Lenses in Swift

Guanshan Liu @guanshanliu

Functional Programming

— First-class and higher-order functions

First-class and higher-order functions

Assign functions to variables

```
func greeting() {
    print("Hello, world!")
}

let welcome = greeting

welcome() // print: Hello, world!
```

First-class and higher-order functions

Use functions as arguments

```
func greeting() -> String {
    return "Hello, world!"
}

func shoutOut(_ f: () -> String) {
    print(f())
}

shoutOut(greeting) // print: Hello, world!
```

First-class and higher-order functions

Return a function

```
func shoutOut() -> (String) -> Void {
    return { str in
        print(str)
    }
}

shoutOut()("Hello, world!") // print: Hello, world!
```

MVVM: A non-reactive introduction by Ian Keen

```
class FriendCellViewModel {
    var didError: ((ErrorProtocol) -> Void)?
    var didUpdate: ((FriendCellViewModel) -> Void)?
    var didSelectFriend: ((Friend) -> Void)?
}
```

Object-Oriented Functional Programming by Saul Mora

```
func expired(fileURL: NSURL) -> Bool {
    let fileManager = NSFileManager()
    var error : NSError?

    let filePath = fileURL.path
    let fileExists : (String) -> (String?) =
    { path in fileManager.fileExistsAtPath(path) ? path : nil }
    let retrieveFileAttributes : (String) -> ([NSObject: AnyObject]?) =
    { path in
        var error: NSError?
        return fileManager.attributesOfItemAtPath(path, error: &error)
    }
    let extractCreationDate : ([NSObject:AnyObject]) -> NSDate? =
    { $0[NSFileModificationDate] as? NSDate }
    let checkExpired: NSDate -> Bool? =
    { $0.isBefore(NSDate.oneDayAgo()) }

    return filePath >>= fileExists >>= retrieveFileAttributes >>= extractCreationDate >>= checkExpired ?? false
}
```

Functional Programming

— First-class and higher-order functions
— Pure functions

Pure functions

The return value of a function is only determined by its input values, no side-effects.

```swift
func increment(_ number: Int) -> Int {
    return number + 1
}

let result = increment(100) // 101
```

Functional Programming

- First-class and higher-order functions
- Pure functions
- Value types
- Immutability

```swift
struct Contact {
    let email: String
}

struct Account {
    let username: String
    let contact: Contact
}

let user = Account(username: "guanshanliu",
                   contact: Contact(email: "guanshan.liu@gmail.com"))

❌ user.username = "liuguanshan"
❌ user.contact.email = "guanshanliu@icloud.com"
```

Lenses are functional getters and setters.

Lenses are functional getters and setters.

```
struct Lens<Whole, Part> {
    let get: (Whole) -> Part
    let set: (Part, Whole) -> Whole
}
```

```swift
extension Contact {
    static let emailLens = Lens<Contact, String>(
        get: { contact in
            return contact.email
        },
        set: { newEmail, _ in
            return Contact(email: newEmail)
        }
    )
}
```

```swift
extension Account {
    static let usernameLens = Lens<Account, String>(
        get: { account in
            return account.username
        },
        set: { newUsername, account in
            return Account(username: newUsername, contact: account.contact)
        }
    )

    static let contactLens = Lens<Account, Contact>(
        get: { account in
            return account.contact
        },
        set: { newContact, account in
            return Account(username: account.username, contact: newContact)
        }
    )
}
```

```
let user = Account(username: "guanshanliu",
                   contact: Contact(email: "guanshan.liu@gmail.com"))

Account.usernameLens.get(user)
// "guanshanliu"
Account.usernameLens.set("liuguanshan", user)
// username: liuguanshan, contact: email: guanshan.liu@gmail.com

Account.contactLens.set(
    Contact.emailLens.set("guanshanliu@icloud.com", user.contact),
    user)
// username: guanshanliu, contact: email: guanshanliu@icloud.com
```

Composition

```
extension Lens {
    func compose<SubPart>(_ other: Lens<Part, SubPart>) -> Lens<Whole, SubPart> {
        return Lens<Whole, SubPart>(
            get: { whole in
                let part = self.get(whole)
                return other.get(part)
            },
            set: { subPart, whole in
                let part = self.get(whole)
                let newPart = other.set(subPart, part)
                return self.set(newPart, whole)
            }
        )
    }
}
```

Composition

```
Account.contactLens.set(
    Contact.emailLens.set("guanshanliu@icloud.com", user.contact),
    user)

Account.contactLens.compose(Contact.emailLens).set("guanshanliu@icloud.com", user)
```

Composition

```
func *<A, B, C>(left: Lens<A, B>, right: Lens<B, C>) -> Lens<A, C> {
    return left.compose(right)
}

(Account.contactLens * Contact.emailLens).set("guanshanliu@icloud.com", user)
```

```swift
infix operator |> { associativity left precedence 80 }
func |><A, B>(x: A, f: (A) -> B) -> B {
    return f(x)
}

func |><A, B, C>(f: (A) -> B, g: (B) -> C) -> (A) -> C {
    return { g(f($0)) }
}

infix operator *~ { associativity left precedence 100 }
func *~<Whole, Part>(left: Lens<Whole, Part>, right: Part) -> (Whole) -> Whole {
    return { whole in
        left.set(right, whole)
    }
}

user |>
    Account.contactLens * Contact.emailLens
    *~ "guanshanliu@icloud.com"
```

Why Lenses?

Why Lenses?

```
struct Contact {
    var email: String
}

struct Account {
    var username: String
    var contact: Contact
}

let user = Account(username: "guanshanliu",
                   contact: Contact(email: "guanshan.liu@gmail.com"))

user.username = "liuguanshan"
user.contact.email = "guanshanliu@icloud.com"
```

Case Study

```
class ViewModel {
    private(set) var model: Account

    init(model: Account) {
        self.model = model
    }

    func update(email: String) {
        model = (model |>
            Account.contactLens * Contact.emailLens
            *~ email)
    }

    func update(username: String) {
        model = (model |>
            Account.usernameLens
            *~ username)
    }
}
```

References

- Lenses in Swift talk by Brandon Williams
- Lenses in Swift post by Chris Eidhof
- MVVM: A non-reactive introduction post by Ian Keen
- Object-Oriented Functional Programming talk by Saul Mora

Guanshan Liu

- Twitter: @guanshanliu
- WeChat Official Account

Thank you

Questions?