Chapter 15: Error Handling

Section 15.1: Error handling basics

Functions in Swift may return values, **throw** errors, or both:

```
func reticulateSplines()                // no return value and no error
func reticulateSplines()
```

nt // always returns a value func reticulateSplines() throws // no return value, but may throw an error func reticulateSplines() throws -> *Int* // may either return a value or throw an error

Any value which conforms to the ErrorType protocol (including NSError objects) can be thrown as an error. Enumerations provide a convenient way to define custom errors:

Version ≥ 2.0 Version ≤ 2.2

```
enum NetworkError: ErrorType {
    case Offline
    case ServerError(String)
}
```

Version = 3.0

```
enum NetworkError: Error {
    // Swift 3 dictates that enum cases should be `lowerCamelCase`
    case offline
    case serverError(String)
}
```

An error indicates a non-fatal failure during program execution, and is handled with the specialized control-flow constructs `do`/`catch`, `throw`, and `try`.

```
func fetchResource(resource: NSURL) throws -> String {
    if let (statusCode, responseString) = /* ...from elsewhere...*/ {
        if case 500..<600 = statusCode {
            throw NetworkError.serverError(responseString)
        } else {
            return responseString
        }
    } else {
        throw NetworkError.offline
    }
}
```

Errors can be caught with `do`/`catch`:

```
do {
    let response = try fetchResource(resURL)
    // If fetchResource() didn't throw an error, execution continues here:
    print("Got response: \\(response)")
    ...
} catch {
    // If an error is thrown, we can handle it here.
    print("Whoops, couldn't fetch resource: \\(error)")
}
```

Any function which can throw an error ***must*** be called using `try`, `try?`, or `try!`:

```
// error: call can throw but is not marked with 'try'
```

```
let response = fetchResource(resURL)

// "try" works within do/catch, or within another throwing function:
do {
    let response = try fetchResource(resURL)
} catch {
    // Handle the error
}

func foo() throws {
    // If an error is thrown, continue passing it up to the caller.
    let response = try fetchResource(resURL)
}

// "try?" wraps the function's return value in an Optional (nil if an error was thrown).
if let response = try? fetchResource(resURL) {
    // no error was thrown
}

// "try!" crashes the program at runtime if an error occurs.
let response = try! fetchResource(resURL)
```

Section 15.2: Catching different error types

Let's create our own error type for this example.

Version = 2.2

```
enum CustomError: ErrorType {
    case SomeError
    case AnotherError
}

func throwing() throws {
    throw CustomError.SomeError
}
```

Version = 3.0

```
enum CustomError: Error {
    case someError
    case anotherError
}

func throwing() throws {
    throw CustomError.someError
}
```

The Do-Catch syntax allows to catch a thrown error, and *automatically* creates a constant named `error` available in the `catch` block:

```
do {
    try throwing()
} catch {
    print(error)
}
```

You can also declare a variable yourself:

```
do {
    try throwing()
} catch let oops {
```

```
    print(oops)
}
```

It's also possible to chain different `catch` statements. This is convenient if several types of errors can be thrown in the Do block.

Here the Do-Catch will first attempt to cast the error as a `CustomError`, then as an `NSError` if the custom type was not matched.

Version = 2.2

```
do {
    try somethingMayThrow()
} catch let custom as CustomError {
    print(custom)
} catch let error as NSError {
    print(error)
}
```

Version = 3.0

In Swift 3, no need to explicitly downcast to NSError.

```
do {
    try somethingMayThrow()
} catch let custom as CustomError {
    print(custom)
} catch {
    print(error)
}
```

Section 15.3: Catch and Switch Pattern for Explicit Error Handling

```
class Plane {

    enum Emergency: ErrorType {
        case NoFuel
        case EngineFailure(reason: String)
        case DamagedWing
    }

    var fuelInKilograms: Int

    //... init and other methods not shown

    func fly() throws {
        // ...
        if fuelInKilograms <= 0 {
            // uh oh...
            throw Emergency.NoFuel
        }
    }

}
```

In the client class:

```
let airforceOne = Plane()
do {
```

```
    try airforceOne.fly()
} catch let emergency as Plane.Emergency {
    switch emergency {
    case .NoFuel:
        // call nearest airport for emergency landing
    case .EngineFailure(let reason):
        print(reason) // let the mechanic know the reason
    case .DamagedWing:
        // Assess the damage and determine if the president can make it
    }
}
```

Section 15.4: Disabling Error Propagation

The creators of Swift have put a lot of attention into making the language expressive and error handling is exactly that, expressive. If you try to invoke a function that can throw an error, the function call needs to be preceded by the try keyword. The try keyword isn't magical. All it does, is make the developer aware of the throwing ability of the function.

For example, the following code uses a loadImage(atPath:) function, which loads the image resource at a given path or throws an error if the image can't be loaded. In this case, because the image is shipped with the application, no error will be thrown at runtime, so it is appropriate to disable error propagation.

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

Section 15.5: Create custom Error with localized description

Create `enum` of custom errors

```
enum RegistrationError: Error {
    case invalidEmail
    case invalidPassword
    case invalidPhoneNumber
}
```

Create `extension` of `RegistrationError` to handle the Localized description.

```
extension RegistrationError: LocalizedError {
    public var errorDescription: String? {
        switch self {
        case .invalidEmail:
            return NSLocalizedString("Description of invalid email address", comment: "Invalid
Email")
        case .invalidPassword:
            return NSLocalizedString("Description of invalid password", comment: "Invalid
Password")
        case .invalidPhoneNumber:
            return NSLocalizedString("Description of invalid phoneNumber", comment: "Invalid Phone
Number")
        }
    }
}
```

Handle error:

```
let error: Error = RegistrationError.invalidEmail
```

```
print(error.localizedDescription)
```