Protocol

G4ENG

18 JUL 19

Agenda

- Justice
- Property, Method
- init
- implement

Justice

- 클래스나 구조체가 어떤 기준을 만족하거나 특수한 목적을 달성하기 위해 구현해야 하는 메소드와 프로퍼티의 목록 -> Java의 Interface와 비슷한 개념
- 특정 기능이나 속성에 대한 설계도

Justice

```
protocol <프로토콜명> {
    <구현할 프로퍼티 명세 1>
    <구현할 프로퍼티 명세 2>
    <구현할 프로퍼티 명세 3>
    ...
    <구현할 메소드 명세 1>
    <구현할 메소드 명세 2>
    <구현할 메소드 명세 3>
    ...
}
```

Justice

```
struct/class/enum/extention 객체명: 프로토콜 {
}
```

Property, Method

- 프로퍼티
  - 프로토콜에 선언되는 프로퍼티는 초기값을 할당할 수 없다.
- 메소드
  - 프로퍼티와 크게 다르지 않다

Property, Method

```
protocol SomePropertyProtocol {
    var name: String { get set }
    var description: String { get }
}
// name은 읽고 쓰기가 가능한 변수, description 은 읽기만 가능함

struct RubyMember: SomePropertyProtocol {
    var name = "홍길동"
    var description: String {
        return "Name: \(self.name)"
    }
}
```

Property, Method

```swift
protocol SomeMethodProtocol {
    func execute(cmd: String)
    func showPort(p: Int) -> String
}

// 프로토콜에 선언되는 메소드는 메소드 종류, 메소드 이름, 파라미터 타입, 파라미터 이름, 반환 타입까지는 정의할 수 있지만 실행할 내용을 작성할 수 없기때문에 중괄호를 붙이지 않는다.

struct RubyServices: SomeMethodProtocol {
    func execute(cmd: String) {
        if cmd == "start" {
            print("run")
        }
    }
    func showPort(p: Int) -> String {
        return "Port : \(p)"
    }
}
```

Property, Method

```
// 일치시켜야 하는 매개변수명은 외부로 드러나는 매개변수명에 국한됨. 다시 말해 외부 매개변수 명은 프로토콜은 그대로 따라가야 하지만 내부 매개변수명은 임의로 바꾸어 사용해도 된다는 뜻. 또한 내부 매개변수명과 외부 매개변수명이 프로토콜에서
    통합되어 선언되어 있을 경우 구현체에서는 이를 분리하여 내부와 외부 매개변수명으로 나누어 따로 작성해도 된다.
struct RubyNewServices2: NewMethodProcotol {
    func execute(cmd comm: String, desc d: String) {
        if comm == "start" {
            print("\(d) run")
        }
    }
    func showPort(p: Int, memo description: String) -> String {
        return "port: \(p), memo: \(description)"
    }
}
```

Property, Method

- 프로토콜에서 mutating, static 사용
    - 구조체 내의 메소드가 프로퍼티를 변경하는 경우, 메소드 앞에 반드시 mutating 키워드를 붙여야 한다. 만약 메소드가 만약 프로토콜에서 선언된 메소드라면 mutating 키워드를 붙이기 위해서는 반드시 프로토콜에 키워드가 추가되어 있어야 한다.

Property, Method

- 프로토콜에서 메소드 선언에 mutating 키워드가 붙지 않는 것을 다음 두 가지 중 하나로 해석 가능
  - 구조체나 열거형 등 값 타입의 객체에서 내부 프로퍼티의 값을 변경하기를 원치 않을 때
  - 주로 클래스를 대상으로 간주하고 작성된 프로토콜일 때

Property, Method

```swift
protocol MService {
    mutating func execute(cmd: String)
    func showPort(p: Int) -> String
}

struct RubyMService: MService {
    var paramCommand: String?

    // execute 메소드가 메소드 내에서 프로퍼티를 변경하지 않기 때문에 mutating 생략 가능.
    mutating func execute(cmd: String) {
        self.paramCommand = cmd

        if cmd == "start" {
            print("run")
        }
    }

    func showPort(p: Int) -> String {
        return "Port: \(p), now command: \(self.paramCommand!)"
    }
}
```

Property, Method

```
// 프로토콜          구조체          결과
// mutating      mutating        OK
// mutating          -           OK          단, 실제로 구조체의 메소드에서 프로퍼티 값 변경이 없을때에만
//   -           mutating        error
//   -               -           OK
```

Property, Method

```swift
// 타입 메소드나 타입 프로퍼티도 프로토콜에 정의할 수 있다. 프로토콜의 각 선언 앞에 static 키워드를 붙이면 된다. class는 클래스에서만 사용 가능하고 나머지는 다 static으로

protocol SomeTypeProperty {
    static var defaultValue: String { get set}
    static func getDefaultValue() -> String
}

struct TypeStruct: SomeTypeProperty {
    static var defaultValue = "default"

    static func getDefaultValue() -> String {
        return defaultValue
    }
}

class ValueObject: SomeTypeProperty {
    static var defaultValue = "default"

    class func getDefaultValue() -> String {
        return defaultValue
    }
}
```

init

- 초기화 메소드를 포함한 프로토콜을 구현할 때 주의할 점이 있다.
- 먼저 외부 매개변수명까지는 완전히 일치해야 한다. 임의로 변경할 경우 프로토콜을 제대로 구현하지 않은 것으로 간주함. 이 점은 일반 메소드와 동일
- 구조체는 모든 프로퍼티의 초기값을 한 번에 설정할 수 있는 멤버 와이즈 메소드가 기본적으로 제공되지만, 만약 프로토콜에 멤버와이즈 메소드가 선언되었다면 좋지 않은 경우다.
- 프로토콜에 선언된 초기화 메소드는 기본 제공 여부에 상관없이 모두 직접 구현해주어야 하기 때문이다
- 클래스에서 초기화 메소드를 구현할 때에는 required 키워드를 붙여야 한다

init

1. 구현되는 초기화 메소드의 이름과 매개변수명은 프로토콜의 명세에 작성된 것과 완전히 일치해야 함

2. 프로토콜에 명세에 선언된 초기화 메소드는 그것이 기본 제공되는 초기화 메소드일지라도 직접 구현해야 함

3. 클래스에서 초기화 메소드를 구현할 때는 required 키워드를 붙여야 한다

init

```
struct SInit: SomeInitProtocol {
    var cmd: String

    init() {
        self.cmd = "start"
    }

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

class CInit: SomeInitProtocol {
    var cmd: String

    required init() {
        self.cmd = "start"
    }

    required init(cmd: String) {
        self.cmd = cmd
    }
}
```

init

```swift
protocol Init {
    init()
    func getValue()
}

// init() 메소드를 가지는 부모 클래스
class Parent {
    init() {
    }
    func getValue() {
    }
}

class Child: Parent, Init {
    override required init() {
    }
    override func getValue() {
    }
}
```

init

```
struct MultiImplement: NewMethodProcotol, SomeInitProtocol {
    var cmd: String

    init() {
        self.cmd = "default"
    }

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

    mutating func execute(cmd: String, desc: String) {
        self.cmd = cmd
        if cmd == "start" {
            print("start")
        }
    }

    func showPort(p: Int, memo desc: String) -> String {
        return "port: \(p), memo: \(desc)"
    }
}
```