

---

# Delegate Pattern

---

# Protocol

---

- 프로토콜은 원하는 작업이나 기능을 구현되도록 메서드, 프로퍼티등으로 요구 사항의 청사진을 정의합니다.
- 클래스, 구조체, 열거형은 프로토콜을 채택하면, 프로토콜에서 요구한 사항에 대해 구현해야 됩니다.
- 프로토콜을 통해 공통적인 작업을 강제 할수 있으며, 해당 프로토콜을 채택한 사람이 구현한 메소드에 대한 정보도 알수 있다.

# Protocol 문법

---

```
protocol Runnable {  
    var regCount: Int {get set}  
    func run()  
}
```

```
class Animal: Runnable{  
    var regCount: Int = 0  
    func run()  
    {  
  
    }  
}
```

# 프로토콜의 프로토콜 채택

---

```
protocol Runnable {  
    var regCount: Int {get set}  
    func run()  
}
```

```
protocol Flying : Runnable {  
    var wingCount: Int {get set}  
}
```

```
class Animal: Flying {  
    var wingCount: Int = 0  
    var regCount: Int = 0  
    func run()  
    {  
  
    }  
}
```

# 추상클래스로의 Protocol

---

- 프로토콜을 추상 클래스처럼 사용할수 있다.
- 다음과 같은 클래스가 있고, racing 이라는 함수를 구현하려고 한다면!

```
class Dog: Runnable{  
    //...  
}
```

```
class Horse: Runnable{  
    //...  
}
```

```
func racing(animals:[Runnable]) -> Runnable  
{  
  
}
```

- 프로토콜타입으로 사용가능하다.

```
let winner:Runnable = racing(animals: [Dog(),Horse()])
```

# Delegate

---

- 델리게이트는 클래스나 구조체에서의 일부분의 할 일을 다른 인스턴스에게 대신 하게 하는 디자인 패턴!
- 뷰가 받은 이벤트나 상태를 ViewController에게 전달해주기 위해 주로 사용된다.(ex:UIScrollViewDelegate...)
- ViewController를 통해 View구성에 필요한 데이터를 받는 용도로도 사용(ex:UITableViewDataSource)

# 직접 예제를 통해 확인해보자!

---

- CustomDelegate만들기!!

# Delegate 선언부

---

```
class CustomView: UIView {  
    var delegate: CustomViewDelegate?  
  
    override func layoutSubviews() {  
        delegate?.viewFrameChanged(newFrame: self.frame)  
    }  
}  
  
protocol CustomViewDelegate {  
    func viewFrameChanged(newFrame: CGRect)  
}
```



# Delegate 구현부

---

```
class ViewController: UIViewController, CustomViewDelegate{  
    override func viewDidLoad() {  
        super.viewDidLoad()  
  
        let custom = CustomView()  
        custom.delegate = self  
    }  
  
    func viewFrameChanged(newFrame: CGRect) {  
        //뷰의 프레임이 변경될때마다 불림  
    }  
}
```

# 구조

## <Delegate 구현부>

```
class ViewController: CustomDelegate {  
  
    override func viewDidLoad() {  
        super.viewDidLoad()  
  
        let custom = CustomView()  
        custom.delegate = self  
    }  
  
    func viewFrameChanged(newFrame:...) {  
        //뷰의 프레임이 변경될때마다 불림  
    }  
}
```

## <Delegate 선언부>

```
class CustomView: UIView {  
  
    var delegate: CustomViewDelegate?  
  
    func layoutSubviews() {  
        delegate?.viewFrameChanged...  
    }  
}  
  
protocol CustomViewDelegate {  
  
    func viewFrameChanged...  
}
```

# 구조

## 1. 프로토콜 생성 <Delegate 구현부>

- 클래스에 delegate 프로퍼티 생성
  - class ViewController: CustomDelegate {
  - 일반적으로 delegate란 이름 사용
  - 타입은 프로토콜 추상화 타입
  - override func viewDidLoad() {
  - super.viewDidLoad()
- delegate 인스턴스의 메소드 실행
  - let custom = CustomView()
  - custom.delegate = self
  - } delegate instance가 존재하는지는  
모른다.
  - func viewFrameChanged(newFrame:...) {
  - 하지만 만약 어떤 instance(A)가 나의 delegate instance 값을 할당했다면, 분명 A는 나의 프로토콜을 채택했으며 (타입이 같기때문에)메소드를 구현 했다는 것을 인지!
  - delegate method를 사용해서 메소드 실행 및 리턴값을 받아와 사용

## <Delegate 선언부>

```
class CustomView: UIView {  
    2 var delegate: CustomViewDelegate?  
  
    func layoutSubviews() {  
        3 delegate?.viewFrameChanged...  
    }  
}  
  
1 protocol CustomViewDelegate {  
    func viewFrameChanged...  
}
```

# 구조

## <Delegate 구현부>

```
class ViewController: CustomDelegate {  
  
    override func viewDidLoad() {  
        super.viewDidLoad()  
  
        let custom = CustomView()  
        custom.delegate = self  
    }  
  
    func viewFrameChanged(newFrame:...) {  
        //뷰의 프레임이 변경될때마다 불림  
    }  
}
```

## <Delegate 선언부>

### 1. CustomView Delegate 채택

```
class CustomView: UIView {  
    2. 채택한 Delegate 메소드 구현  
    var delegate: CustomViewDelegate?  
  
    3. custom instance의 delegate 프로퍼티  
    // 자기 자신의 인스턴스를 할당 (프로  
    to콜 추상화 타입)  
    • ViewController입장에선, 내가 구현  
    한 메소드를 실행하진 않지만,  
    customView가 적절한 곳에서 호출  
    했을것이다.  
    • customView가 특정 위치에서 해당  
    메소드를 호출할 것을 예상하여 필요  
    한 행동을 구현한다.
```