며 위험할 수도 있으므로 오류를 처리하는 코드 작성 권장
- 오류 내용을 유저에게 알려주거나, 다시 해당 오류를 처리하는 코드를 수행하여 처리하는 것이 일반적

## 3. Universal Error (범용적, 보편적 오류)

- 시스템이나 어떤 다른 요인에 의한 오류
- 이론적으로는 복구가 가능하지만, 어느 지점에서 오류가 발생하는 지 예상하기 어려움

## 4. Logic Failure

- Logic 에 대한 오류는 프로그래머의 실수로 발생하는 것으로 프로그램적으로 컨트롤할 수 없는 오류에 해당
- 시스템에서 메시지를 남기고 abort()를 호출하거나 Exception 발생

# Error

---

## 1. Simple Domain Error (단순 도메인 오류)

- 명백하게 실패하도록 되어 있는 연산 또는 추측에 의한 실행 등으로 발생  
예) 1. 숫자가 아닌 문자로부터 정수를 파싱, 2. 빈 배열에서 어떤 요소를 꺼내는 동작 등
- 오류에 대한 자세한 설명이 필요하지 않으며 대개 쉽게 또 즉시 에러를 처리할 수 있음.
- Swift에는 Optional 등을 통해 잘 모델링되어 있음

## 2. Recoverable (복구 가능한 오류)

- 복잡한 연산을 수행하는 도중 실패가 발생할 수 있지만 사전에 미리 오류를 합리적으로 예측할 수 있는 작업  
예) 파일을 읽고 쓰는 작업, 네트워크 연결을 통해 데이터 읽기 등
- iOS에서는 NSError 또는 Error 를 이용하여 처리
- 일반적으로 이런 오류의 무시는 좋지 않으며 위험할 수도 있으므로 오류를 처리하는 코드 작성 권장
- 오류 내용을 유저에게 알려주거나, 다시 해당 오류를 처리하는 코드를 수행하여 처리하는 것이 일반적

## 3. Universal Error (범용적, 보편적 오류)

- 시스템이나 어떤 다른 요인에 의한 오류
- 이론적으로는 복구가 가능하지만, 어느 지점에서 오류가 발생하는 지 예상하기 어려움

## 4. Logic Failure

- Logic 에 대한 오류는 프로그래머의 실수로 발생하는 것으로 프로그램적으로 컨트롤할 수 없는 오류에 해당
- 시스템에서 메시지를 남기고 abort()를 호출하거나 Exception 발생

# Error Handling

---

프로그램 동작 중 예상 가능한 오류가 발생했을 때 이를 감지하고 복구하기 위한 일련의 처리 과정

*Error handling* is the process of responding to and recovering from error conditions in your program. Swift provides first-class support for throwing, catching, propagating, and manipulating recoverable errors at runtime.

# Error Handling

---

write 메서드를 사용할 때 발생할 수 있는 문제는?


```
func write<T>(toFile path: T, atomically useAuxiliaryFile: Bool, encoding enc: String.Encoding) throws where T : StringProtocol
```

```
“Swift”.write(  
    toFile: filePath,  
    atomically: true,  
    encoding: .utf8  
)
```

# Error Handling

---

오류가 발생할 수 있는 함수를 다룰 때는 그것에 대한 처리 필요

**"Swift"**  Call can throw, but it is not marked with 'try' and the error is not handled

```
.write(toFile: filePath,  
      atomically: true,  
      encoding: .utf8)
```

# 4 ways to handle errors

---

- **Propagating Errors Using Throwing Functions**
- **Handling Errors Using Do-Catch**
- **Converting Errors to Optional Values**
- **Disabling Error Propagation**



# Propagating Errors

---

오류에 대한 처리를 코드의 다른 부분에서 처리하도록 `throws` 키워드를 명시적으로 선언  
`throws` 키워드가 없을 때는 오류를 해당 함수 내에서 처리해야 함

```
// This function is not permitted to throw.
```

```
func foo() -> Int { }
```

```
// This functions are permitted to throw.
```

```
func bar() throws -> Int { }
```

```
func foo() throws { }
```

```
func bar(_ callback: () throws -> ()) throws { }
```

# Propagating Errors

---

```
func throwError() throws {  
    try "Swift".write(  
        toFile: filePath,  
        atomically: true,  
        encoding: .utf8  
    )  
}
```

# Distinct types

---

let a: () -> () -> () 인풋이 없고 아웃풋이 없는, 아웃풋들은 클로저들

let b: () throws -> () -> ()

let c: () -> () throws -> ()

let d: () throws -> () throws -> ()

# throws function > non throws function

---

throws는 non throws를 사용할 수 있지만 non throws는 throws를 사용 못함

**// OK**

```
func cannotThrowFunction() -> Int { return 10 }
```

```
func canThrowFunction(_ generator: () throws -> Int) -> Void { }
```

```
canThrowFunction(cannotThrowFunction)
```

**// Error**

```
func canThrowFunction() throws -> Int { return 10 }
```

```
func cannotThrowFunction(_ generator: () -> Int) -> Void { }
```

```
cannotThrowFunction(canThrowFunction)
```

# Java, C++ Exception Handling

## Java [\[ edit \]](#)

Further information: [Java \(programming language\)](#)

```
try {  
    // Normal execution path  
    throw new EmptyStackException();  
} catch (ExampleException ee) {  
    // deal with the ExampleException  
}  
finally {  
    // This optional section is executed upon termination of any of the try or catch  
    // blocks above,  
    // except when System.exit() is called in "try" or "catch" blocks;  
}
```



The Wikibook [Java Programming](#) has a page on the topic of: [Exceptions](#)

in analogy with this C++

```
#include <iostream>  
using namespace std;  
int main()  
{  
    try  
        {throw (int)42;}  
    catch(double e)  
        {cout << "(0," << e << ")" << endl;}  
    catch(int e)  
        {cout << "(1," << e << ")" << endl;}  
}
```

# Handling Errors Using Do-Catch

---

```
do {  
    try expression  
    statements  
} catch pattern 1 {  
    statements  
} catch pattern 2 where condition {  
    statements  
} catch {  
    statements  
}
```

# Catch error

---

```
do {  
    let contents = try String(contentsOfFile: path)  
    print(contents)  
} catch {  
    print(error)  
}
```

# Catch error cases

---

자판기에서 음료수를 구매하려고 할 때 발생할 수 있는 오류는?

```
do {  
    try buyDrink(vendingMachine: vendingMachine)  
    print("Success!")  
} catch VendingMachineError.outOfStock {  
    print("Out of Stock.")  
} catch VendingMachineError.insufficientFunds {  
    print("Please insert an additional coins.")  
} catch {  
    print("Unexpected error: \(error).")  
}
```



# Converting Errors to Optional Values

try? 를 사용하여 do ~ catch 구문 없이 오류 처리 가능  
정상 수행 시 Optional 값 반환, 오류 발생 시 nil 반환

```
func someThrowingFunction() throws -> Int {  
    // ...  
}  
  
let x = try? someThrowingFunction()  
  
let y: Int?    // x와 동일  
do {  
    y = try someThrowingFunction()  
} catch {  
    y = nil  
}
```

# Converting Errors to Optional Values

try? 를 사용하여 do ~ catch 구문 없이 오류 처리 가능  
정상 수행 시 Optional 값 반환, 오류 발생 시 nil 반환

```
func fetchData() -> Data? {  
    if let data = try? fetchDataFromDisk() { return data }  
    if let data = try? fetchDataFromServer() { return data }  
    return nil  
}
```

# Disabling Error Propagation

---

do ~ catch 구문 없이 throws 메서드 처리 가능하지만 오류 발생 시 앱 Crash  
오류가 발생하지 않는다고 확신할 수 있는 경우에만 try! 사용  
e.g. 앱 번들에 함께 제공되는 이미지 로드 등

```
let photo = try! loadImage(atPath: "./Resources/John Appleseed.jpg")
```

# Specifying Cleanup Actions

---

**defer** - 현재 코드 블록이 종료되기 직전에 반드시 실행되어야 하는 코드 등록  
해당 범위가 종료될 때까지 실행을 연기하며 소스 코드에 기록된 순서의 역순으로 동작

```
func processFile(filename: String) throws {  
    if exists(filename) {  
        let file = open(filename)  
        defer {  
            close(file)  
        }  
        while let line = try file.readline() {  
            // Work with the file.  
        }  
        // close(file) is called here, at the end of the scope.  
    }  
}
```

# Error

---

스위프트

```
public protocol Error { }
```

---

오브젝트 C

```
open class NSError : NSObject, NSCopying, NSSecureCoding {  
}  
  
extension NSError : Error {  
}
```

# Define Custom Error

---

```
enum IntParsingError: Error {  
    case overflow  
    case invalidInput(String)  
}
```

```
func parsingInteger(numString: String) throws -> Int {  
    guard let num = Int(numString) else {  
        throw IntParsingError.invalidInput(numString)  
    }  
    guard num <= Int64.max, num >= Int64.min else {  
        throw IntParsingError.overflow  
    }  
    return num  
}
```

# Define Custom Error

---

```
do {  
    let price = try parsingInteger(numString: "$100")  
    print(price)  
} catch IntParsingError.invalidInput(let invalid) {  
    print("Invalid String: '\(invalid)'")  
} catch IntParsingError.overflow {  
    print("Overflow error")  
} catch {  
    print("Other error")  
}
```

# Define Custom Error

---

```
struct XMLParsingError: Error {  
    enum ErrorKind {  
        case invalidCharacter  
        case mismatchedTag  
    }  
  
    let line: Int  
    let column: Int  
    let kind: ErrorKind  
}  
  
func parse(_ source: String) throws -> XMLDoc {  
    // ...  
    throw XMLParsingError(line: 19, column: 5, kind: .mismatchedTag)  
    // ...  
}
```



# XML

---

```
<document type="com.apple.InterfaceBuilder3.CocoaTouch.Storyboard.XIB" version="3.0"
  toolsVersion="14490.70" targetRuntime="iOS.CocoaTouch" propertyAccessControl="none"
  useAutolayout="YES" useTraitCollections="YES" useSafeAreas="YES" colorMatched="YES"
  initialViewController="EAG-df-4FH">
  <device id="retina6_1" orientation="portrait">
    <adaptation id="fullscreen"/>
  </device>
  <dependencies>
    <deployment identifier="iOS"/>
    <plugIn identifier="com.apple.InterfaceBuilder.IBCocoaTouchPlugin" version="14490.49"/>
    <capability name="Safe area layout guides" minToolsVersion="9.0"/>
    <capability name="documents saved in the Xcode 8 format" minToolsVersion="8.0"/>
  </dependencies>
  <scenes>
    <!--SecondVC-->
    <scene sceneID="tne-QT-ifu">
      // ...
      // ...
    </scene>
  </scenes>
</document>
```

# Catch Error Types

---

```
do {  
    throw XMLParsingError(line: 12, column: 15, kind: .mismatchedTag)  
} catch where error is XMLParsingError {  
    // ...  
} catch where error is IntParsingError {  
    // ...  
} catch {  
    // ...  
}
```