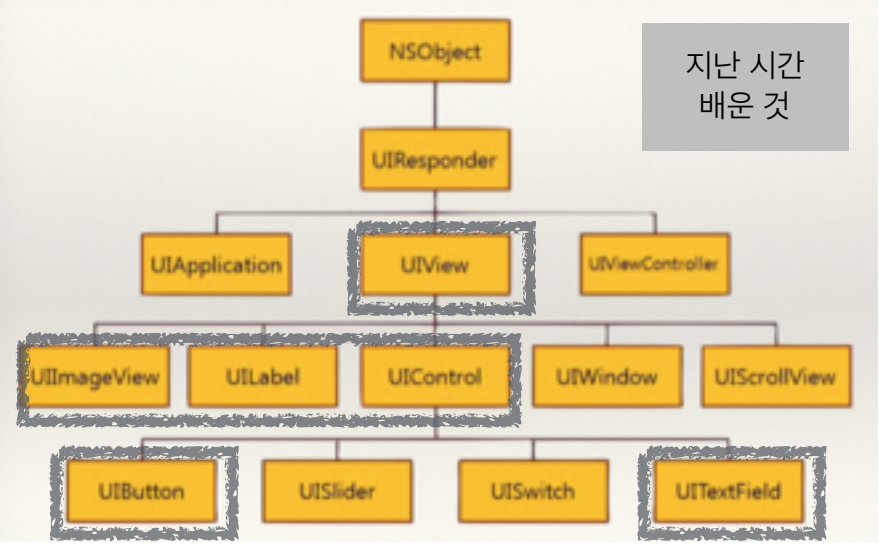
ios 입문 강의

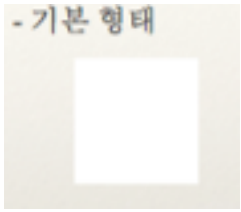
10강 <UI 기본 컴포넌트 (3) >

1. 지난 시간 복습

(1) UI 상속도



(2) UIView



* 기본형태 : 하얀 네모
* UIComponent들의 기본이 되는 View
* iOS에서는 view들의 집합으로 화면이 구성된다

*** 예제**

```
//뷰 객체 생성 - 위치 설정
let secondView:UIView = UIView(frame: CGRect(x: 0, y: 0,
width: 100, height: 100))

//뷰 위에 서브뷰로 추가
self.view.addSubview(secondView)
//뷰의 배경색 설정
secondView.backgroundColor = UIColor.blueColor()
```

* 뷰 객체 생성 - 위치 생성

    * frame의 초기화값을 넣음 (x, y - 위치 / width, height - 크기)
    * 기본 뷰는 UIviewcontroller - > 기본적으로 뷰가 하나씩 있다. -> view는 항상 전체 화면을 포함
  * self.view.addSubview(secondView) : 뷰 위에 서브뷰로 추가
    * 상위뷰가 내 자식 뷰를 갖는 것
  * secondview.backgroundColor
    * 배경 색상을 넣어줌

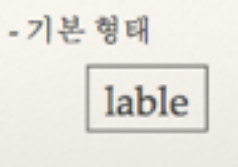
* 순서

1. 객체생성
2. 프레임 - 위치, 크기
3. addsubview로 추가 - 화면에 보이게끔 한다

  * 각각의 것들이 어떤 속성을 가지는지
  * 내가 쓸 것들이 무엇인지 찾아내는 것

(3) UILable : UIView

  * UIview를 상속받음
  * 텍스트 정보를 표시해주는 UI



* UILable - 예제

```
//레이블 객체 생성
let newLb:UILabel = UILabel(frame: CGRect(x: 0, y: 0, width:
100, height: 100))
//레이블 추가
self.view.addSubview(newLb)
//텍스트 입력
newLb.text = "텍스트 입력"
//텍스트 폰트
newLb.font = UIFont.systemFontOfSize(16)
//텍스트 컬러
newLb.textColor = UIColor.whiteColor()
//텍스트 정렬
newLb.textAlignment = NSTextAlignment.Left
```

(4) UIImage : UIView

* UIImageView
  * 이미지 정보를 표시해주는 UI

```
//생성자
init(image: UIImage!)
init(image: UIImage!, highlightedImage: UIImage?)
//속성
var image: UIImage?
```

☆ 중요한 것 - 이미지 객체를 만들어 넣어줘야 한다.

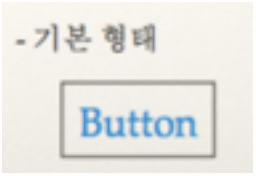
*** 예제**

```
//이미지 객체 생성
let newImgView:UIImageView = UIImageView(frame: CGRect(x: 0,
y: 0, width: 100, height: 100))
//이미지 추가
self.view.addSubview(newLb)
//이미지 입력
newImgView.image = UIImage(named: "imgName")


//이미지 객체 생성
let img:UIImageView = UIImageView(image: UIImage(named:
"imgName"))
//이미지 추가
self.view.addSubview(imp)
//이미지 frame설정
img.frame = CGRect(x: 0, y: 0, width: 100, height: 100)
```

(5) UIButton : View

* 사용자의 액션을 버튼의 형태로 받는 UI
* UIview를 상속받음
* 텍스트 / 이미지 로 구성되어있다.



* Button - 예제

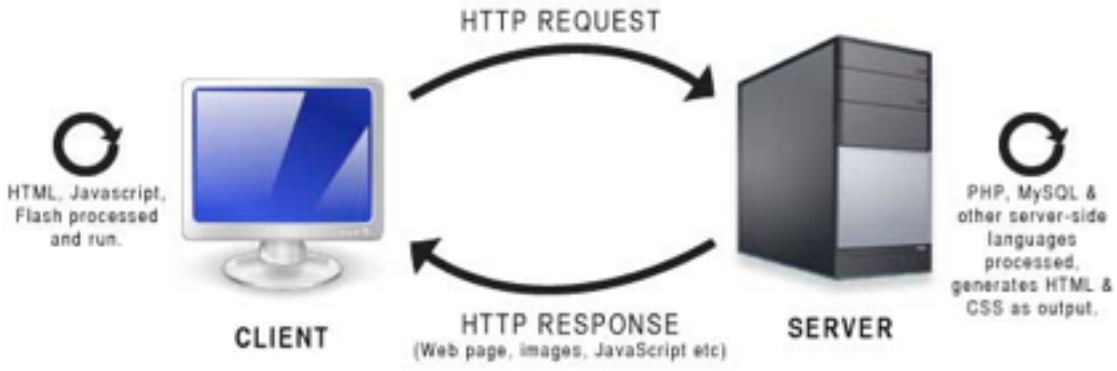
```
//버튼 객체 생성
let newBtn:UIButton = UIButton(frame: CGRect(x: 0, y: 0,
                                             width: 100, height: 100))
//버튼 추가
self.view.addSubview(newBtn)
//액션 추가
newBtn.addTarget(self, action: #selector(ClassName.method(_:),
                 forControlEvents: UIControlEvents.TouchUpInside)
//텍스트 변경
newBtn.setTitle("newBtn", forState: UIControlState.Normal)
newBtn.setTitle("onTouch", forState: UIControlState.Highlighted)


//버튼 액션
func method(sender:UIButton)
{
        //액션 함수 내용
}
```

2. Protocol & Delegate

(1) Protocol (프로토콜)

* delegate 배우기 전에 알아야 할 것



* 복수의 컴퓨터 사이나 중앙 컴퓨터와 단말기 사이에 데이터 통신을 원활하게 하기 위해 필요한 **통신규약**.
  * 신호 송신의 순서, 데이터의 표현법, 오류 검출법 등을 정함.

* 인터넷상에는 여러 개발자들이 개발. 데이터를 보낼 때,
  * 주소, 내용, 비트값 등을 어떤 순서로 보낼 지,
  * 어떤 데이트를 암호화 하고 안할 것인지 등등
  * 순서나 형태에 대한 약속을 한다.

<프로토콜의 예시>

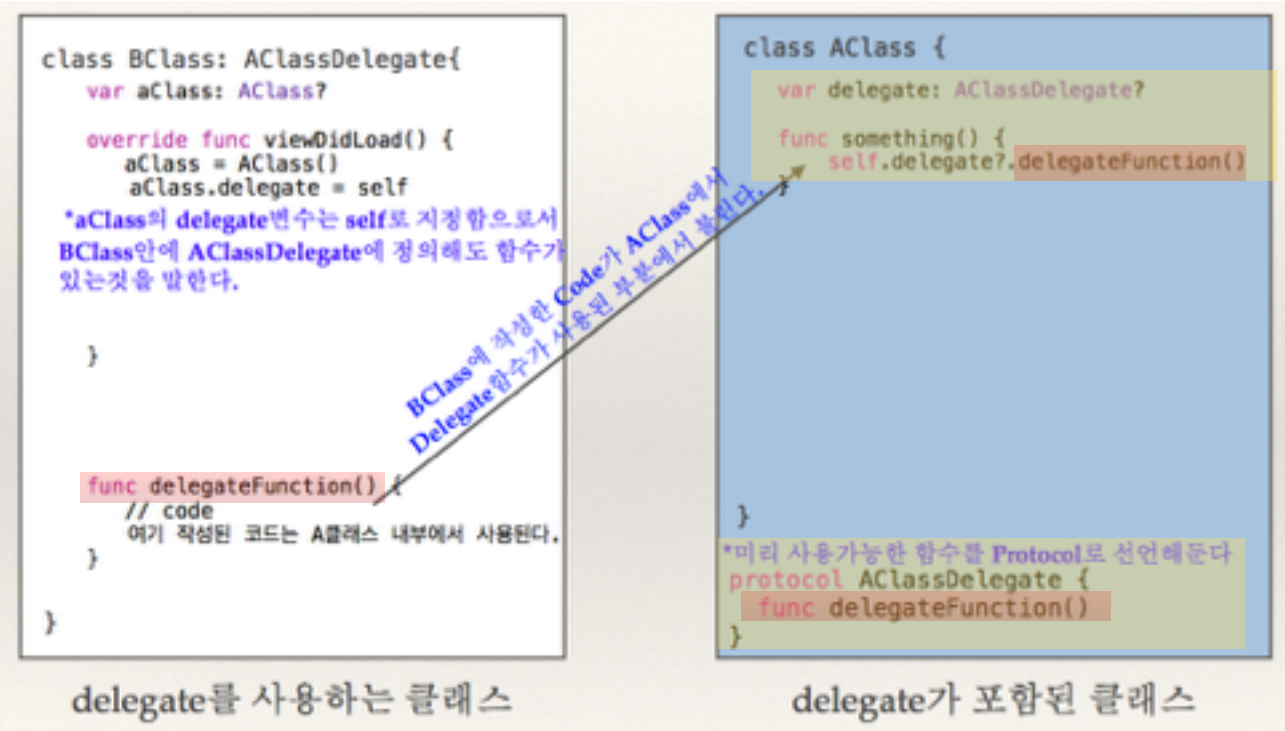
HTTP : Hyper Text Transfer Protocol
HTTPS : Secure Hyper Text Transfer Protocol
FTP : File Transfer Protocol
SFTP : Secure File Transfer Protocol
Telnet : TEminaL NETwork
POP3 : Post Office Protocol version 3
SMTP : Simple Mail Transfer Protocol
SSH : Secure Shell
SSL : Secure Socket Layer
SOAP : Simple Object Access Protocol

* iOS에서도 프로토콜을 쓴다.

```
protocol AClassDelegate {
  func delegateFunction()
}
```

  * 객체와 객체간에 약속을 일컫는데, 보통 함수 이름을 가지고 약속을 한다.
    * 이런 이름으로 함수를 만들어라 = 델리게이트
  * 델리게이트 function => 이걸로 정의를 하면 이걸로 써야한다.

(2) delegate 기본 구조



* delegate가 포함된 클래스 (오른쪽부터 보기)
  * A라는 Class (AClass)에 delegate라는 변수를 만들었다. 이 변수의 타입은 AClassDelegate라고 하는 프로토콜 타입으로 하였다.
  * 그리고 그 안에 어떤 함수(현재 예제에서는 something 함수에서 protocol AClassDelegate (오른쪽 맨 아래) 안에 있는 함수(delegateFunction)를 사용한다.
  * AClassDelegate 안에 delegateFunction는 내용이 없는 함수다 -> 때문에 아무런 일도 안 일어남
  * 텍스트필드, 스크롤뷰 등 이렇게 내용이 없고, **이름만 있는 델리게이트가 포함된 클래스**들이 있다.

* delegate를 사용하는 클래스 (왼쪽 그림, BClass)
  * AClass를 타입으로 받는 변수 (현재 예제에서는 vae aClass)를 만들고
  * 클래스에서는 프로토콜(현재 예제에서는 protocal ACalssDelegate)을 상속받았다
    * -> 때문에 viewDidLoad에서는 delegate를 self로 지정한다
  * 그 다음에 그 아래에서 delegateFunction을 사용한다.
    * BClass에 작성한 Code가 AClass에서 Delegate함수가 사용된 부분에서 불린다.
    * protocal로 선언해둔 함수의 이름은 똑같이 하고 하고자 하는 내용을 넣으면 된다.
* 즉, something이 실행될 때, 오른쪽 클래스에서 사용하지 않고 그것을 사용하는 클래스(왼쪽)에서 규정해서 사용 = 이런 식으로 동작하는 애들을 Delegate라고 부른다.

5. UITextField : UIControl

* 텍스트를 입력받는 UI Compenet
* 주요 요소

```
var text: String?
var font: UIFont!
var textColor: UIColor!
```

  * 레이블과 비슷
  * + 객체명.placeholder - 하나도 입력되지 않은 상태에서의 안내문구 (ex 텍스트를 입력하세요)

```
- UITextFieldDelegate

 optional func textFieldDidBeginEditing(textField: UITextField)
 optional func textFieldShouldClear(textField: UITextField) -> Bool
 optional func textFieldShouldReturn(textField: UITextField) -> Bool
```

  * textFieldDidBeginEditing
    * 텍스트를 시작할 때 딱 불림
  * textFieldShouldClear
    * 글씨를 다 지웠을 때 불림
      * ex 관련 검색어 또는 최근 검색어 보여주는 것 등
  * textFieldShouldReturn
    * -> 내가 UITextFieldDelegate Protocol을 상속 받아서 그 내용을 내가 작성하면
    * -> 반응은 원래 Protocol이 있는 클래스에서 하지만 액션은 내용이 작성된 클래스 내의 내용으로 된다.
    * ※ optional이 다들 붙어 있는 이유 : 안써도 되기 때문에 옵셔널을 붙인 것

* UITextField에서 특이한 점은 가 있다는 것
  * 어떤 특이한 행동일 때 그 함수가 자동으로 불리워짐.

* **예제**

```
//텍스트 필드 객체 추가
let newTF:UITextField = UITextField(frame: CGRect(x: 10, y: 20,
width: 100, height: 30))

//텍스트 필드 추가
self.view.addSubview(newTF)

//델리게이트 설정
newTF.delegate = self

//안내 문구 설정
newTF.placeholder = "id를입력하세요"


//추가 델리게이트 함수 작성
func textFieldShouldReturn(textField: UITextField) -> Bool
{
    newTF.resignFirstResponder()
}
```

* 여태 배운 것들과 다 똑같은데, Delegate 설정이 있다
  * __ .delegate = self
  * UITextField에서 90%이상 Delegate 설정에서 self를 사용
* //추가 델리게이트 함수 작성
  * textFieldShouldReturn : return키를 눌렀을 때
    * resignFirstResponder -> 포커싱을 제거하겠다. (관심 없어. 키보드 내려가.)
      * <—-> becomeFirstResponder - > 포커싱 하겠다.

<실습>

TextField- 입력 & 표시

1. 텍스트 필드 하나 만들기
2. 버튼 클릭 시 텍스트필드에 입력한 데이터 레이블에 표시하기
3. return버튼 클릭 시 키보드 내리기

텍스트 필드 델리게이트 상속

< 예시 코드>

```
class TextfiledViewController: UIViewController,UITextFieldDelegate {

    var textfield:UITextField = UITextField()
    var label:UILabel = UILabel()

    override func viewDidLoad() {
        super.viewDidLoad()

        //텍스트 필드 객체 추가
        self.textfield = UITextField(frame: CGRect(x: 10, y: 20, width: self.view.
            frame.size.width - 100, height: 30))
        //델리게이트 설정
        self.textfield.delegate = self
        //안내 문구 설정
        self.textfield.placeholder = "id를 입력하세여"
        self.textfield.borderStyle = .RoundedRect
        //텍스트 필드 추가
        self.view.addSubview(self.textfield)

        //버튼 객체 생성
        let newBtn:UIButton = UIButton(frame: CGRect(x: self.view.frame.size.width -
            70, y: 20, width: 69, height: 30))
        newBtn.addTarget(self, action: #selector(TextfiledViewController.
            onTouchUpInsideButton(_:)), forControlEvents: UIControlEvents.
            TouchUpInside)
        newBtn.setTitle("입력완료", forState: UIControlState.Normal)
        //텍스트 변경
        newBtn.setTitle("입력완료", forState: UIControlState.Highlighted)
        newBtn.backgroundColor = UIColor.brownColor()
        self.view.addSubview(newBtn)

        //레이블 객체 생성
        self.label = UILabel(frame: CGRect(x: 10, y: 70, width: self.view.frame.size.
            width - 30, height: 100))
        self.label.text = " "
        self.label.textAlignment = NSTextAlignment.Center
        self.label.textColor = UIColor.whiteColor()
        self.label.backgroundColor = UIColor.yellowColor()
        self.view.addSubview(self.label)


    }

    //버튼 액션
    func onTouchUpInsideButton(sender:UIButton) {
        // 라벨에 텍스트필드 글자를 넣어준다.
        self.label.text = textfield.text
        // 키보드를 내려준다.
        textfield.resignFirstResponder()
    }

    // 텍스트필드에서 return키 눌렀을 때
    func textFieldShouldReturn(textField: UITextField) -> Bool {
        // 키보드를 내려준다.
        textfield.resignFirstResponder()
        return true
    }
```

해당 버튼에 액션에 따른 이벤트를 onTouchUpinsideButton에서 설정

2. UIScrollView

(1) UIScrollView:UIView

* UIScrollView 란?
  * view를 확장시켜 스크롤을 통해 확장된 View를 볼수 있게 도와주는 View
  * 내부 contentsView에 다른 view를 addSubview한후 contentsView를 컨트롤한다
* ContentsView란?
  * 원래는 화면 내 View가 사이즈인데, 그것을 넘어선 확장된 뷰가 있을 때
  * 확장된 영역은 보이는 View 사이즈보다 커야한다.
* **비유하자면, 스크롤뷰는 창문역할 - 창문으로 보면 컨텐츠뷰가 움직여 안보이던 곳을 보이게 한다.**
  * 사이즈 설정을 1. 스크롤뷰 사이즈, 2. 컨텐츠 뷰 사이즈 둘 다 설정해야한다.

(2) UIScrollView

```
- 주요 요소
//contentView setting
var contentOffset: CGPoint
var contentSize: CGSize
var contentInset: UIEdgeInsets
//scrollvew setting
var bounces: Bool
var pagingEnabled: Bool
var scrollEnabled: Bool
//스크롤 인디케이터 표시
var showsHorizontalScrollIndicator: Bool
var showsVerticalScrollIndicator: Bool

func setContentOffset(contentOffset: CGPoint, animated: Bool)

- UIScrollViewDelegate
optional func scrollViewDidScroll(scrollView: UIScrollView)
```

* contentOffset
  * 지금 보고 있는 화면이 옆으로 움직인 거리 (x값이 증가하거나 y값이 증가하거나 등등)

    [스크롤 뷰 + 컨텐츠뷰] 컨트롤 하는 것 —> 확장형 뷰
* contentInset
  * 안쪽 여백 (많이 쓰진 않음) <—> offset (바깥쪽 여백)
* bounces: 맨 아래 화면에서(또는 맨위에서 더 올리려 하거나 내리려 했을 때 컨텐츠가 밀리는 거)

<bounces 화면 예시>

* pagingEnabled

    * 페이지 단위로 움직일 것인지
        * 페이지 전환할 때 50%이상 넘어가면 알아서 넘어감

* 스크롤 인디케이터 표시
    * 스크롤 막대 표시 보이냐 안보이냐 여부

* func setContentOffset(contentOffset: CGPoint, animated: Bool)
    * **contentOffset: (0,0), animated: true**
        * 스크롤 시, 드르르르륵 올라가는 애니메이션 효과
        * 안하면 그냥 슝 올라감
    * ex) 쇼핑몰 이미지 넘기는것 같은 것

* 스크롤뷰 델리게이트
    * optional func scrollViewDidScroll (scrollView: UIScrollView)
        * 현재 offset이 어디 있는지 알 수 있다.
        * 조금 내렸을 땐 아니고 많이 내릴 때 쓴다. 많이 쓰이는 함수
            * ex) ContentOffset의 y 좌표가 -50보다 작을 때 내려라
            * ex2) 페이스북 내리면 로딩 되는 것

(3) UIScrollView - 예제

```
//ScrollView객체 생성
let scrollView:UIScrollView = UIScrollView(frame: CGRect(x: 0, y:
0, width: 320, height: 456))

//ScrollView 뷰 추가
self.view.addSubview(scrollView)

//ScrollView contentSize 설정
scrollView.contentSize = CGSizeMake(scrollView.frame.size.width*2,
scrollView.frame.size.height)
//ScrollView 델리게이트 설정
scrollView.delegate = self


//추가 델리게이트 함수 작성
func scrollViewDidScroll(scrollView: UIScrollView) {
        //스크롤시 불린다.

}
```

* 컨텐츠 사이즈 추가하는 것이 특이점
    * 현재 예제는 가로로만 2칸 움직이는 구조 (2배니까)

- 실습 2 -

ScrollView 만들기

1. 스크롤뷰 만들기(가로사이즈 3배)

2. 각 화면 별 ImageView 올리기

화면에 이미지를 3개 올렸습니다 (예시 이미지 입니다)

————————————> 스크롤 하는 모습 (옆으로)

```swift
class ScrollViewController: UIViewController, UIScrollViewDelegate {

    /* ScrollView 전역 변수로 객체 생성 */
    var mainScrollView:UIScrollView = UIScrollView()

    override func viewDidLoad() {
        super.viewDidLoad()

        /* 전역변수로 만든 스크롤뷰 객체 세팅 */
        self.mainScrollView = UIScrollView(frame: CGRect(x: 0, y: 0, width:
            self.view.frame.size.width , height: self.view.frame.size.height))
        //ScrollView contentSize 설정
        self.mainScrollView.contentSize = CGSizeMake(self.mainScrollView.frame.size.
            width*3, self.mainScrollView.frame.size.height)
        //ScrollView 델리게이트 설정
        self.mainScrollView.delegate = self
        self.view.addSubview(self.mainScrollView)

        /* 이미지 객체 생성 및 세팅 */

        var offsetX:CGFloat = 0
        let offsetWidth:CGFloat = self.view.frame.size.width

        //이미지 객체 생성
        let newImgView:UIImageView = UIImageView(frame: CGRect(x: offsetX, y: 0,
            width: offsetWidth, height: self.view.frame.height))
        //이미지 추가( ⭐ 스크롤뷰 위에 놓아야 한다)
        self.mainScrollView.addSubview(newImgView)
        //이미지 입력
        newImgView.image = UIImage(named:"1.jpg")

        // X값 조절 = 이전 이미지 가로사이즈만큼 옆으로 가 있어야 한다.
        offsetX += offsetWidth

        // 두 번째 이미지 객체 세팅
        let newImgView2:UIImageView = UIImageView(frame: CGRect(x: offsetX, y: 0,
            width: offsetWidth, height: self.view.frame.height))
        self.mainScrollView.addSubview(newImgView2)
        newImgView2.image = UIImage(named: "2.jpg")

        // X값 조절
        offsetX += offsetWidth

        // 세 번째 이미지 객체 세팅
        let newImgView3:UIImageView = UIImageView(frame: CGRect(x: offsetX, y: 0,
            width: offsetWidth, height: self.view.frame.height))
        self.mainScrollView.addSubview(newImgView3)
        newImgView3.image = UIImage(named: "3.jpg")

    }

    //추가 델리게이트 함수 작성
    func scrollViewDidScroll(scrollView: UIScrollView) {
        print("scrollViewDidScroll")
    }
```

스크롤뷰 객체 생성