

Swift OOP Eğitimi

İleri Swift

Kasım ADALAN

Elektronik ve Haberleşme Mühendisi
Freelance Software Developer

Eğitim İçeriği

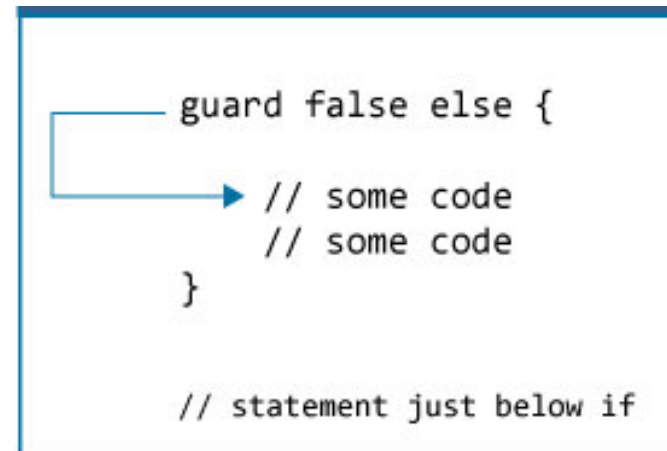
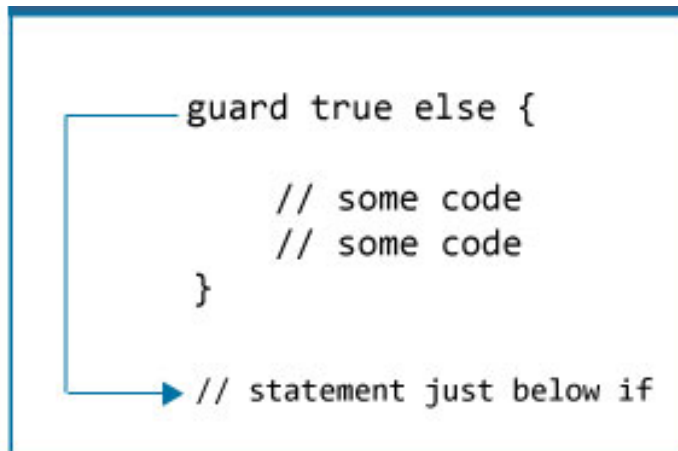
1. Guard
2. Exception Nedir ?
3. Do - Try – Catch Yapısı
4. Thread

guard

Guard

- Guard if yapısı ile aynıdır.
- If gibi koşula göre işlem yapar.
- Guard if tersi gibi çalışır.
- Koşul false olduğu durumda çalışır.True durumu yoktur.
- **return** veya **throw** ifadesi ile kullanılır.Bundan dolayı genelde **metod** içinde yer alırlar.

```
guard condition else {  
    statements  
}
```



if

```
func kisiTanıma(ad:String){  
    if ad == "Ahmet" {  
        print("Merhaba Ahmet")  
    }else{  
        print("Tanınmayan Kişi")  
    }  
}
```

```
kisiTanıma(ad: "Ahmet")
```

Guard

```
func kisiTanıma(ad:String){  
    guard ad == "Ahmet" else {  
        print("Tanınmayan Kişi")  
        return  
    }  
    print("Merhaba Ahmet")  
}
```

```
kisiTanıma(ad: "Ahmet")
```

Return :

Hata oluştuğunda veya şart sağlanmadığında metodu bitirir.

Okunabilirliği Artırır

```
if let username = usernameField.text, username != "" {  
    if let email = emailField.text, email != "" {  
        if let password = passwordField.text, password != "" {  
            // do awesome stuff here  
        }  
    }  
}
```

İf KULLANIMI

```
guard let username = usernameField.text else {  
    throw SignupError.NoUsername  
}  
guard let email = emailField.text else {  
    throw SignupError.NoEmail  
}  
guard let password = passwordField.text else {  
    throw SignupError.NoPassword  
}
```

GUARD KULLANIMI

```
print("username: \username, "email: \email, password: \password")
```

Optional ifade

if

```
func büyükHarfYap(str:String?){  
    if let temp = str {  
        print("\(temp.uppercased())")  
    }else{  
        Otomatik unwrapping  
        print("str nil dir.İşlem yapılamaz")  
        return  
    }  
}
```

buyukHarfYap(str:nil)

Guard

```
func büyükHarfYap(str:String?){  
    guard let temp = str else {  
        print("str nil dir.İşlem yapılamaz")  
        return  
    }  
    Otomatik unwrapping  
    print("\(temp.uppercased())")  
}
```

buyukHarfYap(str:nil)

Not : temp değeri guard'dan geçtikten
sonra kullanılabilir ve doğal olarak
unwrapping olur.

str nil dir.İşlem yapılamaz

Çoklu Şartlar

```
func büyükHarfYap(str:String?){  
    guard let temp = str , temp.count > 0 else {  
        print("str nil dir.İşlem yapılamaz")  
        return  
    }  
    print("\n(temp.uppercased())")  
}
```

```
büyükHarfYap(str:"")
```

str nil dir.İşlem yapılamaz

Exception Nedir ?

- Derleyici Hatası (Compiler Error) : Derleme öncesi yakalanan hatalar
 - Örn: karakter hataları, sentaks hatası, ...
- Hata (Exception): Çalışma anında (runtime) gerçekleşen hatalar
 - Örn: Sistem hataları, cihaz hataları, dosya bulunamadı, dizi indeksi aşıldı, ...

Exception Hata Ayıklama

Do try catch

- Derleme sırasında oluşabilecek hatalar için kullanılır.
- Genelde swift input – output işlemleri için kullanılır. Yani veri alışveriş işlemlerinde kullanılır.
- Kullanılacak yer mutlaka hata fırlatmalıdır.

```
do {  
    try expression  
    statements  
} catch pattern 1 {  
    statements  
} catch pattern 2 where condition {  
    statements  
}
```

***Not : Hata olmasını beklediğimiz satıra try ifadesini koyarız.
Block halinde bir hata bekleme durumu olmaz.***

Do try catch bloğu

Kendisinden önceki ifadenin throw özelliği olmalıdır.

```
var s1 = 10  
var s2 = 0
```

Kendisinden önceki ifadeden sorumludur.

```
do {  
    let sonuc = try bolme(s1: s1, s2: s2) // hata oluşma ihtimali olan kod  
    print(sonuc)  
} catch Hatalar.sifiraBölünmeHatasi { //Bu hata ile bizim hatamız eşleşirse mesaj vericek.  
    print("Sayı sifıra bölünemez")  
}
```

Hata oluşursa catch bloğuna geçer.

throw özelliği : Hatayı Fırlatmak

- Kodları koruma altına almaya zorlama işlemidir.
- Metotlara eklenen bu özellik bu metodu kullanan kişiye hata olabileceğini ve bunu koruma altına almasına zorlamaktadır.
- Eğer metodun **throw** özelliği yoksa **try** ifadelerinde kullanılamaz

```
func bolme(s1:Int,s2:Int) throws -> Int { //hata fırlatma
    if s2 == 0 { //s2 0 olursa hata fırlatacak
        throw Hatalar.sifiraBölünmeHatasi //Oluşacak hata
    }
    return s1 / s2
}
```



Oluşacak hata için enum oluşturmalıyız.

```
enum Hatalar: Error { //Enum kendi hatamızı oluşturmak için
    case sifiraBölünmeHatasi
}
```

Örnek

```
enum Hatalar: Error { //Enum kendi hatamızı oluşturmak için
    case sifiraBölünmeHatasi
}

func bolme(s1:Int,s2:Int) throws -> Int { //hata fırlatma
    if s2 == 0 {//s2 0 olursa hata fırlatacak
        throw Hatalar.sifiraBölünmeHatasi//Oluşacak hata
    }
    return s1 / s2
}

var s1 = 10
var s2 = 0

do {
    let sonuc = try bolme(s1: s1, s2: s2) // hata oluşma ihtimali olan kod
    print(sonuc)
} catch Hatalar.sifiraBölünmeHatasi { //Bu hata ile bizim hatamız eşleşirse mesaj vericek.
    print("Sayı sifıra bölünemez")
}
```

try?

- Hata yok sayılır veya görmezden gelinir fakat hata oluşursa değişkeni *nil* yapar.
- **Do** ve **catch** bloğuna ihtiyaç yoktur.
- try ? dan önce yer alan ifade **throw** özelliği olan metod olmalıdır.

```
var s1 = 10
var s2 = 2

let sonuc1 = try? bolme(s1: s1, s2: s2)

if sonuc1 == nil {
    print("hata oluştuğu için sonuc1 içinde nil")
}else{
    print("Hata Yok : \(sonuc1!)")
}
```

Hata olursa sonuc1
içine nil verisi atılır

Bu ifade throw özelliği
olan metod olmalıdır.

try!

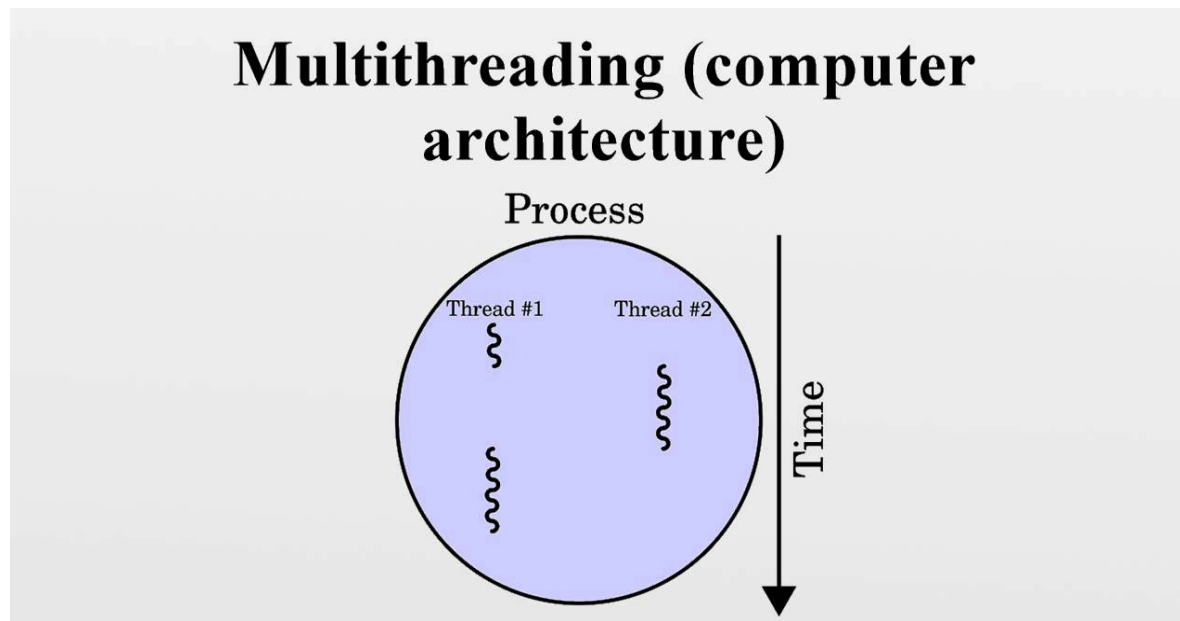
- Hata olma ihtimali var fakat neredeyse hata hiç olmayacaktır. Olursa da hata bastırılır.
- **Do** ve **catch** bloğuna ihtiyaç yoktur.
- Kullanırken çok dikkatli olunmalıdır.

```
if let filename = Bundle.main.path(forResource: "input", ofType: "txt") {  
    let str = try! String(contentsOfFile: filename)  
}
```


Dispatch Queue (Thread)

Dispatch Queue (Thread)

- Thread'ler iş parçacıklarıdır.
- Aynı anda birden fazla iş yapılabilir.



Senkron Thread – Senkronize İşlem

```
let queue = DispatchQueue(label: "etiket")
```

```
queue.sync {  
    for i in 1...10 {  
        print("Thread : \(i)")  
    }  
}
```

```
for i in 100...110 {  
    print("Main : \(i)")  
}
```

Main Thread

```
Thread : 1  
Thread : 2  
Thread : 3  
Thread : 4  
Thread : 5  
Thread : 6  
Thread : 7  
Thread : 8  
Thread : 9  
Thread : 10  
Main : 100  
Main : 101  
Main : 102  
Main : 103  
Main : 104  
Main : 105  
Main : 106  
Main : 107  
Main : 108  
Main : 109  
Main : 110
```

Asenkron Thread – Asenkron İşlem

```
let queue = DispatchQueue(label: "etiket")
```


```
queue.async {  
    for i in 1...10 {  
        print("Thread : \(i)")  
    }  
}
```

```
for i in 100...110 {  
    print("Main : \(i)")  
}
```

Main Thread

```
Thread : 1  
Main : 100  
Main : 101  
Thread : 2  
Main : 102  
Main : 103  
Thread : 3  
Main : 104  
Main : 105  
Main : 106  
Main : 107  
Main : 108  
Main : 109  
Main : 110  
Thread : 4  
Thread : 5  
Thread : 6  
Thread : 7  
Thread : 8  
Thread : 9  
Thread : 10
```

Thread Öncelik Seviyeleri

- `userInteractive` **Yüksek**
 - `userInitiated`
 - `default`
 - `utility`
 - `background`
 - `unspecified` **Düşük**
- 

Örnek : Aynı Öncelik – Farklı Thread

```
let queue1 = DispatchQueue(label: "Thread1", qos:DispatchQoS.userInitiated)
let queue2 = DispatchQueue(label: "Thread2", qos:DispatchQoS.userInitiated)
```

```
queue1.async {
    for i in 1...10 {
        print("A : \(i)")
    }
}
```

```
queue2.async {
    for i in 100...110 {
        print("B : \(i)")
    }
}
```

```
A : 1
B : 100
B : 101
B : 102
B : 103
B : 104
B : 105
B : 106
A : 2
B : 107
B : 108
A : 3
B : 109
B : 110
A : 4
A : 5
A : 6
A : 7
A : 8
A : 9
A : 10
```

Örnek : Farklı Öncelik – Farklı Threadler

```
let queue1 = DispatchQueue(label: "Thread1", qos:DispatchQoS.userInitiated)
let queue2 = DispatchQueue(label: "Thread2", qos:DispatchQoS.background)
```

```
queue1.async {
    for i in 1...10 {
        print("A : \(i)")
    }
}
```

```
queue2.async {
    for i in 100...110 {
        print("B : \(i)")
    }
}
```

```
A : 1
B : 100
A : 2
A : 3
A : 4
A : 5
A : 6
A : 7
A : 8
A : 9
A : 10
B : 101
B : 102
B : 103
B : 104
B : 105
B : 106
B : 107
B : 108
B : 109
B : 110
```

Örnek : Main Thread Her zaman Önceliklidir.

```
let queue1 = DispatchQueue(label: "Thread1", qos:DispatchQoS.userInitiated)
let queue2 = DispatchQueue(label: "Thread2", qos:DispatchQoS.background)

queue1.async {
    for i in 1...10 {
        print("A : \(i)")
    }
}

queue2.async {
    for i in 100...110 {
        print("B : \(i)")
    }
}

for i in 1000...1010 {
    print("Main : \(i)")
}
```

A : 1
B : 100
Main : 1000
A : 2
Main : 1001
B : 101
Main : 1002
B : 102
Main : 1003
B : 103
Main : 1004
B : 104
Main : 1005
B : 105
Main : 1006
Main : 1007
Main : 1008
A : 3
Main : 1009
A : 4
Main : 1010
A : 5
A : 6
A : 7
A : 8
A : 9
A : 10
B : 106
B : 107
B : 108
B : 109
B : 110

Örnek :

Tek Thread Aynı öncelik ise sırayla bitene kadar çalışır.

```
let queue1 = DispatchQueue(label: "Thread1", qos:DispatchQoS.utility)

queue1.async {
    for i in 1...10 {
        print("A : \(i)")
    }
}

queue1.async {
    for i in 100...110 {
        print("B : \(i)")
    }
}

queue1.async {
    for i in 1000...1010 {
        print("C : \(i)")
    }
}
```

```
A : 1
A : 2
A : 3
A : 4
A : 5
A : 6
A : 7
A : 8
A : 9
A : 10
B : 100
B : 101
B : 102
B : 103
B : 104
B : 105
B : 106
B : 107
B : 108
B : 109
B : 110
C : 1000
C : 1001
C : 1002
C : 1003
C : 1004
C : 1005
C : 1006
C : 1007
C : 1008
C : 1009
C : 1010
```

Thread Delay - Gecikme Oluşturma

```
var gecikmeSaniye:DispatchTimeInterval = .seconds(2)
var gecikmeMilliSaniye:DispatchTimeInterval = .milliseconds(2)
var gecikmeMikroSaniye:DispatchTimeInterval = .microseconds(2)
var gecikmeNanoSaniye:DispatchTimeInterval = .nanoseconds(2)
```

Örnek

```
let queue1 = DispatchQueue(label: "Thread", qos: DispatchQoS.userInitiated)

var gecikmeMilliSaniye:DispatchTimeInterval = .milliseconds(2)

queue1.asyncAfter(deadline: .now() + gecikmeSaniye) {
    // .now() şimdiki zamanı temsil eder
    // şimdiki zamana 2 saniye gecikme eklenir ve yapılacak işlem geçikmeli olur.
    print(Date())//print içindeki Date() 2 saniye öncesini gösterir.
}
```

MainDispatch Kullanımı

```
if let data = imageData {  
    print("Did download image data")  
  
    DispatchQueue.main.async {  
        self.imageView.image = UIImage(data: data)  
    }  
}
```

UIView'lere veri yüklerken veya işlerken sık kullanırız.

Teşekkürler...



kasım-adalan



kasimadalan@gmail.com



kasimadalan