접근 제어

Fast campus

접근 수준

은닉화나 캡슐화를 위해 접근 제어를 함

접근에 대한 권한 설정

- 외무 모듈에서의 접근을 제어하는 수단.
- 캡슐화, 은닉화를 위해 사용

Fast campus

모듈 & 소스파일

하나의 묶음 단위

- 모듈 : 배포할 코드의 묶음 단위, 통상 프레임워크나 라이브러리, 어플리케이션이 모듈의 단위가 될수 있다.

- 소스파일 : 하나의 스위프트 소스코드 파일 ".swift"

Fast campus

접근제어

모듈 밖

- Open (개방 접근수준) : 모듈 외부까지 접근 가능
- public (공개 접근수준) : 모듈 외부까지 접근 가능

모듈 안

- internal (내부 접근수준) : 모듈 내부에서 접근가능, 기본 지정값
- fileprivate (파일외 비공개) : 파일 내부에서만 접근가능

비공개

- private (비공개) : 기능 정의 내부에서만 가능

extention

swift 4에서 추가됨

클래스의 확장

Fast campus

Open VS Public

상속의 유무
외부에 있는 class를 내가 상속을 받을 수 있는가?
open - 상속 가능
public - 상속 불 가능

ㄷ

- Open을 제외한 다른 모든 접근수준의 클래스는 그 클래스가 정의된 모듈 안에서만 상속될 수 있다.

- Open을 제외한 다른 모든 접근수준의 클래스 멤버는 그 멤버가 정의된 모듈 안에서만 재정의 될 수 있다.

- Open 수준의 클래스는 그 클래스가 정의된 모듈 밖의 다른 모듈에서도 상속되고, 재정의 될수 있다.

- 클래스를 Open으로 명시하는 것은 그 클래스를 다른 모듈에서도 부모클래스로 사용할수 있다는 얘기

Fast campus

예시

```
public class SomePublicClass {
    public var somePublicProperty = 0
    var someInternalProperty = 0
    fileprivate func someFilePrivateMethod() {}
    private func somePrivateMethod() {}
}

class SomeInternalClass {
    var someInternalProperty = 0
    fileprivate func someFilePrivateMethod() {}
    private func somePrivateMethod() {}
}

fileprivate class SomeFilePrivateClass {
    func someFilePrivateMethod() {}
    private func somePrivateMethod() {}
}

private class SomePrivateClass {
    func somePrivateMethod() {}
}
```

Fast campus

접근수준 확인하기

- Test클래스 생성
- Public, internal, fileprivate, private 접근 수준을 포함한 메소드 만들기
- 각각의 상황에서 메소드 호출 해보기

Fast campus

Property

강사 주영민

Fast campus

프로퍼티

프로퍼티 > 변수

- 변수의 다른 이름
- 클래스, 구조체, 열거형에서 전체의 속성으로 사용되는 변수를 프로퍼티라고 부른다.

Fast campus

프로퍼티의 종류

- 저장 프로퍼티 (Stored Properties)
- 연산 프로퍼티 (Computed Properties)
- 타입 프로퍼티 (TypeProperties)

Fast campus

저장 프로퍼티(Stored Properties)

- 가장 일반적인 프로퍼티
- 값을 저장하는 용도로 사용된다.
- 클래스, 구조체에서만 인스턴스와 연관된 값을 저장한다.
- 초기값을 설정 할 수 있습니다.

Fast campus

저장 프로퍼티(Stored Properties)

```
struct Subject {

    var name:String
    var score:Int
}
```

```
class Person
{
    var name:String
    var gender:String
    var blood:String?
}
```

Fast campus

지연 저장 프로퍼티 (Lazy Stored Properties)

속도 이슈가 있을 때 유용함

- 지연 저장된 속성은 처음 프로퍼티가 사용되기 전 까지 초기값이 계산되지 않은 특성을 가지고 있는 프로퍼티이다.

- 지연 저장 속성은 **lazy** keyword를 선언 앞에 작성한다.

- let은 지연 저장 프로퍼티로 설정할 수 없다.
  시점이 달라서 바로 선언 그래서 var 로 선언함
- **초기화하는데 오래걸리거나, 복잡한 초기화 과정이 있는 변수의 경우 지연저장**을 사용하면 좋다 .

Fast campus

지연 저장 프로퍼티 (Lazy Stored Properties)

```
class ViewController: UIViewController {

    //init 시점이 아닌 사용이 되는 시점에 초기화 한다.
    lazy var cal:Calculator = Calculator()

    override func viewDidLoad() {
        super.viewDidLoad()

        cal.average(student: Student())
    }
}
```

Fast campus

연산 프로퍼티(Computed Properties)

중요함!

함수랑 같은 행동
하는 일 : 프로퍼티에 값을 넣고 뺄 때 사용
get, set함수를 프로퍼티로 봄
연산만 진행함

- 실제로 값을 저장하지 않지만, get, set키워드를 통해서 값을 간접적으로 설정하거나 받을 수 있다.
- 일반적으로 메소드중 속성에 관련된 메소드를 연산프로퍼티로 사용한다.
- 클래스, 구조체, 열거형 모두에서 사용 가능하다.

Fast campus

연산 프로퍼티 예제

```
struct Point {
    var x = 0.0, y = 0.0
}
struct Size {
    var width = 0.0, height = 0.0
}
struct Rect {
    var origin = Point()
    var size = Size()
    var center: Point {
        get {
            let centerX = origin.x + (size.width / 2)
            let centerY = origin.y + (size.height / 2)
            return Point(x: centerX, y: centerY)
        }
        set(newCenter) {
            origin.x = newCenter.x - (size.width / 2)
            origin.y = newCenter.y - (size.height / 2)
        }
    }
}
```

Fast campus

연산 프로퍼티 예제

연산 프로퍼티안에서는 자기 자신의 프로퍼티를 호출하면 절대 안된다.
저장 공간이 아니라 메서드 처럼 사용해야 한다.

- 연산프로퍼티 자신을 바로 사용하면 어떻게 될까요?

```
struct Rect {
    //var origin = Point()
    var size = Size()
    var center: Point {
        get {
            let centerX = center.x + (size.width / 2)
            let centerY = center.y + (size.height / 2)
            return Point(x: centerX, y: centerY)
        }
        set(newCenter) {
            center = newCenter
        }
    }
}
```

Fast campus

Setter ValueName 미 지정

- set의 값이름 미지정시 newValue 가 기본 값으로 사용된다.

```
struct Rect {
    var origin = Point()
    var size = Size()
    var center: Point {
        get {
            let centerX = origin.x + (size.width / 2)
            let centerY = origin.y + (size.height / 2)
            return Point(x: centerX, y: centerY)
        }
        set {
            origin.x = newValue.x - (size.width / 2)
            origin.y = newValue.y - (size.height / 2)
        }
    }
}
```

Fast campus

Read Only 연산 프로퍼티

- 읽기 전용 연산프로퍼티 작성시 get 키워드 없이 바로 작성할수 있다.

- **쓰기 전용 연산 프로퍼티는 작성할수 없다.**

```
struct Cuboid {
    var width = 0.0, height = 0.0, depth = 0.0
    var volume: Double {
        return width * height * depth
    }
}
```

Fast campus

연산프로퍼티 연습

- 같이 해봅시다.

Fast campus

Property Observers

Programming에 design pattern이 있다.
그 중에 observer pattern이 있다.
일반적으로 순서대로 나가는데,
등록을 하고 그 이벤트가 발생하면 실행하는 것
그것을 위해서 관찰자를 만들고 어떤 행동을 하면 그러한 행동을 하게 함

저장 프로퍼티여야 하고
초기값이 설정되어 있어야 함.

Property에 어떠한 행동이 나타나면 실행
Objective-C에서는 KVO pattern,
key value observer가 되나
Swift에서는 property observer가 함.

isUniversityStudent 가 들어오면,
schoolGrade만 할당

특정 값에 할당 되는 것에 따라 변화

- 프로퍼티 값의 변화를 감시하고 그에 따라 대응한다.
- 초기값이 설정된 저장 프로퍼티에서 사용 가능하다. .
- 프로퍼티의 값이 설정될때마다 호출된다.
- didSet, willSet 키워드를 통해 값 변화의 직전 직후를 감지 할수 있다.
- 값이름 미지정시 oldValue, newValue가 기본값으로 지정된다.

Fast campus

Property Observers 예제

```
var changeValue:Int = 0{
    didSet(oldV){
        print("oldValue \(oldV)")
    }
    willSet(willV)
    {
        print("newValue \(willV)")
    }
}

changeValue = 4
```

초기값에는 반응하지 않고,
값이 바뀌는 시점에 반응한다.

Fast campus

타입 프로퍼티(Type Properties)

객체가 아닌 클래스인 상태에서 주는 속성
변하지 않는 값들
version
제조국 등

- 인스턴스의 속성이 아닌, 타입에 따른 속성을 정의 할수 있다.

메모리의 데이터 영역에 저장되는 변수 :static

- static 키워드를 사용해서 타입 프로퍼티를 설정할수 있으며, 클래스의 경우 연산 프로퍼티의 오버라이드를 지원하기 위해서는 class 키워드를 사용해서 클래스 타입에서 연산 프로퍼티를 설정해야 한다.(class키워드는 저장 프로퍼티는 사용불가)

- 값을 가져올때는 클래스의 이름을 통해서 가져올 수 있다.

Fast campus

타입 프로퍼티 예제

```
struct AudioChannel {
    static let level = 10
    static var maxLevel = 0
    var currentLevel: Int = 0 {
        didSet {
            if currentLevel > AudioChannel.level
            {
                currentLevel = AudioChannel.level
            }
            if currentLevel > AudioChannel.maxLevel
            {
                AudioChannel.maxLevel = currentLevel
            }
        }
    }
}
```

Fast campus

Method

- 메서드는 특정 타입에 관련된 함수를 뜻합니다.
- 함수의 문법과 같다.
- 인스턴스의 기능을 수행하는 인스턴스 메서드와 타입자체의 기능을 수행하는 타입 메서드로 나눌수 있습니다.

Fast campus

self Property

- 모든 인스턴스는 self 프로퍼티를 가지고 있다.
- self 프로퍼티는 자기 자신을 가르키고 있는 프로퍼티이다.
- Type Method안에서의 self는 클래스 자체를 가르키고, instance Method안에서는 self는 인스턴스를 가르킨다.

```
struct Point {
    var x = 0.0, y = 0.0
    func isToTheRightOf(x: Double) -> Bool {
        return self.x > x
    }
}
```

Fast campus

Value Type 프로퍼티 수정

- 기본적으로 구조체와 열거형의 값타입 프로퍼티는 인스턴스 메소드 내에서 수정이 불가능 하다.

- 그러나 특정 메소드에서 수정을 해야 할경우에는 mutating 키워드를 통해 변경 가능하다.

```
struct Point {
    var x = 0.0, y = 0.0
    mutating func moveBy(x deltaX: Double, y deltaY: Double) {
        x += deltaX
        y += deltaY
    }
}
```

Fast campus

타입 메서드

- 타입 프로퍼티랑 마찬가지로 타입 자체에서 호출이 가능한 메서드.

- 메서드 앞에 static키워드를 사용하여 타입메서드를 작성할수 있다. 타입 프로퍼티와 마찬가지로 클래스에서는 class키워드를 사용해 타입메서드를 표현한다.

- 타입 메소드 안에서의 self는 인스턴스가 아닌 타입을 나타낸다.

Fast campus