Error Handling

Что рассмотрим?

- Представление ошибок
- Проброс ошибок
- Обработка ошибок

Errors Representation

```
struct XMLParsingError: Error {
    enum ErrorKind {
        case invalidCharacter
        case mismatchedTag
        case internalError
    }


    let line: Int
    let column: Int
    let kind: ErrorKind
}

func parse(_ source: String) throws -> XMLDoc {
    // ...
    throw XMLParsingError(line: 19, column: 5, kind: .mismatchedTag)
    // ...
}
```

```
public protocol Error { }
```

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

Errors Representation

```swift
/// Describes an error that provides localized messages describing why
/// an error occurred and provides more information about the error.
public protocol LocalizedError : Error {

    /// A localized message describing what error occurred.
    var errorDescription: String? { get }

    /// A localized message describing the reason for the failure.
    var failureReason: String? { get }

    /// A localized message describing how one might recover from the failure.
    var recoverySuggestion: String? { get }

    /// A localized message providing "help" text if the user requests help.
    var helpAnchor: String? { get }
}
```

Errors Representation

```
/// Describes an error that may be recoverable by presenting several
/// potential recovery options to the user.
public protocol RecoverableError : Error {

    /// Provides a set of possible recovery options to present to the user.
    var recoveryOptions: [String] { get }

    /// Attempt to recover from this error when the user selected the
    /// option at the given index. This routine must call handler and
    /// indicate whether recovery was successful (or not).
    ///
    /// This entry point is used for recovery of errors handled at a
    /// "document" granularity, that do not affect the entire
    /// application.
    func attemptRecovery(optionIndex recoveryOptionIndex: Int, resultHandler
        handler: @escaping (Bool) -> Void)

    /// Attempt to recover from this error when the user selected the
    /// option at the given index. Returns true to indicate
    /// successful recovery, and false otherwise.
    ///
    /// This entry point is used for recovery of errors handled at
    /// the "application" granularity, where nothing else in the
    /// application can proceed until the attempted error recovery
    /// completes.
    func attemptRecovery(optionIndex recoveryOptionIndex: Int) -> Bool
}
```

Errors Representation

```swift
/// Для необходимости вывести пользователю предупреждение
protocol ErrorWithDescription: Error {
    var localizedDescription: String { get }
}

struct SimpleError: ErrorWithDescription {
    var localizedDescription: String { "Simple Error" }
}
```

- NSError

Throwing Errors

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

```
func container<Key>(keyedBy type: Key.Type) throws ->
KeyedDecodingContainer<Key> where Key : CodingKey
```

- Ошибка обязательно должна быть обработана
- Исправление проблемы
- Альтернативный подход
- Информирование пользователя

Handling Errors

- Вернуть Result
- Пробросить ошибку выше по стеку вызовов
- Использовать do catch
- Обработать ошибку как опциональное значение
- Добавить assert, что ошибка никогда не возникнет

Result

```swift
/// A value that represents either a success or a failure, including an
/// associated value in each case.
@frozen public enum Result<Success, Failure> where Failure : Error {

    /// A success, storing a `Success` value.
    case success(Success)

    /// A failure, storing a `Failure` value.
    case failure(Failure)
```

Result

```swift
enum FileError: Error {
    case invalidFilename
    case noPermissions
    case someError
}

func contents(of filename: String) -> Result<String, FileError> {
    guard let url = URL(string: filename) else {
        return .failure(.invalidFilename)
    }

    do {
        let fileContents = try String(contentsOf: url)
        return .success(fileContents)
    } catch let error as NSError {
        switch error.code {
        case -1102:
            return .failure(.noPermissions)
        default:
            return .failure(.someError)
        }
    }
}
```

Практика

```
func months(salary: Double, amount: Double) -> Double {
    amount / salary
}
```

- Улучшить функцию с возвращаемой ошибкой
- Деление на 0
- В некоторых случаях не только возвращать количество месяцев, но и текст предупреждения

Propagating Errors Using Throwing Functions

```
func canThrowErrors() throws -> String

func cannotThrowErrors() -> String
```

Propagating Errors Using Throwing Functions

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

```
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)")
    }
}
```

Propagating Errors Using Throwing Functions

```
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)
}
```

```
struct PurchasedSnack {
    let name: String
    init(name: String, vendingMachine: VendingMachine) throws {
        try vendingMachine.vend(itemNamed: name)
        self.name = name
    }
}
```

Handling Errors Using Do-Catch

```
do {
    try expression
    statements
} catch pattern 1 {
    statements
} catch pattern 2 where condition {
    statements
} catch pattern 3, pattern 4 where condition {
    statements
} catch {
    statements
}
```

Handling Errors Using Do-Catch

```
var vendingMachine = VendingMachine()
vendingMachine.coinsDeposited = 8
do {
    try buyFavoriteSnack(person: "Alice", vendingMachine: vendingMachine)
    print("Success! Yum.")
} 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.")
} catch {
    print("Unexpected error: \(error).")
}
// Prints "Insufficient funds. Please insert an additional 2 coins."
```

Handling Errors Using Do-Catch

```
func nourish(with item: String) throws {
    do {
        try vendingMachine.vend(itemNamed: item)
    } catch is VendingMachineError {
        print("Couldn't buy that from the vending machine.")
    }
}

do {
    try nourish(with: "Beet-Flavored Chips")
} catch {
    print("Unexpected non-vending-machine-related error: \(error)")
}
// Prints "Couldn't buy that from the vending machine."
```

Handling Errors Using Do-Catch

```
func eat(item: String) throws {
    do {
        try vendingMachine.vend(itemNamed: item)
    } catch VendingMachineError.invalidSelection,
  VendingMachineError.insufficientFunds, VendingMachineError.outOfStock {
        print("Invalid selection, out of stock, or not enough money.")
    }
}
```

Converting Errors to Optional Values

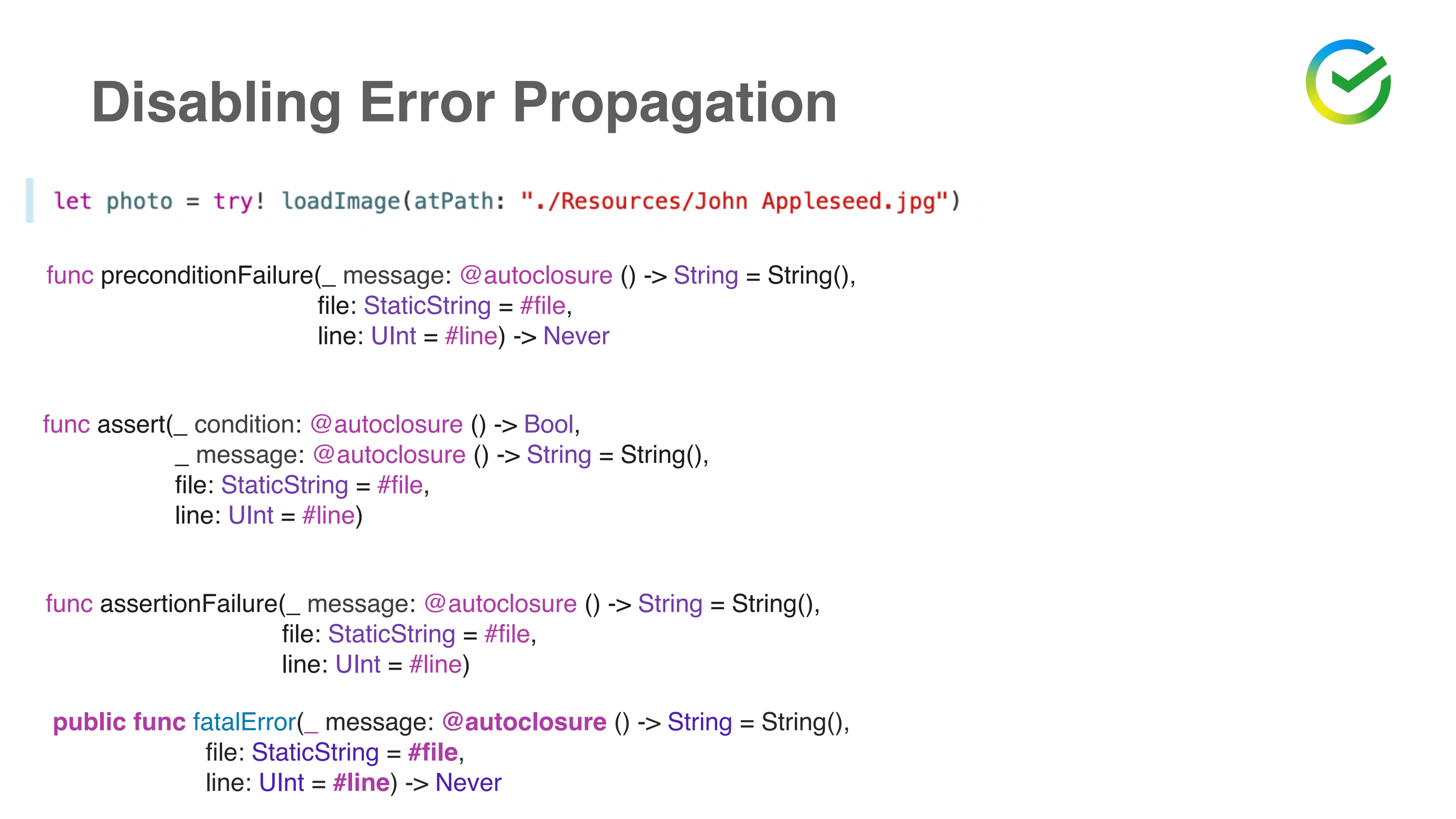
```
func someThrowingFunction() throws -> Int {
    // ...
}

let x = try? someThrowingFunction()

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

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

Disabling Error Propagation



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

```
func preconditionFailure(_ message: @autoclosure () -> String = String(),
                         file: StaticString = #file,
                         line: UInt = #line) -> Never
```

```
func assert(_ condition: @autoclosure () -> Bool,
            _ message: @autoclosure () -> String = String(),
            file: StaticString = #file,
            line: UInt = #line)
```

```
func assertionFailure(_ message: @autoclosure () -> String = String(),
                      file: StaticString = #file,
                      line: UInt = #line)
```

```
public func fatalError(_ message: @autoclosure () -> String = String(),
                       file: StaticString = #file,
                       line: UInt = #line) -> Never
```

Specifying Cleanup Actions

```
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.
    }
}
```

Fun Stuff

```
enum SomeError: Error {
    case blah
}

extension Optional {
    func or(error: Error) throws -> Wrapped {
        switch self {
        case let x?: return x
        case nil: throw error
        }
    }
}


do {
    let _ = try Int("8").or(error: SomeError.blah)
} catch {
    print(error)
}
```

Что почитать

- Error
- Handling Errors
- Patterns
- NSError

Домашнее задание

- Есть массив строк. Но это не обычные строки - это примеры. «32 + 16 = 48». В нём два числа слева от равно, одно число справа. Операции: сложение и вычитание
- Функция checkHomework: принимает массив примеров, и возвращает:
  - Если больше 75% ошибок в примерах - функция возвращает список всех примеров с ошибками и подписью «делай заново»
  - Если пример решён верно - возвращает строку «молодец»
  - Если нет примеров - возвращает строку «нет примеров»
  - Если ей на вход дать хотя бы один невалидный пример - выводит "переделывай" и не оценивает другие примеры
- Функция checkTask: принимает пример
  - Если это не пример - надо кинуть исключение
  - Если пример решён правильно - возвращаем строку с похвалой
  - Если нет - возвращаем ошибку с правильным ответом
- Срок до 12 июля включительно