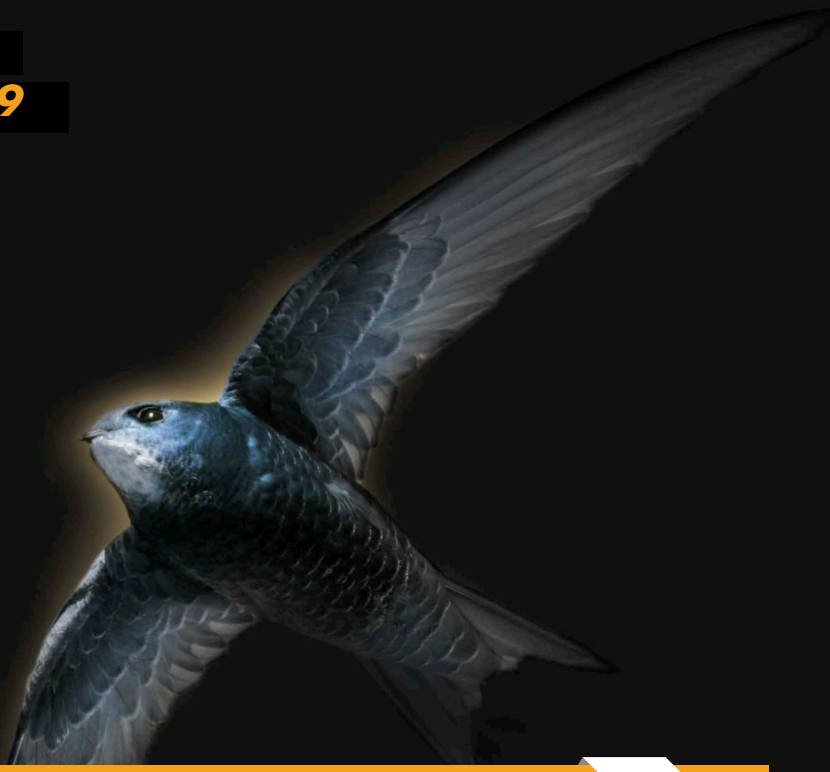
Hanbit eBook

Realtime 79



Swift 2.0 대응

만들면서 배우는 Swift

스위프트로 시작하는 iOS 개발

야곰 지음

한빛미디어 Hanbit Media, Inc.

만들면서 배우는 Swift

스위프트로 시작하는 iOS 개발

만들면서 배우는 Swift 스위프트로 시작하는 iOS 개발

초판발행 2014년 07월 30일
3판발행 2015년 10월 01일

지은이 야곰 / **펴낸이** 김태헌
펴낸곳 한빛미디어(주) / **주소** 서울시 마포구 양화로 7길 83 한빛미디어(주) IT출판부
전화 02-325-5544 / **팩스** 02-336-7124
등록 1999년 6월 24일 제10-1779호
ISBN 978-89-6848-675-3 15000 / **정가** 9,900원

총괄 배용석 / **책임편집** 김창수 / **기획·편집** 정지연
디자인 표지 여동일, 내지 스튜디오 [밈], 조판 최송실
영업 김형진, 김진불, 조유미 / **마케팅** 박상용

이 책에 대한 의견이나 오탈자 및 잘못된 내용에 대한 수정 정보는 한빛미디어(주)의 홈페이지나 아래 이메일로 알려주십시오.
한빛미디어 홈페이지 www.hanbit.co.kr / **이메일** ask@hanbit.co.kr

Published by HANBIT Media, Inc. Printed in Korea

저자 소개

지은이_야곰

yagom's blog(http://blog.yagom.net/)를 운영하는 iOS 개발 블로거로, iOS, OS X 개발자 커뮤니티인 맥부기(http://cafe.naver.com/mcbugi)에서 강좌를 연재하고 있다. 컴퓨터 교육을 전공했으며 비전공자와 학생들에게 컴퓨터 지식을 좀 더 쉽고 재미있게 알리는 데 관심이 많다. 2010년부터 iOS 개발을 시작하여 현재까지도 iOS 관련 개발에 열정을 쏟고 있다. 내일 걱정은 모레 하는 것이 좋다고 생각하며 긍정적인 마음가짐을 빼면 시체라고 말한다. 스스로 개발자라고 생각하지 않는 것을 보면 괴짜임이 틀림없다. 무엇보다 여행과 요리를 좋아한다.

저자 서문

우리는 급변하는 세상 속에 살고 있습니다. 언제 또 변할지 모르지만, 현재는 스마트폰과 PC가 없으면 살아가기 어려운 세상이 되어버렸습니다. 물론 세상을 등지고 살겠다면 모르겠지만, 이 책을 보시는 분들은 그렇지 않은 분들이 대부분이라고 생각합니다. 저는 사실 급변하는 세상에 뛰어들고자 모바일 개발을 시작한 것은 아니지만, 이제는 떼려야 뗄 수 없는 사이가 된 것은 분명합니다. 이 책을 읽고 계신 분들도 그러시겠지요.

그래서 iOS와 Mac OS 개발을 처음 시작하시는 분들을 비롯하여 현직에서 Objective-C로 개발하시는 분들께 Swift를 활용하는 방법에 대하여 조금이나마 도움을 드리고자 이 책을 썼습니다. 비록 모든 부분을 충족할 수 있는 방대한 분량은 아니지만, 충분히 힌트를 얻고 감을 잡으실 정도의 내용이 되리라 믿습니다. 부족한 부분은 언제든지 블로그(http://blog.yagom.net) 또는 이메일(yagomsoft@me.com)을 통하여 말씀해 주시면 좋겠습니다.

끝으로 항상 믿고 응원해 주시며 사랑을 아끼지 않는 우리 가족과 동료들, 이 책의 편집을 멋지게 해주신 한빛미디어 정지연 님, 함께 응원해주신 조희진 님과 이중민 님께 무한한 감사의 말을 전합니다.

사랑합니다.

글쓴이 **야곰**

대상 독자 및 참고사항

초급 **초중급** 중급 중고급 고급

이 책은 Swift로 아주 간단한 iOS 애플리케이션을 만들어보는 책입니다. 기본적으로 프로그래밍 개념을 알고 있으며 iOS 애플리케이션 개발에 입문하려는 초중급 개발자라면 읽을 수 있습니다. 이 책의 샘플 코드를 실행하려면 다음과 같은 환경이 갖춰져 있어야 합니다.

- Mac OS X 매버릭스 이상의 OS가 구동되는 매킨토시 컴퓨터
- Xcode 7을 사용할 수 있는 개발 환경

이 책의 소스 코드는 Swift 2.0 버전에 맞게 작성되었고, Xcode 7.0에서 테스트되었습니다. Cocoa Touch 프레임워크 또는 Swift 버전에 따라 다소 상이한 결과나 오류가 발생할 수 있습니다. iOS, Swift 또는 Xcode의 지난 버전에서 동작하는 코드를 원할 경우 Git에서 과거 내역을 살펴보시면 됩니다.

이 책의 샘플 코드는 다음에서 내려받을 수 있습니다.

- https://bitbucket.org/yagom/swift_yagom.git

한빛 리얼타임

한빛 리얼타임은 IT 개발자를 위한 전자책입니다.

요즘 IT 업계에는 하루가 멀다 하고 수많은 기술이 나타나고 사라져 갑니다. 인터넷을 아무리 뒤져도 조금이나마 정리된 정보를 찾는 것도 쉽지 않습니다. 또한 잘 정리되어 책으로 나오기까지는 오랜 시간이 걸립니다. 어떻게 하면 조금이라도 더 유용한 정보를 빠르게 얻을 수 있을까요? 어떻게 하면 남보다 조금 더 빨리 경험하고 습득한 지식을 공유하고 발전시켜 나갈 수 있을까요? 세상에는 수많은 종이책이 있습니다. 그리고 그 종이책을 그대로 옮긴 전자책도 많습니다. 전자책에는 전자책에 적합한 콘텐츠와 전자책의 특성을 살린 형식이 있다고 생각합니다.

한빛이 지금 생각하고 추구하는, 개발자를 위한 리얼타임 전자책은 이렇습니다.

1. eBook First - 빠르게 변화하는 IT 기술에 대해 핵심적인 정보를 신속하게 제공합니다.

   500페이지 가까운 분량의 잘 정리된 도서(종이책)가 아니라, 핵심적인 내용을 빠르게 전달하기 위해 조금은 거칠지만 100페이지 내외의 전자책 전용으로 개발한 서비스입니다. 독자에게는 새로운 정보를 빨리 얻을 수 있는 기회가 되고, 자신이 먼저 경험한 지식과 정보를 책으로 펴내고 싶지만 너무 바빠서 엄두를 못 내는 선배, 전문가, 고수 분에게는 보다 쉽게 집필할 수 있는 기회가 될 수 있으리라 생각합니다. 또한 새로운 정보와 지식을 빠르게 전달하기 위해 O'Reilly의 전자책 번역 서비스도 하고 있습니다.

2. 무료로 업데이트되는 전자책 전용 서비스입니다.

   종이책으로는 기술의 변화 속도를 따라잡기가 쉽지 않습니다. 책이 일정 분량 이상으로 집필되고 정리되어 나오는 동안 기술은 이미 변해 있습니다. 전자책으로 출간된 이후에도 버전 업을 통해 중요한 기술적 변화가 있거나 저자(역자)와 독자가 소통하면서 보완하여 발전된 노하우가 정리되면 구매하신 분께 무료로 업데이트해 드립니다.

3. 독자의 편의를 위해 DRM-Free로 제공합니다.

구매한 전자책을 다양한 IT 기기에서 자유롭게 활용할 수 있도록 DRM-Free PDF 포맷으로 제공합니다. 이는 독자 여러분과 한빛이 생각하고 추구하는 전자책을 만들어 나가기 위해 독자 여러분이 언제 어디서 어떤 기기를 사용하더라도 편리하게 전자책을 볼 수 있도록 하기 위함입니다.

4. 전자책 환경을 고려한 최적의 형태와 디자인에 담고자 노력했습니다.

종이책을 그대로 옮겨 놓아 가독성이 떨어지고 읽기 힘든 전자책이 아니라, 전자책의 환경에 가능한 한 최적화하여 쾌적한 경험을 드리고자 합니다. 링크 등의 기능을 적극적으로 이용할 수 있음은 물론이고 글자 크기나 행간, 여백 등을 전자책에 가장 최적화된 형태로 새롭게 디자인하였습니다.

앞으로도 독자 여러분의 충고에 귀 기울이며 지속해서 발전시켜 나가겠습니다.

차례

1 | 애플이 선택한 새로운 언어, Swift

1.1 Swift의 특징

Swift는 iOS와 OS X 애플리케이션을 개발할 수 있는 새로운 언어다. 혜성처럼 나타난 이 새로운 언어는 기존 Objective-C에 없던 새로운 기능을 많이 포함하고 있다. 하지만 아직은 갈 길이 많이 남은 불완전한 언어다. 애플은 Swift가 아직 완성되지 않은 언어라는 단서들을 곳곳에 남겨놓고 있다.

Swift는 Objective-C와 마찬가지로 LLVM으로 빌드되고, 코드 내부에서 C 또는 Objective-C와 혼용할 수 있다. 그리고 기존 Objective-C에서 사용하던 명명법을 그대로 사용하므로 기존 Objective-C 개발자들도 큰 어려움 없이 접근할 수 있다. 또한, 스크립트 언어처럼 실시간으로 컴파일하여 테스트할 수 있는 Playground라는 도구로 실시간으로 결과를 확인하며 코딩할 수도 있다. 이는 기존 스크립트 언어 개발자들이 다가오기 쉽도록 배려한 것으로 생각한다.

하지만 스크립트 언어처럼 자료형에 대한 제한이 없다고 생각한다면 오산이다. 오히려 Swift는 Objective-C보다 자료형에 관대하지 않다. 또한, 디버깅이 어렵다고 생각할 수 있으나 Swift는 컴파일하여 동작하는 언어이므로 기존 스크립트 언어보다 디버깅이 훨씬 수월하다.

1.2 이 책에서 살펴볼 내용

2장에서는 Swift의 문법을 간단히 살펴보고, 3장에서는 Swift로 간단한 앱을 만들어 본다. 4장에서는 Swift와 Objective-C 또는 C를 혼용하여 사용하는 방법을 소개한다. 5장에는 이 책을 훑어보기 위한 필수적인 개념들을 부록으로 실었다.

1.3 갖추고 있어야 할 개념

이 책은 독자들이 객체 지향의 개념을 알고 있다는 전제하에 설명하고 있으며, 기초적인 개발 경험이 있는 독자를 대상으로 한다. 기본으로 알고 있어야 하는 개념은 부록을 참고하고, 모르는 용어에 대해서는 검색해 보길 바란다.

자, 그럼 힘차게 Swift와 애플 애플리케이션의 세계로 출발해 보자. 🍏

2 | Swift 문법을 간략히 알아보자

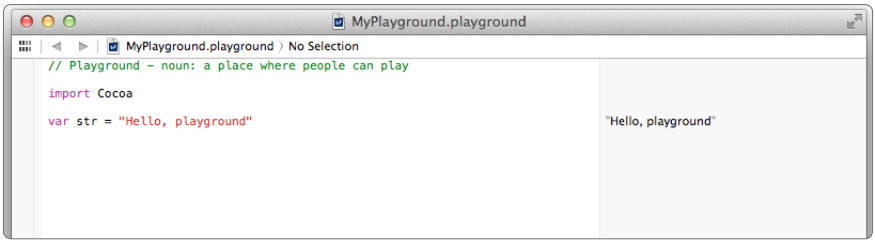
새로운 언어인 만큼 Swift는 나름 독특한 문법이 있다. 아주 기초적인 문법부터 약간은 독특한 문법까지 간략히 알아보겠다. 상세하고 자세한 문법은 애플의 Swift 문서 또는 좋은 레퍼런스 서적을 참고하자.

Swift 문법을 경험해 보기에 아주 안성맞춤인 놀이터가 있다. 바로 Xcode의 Playground라는 도구다. Xcode를 실행한 첫 화면에서 'Get started with a playground'를 선택하여 Playground를 생성한다.

[그림 2-1] Xcode 실행 화면

Playground가 실행되면 [그림 2-2]와 같은 화면이 보인다.

[그림 2-2] Playground 실행 화면



자, 그럼 이제 본격적으로 놀아 보자!

2.1 단순값

단순값Simple Value에는 크게 상수와 변수가 있다. 상수는 변하지 않는 값, 변수는 변할 수 있는 값이다. Swift에서는 기본으로 자료형을 명시하지만, 그렇게 하지 않아도 된다. 그리고 코드 마지막에 세미콜론(;)을 쓰지 않는다. 일단 막 저지르자.

Playground에 [코드 2-1]을 입력해 본다. 모든 예제 코드의 결과값은 Playground에서 실시간으로 확인할 수 있다.

[코드 2-1] 변수와 상수의 선언

```
var hi = "hi"
let hello = "hello"

hi = "bye"
hello = "bye"
```

'`hello = "bye"`'라는 코드는 에러를 내뿜을 것이다. `let`은 상수 키워드 즉, 변경할 수 없는 값이다. `var`는 변경할 수 있는 값인 변수다. 하지만 새로 만든 변수나 상수가 특정 자료형이라는 것을 알려주고 싶다면 다음과 같이 작성하면 된다.

[코드 2-2] 자료형의 명시

```
var itsInteger: Int = 3
let itsString: String = "string"
itsInteger = 3.0
```

세 번째 줄에서 에러가 발생한다. 분명 정수형이라고 알려줬음에도 실수를 대입했기 때문이다. 이 에러를 수정하려면 어떻게 해야 할까?

2.2 형 변환

자료형의 변환Casting은 다음과 같이 할 수 있다.

[코드 2-3] 명시적 형 변환 1

```
itsInteger = Int(3.0)
```

에러가 사라진다. 다음과 같이 정수를 문자열로도 형 변환할 수 있다.

[코드 2-4] 명시적 형 변환 2

```
let intToString: String = String(300)
```

2.3 문자열 포맷

문자열을 다루는 특정 방법이 있다. 경우에 따라 다른 문장들을 표현하고 싶을 때가 있을 것이다. 그럴 때에는 문자열 포맷을 사용할 수도 있고, 문자열을 합쳐 볼 수도 있다.

[코드 2-5] 다양한 문자열 사용

```
var myName: String = "yagom"
```

```
var introduceString: String = "My name is "

"\(introduceString)\(myName)"
introduceString + myName
```

문자열 안에 다른 변수의 값을 가져와서 넣고 싶다면 \(변수명)를 사용한다.

2.4 배열과 딕셔너리 그리고 세트

배열은 값의 목록이고, 딕셔너리는 키와 값이 쌍을 이룬 조합(해쉬맵)이다. 세트는 순서가 없는 키의 모음으로 유일한 값의 모음을 묶기 위해 사용한다(Swift 1.2 버전에서 추가). 세트가 배열과 다른 점은 중복 값이 들어갈 수 없고 순서가 없다는 것이다.

[코드 2-6] 배열과 딕셔너리 그리고 세트의 선언과 사용

```
var numberList = [1,2,3,4,5]                    // 배열 선언
var numberDic = [ "one":1, "two":2, "three":3 ] // 딕셔너리 선언
var numberSet = Set<Int>([1, 2, 3, 3, 3])       // 세트 선언
numberList[2]
numberDic["one"]
numberSet.contains(2)
```

2.5 반복문

반복문은 배열을 가장 손쉽게 제어하는 방법이다. 우선 기본적인 for 반복문을 구현해 보자. for의 조건문을 소괄호로 감싸는 것은 선택사항이다.

[코드 2-7] for 반복문 사용

```
for var i = 0 ; i < numberList.count ; i++
{
     numberList[i]
}
```

for-in 반복문도 사용할 수 있다. 필자는 딕셔너리의 아이템을 열거할 때 자주 사용한다.

[코드 2-8] for-in 반복문 사용 1

```
for number in numberDic
{
    number
}
```

for-in 반복문은 배열에서도 유용하다. [코드 2-9]는 [코드 2-7]과 정확히 같은 동작을 한다.

[코드 2-9] for-in 반복문 사용 2

```
for number in 0..<numberList.count
{
    number
}
```

for 반복문 외에도 while, do-while 반복문 등을 사용할 수 있다.

2.6 조건문

특정 조건에 따라 실행되는 코드를 생성할 수 있다. 먼저 if-else 조건문을 살펴보자. 기존 문법과 크게 다르지 않다.

[코드 2-10] if-else 조건문 사용

```
var number = 30
let hundred = 100
if number > hundred
{
```

```
        "number is over \(hundred)"
}
else
{
        "number is under \(hundred)"
}
```

switch 조건문도 사용할 수 있다. 특이하게 정수형이 아닌 실수, 문자열까지 체크한다. 또한, 조건마다 break 키워드로 종료할 필요가 없다.

[코드 2-11] switch 조건문 사용

```
var string = "hi"

switch string
{
        case "hi":
            "hi!!!!!!!!"

        case "hello":
            "hello!!!"

        default:
            "bye"
}
```

2.7 함수

함수는 특정 동작을 하는 코드의 집합이다. 함수는 특정 값을 전달받아 특정 결과물을 내놓는다. 전달받는 값은 파라미터, 결과값은 반환값이라고 한다. 함수의 구현 방법은 'func 함수명(파라미터명1:자료형, 파라미터명2:자료형, …) -> 반환값의 자료형 { 실행코드 }'다. 예제를 살펴보자.

[코드 2-12] 함수의 구현과 사용

```
func sumOf(number1:Int, number2:Int) -> Int
{
      return number1 + number2
}

sumOf(100, 30)
```

[코드 2-12]는 number1과 number2라는 이름으로 Int형 자료를 받아서 합한 후 Int형으로 반환하는 sumOf라는 함수를 보여준다. 이 함수는 파라미터의 개수를 제한해서 받지 않고 배열 등으로 한 번에 여러 개를 받을 수도 있다.

[코드 2-13] 다수의 파라미터를 전달받는 함수

```
func sumOf(numbers:Int...) -> Int
{
      var sum = 0

      for number in numbers
      {
         sum = sum + number
      }
      return sum
}

sumOf(100, 30, 50, 70, 33)
```

반환하는 값이 하나라서 불편하다면 여러 개의 값을 반환할 수도 있다.

[코드 2-14] 여러 개의 값을 반환하는 함수

```
func castNumber(number: Int) -> (floatValue: Float, boolValue: Bool,
stringValue: String)
{
```

```
    return (Float(number), Bool(number), String(number))
}

let castResult = castNumber(100)

castResult.floatValue
castResult.boolValue
castResult.stringValue
```

2.8 느낌표와 물음표

Swift에는 다른 언어에서 보기 힘든 변수 뒤에 느낌표(!) 또는 물음표(?)가 붙는 경우가 있다. 이는 nil이라는 독특한 녀석으로부터 출발한다. 이미 Objective-C를 경험한 독자라면 nil이 익숙할 것이다.

하지만 Swift의 nil은 Objective-C의 nil과는 사뭇 다르다. Objective-C의 nil은 객체에만 할당할 수 있는 값 즉, 포인터에만 해당하고, 포인터가 아닌 값에는 NSNotFound와 같은 상수값으로 표현해야 한다. Swift에서 nil은 '없음' 그 자체를 의미한다. 객체든 아니든 모든 자료형에 사용될 수 있다.

nil 값이 할당된 변수에 잘못 접근하면 런타임 에러가 발생하는데, Swift에서는 이를 미리 방지하고자 Optional이라는 기능을 도입했다. 이것은 애플이 강조한 Swift의 'Safe' 특징을 보여주는 예다. Optional은 변수에 nil이 들어갈 수 있다는 것을 의미하고, 자료형 뒤에 물음표(?)를 붙여서 표현한다.

[코드 2-15] Optional 변수와 일반 변수

```
var intValueNonOptional: Int = 0
var intValueOptional: Int?
```

[코드 2-15]에서 intValueNonOptional은 Optional 변수가 아니므로 nil 할당에

주의해야 한다. 두 변수 모두 초기화 값이 없으면 nil로 초기화되지만, Optional 변수가 아닌 일반 변수를 선언할 때는 반드시 초기화하는 것이 좋다. Optional 변수에 유효한 값이 있음을 확신하고 값을 꺼내올 때 Forced-Unwrapping 또는 Optional Chaining 기능 등을 사용한다. Forced-Unwrapping을 시도할 때 값이 nil이면 런타임 에러로 앱이 강제 종료될 수 있으므로 항상 nil이 아닌지 확인한 후 사용해야 한다. 즉, Optional 자료형을 사용하려면 값이 nil인지 먼저 확인하고 값이 있다면 Forced-Unwrapping해서 사용한다.

[코드 2-16] Optional 변수의 Forced-Unwrapping

```
let myInfo = ["name": "yagom", "age": "unknown", "gender": "male"]
let girlFriend: String? = myInfo["girlFriend"]

if (girlFriend != nil)
{
     let her = girlFriend!
     print("Your girl friend is \(her)")
}
else
{
     print("YOU DON'T HAVE ANY GIRL FRIEND")
}
```

이러한 불편함을 없애기 위해 Optional Binding과 Optional Chaining을 사용할 수 있다. 먼저 Optional Binding을 살펴보자. Optional Binding은 Optional 변수에 값이 들어 있는지 확인해서 값이 있을 때만 if 조건문이 실행되도록 if-let 조건문을 활용할 수 있게 해준다.

[코드 2-17] Optional Binding을 위한 if-let 조건문

```
if let her = myInfo["girlFriend"]
{
```

```
        print("Your girl friend is \(her)")
}
else
{
        print("YOU DON'T HAVE ANY GIRL FRIEND")
}
```

[코드 2-16]보다 조건문이 훨씬 간단해졌지만, 객체 안 객체의 또 다른 하위 객체일 때에는 if-let 문을 중첩할 수밖에 없다. 이때 Optional Chaining이라는 방식을 사용한다. Optional Chining은 연산자를 통해 Optional 변수에 값이 있을 때만 다음 하위 Optional 변수로 넘어가기를 반복해 최종값을 가져오는 방법이다. Chain 도중에 값이 없는 Optional 변수를 만나면 Unwrapping을 중지하고 nil을 반환한다.

[코드 2-18] Optional Chaining

```
class Person: NSObject {
        var gender: String?
}

class Developer: Person {
        var name:String?
        var girlFriend: Person?

        init(name: String) {
           self.name = name
        }
}

let girl = Person()
girl.gender = " Girl"
var yagom: Developer? = Developer(name: "yagom")
```

```
yagom.girlFriend = girl

var girlFriend = yagom?.girlFriend
var gender = girlFriend?.gender
```

[코드 2-19] Optional Binding과 Optional Chaining의 조합

```
// Optional Binding과 Optional Chaning의 조합을 사용하지 않을 때
if let gom = yagom {
     if let girlFriend = gom.girlFriend {
        if let gender = girlFriend.gender {
           print("yagom's girl freind is \(gender)")
        }
     }
}
// Optional Binding과 Optional Chaning의 조합을 사용할 때
if let gender = yagom?.girlFriend?.gender {
     if gender == "Girl" {
        print("yagom's girl freind is \(gender)")
     }
}
```

완전히 같은 동작을 하는 코드지만, 길이가 현저히 짧아졌다. Optional Chaining에서 쓰는 물음표(?)는 Optional 변수를 선언할 때와는 의미가 다르다. 값을 꺼내올 때 물음표(?)가 사용된다면 이때는 Optional Chaining을 사용한 것으로 이해하면 된다.

Optional Chaining을 사용하여 현저하게 짧아진 코드를 보았다. 그런데 Swift 2.0 에서는 앞의 내용보다도 더 간단하게 코드를 처리할 수 있다. guard 키워드가 그 핵심이다. if-let 문법과 사용법은 유사하지만 guard 키워드를 사용하면 상수를 스코프 밖에서 사용할 수 있다. guard 키워드는 함수 안에서만 사용할 수 있으

므로 Playground에서 테스트할 때에는 따로 함수를 만들어 줘야 한다.

[코드 2-20] guard 키워드를 이용한 Optional Unwrapping

```
func guardTest(developerObject: Developer?) {
      guard let gender = developerObject?.girlFriend?.gender else {
         print("Optional unwrapping failed")
         return
      }
      print("yagom's girl friend is \(gender)")
}
guardTest(yagom)
```

2.9 예외 처리

프로그래밍을 하다보면 어쩔 수 없이 수많은 에러에 직면하게 된다. 그럴 때 현명하게 에러를 핸들링하는 방법을 알고 있어야 한다. Swift 2.0에서는 기존 버전과 다르게 좀 더 우아한 방법으로 에러를 처리할 수 있다. 미리 예상된 에러를 시도해보고 실패할 경우 이를 캐치하는 do-catch 구문을 살펴보자.

[코드 2-21] do-catch 구문을 이용한 예외 처리

```
import AVFoundation
let fileURL = NSURL(fileURLWithPath: "someSoundFileURL")
// 플레이어를 생성하고 초기화한다.
do {
      let audioPlayer = try AVAudioPlayer(contentsOfURL: fileURL)
      audioPlayer.prepareToPlay()
} catch _ {
      // 생성에 실패했을 때는 catch문으로 들어온다.
      print("audioPlayer 초기화 실패")
}
```

여기까지 Swift의 간략한 문법과 굵직굵직한 특이사항들을 알아보았다. 이제 Swift로 애플리케이션을 만들러 가보자!