

Generic Enums that are Sometimes Optional

by Jeremy Kelleher

Generics

A dark, atmospheric photograph of a forest. The scene is dimly lit, with light filtering through the dense canopy of trees, creating a misty or foggy effect. In the center-right background, a small, dark silhouette of a person is walking away from the viewer down a path. The overall mood is mysterious and somber.

Extracting Behavior

```
func myFirstProject() {  
    print("Name = Jeremy Kelleher")  
    print("Age = 18")  
    let heightFeet = 5  
    let heightRemainderInches = 8  
    let totalHeightInches = (heightFeet * 12) + heightRemainderInches  
    print("Height in Inches = \(totalHeightInches)")  
    let totalHeightCM = (Double(totalHeightInches) * 2.54)  
    print("Height in CM = \(totalHeightCM)")  
}
```

myFirstProject()

Name = Jeremy Kelleher

Age = 18

Height in Inches = 68

Height in CM = 172.72

Extracting Behavior

```
func generalizedFirstProject(name: String, age: Int, heightFeet: Int, heightRemainderInches: Int) {  
    print("Name = \(name)")  
    print("Age = \(age)")  
    let totalHeightInches = (heightFeet * 12) + heightRemainderInches  
    print("Height in Inches = \(totalHeightInches)")  
    let totalHeightCM = (Double(totalHeightInches) * 2.54)  
    print("Height in CM = \(totalHeightCM)")  
}
```

```
generalizedFirstProject(name: "Jeremy Kelleher", age: 18, heightFeet: 5, heightRemainderInches: 8)
```

Name = Jeremy Kelleher

Age = 18

Height in Inches = 68

Height in CM = 172.72

```
class Apple { }
```

```
class ApplePickingBag {  
    var apples = [Apple]()  
    func add(apple: Apple) {  
        apples.append(apple)  
    }  
}
```

```
class Textbook { }
```

```
class TextbookBag {  
    var textbooks = [Textbook]()  
    func add(textbook: Textbook) {  
        textbooks.append(textbook)  
    }  
}
```

I should put
that **Thing** in
my **Bag**?

```
class Bag<Thing> {...}
```



```
class Bag<Thing> {  
    var things = [Thing]()  
    func add(thing: Thing) {  
        things.append(thing)  
    }  
}
```

```
let yayFallIsHere = Bag<Apple>()
```


```
class Bag<Thing> {  
    var things = [Thing]()  
    func add(thing: Thing) {  
        things.append(thing)  
    }  
}
```

```
let yayFallIsHere = Bag<Apple>()
```


```
class Bag<Apple> {  
    var things = [Apple]()  
    func add(thing: Apple) {  
        things.append(thing)  
    }  
}
```

```
let booSchoolIsTheWorst = Bag<Textbook>()
```

```
let completeConfusion = Bag()
```

 Generic parameter 'Thing' could not be inferred

```
let bagOfBruisedApples = Bag<Apple & TextBook>()
```

 Protocol-constrained type cannot contain class 'TextBook' because it already contains class 'Apple'

```
protocol Clothing { }
```


```
class LightClothingItem: Clothing { }
```

```
class DarkClothingItem: Clothing { }
```

```
class Rock { }
```

```
class WashingMachine<ClothingItem: Clothing> {  
    func wash(dirtyClothes: [ClothingItem]) { }  
}
```

```
let brockenWashingMachine = WashingMachine<Rock>()
```

 Type 'Rock' does not conform to protocol 'Clothing'



Bougie Way to Wash Clothes


```
protocol Clothing { }
```

```
class LightClothingItem: Clothing { }
```

```
class DarkClothingItem: Clothing { }
```

```
class WashingMachine<ClothingItem: Clothing> {  
    func wash(dirtyClothes: [ClothingItem]) { }  
}
```

```
let whiteShirtForCareerFairs = LightClothingItem()
```

```
let darkShirtForParties = DarkClothingItem()
```

```
let lightsWashingMachine = WashingMachine<LightClothingItem>()  
lightsWashingMachine.wash(dirtyClothes: [whiteShirtForCareerFairs])
```

```
let darksWashingMachine = WashingMachine<DarkClothingItem>()  
darksWashingMachine.wash(dirtyClothes: [darkShirtForParties])
```

```
protocol Clothing { }
```

```
class LightClothingItem: Clothing { }
```

```
class DarkClothingItem: Clothing { }
```

```
class WashingMachine<ClothingItem: Clothing> {  
    func wash(dirtyClothes: [ClothingItem]) { }  
}
```

```
darksWashingMachine.wash(dirtyClothes: [whiteShirtForCareerFairs])
```

❗ Cannot convert value of type 'LightClothingItem' to expected element type 'DarkClothingItem'



```
lightsWashingMachine.wash(dirtyClothes: [darkShirtForParties])
```

❗ Cannot convert value of type 'DarkClothingItem' to expected element type 'LightClothingItem'





the College Student Method

```
class OmniWashingMachine {  
    func wash(dirtyClothes: [Clothing]) { }  
}
```

```
let everythingWashingMachine = OmniWashingMachine()
```

```
everythingWashingMachine.wash(dirtyClothes: [whiteShirtForCareerFairs])
```

```
everythingWashingMachine.wash(dirtyClothes: [darkShirtForParties])
```

```
everythingWashingMachine.wash(dirtyClothes: [whiteShirtForCareerFairs, darkShirtForParties])
```




the
Way-You-Are-Supposed-to-Wash-Clothes
Method

```
class NoBlendingColorsWashingMachine {  
    func wash<ClothingItem: Clothing>(dirtyClothes: [ClothingItem]) { }  
}  
  
let betterWashingMachine = NoBlendingColorsWashingMachine()  
  
betterWashingMachine.wash(dirtyClothes: [whiteShirtForCareerFairs])  
  
betterWashingMachine.wash(dirtyClothes: [darkShirtForParties])  
  
betterWashingMachine.wash(dirtyClothes: [whiteShirtForCareerFairs, darkShirtForParties])
```

❗ In argument type '[Any]', 'Any' does not conform to expected type 'Clothing'



A dark, atmospheric photograph of a forest. The scene is shrouded in a thick, pale fog or mist. Tall, slender trees with dark trunks and branches are visible, their leaves creating a dense canopy. In the center-right of the image, a small, dark silhouette of a person is walking away from the viewer down a path. The overall mood is mysterious and somber.

Enums

```
enum Major {  
    case computerScience  
    case spanish  
    case biochemistry  
}  
  
let myMajor = Major.computerScience  
  
func willIHaveProfessorBarington(in major: Major) -> Bool {  
    switch major {  
    case .computerScience:  
        return true  
    default:  
        return false  
    }  
}  
  
willIHaveProfessorBarington(in: .biochemistry) // returns false
```



```
enum FlexibleMajor {
    case computerScience
    case spanish
    case biochemistry
    case bdic(focus: String)
}

let coolMajor: FlexibleMajor = .bdic(focus: "iOS App Development")

func isCoolestKidOnCampus(myMajor major: FlexibleMajor) -> Bool {
    switch major {
    case .bdic(focus: let focusDescription) where focusDescription == "iOS App Development":
        return true
    default:
        return false
    }
}
```

Optionals

A dark, atmospheric photograph of a forest. The scene is dimly lit, with a misty or foggy atmosphere. Tall, slender trees line a path that leads into the distance. In the far distance, a small, dark silhouette of a person is walking away from the viewer. The overall mood is mysterious and somber.

Check for Existence

```
struct WifiNetwork { }
```

```
let wifiAtAppleStore: WifiNetwork = AppleWifi()
```

```
var wifiAtUMass: WifiNetwork?
```

```
wifiAtUMass = nil
```

```
wifiAtUMass = Eduroam() // probably still nil
```



```
enum Optional<Wrapped> {  
    case none  
    case some(Wrapped)  
}
```

```
enum Optional<Wrapped> {  
    case none  
    case some(Wrapped)  
}
```

```
var wifiAtUMass: WifiNetwork?
```

```
var wifiAtUMass: Optional<WifiNetwork>
```

```
wifiAtUMass = nil
```

```
wifiAtUMass = Optional.none
```

```
wifiAtUMass = Eduroam()
```

```
wifiAtUMass = Optional.some(Eduroam())
```

```
enum Optional<Wrapped> {  
    case none  
    case some(Wrapped)  
}
```

```
func watchAppleKeynote(on maybeWifiNetwork: WifiNetwork?) {  
  
    switch maybeWifiNetwork {  
    case .some(let definiteWifiNetwork):  
        // let's hope for some new MacBooks  
        visitAppleDotCom(on: definiteWifiNetwork)  
    case .none:  
        // no keynote for you  
        return  
    }  
  
}
```

```
func visitAppleDotCom(on wifiNetwork: WifiNetwork) { }
```

```
enum Optional<Wrapped> {  
    case none  
    case some(Wrapped)  
}
```

```
func watchAppleKeynote(on maybeWifiNetwork: WifiNetwork?) {
```

```
    switch maybeWifiNetwork {  
    case .some(let definiteWifiNetwork):
```

```
        // let  
        visit
```

Declaration

```
let definiteWifiNetwork: (WifiNetwork)
```

Declared In

[Swift 4 Generics.playground](#)


```
        work)
```

```
    case .none:  
        // no keynote for you  
        return  
}
```

```
}
```

```
func visitAppleDotCom(on wifiNetwork: WifiNetwork) { }
```

```
func visitAppleDotCom(on wifiNetwork: WifiNetwork) { }
```

```
func watchAppleKeynote(on maybeWifiNetwork: WifiNetwork?) {  
    visitAppleDotCom(on: maybeWifiNetwork )  
     Value of optional type 'WifiNetwork?' must be unwrapped to a value of type 'WifiNetwork'  
}
```



```
func visitAppleDotCom(on wifiNetwork: WifiNetwork) { }
```

```
func watchAppleKeynote(on maybeWifiNetwork: WifiNetwork?) {  
    visitAppleDotCom(on: maybeWifiNetwork!)  
}
```

```
watchAppleKeynote(on: nil)
```



Fatal error: Unexpectedly found nil while unwrapping an Optional value

```
func watchAppleKeynote(on maybeWifiNetwork: WifiNetwork?) {  
    if maybeWifiNetwork != nil {  
        visitAppleDotCom(on: maybeWifiNetwork!)  
    } else {  
        return  
    }  
}
```

```
func watchAppleKeynote(on maybeWifiNetwork: WifiNetwork?) {  
    if let definiteWifiNetwork = maybeWifiNetwork {  
        visitAppleDotCom(on: definiteWifiNetwork)  
    } else {  
        return  
    }  
}
```

```
func watchAppleKeynote(on maybeWifiNetwork: WifiNetwork?) {  
    guard let definiteWifiNetwork = maybeWifiNetwork else {  
        return  
    }  
  
    visitAppleDotCom(on: definiteWifiNetwork)  
}
```

Future Investigation

```
// optional chaining
var commentText: String?
let commentLabel: UILabel = UILabel()
commentLabel.text = commentText?.uppercased()
```

```
// nil coalescing
var likeEmoji: String?
print(likeEmoji ?? "👍")
```

```
enum State {
    case noData
    case requestingData
    case hasData(Data)
}
```

On GitHub!

https://github.com/jeremy6462/Teaching_Generics-Enums-Optionals

jeremykelleh@umass.edu