Existential any in Swift 5.6

Introduction to existential any and what it means to you

Mark Skov Jensen

Existential any in Swift 5.6

Introduction to existential any and what it means to you

- Quick overview of existential types
- Why the change?
- What has changed

Wait… existential types?

Existential types

A quick refresher

```
protocol Animal {
    var name: String { get }
}

struct Dog: Animal {
    let name: String
}

let marksDog: Animal = Dog(name: "Max")

func feed(_ animal: Animal) {
    print("Feeding \(animal.name)")
}
```

Why the change?

- Existential types in Swift have significant limitations and performance implications
- Syntactically, the cost of using one is hidden, and the similar spelling to generic constraints has caused many programmers to confuse existential types with generics

Code changes

```swift
protocol Animal {
    var name: String { get }
}

struct Dog: Animal {
    let name: String
}

let marksDog: Animal = Dog(name: "Max") ⚠️

func feed(_ animal: Animal) { ⚠️
    print("Feeding \(animal.name)")
}
```

Code changes

```swift
protocol Animal {
    var name: String { get }
}

struct Dog: Animal {
    let name: String
}

let marksDog: any Animal = Dog(name: "Max")

func feed(_ animal: any Animal) {
    print("Feeding \(animal.name)")
}
```

Timeframe

What it effects

- Protocols
- Protocol compositions
- Protocol metatypes
- Type aliases to protocols

Caveats

Not everything has changed

Dose not apply to Any and AnyObject

any is unnecessary for Any and AnyObject (unless part of a protocol composition). It’s already in the name, so it’s considered redundant

Dose not work with how we usually use optionals

```
let marksDog: any Animal? = Dog(name: "Max") // Dose not work

let marksDog: (any Animal)? = Dog(name: “Max") // Works
let marksDog: Optional<any Animal> = Dog(name: "Max") // Works
```

Existential types vs Generics

If possible, prefer generic functions over existential variants

```swift
func feed(_ animal: any Animal) {
    print("Feeding \(animal.name)")
}

func feed<A: Animal>(_ animal: A) {
    print("Feeding \(animal.name)")
}
```

Generic functions allows the compiler to create specialized variants for the concrete types

Demo

Sources

There’s more to be seen

Swift Evolution Proposal: SE-0335

https://github.com/apple/swift-evolution/blob/main/proposals/0335-existential-any.md

Hacking with Swift | What’s new in Swift 5.6?

https://www.hackingwithswift.com/articles/247/whats-new-in-swift-5-6

Swift Evolution Proposal: SE-0309

https://github.com/apple/swift-evolution/blob/main/proposals/0309-unlock-existential-types-for-all-protocols.md

Thank You