

# Swift講習会

## #2

Yuhel Tanaka



# 今回やること

- プロトコル
- 構造体
- 型とmutableについての理解
- **Optional型**

# プロトコル

きまりごと？？？なに？？？



# プロトコル

実装すべきメソッドやフィールドの一覧

プロトコル自体は**実装**を持っていない

```
protocol Car {  
    var weight:Double {set get}  
    var horsepower:Double {set get}  
    func run(speed:Int)  
}
```



# Obj-C時代のプロトコル との違い

- メソッドだけでなくフィールドも持てる
- クラスだけでなく構造体や列挙体にも適用可能

# プロトコルに適合させる

class クラス名:プロトコル名で適合させることができる

```
class Benz:Car {
    var weight:Double
    var horsepower:Double
    func run(speed:Int) {
        print(speed)
    }
    init(weight:Double, horsepower:Double) {
        self.weight = weight
        self.horsepower = horsepower
    }
}
```



# 構造体

Objective-Cでは影の薄いヤツ



# 構造体

Swiftではめっちゃ強化されています

```
struct Benz:Car {  
    var weight:Double  
    var horsepower:Double  
    func run(speed:Int) {  
        print(speed)  
    }  
    init(weight:Double, horsepower:Double) {  
        self.weight = weight  
        self.horsepower = horsepower  
    }  
}
```

※Swiftの構造体はメソッドを持てる



# 構造体とクラスの違い

- クラスは継承可能だが構造体は継承できない
- クラスは参照型, 構造体は値型

# 値型??参照型??

変数に何が格納されているかの違い

- 値型 -> 値**そのもの**が入っている
- 参照型 -> 値への**参照**が入っている

# 値型と参照型

- 値型

構造体, 列挙体, タプル

- 参照型

クラス, 関数, クロージャ

# 参照型の落とし穴 #1

```
class Car {  
    var name:String  
    init(name:String){  
        self.name = name  
    }  
}
```

```
let car = Car(name:"Crown")  
car.name = "Benz" // carの参照先を変更しているわけではないのでOK  
car = Car(name:"Mazda") // carの参照先を変更するのでNG  
print(car.name) // "Benz"
```



# 参照型の落とし穴 #2

```
class Car {  
    var name:String  
    init(name:String){  
        self.name = name  
    }  
}
```

```
let car1 = Car(name:"Crown")  
let car2 = car1 // 参照をコピー  
car2.name = "Benz"  
print(car1.name, car2.name) // "Benz" "Benz"
```



# 解決策

- (クラスの)プロパティを読み取り専用にする
  - **immutable**にしよう！
- 構造体を使う
  - 値の受け渡しの際に**値自体がコピー**される

# immutableとmutable

ってなに？



# immutableなオブジェクト

作成後にその状態を変えられないオブジェクト

```
// Example
class Car {
    let name:String
    init(name:String) {
        self.name = name
    }
}

let name = "Benz" // immutable
let car = Car(name:name) // immutable
// carクラスのプロパティは変更できない(letなので)
```



# immutableにする方法

値型 -> letを使う

参照型 -> プロパティを初期化以外のタイミングで変更できなくする

# Optional型

ってなに？



# Optional型

値があるかnilかのいずれかを表す型

-> Optional型にはnilが入っている可能性がある

-> **Optional型でない型にはnilを代入できない**

```
var name1:String? // OptionalなString型  
name1 = nil // OK
```

```
var name2:String // Non-OptionalなString型  
name2 = nil //Error
```



# Unwrap

Optional型の値の**中身を取り出す**操作

- Forced Unwrap
- Implicitly Unwrapped Optionalを使う
- guard let文を使う
- if let文を使う
- ??演算子を使う

# Unwrap

Optional型の値の**中身を取り出す**操作

- **Forced Unwrap**
- Implicitly Unwrapped Optionalを使う
- guard let文を使う
- if let文を使う
- ??演算子を使う

# Forced Unwrap

- 無理やりUnwrapする
- Optional型の変数に『!』をつける
- 中身がnilだったらクラッシュする

```
var name:String?  
print(name!) // クラッシュ
```



# Unwrap

Optional型の値の中身を取り出す操作

- Forced Unwrap
- **Implicitly Unwrapped Optional**を使う
- guard let文を使う
- if let文を使う
- ??演算子を使う

# Implicitly Unwrapped Optional

- !をつけて定義するOptional型
- 勝手にUnwrapされる
- 中身がnilだったらクラッシュする

```
var num1:Int!  
num1 = 10  
print(num1 + 5) // 15
```

```
var num2:Int?  
num2 = 10  
print(num2! + 5) // 15
```





# Unwrap

Optional型の値の**中身を取り出す**操作

- Forced Unwrap
- Implicitly Unwrapped Optionalを使う
- **guard let文**を使う
- if let文を使う
- ??演算子を使う

# guard-let文

Optionalの中身がnilだったときに処理を抜きたい場合に使う

※guard-letのelse節には**break**か**return**が必須

```
var s:String? = "Tanaka" // s は Optional
guard let name = s else {return} // sがnullだったときreturnする
print(name) // name は Non-Optional
```

# Unwrap

Optional型の値の**中身を取り出す**操作

- Forced Unwrap
- Implicitly Unwrapped Optionalを使う
- guard let文を使う
- **if let文を使う**
- ??演算子を使う

# if-let文

Optionalの中身がnilかどうかで処理を分けたいときに使う

Optional Bindingとも呼ばれる

```
var s:Int? = "Tanaka" // s は Optional
if let name = s {
    print(name) // name は Non-Optional
}
else {
    print("nil")
}
```



# Unwrap

Optional型の値の**中身を取り出す**操作

- Forced Unwrap
- Implicitly Unwrapped Optionalを使う
- guard let文を使う
- if let文を使う
- ??演算子を使う

# ??演算子を使う

別名: Nil Coalescing Operator

$A ?? B \rightarrow A! \text{ if } A \neq \text{nil} \text{ else } B$

```
var s:Int? = "Tanaka" // s は Optional
let name = s ?? "nil" // s が nilのときは"nil"
// そうでないときは s!
print(name) // "tanaka"
```

# 終わり

第2回は終わりです。

復習をしっかりとしましょう。