SWIFTUI'DA COORDINATOR YONDASHUVI

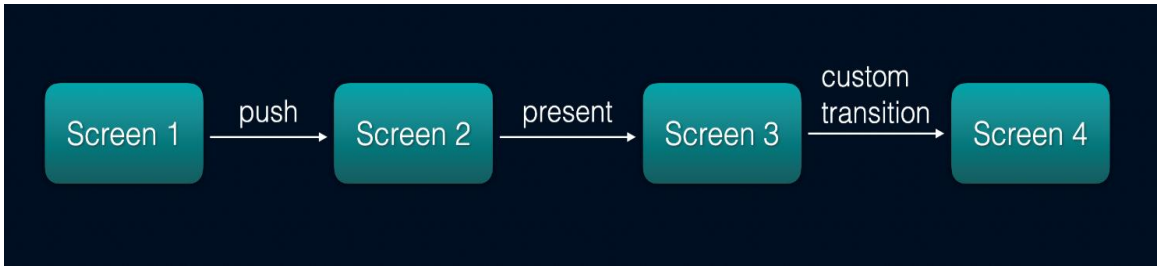
Cho'lliyev Shoxrux Ibadullayevich
Muhammad al-Xorazmiy nomidagi
Toshkent Axborot Texnologiyalari Universiteti

Annotatsiya: Ushbu maqola iOS dasturlashda Coordinator yondashuvini SwiftUI'da qo'llash imkoniyatlarini o'rganadi. Coordinator nima ekanligi, uning UIKit va SwiftUI'dagi amalga oshirilishi solishtiriladi va NavigationStack yordamida SwiftUI'da Coordinator qanday qurilishi mumkinligi misollar bilan ko'rsatiladi. Maqola SwiftUI navigatsiyasining afzalliklari va cheklovlarini tahlil qilib, ushbu yondashuvning katta ilovalarda qo'llanilishi bo'yicha xulosalar chiqaradi. Dasturchilarga UIKit va SwiftUI o'rtasidagi tanlovda yordam berish maqsad qilingan.

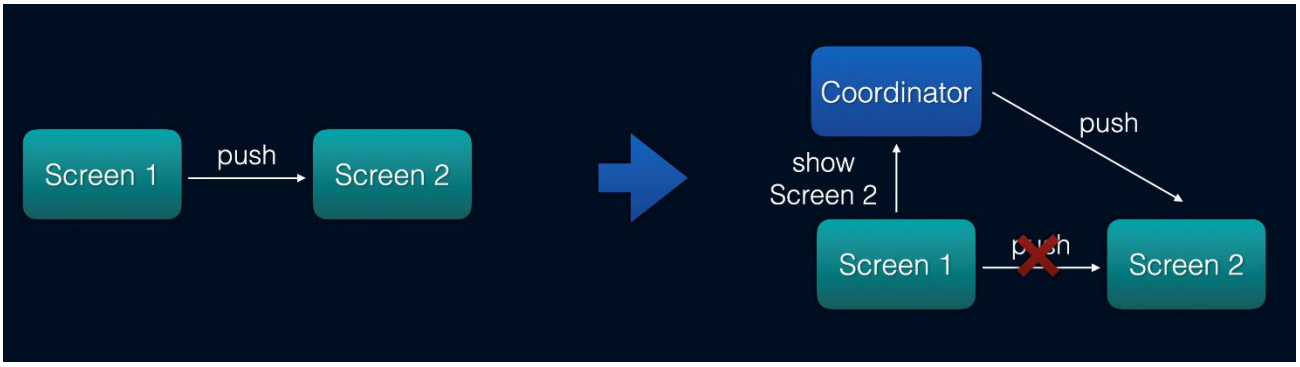
SwiftUI'da Coordinator yondashuvi haqida gaplashmoqchiman. iOS 18 chiqishi bilan ko'plab jamoalar iOS 16'ni minimal talab sifatida qabul qilmoqda va bu SwiftUI'ning yangi imkoniyatlari, xususan NavigationStack'ga e'tiborni oshiradi. SwiftUI navigatsiyasi haqida fikrlar turlicha – ba'zilar uni yaxshi deb hisoblasa, boshqalar hali katta ilovalar uchun yetarli emasligini aytadi. Shu sababli, men SwiftUI'da Coordinator yondashuvini sinab ko'rishga qaror qildim va bu haqda o'z tajribamni sizlar bilan bo'lishmoqchiman.

Coordinator nima?

Coordinator – bu navigatsiya jarayonini boshqaradigan oddiy, lekin kuchli yondashuv. Navigatsiya jarayoni deganda, ilovadagi ekranlar ketma-ketligi va ular o'rtasidagi o'tishlar tushuniladi – bu push, present yoki maxsus o'tishlar bo'lishi mumkin. Coordinatorning vazifasi – ushbu jarayonni to'liq nazorat qilish, ya'ni foydalanuvchi harakatlariga qo'shimcha ravishda, tashqi hodisalar asosida ham navigatsiyani o'zgartirish imkonini beradi.

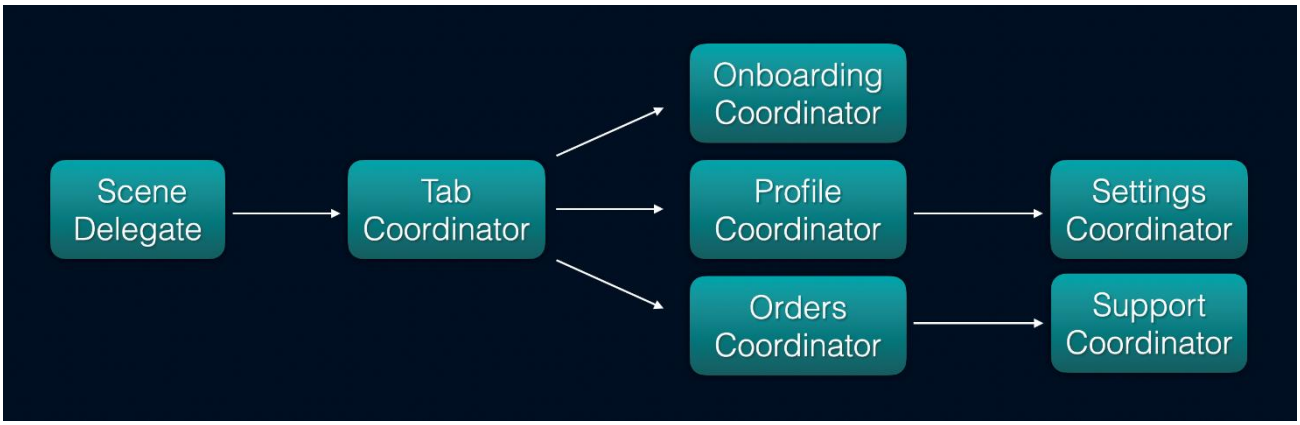


Coordinatorning afzalliklari shundaki, u navigatsiya mantig'ini ko'rinishlardan (views) ajratadi. Bu ko'rinishlar o'rtasidagi bog'liqlikni kamaytiradi, kodni sinash, qo'llab-quvvatlash va qayta ishlatishni osonlashtiradi. Coordinatorlar ierarxik tuzilishi tufayli katta ilovalarda ham yaxshi moslashadi – siz ularni ichma-ich joylashtirib, daraxtsimon tuzilma hosil qilishingiz mumkin.



UIKit'da Coordinator

UIKit'da Coordinatorni oddiy tarzda qurish mumkin. Avval Coordinator protokolini aniqlaymiz:




```
protocol Coordinator: CoordinatorFinishDelegate {
    func start()
    func finish()
    var finishDelegate: CoordinatorFinishDelegate? { get set }
}

protocol CoordinatorFinishDelegate: AnyObject {
    func didFinish(childCoordinator: Coordinator)
}
```

start() navigatsiya jarayonini boshlaydi, finish() esa uni yakunlaydi. finishDelegate orqali bola Coordinator ota Coordinatorni xabardor qiladi. Keyin, ma'lum bir xususiyat uchun Coordinatorni quyidagicha amalga oshirish mumkin:

```
final class FeatureCoordinator: FlowCoordinator {
    weak var finishDelegate: CoordinatorFinishDelegate?
    var childCoordinator: Coordinator?
    private let navigationController: UINavigationController

    init(navigationController: UINavigationController) {
        self.navigationController = navigationController
```

```
    }
    func start() {
        let rootScreen = FeatureScreen(coordinator: self)
        let vc = UIHostingController(rootView: rootScreen)
        navigationController.setViewControllers([vc], animated: false)
    }
    func pushNextScreen() {
        let nextScreen = NextScreen(coordinator: self)
        let vc = UIHostingController(rootView: nextScreen)
        navigationController.pushViewController(vc, animated: true)
    }
}
```

Bu yondashuvda navigatsiya Controller orqali boshqariladi va SwiftUI ko‘rinishlari UIHostingControllerga o‘raladi. Bu sodda, moslashuvchan va katta ilovalarda sinovdan o‘tgan usul.

SwiftUI’da navigatsiya

SwiftUI’da avvalgi NavigationView ko‘p cheklovlarga ega edi – masalan, bir nechta ekranlarni push qilish yoki root ekranga qaytish qiyin edi. iOS 16’da chiqarilgan NavigationStack esa holatni boshqarish imkonini beradi:

```
struct ContentView: View {
    @State private var path = NavigationPath()
    var body: some View {
        NavigationStack(path: $path) {
            RootView()
                .navigationDestination(for: HashableRoute.self) { route in
                    switch route {
                    case .firstScreen: FirstScreen()
                    case .secondScreen: SecondScreen()
                    }
                }
        }
    }
}
```

NavigationStack yordamida holatni path orqali boshqarish mumkin, bu esa dasturiy navigatsiyani osonlashtiradi.

SwiftUI’da Coordinator

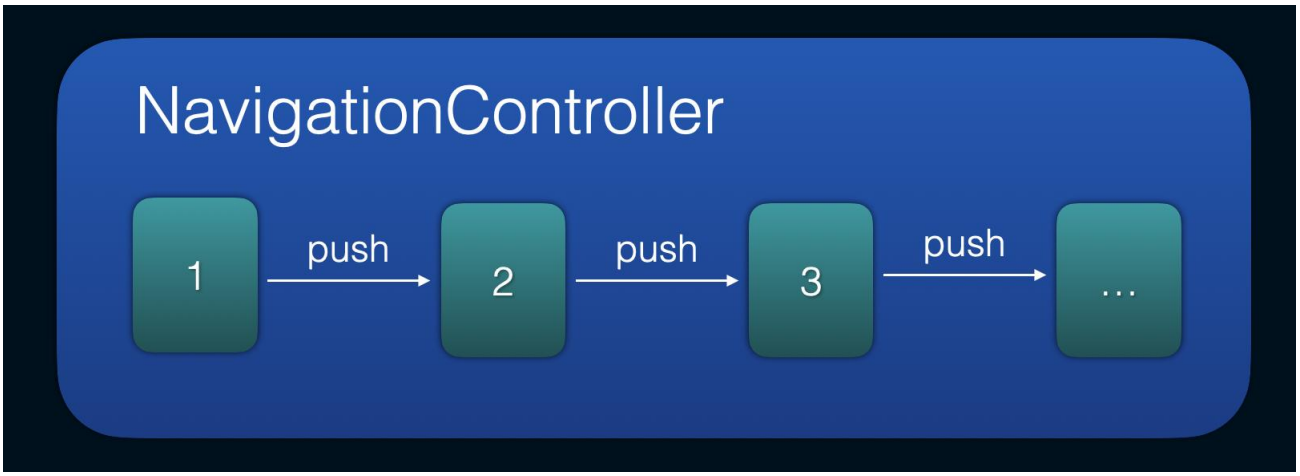
SwiftUI’da Coordinatorni qurish uchun protokollarni quyidagicha aniqlaymiz:

```
protocol Coordinator: ObservableObject, CoordinatorFinishDelegate {
    associatedtype Content: View
    @ViewBuilder var rootView: Content { get }
```

```
    var finishDelegate: CoordinatorFinishDelegate? { get set }
    func finish()
}
protocol FlowCoordinator: Coordinator {
    associatedtype Route: Routable
    @ViewBuilder func destination(for route: Route) -> some View
    var childCoordinator: (any Coordinator)? { get set }
    var navigationControllers: [NavigationController<Route>] { get set }
}
```

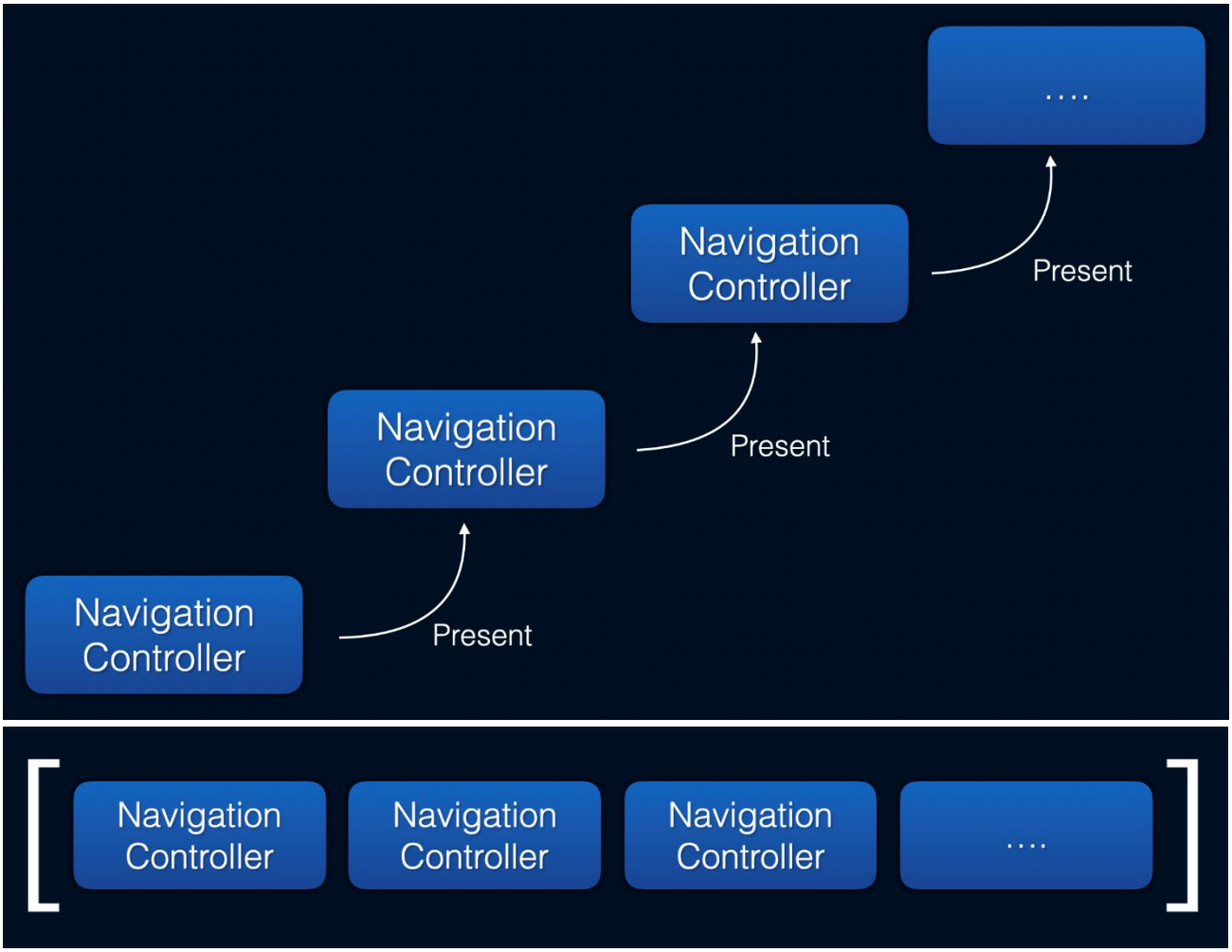
Bu yerda ObservableObject qo‘shildi, chunki SwiftUI’da holat o‘zgarishlarini kuzatish kerak. rootView esa Coordinatorni SwiftUI ko‘rinishlari bilan bog‘laydi. Misol uchun:

```
final class FeatureCoordinator: FlowCoordinator {
    @Published var childCoordinator: (any Coordinator)?
    @Published var navigationControllers = [NavigationController<FeatureRoute>]()
    func destination(for route: FeatureRoute) -> some View {
        switch route {
        case .next: NextScreen()
        case .another: AnotherScreen()
        }
    }
```




```
    var rootView: some View {
        NavigatingView(nc: rootNavigationController, coordinator: self) {
            FeatureScreen()
        }
        .environmentObject(self)
    }
    func pushNextScreen() { push(route: .next) }
}
```

NavigatingView – bu NavigationStackni o‘z ichiga olgan yordamchi ko‘rinish bo‘lib, push va present kabi navigatsiyalarni boshqaradi.



Cheklovlar

SwiftUI’dagi Coordinator bir qator cheklovlarga ega:

1. **Ko‘rinishga bog‘liqlik**: UIKit’da navigatsiya ko‘rinishlardan ajratilgan bo‘lsa, SwiftUI’da rootView va destination kabi xususiyatlar Coordinatorni ko‘rinishlarga bog‘laydi.
2. **Murakkablik**: SwiftUI’da NavigationStack faqat push/pop bilan cheklangan, present kabi operatsiyalar uchun qo‘shimcha kod yozish kerak.
3. **Bola Coordinatorlar**: UIKit’da bola Coordinatorlarni boshqarish moslashuvchan bo‘lsa, SwiftUI’da bu faqat sheet bilan cheklangan.
4. **Navigatsiya yo‘lini o‘zgartirish**: NavigationPath faqat oxiridan elementlarni o‘chirishga ruxsat beradi, o‘rtadan o‘chirish qiyin.

Xulosa

Hozircha men UIKit’da Coordinator yondashuvidan foydalanishni afzal ko‘raman, chunki u kuchli, bashorat qilinadigan va sinovdan o‘tgan. SwiftUI’da navigatsiya hali to‘liq pishmagan, ammo kelajakda yaxshilanishi mumkin. Agar sizga

oddiy navigatsiya kerak bo‘lsa, SwiftUI yetarli bo‘lishi mumkin, lekin murakkab ilovalar uchun UIKit hali ham ustun turadi.

Foydalanilgan adabiyotlar:

1. Apple Developer Documentation – SwiftUI va NavigationStack bo‘yicha rasmiy hujjatlar.
2. WWDC 2022 – NavigationStack haqida sessiya.
3. SwiftUI Navigation bo‘yicha onlayn munozarala