Optional Chaining

`옵셔널 체이닝은 옵셔널에 속해 있는 nil일지도 모르는 프로퍼티, 메서드, 서브스크립션 등을 가져오거나 호출할 때 사용할 수 있는 일련의 과정입니다.`

옵셔널에 값이 있다면 프로퍼티, 메서드, 서브스크립트 등을 호출할 수 있고, 옵셔널이 nil이라면 프로퍼티, 메서드, 서브스크립트 등은 nil을 반환합니다.

즉, 옵셔널을 반복 사용하여 옵셔널이 자전거 체인처럼 서로 꼬리를 물고 있는 모양이기 때문에 옵셔널 체이닝이라고 부릅니다.

자전거 체인에서 한 칸이라도 없거나 고장 나면 체인 전체가 동작하지 않듯이 중첩된 옵셔널 중 하나라도 값이 존재하지 않는다면 결과적으로 nil을 반환합니다.

옵셔널 체이닝은 프로퍼티나 메서드 또는 서브스크립트를 호출하고 싶은 옵셔널 변수나 상수 뒤에 물음표(?)를 붙여 표현합니다.

`옵셔널이 nil이 아니라면 정상적으로 호출될 것이고, nil이라면 결괏값으로 nil을 반환할 것입니다.`

결과적으로 nil이 반환될 가능성이 있으므로 옵셔널 체이닝의 반환된 값은 항상 옵셔널입니다.

📖 NOTE _ 느낌표(!)

물음표 대신에 느낌표(!)를 사용할 수도 있는데 이는 옵셔널에서 값을 강제 추출하는 효과가 있습니다.
물음표를 사용하는 것과 가장 큰 차이점은 값을 강제 추출하기 때문에 옵셔널에 값이 없다면 런타임 오류가 발생한다는 점입니다.
또 다른 점은 옵셔널에서 값을 강제 추출해 반환하기 때문에 반환 값이 옵셔널이 아니라는점 입니다.
하지만 정말 100% nil이 아니라는 확신을 하더라도 사용을 지양하는 편이 좋습니다.

옵셔널 체이닐에 대해 알아보기 위해 아래의 코드에서 기본 클래스를 설계하겠습니다.

```
/// 사람의 주소 정보 표현 설계

class Room { // 호실
  var number: Int // 호실 번호

  init(number: Int) {
    self.number = number
  }
}

class Building { // 건물
  var name: String // 건물 이름
  var room: Room? // 호실 정보

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

struct Address { // 주소
  var province: String // 광역시/도
  var city: String // 시/군/구
  var street: String // 도로명
  var building: Building? // 건물
  var detailAddress: String? // 건물 외 상세 주소
}

class Person { // 사람
  var name: String // 이름
  var address: Address? // 주소
```

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

사람의 정보를 표현하기 위해 Person 클래스를 설계했습니다.

Person 클래스는 이름이 있으며 주소를 옵셔널로 갖습니다.

주소 정보는 Address 구조체로 설계했습니다.

주소에는 광역시/도, 시/군/구, 도로명이 필수며, 건물 정보가 있거나 건물이 아니면 상세주소를 기재할 수 있도록 했습니다.

건물 정보는 Building 클래스로 설계했습니다.

건물은 이름이 있으며, 호실의 정보를 옵셔널로 갖습니다.

호실 정보는 Room 클래스로 설계했으며 각 호실은 번호를 갖습니다.

먼저, 아래의 코드에서 kobe 라는 사람의 인스턴스를 생성합니다.

```
/// kobe 인스턴스 생성
let kobe: Person = Person(name: "kobe")
```

kobe가 사는 호실 번호를 알고 싶습니다.

옵셔널 체이닝과 강제 추출을 사용하여 프로퍼티에 접근해보면 아래의 코드와 같은 결과를 볼 수 있습니다.

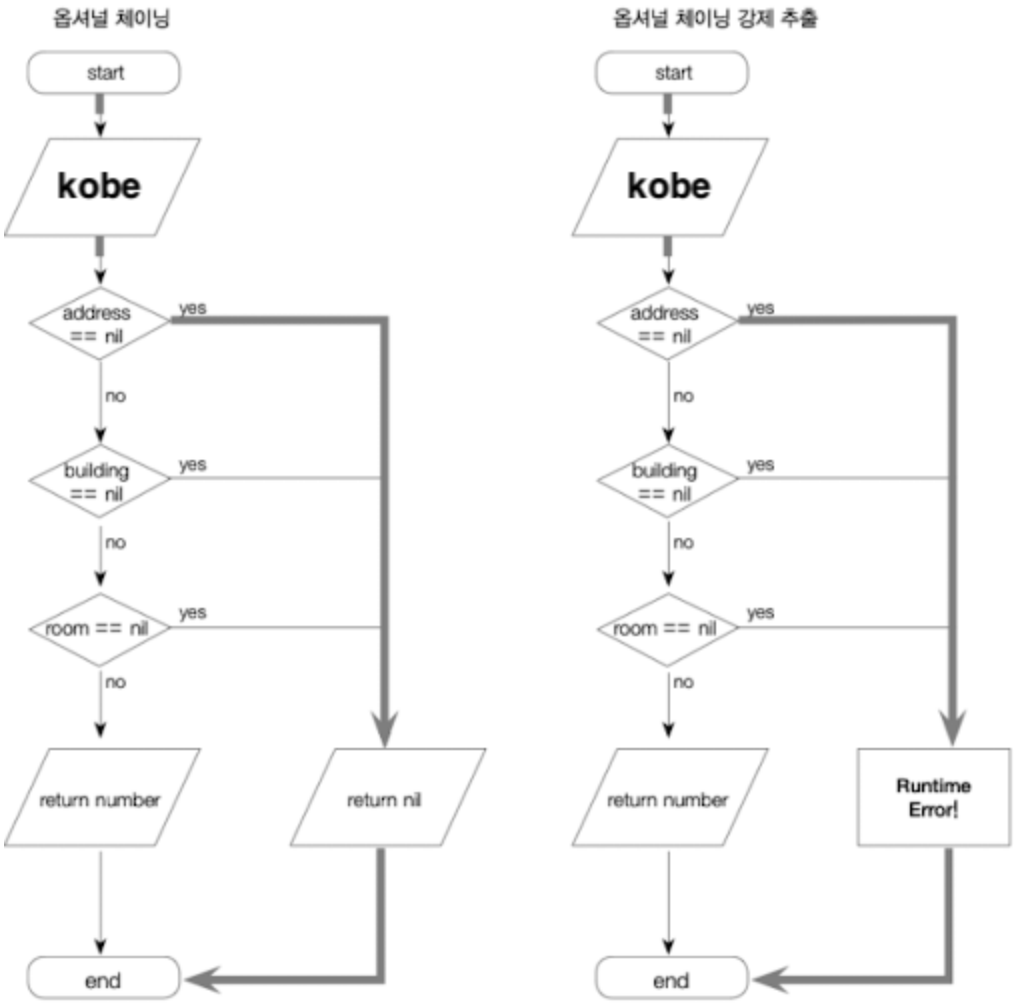
```
/// 옵셔널 체이닝 문법
let kobeRoomViaOptionalChaining: Int? = kobe
                                        .address?
                                        .building?
                                        .room?
                                        .number

// nil
```

```
let kobeRoomViaOptionalUnwraping: Int = kobe
                                        .address!
                                        .building!
                                        .room!
                                        .number
// 오류 발생!
```

kobe에는 아직 주소, 건물, 호실 정보가 없습니다.

kobeRoomViaOptionalChaining 상수에 호실 번호를 할당하려고 옵셔널 체이닝을 사용하면 kobe의 address 프로퍼티가 nil이므로 옵셔널 체이닝 도중 nil이 반환됩니다.

그러나 kobeRoomViaOptionalUnwraping 상수에 호실 번호를 할당할 때는 강제 추출을 시도했기 때문에 nil인 address 프로퍼티에 접근하려 할 때 런타임 오류가 발생합니다.



옵셔널 체이닝을 간략히 알아보았습니다.

아래 코드는 옵셔널 바인딩을 사용하여 kobe가 사는 호실 정보를 가져오는 코드를 표현한 것입니다.

```
/// 옵셔널 바인딩의 사용
let kobe: Person = Person(name: "kobe")

var roomNumber: Int? = nil

if let kobeAddress: Address = kobe.address {
  if let kobeBuilding: Building = kobeAddress.building {
    if let kobeRoom: Room = kobeBuilding.room {
      roomNumber = kobeRoom.number
```

```
    }
  }
}

if let number: Int = roomNumber {
  print(number)
} else {
  print("Can't find room number")
}
```

위 코드를 아래의 코드처럼 옵셔널 체이닝으로 표현해보면 훨씬 간단해집니다.

```
/// 옵셔널 체이닝의 사용
let kobe: Person = Person(name: "kobe")

if let roomNumber: Int = kobe.address?.building?.room?.number {
  print(roomNumber)
} else {
  print("Can't find room number")
}
```

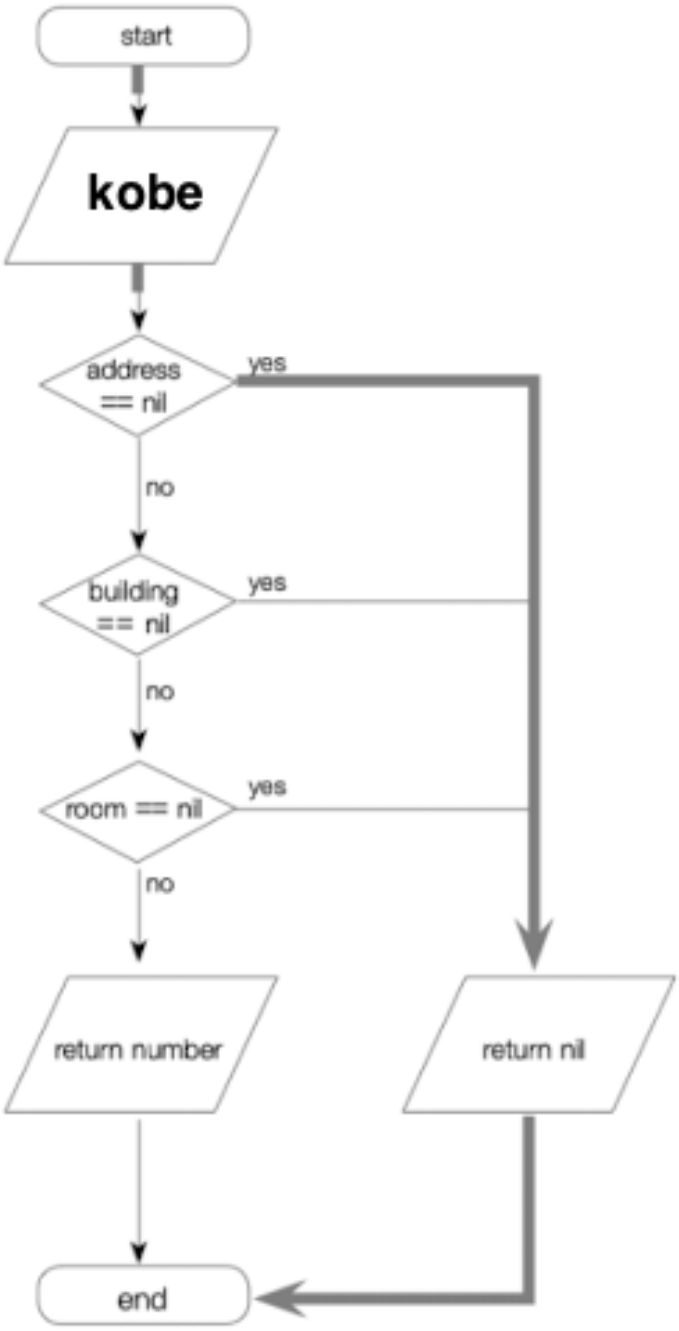
위의 코드와 아래의 코드는 완전히 똑같은 결과를 내놓지만, 코드의 간결함과 분량은 꽤 차이가 큽니다.

그런데 재미있는 점은 아래 코드의 옵셔널 체이닝 코드가 옵셔널 바인딩 기능과 결합했다는 점입니다.

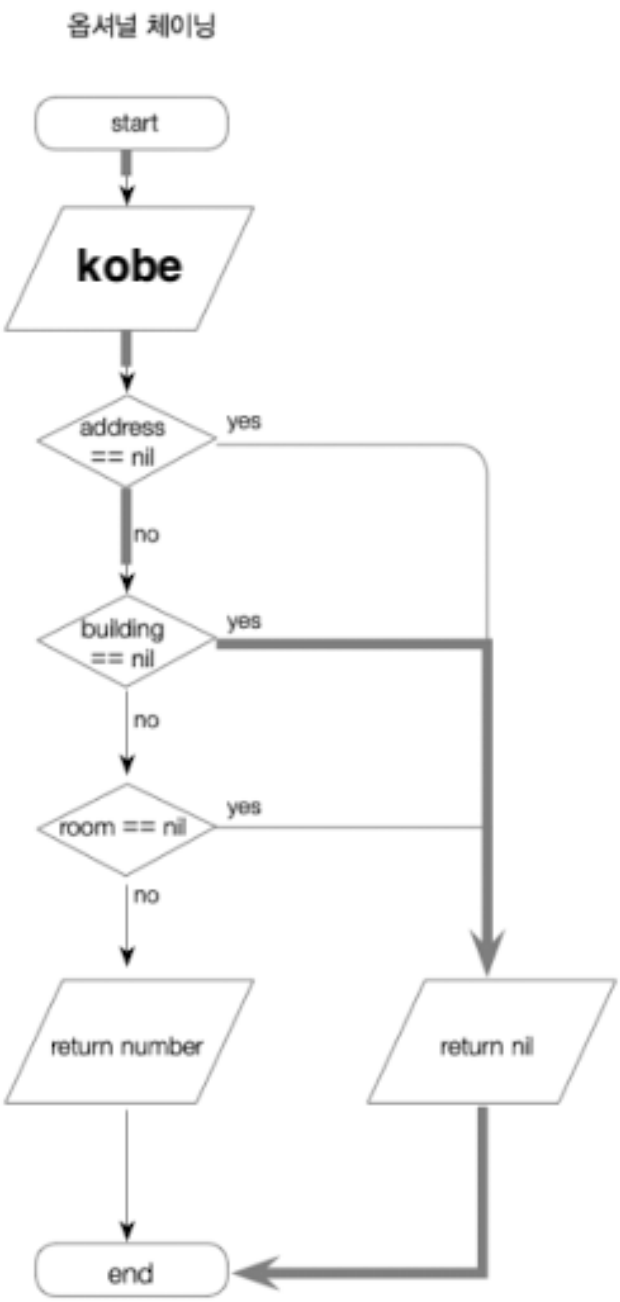
옵셔널 체이닝의 결괏값은 옵셔널 값이기 때문에 옵셔널 바인딩과 결합할 수 있는 것입니다.

옵셔널 바인딩을 통해 kobe.address?.building?.room?.number의 결괏값이 nil이 아님을 확인하는 동시에 roomNumber라는 상수로 받아올 수 있습니다.

'옵셔널 체이닝의 사용' 코드에서 옵셔널 체이닝을 실행할 때, kobe의 address가 nil이기 때문에 더 이상 다음 체인의 building을 체크하지 않고 nil을 반환합니다.



만약에 address의 값이 있었다면 아래 그림과 같이 다음 체인인 building을 체크해보았을 것입니다.



이처럼 옵셔널 체이닝을 통해 한 단계뿐만 아니라 여러 단계로 복잡하게 중첩된 옵셔널 프로퍼티나 메서드 등에 매번 nil 체크를 하지 않아도 손쉽게 접근할 수 있습니다.

또한 옵셔널 체이닝을 통해 값을 받아오기만 하는 것이 아니라 반대로 값을 할당해줄 수도 있습니다.

```
/// 옵셔널 체이닝을 통한 값 할당 시도
kobe.address?.building?.room?.number = 505
print(kobe.address?.building?.room?.number) // nil
```

```
/// 옵셔널 바인딩의 사용
let kobe: Person = Person(name: "kobe")

var roomNumber: Int? = nil

if let kobeAddress: Address = kobe.address {
  if let kobeBuilding: Building = kobeAddress.building {
    if let kobeRoom: Room = kobeBuilding.room {
      roomNumber = kobeRoom.number
    }
  }
}

if let number: Int = roomNumber {
  print(number)
} else {
  print("Can't find room number")
}

/// 옵셔널 체이닝의 사용
let kobe: Person = Person(name: "kobe")

if let roomNumber: Int = kobe.address?.building?.room?.number {
  print(roomNumber)
} else {
  print("Can't find room number")
}

/// 옵셔널 체이닝을 통한 값 할당 시도
kobe.address?.building?.room?.number = 505
print(kobe.address?.building?.room?.number) // nil
```

위 코드를 보면 아직 kobe의 address 프로퍼티가 없으며 그 하위의 building 프로퍼티도 room 프로퍼티도 없습니다.

때문에 ‘옵셔널 체이닝을 통한 값 할당 시도’ 코드의 옵셔널 체이닝은 동작 도중에 중지될 것입니다.

number 프로퍼티는 존재조차 하지 않으므로 505가 할당되지 않는 것은 물론입니다.

```
/// 옵셔널 체이닝을 통한 값 할당
kobe.address = Address(province: "서울",
                       city: "특별시 성동구",
                       street: "송정4가길",
                       building: nil,
                       detailAddress: nil)

kobe.address?.building = Building(name: "농구코트")
kobe.address?.building?.room = Room(number: 0)
kobe.address?.building?.room?.number = 2

print(kobe.address?.building?.room?.number) // Optional(2)
```

위 코드를 통해 옵셔널 체인에 존재하는 프로퍼티를 실제로 할당해준 후 옵셔널 체이닝을 통해 값이 정상적으로 반환되는 것을 확인할 수 있습니다.

옵셔널 체이닝을 통해 메서드와 서브스크립트 호출도 가능합니다.

서브스크립트는 인덱스를 통해 값을 넣고 빼올 수 있는 기능입니다.

먼저, 옵셔널 체이닝을 통한 메서드 호출입니다.

호출 방법은 프로퍼티 호출과 동일합니다.

만약 메서드의 반환 타입이 옵셔널이라면 이 또한 옵셔널 체인에서 사용 가능합니다.

아래의 코드에서 Address 구조체에 메서드 코드를 추가하고 옵셔널 체인을 통해 호출해봅시다.

```
/// 옵셔널 체이닝을 통한 메서드 호출
struct Address { // 주소
  var province: String // 광역시/도
  var city: String // 시/군/구
  var street: String // 도로명
  var building: Building? // 건물
  var detailAddress: String? // 건물 외 상세주소
```

```
  init(province: String, city: String, street: String) {
    self.province = province
    self.city = city
    self.street = street
  }

  func fullAddress() -> String? {
    var restAddress: String? = nil

    if let buildingInfo: Building = self.building {
      restAddress = buildingInfo.name
    } else if let detail = self.detailAddress {
      restAddress = detail
    }

    if let rest: String = restAddress {
      var fullAddress: String = self.province

      fullAddress += " " + self.city
      fullAddress += " " + self.street
      fullAddress += " " + rest

      return fullAddress
    } else {
      return nil
    }
  }

  func printAddress() {
    if let address: String = self.fullAddress() {
      print(address)
    }
  }
}

kobe.address?.fullAddress()?.isEmpty // false
kobe.address?.printAddress() // 서울 특별시 성동구 송정4가길 농구코트
```

우리가 서브스크립트를 가장 많이 사용하는 곳은 Array와 Dictionary입니다.

옵셔널의 서브스크립트를 사용하고자 할 때는 대괄호([ ])보다 앞에 물음표(?)를 표기해주어야 합니다.

이는 서브스크립트 외에도 언제나 옵셔널 체이닝을 사용할 때의 규칙입니다.

아래의 코드에서 서브스크립트의 옵셔널 체이닝을 살펴봅시다.

```
/// 옵셔널 체이닝을 통한 서브스크립트 호출
let optionalArray: [Int]? = [1, 2, 3]
optionalArray?[1] // 2

var optionalDictionary: [String: [Int]]? = [String: [Int]]()
optionalDictionary?["numberArray"] = optionalArray
optionalDictionary?["numberArray"]?[2] // 3
```