Introducing Combine

Tony Parker, Foundation
App Requirements

Valid user name
Matching passwords
Responsive user interface
Wizard School Signup

- Wizard name
- Password
- Repeat password

Create Account
URLSession
Merge
KVC
Asynchronous Interfaces

Target/Action

Notification center

URLSession

Key-value observing

Ad-hoc callbacks
A unified, declarative API for processing values over time
Combine Features

Generic
Type safe
Composition first
Request driven
Key Concepts

Publishers
Subscribers
Operators
Publisher

Defines how values and errors are produced

Value type

Allows registration of a Subscriber
protocol Publisher {
    associatedtype Output
    associatedtype Failure: Error

    func subscribe<S: Subscriber>(_ subscriber: S)
        where S.Input == Output, S.Failure == Failure
}
extension NotificationCenter {
    struct Publisher: Combine.Publisher {
        typealias Output = Notification
        typealias Failure = Never
        init(center: NotificationCenter, name: Notification.Name, object: Any? = nil)
    }
}
Subscriber

Receives values and a completion

Reference type
protocol Subscriber {
    associatedtype Input
    associatedtype Failure: Error

    func receive(subscription: Subscription)
    func receive(_ input: Input) -> Subscribers.Demand
    func receive(completion: Subscribers.Completion<Failure>)
}
extension Subscribers {
    class Assign<Root, Input>: Subscriber, Cancellable {
        typealias Failure = Never
        init(object: Root, keyPath: ReferenceWritableKeyPath<Root, Input>)
    }
}
The Pattern
The Pattern
Subscriber is attached to Publisher
The Pattern

Subscriber **is attached to** Publisher

Publisher **sends a** Subscription

Publisher sends a Subscription

receive(subscription:)

subscribe( )
The Pattern

**Subscriber** is attached to **Publisher**

**Publisher** sends a **Subscription**

**Subscriber** requests $N$ values
The Pattern

**Subscriber** is attached to **Publisher**

**Publisher** sends a **Subscription**

**Subscriber** requests $N$ values

**Publisher** sends $N$ values or less
The Pattern

Subscriber is attached to Publisher

Publisher sends a Subscription

Subscriber requests $N$ values

Publisher sends $N$ values or less

Publisher sends completion

```swift
Publisher
  subscribe(
    Subscriber
  )

receive(subscription:)

request(_ : Demand)

receive(_ : Input)

receive(_ : Input)

receive(completion:)
```
// Using Publisher and Subscriber

class Wizard {
    var grade: Int
}

let merlin = Wizard(grade: 5)

let graduationPublisher = NotificationCenter.Publisher(center: .default, name: .graduated, object: merlin)

let gradeSubscriber = Subscribers.Assign(object: merlin, keyPath: \.grade)

graduationPublisher.subscribe(gradeSubscriber)
// Using Publisher and Subscriber

class Wizard {
    var grade: Int
}

let merlin = Wizard(grade: 5)

let graduationPublisher = NotificationCenter.Publisher(center: .default, name: .graduated, object: merlin)

let gradeSubscriber = Subscribers.Assign(object: merlin, keyPath: ".grade")

graduationPublisher.subscribe(gradeSubscriber)
// Using Publisher and Subscriber

class Wizard {
    var grade: Int
}

let merlin = Wizard(grade: 5)

let graduationPublisher = NotificationCenter.Publisher(center: .default, name: .graduated, object: merlin)

let gradeSubscriber = Subscribers.Assign(object: merlin, keyPath: ".grade")

graduationPublisher.subscribe(gradeSubscriber)
// Using Publisher and Subscriber

class Wizard {
    var grade: Int
}

let merlin = Wizard(grade: 5)

let graduationPublisher = NotificationCenter.Publisher(center: .default, name: .graduated, object: merlin)

let gradeSubscriber = Subscribers.Assign(object: merlin, keyPath: \.grade)

graduationPublisher.subscribe(gradeSubscriber)
class Wizard {
    var grade: Int
}

let merlin = Wizard(grade: 5)

let graduationPublisher = NotificationCenter.Publisher(center: .default, name: .graduated, object: merlin)

let gradeSubscriber = Subscribers.Assign(object: merlin, keyPath: ".grade")

graduationPublisher.subscribe(gradeSubscriber)

⚠️ Instance method 'subscribe' requires the types 'NotificationCenter.Publisher.Output' (aka 'Notification') and 'Int' be equivalent
Notification Center

Notification

?

Int

Assign
Operator

Adopts Publisher

Describes a behavior for changing values

Subscribes to a Publisher ("upstream")

Sends result to a Subscriber ("downstream")

Value type
extension Publishers {
    struct Map<Upstream: Publisher, Output>: Publisher {
        typealias Failure = Upstream.Failure

        let upstream: Upstream
        let transform: (Upstream.Output) -> Output
    }
}
Notification Center

Notification

Map

Int

Assign
// Using Operators

let graduationPublisher = NotificationCenter.Publisher(center: .default, name: .graduated, object: merlin)

let gradeSubscriber = Subscribers.Assign(object: merlin, keyPath: ".grade")

let converter = Publishers.Map(upstream: graduationPublisher) { note in
    return note.userInfo?["NewGrade"] as? Int ?? 0
}

converter.subscribe(gradeSubscriber)
// Using Operators

let graduationPublisher = 
    NotificationCenter.Publisher(center: .default, name: .graduated, object: merlin)

let gradeSubscriber = Subscribers.Assign(object: merlin, keyPath: \.grade)

let converter = Publishers.Map(upstream: graduationPublisher) { note in
    return note.userInfo?["NewGrade"] as? Int ?? 0
}

converter.subscribe(gradeSubscriber)
// Using Operators
let graduationPublisher = 
    NotificationCenter.Publisher(center: .default, name: .graduated, object: merlin)

let gradeSubscriber = Subscribers.Assign(object: merlin, keyPath: \.grade)

let converter = Publishers.Map(upstream: graduationPublisher) { note in
    return note.userInfo?["NewGrade"] as? Int ?? 0
}

canverter.subscribe(gradeSubscriber)
// Operator Construction
extension Publisher {
    func map<T>(_ transform: @escaping (Output) -> T) -> Publishers.Map<Self, T> {
    }
}
// Operator Construction

extension Publisher {

    func map<T>(_ transform: @escaping (Output) -> T) -> Publishers.Map<Self, T> {
    }
}

// Operator Construction

extension Publisher {

    func map<T>(_ transform: @escaping (Output) -> T) -> Publishers.Map<Self, T> {
    }

}
extension Publisher {
    func map<T>(_ transform: @escaping (Output) -> T) -> Publishers.Map<Self, T> {
    }
}
// Chained Publishers

let cancellable =
    NotificationCenter.default.publisher(for: .graduated, object: merlin)
        .map { note in
            return note.userInfo?["NewGrade"] as? Int ?? 0
        }
        .assign(to: \.grade, on: merlin)
let cancellable = 
NotificationCenter.default.publisher(for: .graduated, object: merlin)
    .map { note in
        return note.userInfo?["NewGrade"] as? Int ?? 0
    }
    .assign(to: \.grade, on: merlin)
let cancellable =  
    NotificationCenter.default.publisher(for: .graduated, object: merlin)  
    .map { note in  
        return note.userInfo?["NewGrade"] as? Int ?? 0  
    }  
    .assign(to: \.grade, on: merlin)
let cancellable = 
NotificationCenter.default.publisher(for: .graduated, object: merlin)
    .map { note in
        return note.userInfo?["NewGrade"] as? Int ?? 0
    }
    .assign(to: \.grade, on: merlin)
Declarative Operator API

Functional transformations
List operations
Error handling
Thread or queue movement
Scheduling and time
catch, append, count, first, min, dropFirst, map, abortOnError, merge, log, last, allSatisfy, filter, breakpointOnError, handleEvents, mapError, output, breakpoint, flatMap, ignoreOutput, max, drop, setFailureType, removeDuplicates, reduce, combineLatest, replaceEmpty, prepend, contains, switchToLatest, collect, compactMap, replaceError, replaceNil, scan, retry, print, zip
Try composition first
<table>
<thead>
<tr>
<th>Synchronous</th>
<th>Asynchronous</th>
</tr>
</thead>
<tbody>
<tr>
<td>One</td>
<td>Many</td>
</tr>
</tbody>
</table>
// Composing Operators

let cancellable = 
    NotificationCenter.default.publisher(for: .graduated, object: merlin) 
    .map { note in 
        return note.userInfo?"NewGrade" as? Int ?? 0 
    } 
    .assign(to: \.grade, on: merlin)
// Composing Operators

let cancellable = 
    NotificationCenter.default.publisher(for: .graduated, object: merlin) 
        .map { note in 
            return note.userInfo?["NewGrade"] as? Int ?? 0 
        } 
        .assign(to: \.grade, on: merlin)
// Composing Operators

let cancellable =
    NotificationCenter.default.publisher(for: .graduated, object: merlin)
        .compactMap { note in
            return note.userInfo?"NewGrade" as? Int
        }
        .assign(to: \.grade, on: merlin)
let cancellable = 
    NotificationCenter.default.publisher(for: .graduated, object: merlin)
    .compactMap { note in
        return note.userInfo?"NewGrade" as? Int
    }
    .filter { $0 >= 5 }
    .assign(to: \.grade, on: merlin)
// Composing Operators

let cancellable = 
    NotificationCenter.default.publisher(for: .graduated, object: merlin)
        .compactMap { note in
            return note.userInfo?["NewGrade"] as? Int
        }
        .filter { $0 >= 5 }
        .assign(to: \.grade, on: merlin)
let cancellable = 
  NotificationCenter.default.publisher(for: .graduated, object: merlin)
    .compactMap { note in
      return note.userInfo?['NewGrade'] as? Int
    }
    .filter { $0 >= 5 }
    .prefix(3)
    .assign(to: \.grade, on: merlin)
// Composing Operators

let cancellable = 
    NotificationCenter.default.publisher(for: .graduated, object: merlin)
    .compactMap { note in
        return note.userInfo?["NewGrade"] as? Int
    }
    .filter { $0 >= 5 }
    .prefix(3)
    .assign(to: \.grade, on: merlin)
// Composing Operators

let cancellable = 
    NotificationCenter.default.publisher(for: .graduated, object: merlin)
        .compactMap { note in
            return note.userInfo?["NewGrade"] as? Int
        }
        .filter { $0 >= 5 }
        .prefix(3)
        .assign(to:\.grade, on: merlin)
Combining Publishers

Zip

CombineLatest
Creating wand...

- Organizing sparkles...
- Decomposing cellular material...
- Arranging discontinuity matrix...

Continue
Zip

Converts several inputs into a single tuple

A “when/and” operation

Requires input from all to proceed
Zip

Converts several inputs into a single tuple

A “when/and” operation

Requires input from all to proceed
Zip3(organizing, decomposing, arranging)
.map { $0 && $1 && $2 }
.assign(to: \.isEnabled, on: continueButton)
Terms & Conditions

Read manual
Practiced in simulator
Teacher approved

Play
Combine Latest

Converts several inputs into a single value

A “when/or” operation

Requires input from any to proceed

Stores last value
Combine Latest

Converts several inputs into a single value

A “when/or” operation

Requires input from any to proceed

Stores last value
CombineLatest3(read, practiced, approved) {
    $0 && $1 && $2
}
.assign(to: \.isEnabled, on: playButton)
Try It

Process a NotificationCenter post with filter

Await completion of two network requests with zip
decode the data of a URLResponse
More to Combine

Error handling and cancellation

Schedulers and time

Design patterns
More Information

developer.apple.com/wwdc19/722