에러처리

강사 주영민

예외처리

- 프로그램의 오류 조건에 응답하고 오류 조건에서 복구하는 프로세스입니다

- Swift는 런타임시 복구 가능한 오류를 던지고, 포착하고, 전파하고, 조작하는 기능을 제공합니다.

- 에러는 Error 프로토콜을 준수하는 유형의 값으로 나타냅니다. 실제로 Error프로토콜은 비어 있으나 오류를 처리할수 있는 타입임을 나타냅니다.

열거형의 에러 표현

- 열거형은 에러를 표현하는데 적합합니다.

```
enum VendingMachineError: Error {
    case invalidSelection
    case insufficientFunds(coinsNeeded: Int)
    case outOfStock
}
```

에러발생

- throw 키워드를 통해 에러를 발생 시킵니다.

```
throw VendingMachineError.insufficientFunds(coinsNeeded: 5)
```

에러전달

- 함수의 작성중 에러가 발생할수 있는 함수에는 매개변수 뒤에 throws 키워드를 작성하여 에러가 전달될수 있는 함수를 선언합니다.

```
//에러전달 가능성 함수
func canThrowErrors() throws -> String

//에러전달 가능성이 없는 함수
func cannotThrowErrors() -> String
```

에러처리

- 함수가 에러를 throw하면 프로그램의 흐름이 변경되므로 에러가 발생할 수있는 코드의 위치를 신속하게 식별 할 수 있어야합니다.

- 이 장소를 식별하기 위해 try 나 try?, try!를 사용할수 있습니다.

- 발결된 에러를 처리하기 위해 do-catch 문을 사용해서 에러를 처리 합니다.

do - catch

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

예제

```
let favoriteSnacks = [
    "Alice": "Chips",
    "Bob": "Licorice",
    "Eve": "Pretzels",
]


func buyFavoriteSnack(person: String, vendingMachine: VendingMachine) throws {
    let snackName = favoriteSnacks[person] ?? "Candy Bar"
    try vendingMachine.vend(itemNamed: snackName)
}
```

```
var vendingMachine = VendingMachine()
vendingMachine.coinsDeposited = 8

do {
    try buyFavoriteSnack(person: "Alice", vendingMachine: vendingMachine)
} catch VendingMachineError.invalidSelection {
    print("Invalid Selection.")
} catch VendingMachineError.outOfStock {
    print("Out of Stock.")
} catch VendingMachineError.insufficientFunds(let coinsNeeded) {
    print("Insufficient funds. Please insert an additional \(coinsNeeded) coins.")
}
```

예제

```
class VendingMachine {
    var inventory = [
        "Candy Bar": Item(price: 12, count: 7),
        "Chips": Item(price: 10, count: 4),
        "Pretzels": Item(price: 7, count: 11)
    ]
    var coinsDeposited = 0

    func vend(itemNamed name: String) throws {
        guard let item = inventory[name] else {
            throw VendingMachineError.invalidSelection
        }
        guard item.count > 0 else {
            throw VendingMachineError.outOfStock
        }

        guard item.price <= coinsDeposited else {
            throw VendingMachineError.insufficientFunds(coinsNeeded: item.price - coinsDeposited)
        }

        coinsDeposited -= item.price

        var newItem = item
        newItem.count -= 1
        inventory[name] = newItem

        print("Dispensing \(name)")
    }
}
```

```
struct Item {
    var price: Int
    var count: Int
}

enum VendingMachineError: Error {
    case invalidSelection
    case
insufficientFunds(coinsNeeded: Int)
    case outOfStock
}
```

Converting Errors to Optional Value

```swift
func someThrowingFunction() throws -> Int {
    // ...
}

let x = try? someThrowingFunction()

let y: Int?
do {
    y = try someThrowingFunction()
} catch {
    y = nil
}
```

Specifying Cleanup Actions (후처리)

- 에러에 의해 함수의 문제가 생기더라도 꼭! 해야할 행동이 있다면!!
- defer 구문은 블럭이 어떻게 종료되던 꼭 실행된다는 것을 보장.

```
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.
    }
}
```