Codable

Giftbot

참고 링크

Encoding and Decoding Custom Types
https://goo.gl/cdPsKM

Using JSON with Custom Types
https://developer.apple.com/documentation/foundation/archives_and_serialization/using_json_with_custom_types

What's New in Foundation - WWDC 2017
https://developer.apple.com/videos/play/wwdc2017/212/

try! Swift NYC 2017 - Swift 4 Codable
https://academy.realm.io/posts/try-swift-nyc-2017-yasuhiro-inami-swift-4-codable/

JSON Data Sample
http://json.org/example.html
https://adobe.github.io/Spry/samples/data_region/JSONDataSetSample.html

Swift 4 Decodable: Beyond The Basics
https://medium.com/swiftly-swift/swift-4-decodable-beyond-the-basics-990cc48b7375

Everything about Codable in Swift 4
https://hackernoon.com/everything-about-codable-in-swift-4-97d0e18a2999

Codable

A type that can convert itself into and out of an external representation.

```
public typealias Codable = Decodable & Encodable

public protocol Encodable {

    /// Encodes this value into the given encoder.
    /// - Parameter encoder: The encoder to write data to.
    public func encode(to encoder: Encoder) throws
}

public protocol Decodable {

    /// Creates a new instance by decoding from the given decoder.
    /// - Parameter decoder: The decoder to read data from.
    public init(from decoder: Decoder) throws
}
```

Encoding & Decoding

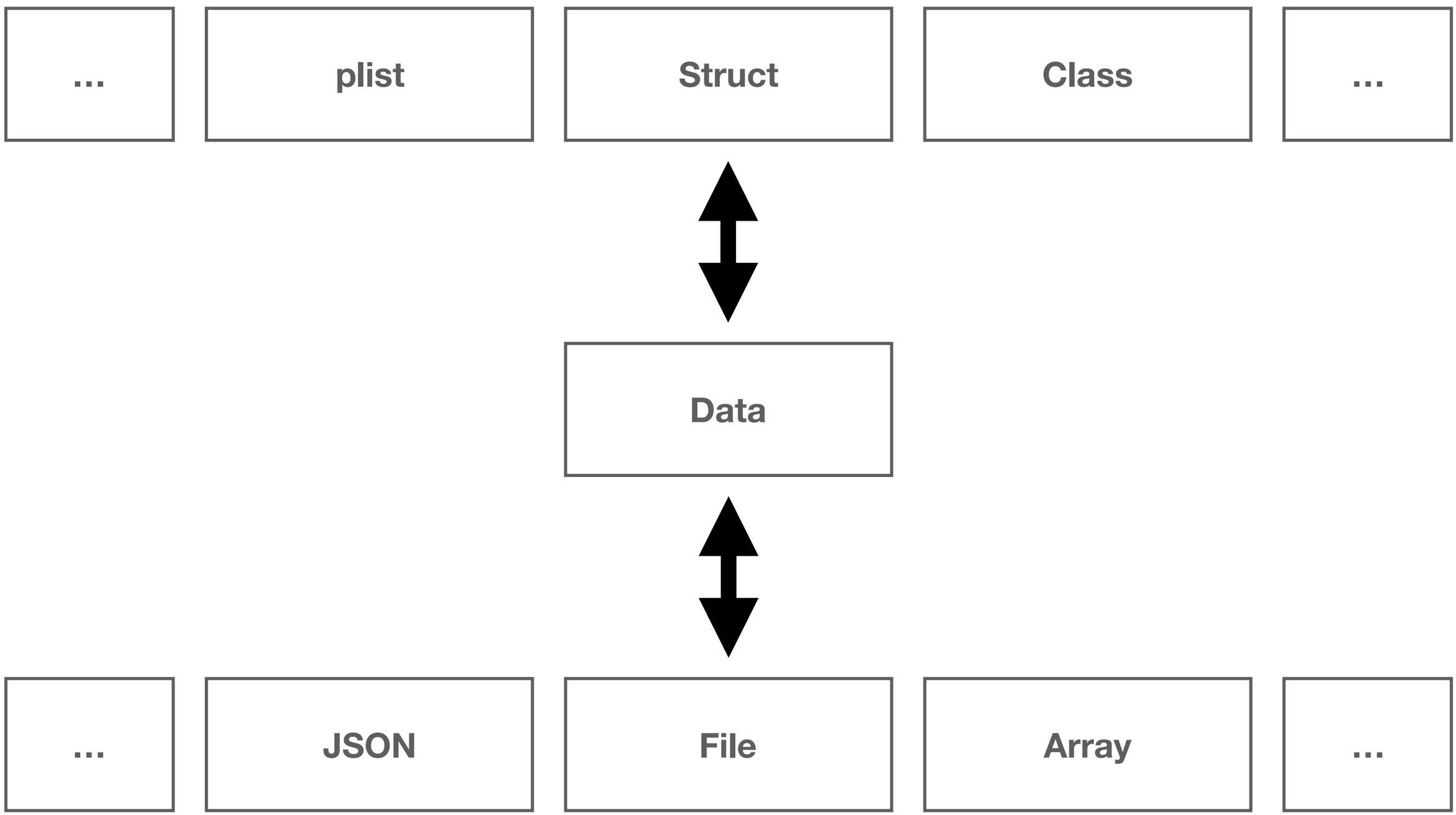
[Encoding, 부호화]

- 정보의 형태나 형식을 표준화, 보안, 처리 속도 향상, 저장 공간 절약 등을 위해서 목적에 맞는 다른 형태나 형식으로 변환하는 처리 혹은 그 처리 방식.
- Encoder : 인코딩을 수행하는 장치나 회로, 컴퓨터 소프트웨어, 알고리즘
- A type that can encode values into a native format for external representation.

[Decoding, 복호화]

- Encoding(부호화)된 대상을 원래의 형태로 되돌리는 일
- 예를 들어, 압축 파일을 다시 풀거나 암호화된 내용을 원래 내용으로 되돌리는 일
- A type that can decode values from a native format into in-memory representations.

Encode & Decode



Built-in Codable Types

- Swift Basic Types
  - `Bool`, `Int`, `Int(N)`, `UInt(N)`, `Float`, `Double`, `String`, `RawRepresentable`, `Optional`, `Array`, `Set`, `Dictionary`
- Foundation / CoreGraphics Types
  - `AffineTransform`, `Calendar`, `CharacterSet`, `Data`, `Date`, `DateComponents`, `DateInterval`, `Decimal`, `IndexPath`, `IndexSet`, `Locale`, `Measurement`, `NSRange`, `TimeZone`, `URL`, `UUID`, `CGFloat`, `CGPoint`, `CGSize`, `CGRect`, ...

Prepare JSON

```
let jsonData = """
{
  "bool": true,
  "int": 0,
  "double": 2.9,
  "string": "Hellow, World!",
  "array": [1,2,3,4],
  "dict": { "key": "value" },
}
""".data(using: .utf8)!
```

Conform to protocol Codable

```
struct CodableExample: Codable {
  let bool: Bool
  let int: Int
  let double: Double
  let string: String
  let array: [Int]
  let dict: [String: String]
}
```

Use Decoder & Encoder

```
let decoder = JSONDecoder()
let decoded = try decoder.decode(CodableExample.self, from: json)
print(type(of: decoded))        // CodeExample.Type


let encoder = JSONEncoder()
let encodedData = try encoder.encode(decoded)
let encodedString = String(data: encodedData, encoding: .utf8)!
print(type(of: encodedString)) // String.Type
```

Built-in Decoder / Encoder

```
/// `PropertyListEncoder` facilitates the encoding of `Encodable` values
into property lists.
open class PropertyListEncoder { }

/// `PropertyListDecoder` facilitates the of property list values into
semantic `Decodable` types.
open class PropertyListDecoder { }


/// `JSONEncoder` facilitates the encoding of `Encodable` values
into JSON.
open class JSONEncoder { }

/// `JSONDecoder` facilitates the decoding of JSON into semantic
`Decodable` types.
open class JSONDecoder { }
```

PropertyListDecoder

```
let fileURL = URL(fileURLWithPath: "file.plist")
guard let plistDict = NSDictionary(contentsOf: fileURL) as? [String: Any] else {
  return
}
if let value1 = plistDict["key1"] as? Int,
  let value2 = plistDict["key2"] as? String {
  print(value1, value2)
}
```

```
let fileURL = URL(fileURLWithPath: "file.plist")
guard let fileData = try? Data(contentsOf: fileURL) else { return }
let plistContent = PropertyListDecoder().decode(CodableType.self, from: fileData)
print(plistContent.key)
```

JSONDecoder

```
let fileURL = URL(fileURLWithPath: "file.json")
let jsonData = try! Data(contentsOf: fileURL)
guard let jsonObject = try? JSONSerialization.jsonObject(
  with: jsonData,
  options: .allowFragments
  ) else { return }
guard let jsonDict = jsonObject as? [String: Any] else { return }

if let value1 = jsonDict["key1"] as? Int {
  print(value1)
}
```

```
let fileURL = URL(fileURLWithPath: "file.json")
let jsonData = try! Data(contentsOf: fileURL)
let decodedContent = JSONDecoder().decode(CodableType.self, from: jsonData)
print(decodedContent.key)
```

SwiftyJSON

SwiftyJSON / SwiftyJSON

Watch 638 | Star 16,196 | Fork 2,794

Code | Issues 48 | Pull requests | Projects 1 | Insights

The better way to deal with JSON data in Swift

swiftyjson | json | swift

689 commits | 5 branches | 23 releases | 124 contributors | MIT

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

wongzigii Update README.md — Latest commit fb67763 on 8 Nov 2017

.github	Move files into .github directory	11 months ago
Example	Fix violations	3 months ago
Source	Fix SwiftLint violations	2 months ago
SwiftyJSON.xcodeproj	Fix broken path after renaming	4 months ago
SwiftyJSON.xcworkspace	Change example path for carthage. #276	3 years ago
Tests/SwiftyJSONTests	Fix SwiftLint violations	2 months ago

ObjectMapper

Hearst-DD / ObjectMapper

Watch 221 | Star 6,740 | Fork 737

Code | Issues 43 | Pull requests 5 | Projects 0 | Wiki | Insights

Simple JSON Object mapping written in Swift

924 commits | 8 branches | 48 releases | 83 contributors | MIT

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

tristanhimmelman Updated install versions in read me — Latest commit 883a6f1 15 days ago

.github	renamed github folder to github	11 months ago
ObjectMapper.xcodeproj	swift 4 no depreceted warnings	3 months ago
ObjectMapper.xcworkspace	Xcode 8 beta 4 changes..	2 years ago
Sources	swift 4 no depreceted warnings	3 months ago
Tests	Fix typo	3 months ago

Auto-synthesis example

```
struct User: Codable {
    var userName: String
    var score: Int
}
```

Auto-synthesis by compiler

```swift
struct User: Codable { // Auto-synthesis example
    var userName: String
    var score: Int

    @derived private enum CodingKeys: String, CodingKey { // @derived = auto-synthesized
        case userName
        case score
    }

    @derived init(from decoder: Decoder) throws {
        let container = try decoder.container(keyedBy: CodingKeys.self)
        userName = try container.decode(String.self, forKey: .userName)
        score = try container.decode(Int.self, forKey: .score)
    }

    @derived func encode(to encoder: Encoder) throws {
        var container = encoder.container(keyedBy: CodingKeys.self)
        try container.encode(userName, forKey: .userName)
        try container.encode(score, forKey: .score)
    }
}
```

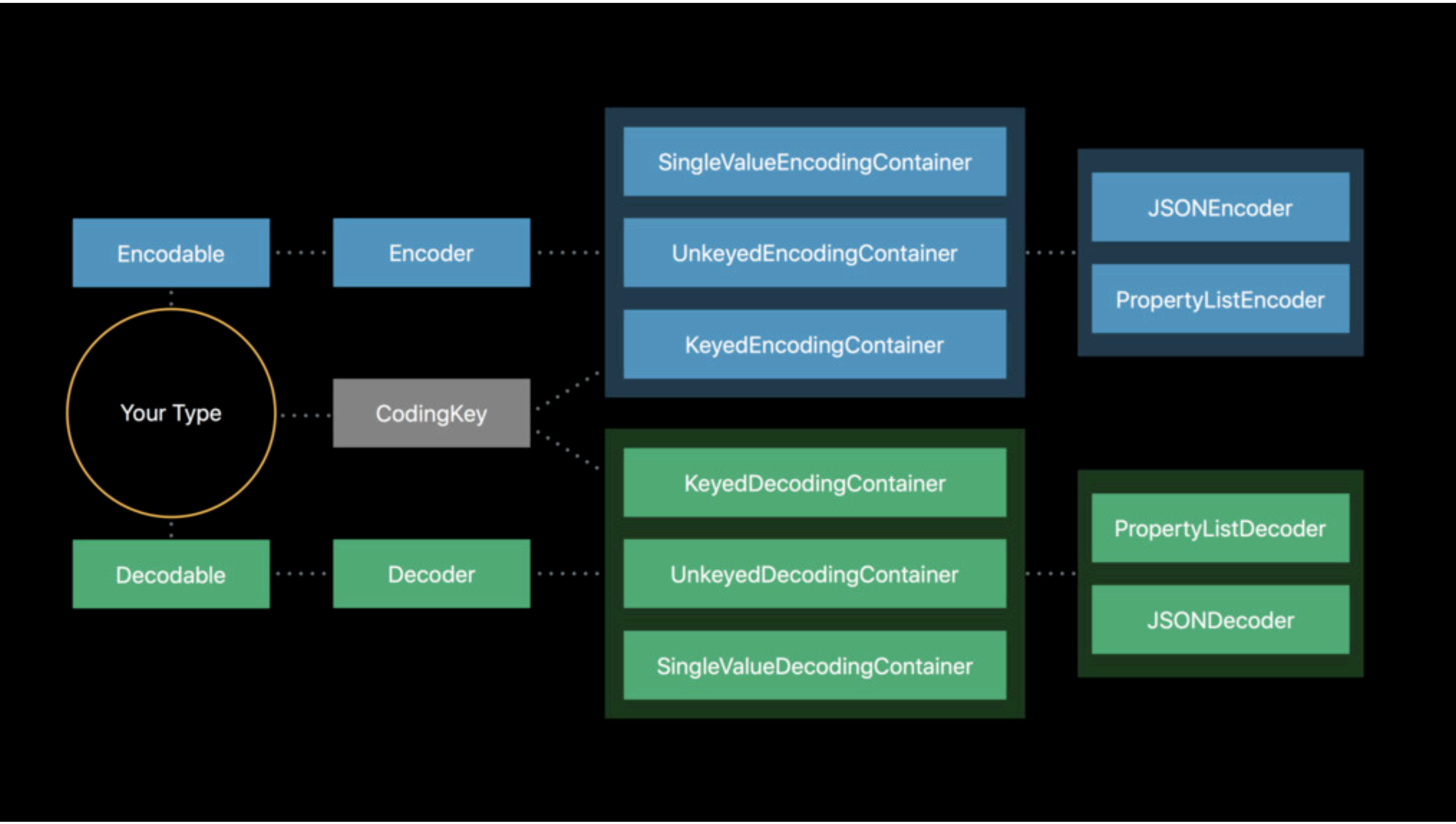
Manual Implementation

```swift
struct User: Codable { // Manual implementation example
    var userName: String  // Let's say, JSON has "user_name" key
    var score: Int        // Let's limit to 0...100

    private enum CodingKeys: String, CodingKey {  // manual implementation
        case userName = "user_name"  // rename key
        case score
    }

    init(from decoder: Decoder) throws {  // manual implementation
        let container = try decoder.container(keyedBy: CodingKeys.self)
        score = try container.decode(Int.self, forKey: .score)
        guard (0...100).contains(score) else {  // add validation
            throw DecodingError.dataCorrupted(
                codingPath: container.codingPath + [CodingKeys.score],
                debugDescription: "score is not in range 0...100"
            )
        }
        userName = try container.decode(String.self, forKey: .userName)
    }
}
```

Overview Codable Protocol



CodingKey

인코딩과 디코딩을 위한 키로 사용하기 위해 쓰이는 프로토콜

```
public protocol CodingKey {
    /// The string to use in a named collection(e.g. a string-keyed dict)
    public var stringValue: String { get }

    /// Creates a new instance from the given string.
    public init?(stringValue: String)

    /// The value to use in an integer-indexed collection
    /// (e.g. an int-keyed dictionary).
    public var intValue: Int? { get }

    /// Creates a new instance from the specified integer.
    public init?(intValue: Int)
}
```

Basic

```
struct Dog: Decodable {
  let age: Int
  let name: String

  private enum CodingKeys: String, CodingKey {
    case age
    case name
  }
}
```

Basic

```
let jsonData = """
{
  "age": 3,
  "name": "Tory"
}
""".data(using: .utf8)!

let dog = try? JSONDecoder().decode(Dog.self, from: jsonData)
print(dog)
```

Array

```
let jsonData = """
[
  {
    "age": 3,
    "name": "Tory"
  },
  {
    "age": 3,
    "name": "Tory"
  }
]
""".data(using: .utf8)!

let dogs = try! JSONDecoder().decode([Dog].self, from: jsonData)
print(dogs)
```

Dictionary

```
let jsonData = """
{
  "first": {
    "age": 3,
    "name": "Tory"
  },
  "second": {
    "age": 3,
    "name": "Tory"
  }
}
""".data(using: .utf8)!

let dogs = try! JSONDecoder().decode([String: Dog].self, from:
jsonData)
print(dogs)
```

Decode Manually

```
struct Dog: Decodable {
  let age: Int
  let name: String

  private enum CodingKeys: String, CodingKey {
    case age
    case name
  }

  init(from decoder: Decoder) throws {
    let values = try decoder.container(keyedBy: CodingKeys.self)
    age = try values.decode(Int.self, forKey: .age)
    name = try values.decode(String.self, forKey: .name)
  }
}
```

Nested Keys

```
struct Coordinate {
  var latitude: Double
  var longitude: Double
  var elevation: Double

  enum CodingKeys: String, CodingKey {
    case latitude
    case longitude
    case additionalInfo
  }
  enum AdditionalInfoKeys: String, CodingKey {
    case elevation
  }
}
```

Nested Keys

```
extension Coordinate: Decodable {
  init(from decoder: Decoder) throws {
    let values = try decoder.container(keyedBy: CodingKeys.self)
    latitude = try values.decode(Double.self, forKey: .latitude)
    longitude = try values.decode(Double.self, forKey: .longitude)

    let additionalInfo = try values.nestedContainer(
      keyedBy: AdditionalInfoKeys.self, forKey: .additionalInfo
    )
    elevation = try additionalInfo.decode(
      Double.self, forKey: .elevation
    )
  }
}
```

EncodingError

```
/// An error that occurs during the encoding of a value.
public enum EncodingError : Error {
  /// 주어진 값으로 인코딩을 하지 못할 때
  case invalidValue(Any, EncodingError.Context)
}
```

DecodingError

```
/// An error that occurs during the decoding of a value.
public enum DecodingError : Error {
  /// 프로퍼티 타입 미스매치
  case typeMismatch(Any.Type, DecodingError.Context)
  /// 디코딩할 데이터의 키에 해당하는 Value 가 없을 경우
  case valueNotFound(Any.Type, DecodingError.Context)]
  /// 디코딩할 데이터에 지정한 키가 없는 경우
  case keyNotFound(CodingKey, DecodingError.Context)
  /// 데이터가 망가졌을 경우
  case dataCorrupted(DecodingError.Context)
}
```

Overview

Overview of Codable (1)

- Container Protocols
  - Keyed ... for dictionary coding
  - Unkeyed ... for array coding
  - SingleValue ... for single primitive value coding
- 3 containers are used as **intermediate hierarchical representation** for arbitrary (de)serialization (e.g. `JSONSerialization` with `[String: Any]`).

Overview

Overview of Codable (2)

- `CodingKey`
  - Type-safe `String`-based key (and optionally, `Int` key)
- `superDecoder` / `superEncoder`
  - Used when subclassing
- Error Handling (`DecodingError` / `EncodingError`)
  - `codingPath` records the error path and outputs its detail

Quicktype-xcode Library

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help 00:09.9

Ready | Today at 10:45 PM

CoolApp

```
1  import UIKit
2
   let json = """
4  {
5      "message": "Hello, App Store!",
6      "desired_rating": 5,
7      "creators": [
8          { "name": "Mark" },
9          { "name": "David" }
10     ]
11 }
12 """
13
14
15
```