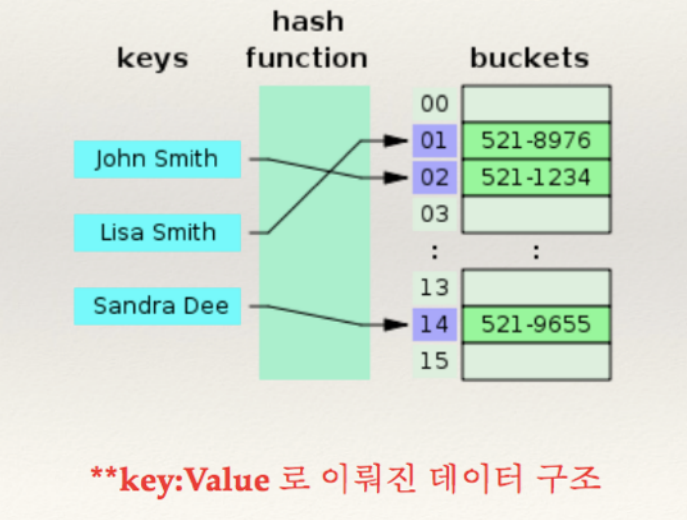
ios 입문 강의

6강 클래스와 객체

1. Dictionary와 tuple (지난 수업 복습)

(1) Dictionary란?

* 키를 값에 매핑할 수 있는 구조인, 연관 배열 추가에 사용되는 자료구조



* 키를 값(value)에 매핑 할 수 있는 구조
* 배열은 값을 인덱스로 구별할 수 있었으나 <—> 딕셔너리는 키값을 통해 매칭된 버킷을 가져올 수 있음

(2) Dictionary - 문법

* 선언
  * var vName:[keyType:valueType] = [key1:value1, key2:vlaue2 ……]
  * var vName:Dictionary<keyType,VlaueType> = [key1:value1, key2:vlaue2 ……]
    * key의 타입은 보통 String을 쓴다. 이용을 용이하게 하기 위해.
* 사용
  * 데이터 추가 : vName.updateVlaue(“newVale”,forkey:”key2”) <———> 배열에서는 append
    * (참고) 스위프트 함수를 만들 때에는 보통 앞에 것이 함수에 대한 설명을 하고, 그에 해당되는 인자를 하나씩 들어간다. 때문에 [딕셔너리 이름 - 데이터를 업데이트 한다는 updateValue - 그 다음이 그에 대한 값 - 그리고 forkey] 순서로 써져있는 것.
  * 값을 불러오기 : vName[“key2”]
  * var value2 = vName["key2"]

* 딕셔너리는 순서가 없다. 키값을 통해 정해지기 때문에 순서를 보장할 수 없음.
  * <——> 배열에서는 순차적으로 배열

(3) 튜플 (tuple)

* 두 개 이상의 자료를 묶음으로 다루는 구조
* 변경 불가능한 배열
* 다양한 자료형을 쓸 수 있다.

(4) 튜플 - 문법

* 선언

```
var tName:(Int, String, Int) = (1,"joo",3)
```

* 사용
  * 인덱스 번호를 통해 사용한다.

```
tName.0
tName.1
tName.2
```

※ 참고

* 배열을 반환하는 함수를 만들 때에는 반환 타입을 [ ] (대괄호)로 묶어줘야 한다.

  ex. func exampleFunc () -> **[Int]**

* 여러 데이터를 한 개씩 끄집어내려면 for 문을 사용하여야 한다.

  ex. for (name, cost) in dates

  for (_ , cost) in dates

[실습] 자판기의 기본 데이터 만들기

- 음료수 이름, 가격의 딕셔너리 필요!
- 딕셔너리 데이터를 쉽게 가져올수 있는 함수 만들기

예제 답안은 맨 마지막 장에 있습니다!
꼭 한 번씩 풀어보세요!

자판기 동작을 고민해 봅시다.

	사용자 입장	자판기 입장
1	동전 넣기	입력 금액 저장
2	음료선택	1. 입력된 금액으로 음료 구입가능 확인 2. 입력금액에서 선택된 음료가격 빼기
3	음료받기	음료반환
4	잔돈 받기	음료가격이 빠진 잔돈반환

2. 객체 지향 프로그래밍 (Object-Oriented Programming)

(1) What is OOP?

* 객체지향 프로그래밍(OOP)은 컴퓨터 프로그래밍의 패러다임 중 하나다.
  * 이 외에도 절차형, 함수형, 논리형 등등 여러가지 패러다임이 있다.
* “객체를 지향한다” = **객체 위주로 만든다**
* 명령어의 목록으로부터 보는 시각에서 벗어나 **여러 개의 독립된 단위, “객체”들의 모임으로 파악하고자 한다.**
  * 객체지향 이전의 프로그래밍은 순차적으로 이루어졌고, 그렇게 진행되어 왔다.
    ㄴ> 이를 ‘절차 지향형’이라고 말한다.
* 객체지향은 이러한 절차지향을 분할시켜서, 분할된 단위를 일정한 설계에 따라 ‘조합’하는 것으로 프로그램이 실행되도록 한다. 그 단위가 객체이다.
* 각각의 객체는 메세지를 주고 받고, 데이터를 처리할 수 있다.
* 단위가 쪼개지기 때문에 프로그램이 유연하고, 변경도 용이하다.
  * 이렇게 쪼개기 때문에 규모도 커지기 용이하다.

(2) 클래스 / 객체 / 메서드

클래스	객체	메서드
같은 종류의 집단에 속하는 속성과 행위를 정의한 것	클래스의 인스턴스	클래스로부터 생성된 객체를 사용하는 방법으로서 객체에 명령을 내리는 메시지 (지금은 일단, 함수로 이해하시면 될 것 같습니다)
속성과 행위로 이루어진 집합체 속성과 행위를 가지고있는 뼈대	속성과 행위를 가진 집합체 (클래스)가 일정한 수행을 위해 메모리에 들어간 것이 객체	
ex 속성 = 변수 / 행위 = 함수 사람 객체가 있다고 예를 들면, 속성 = 사람의 관련된 속성 행위 = 사람에 관련된 행위		

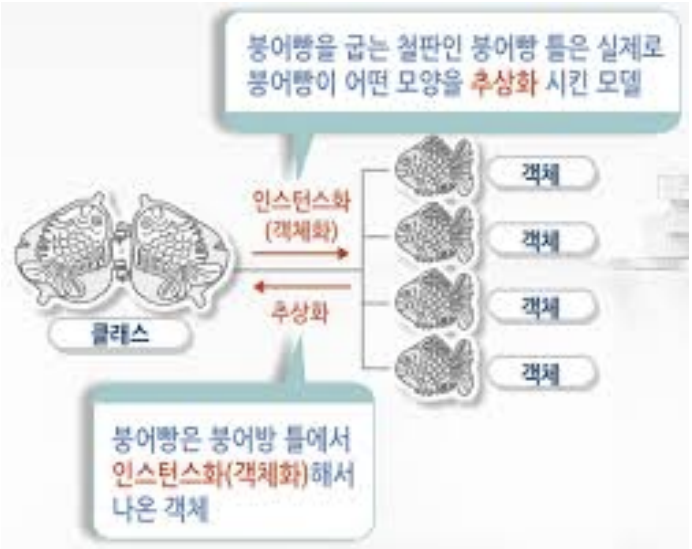
(3) 상속

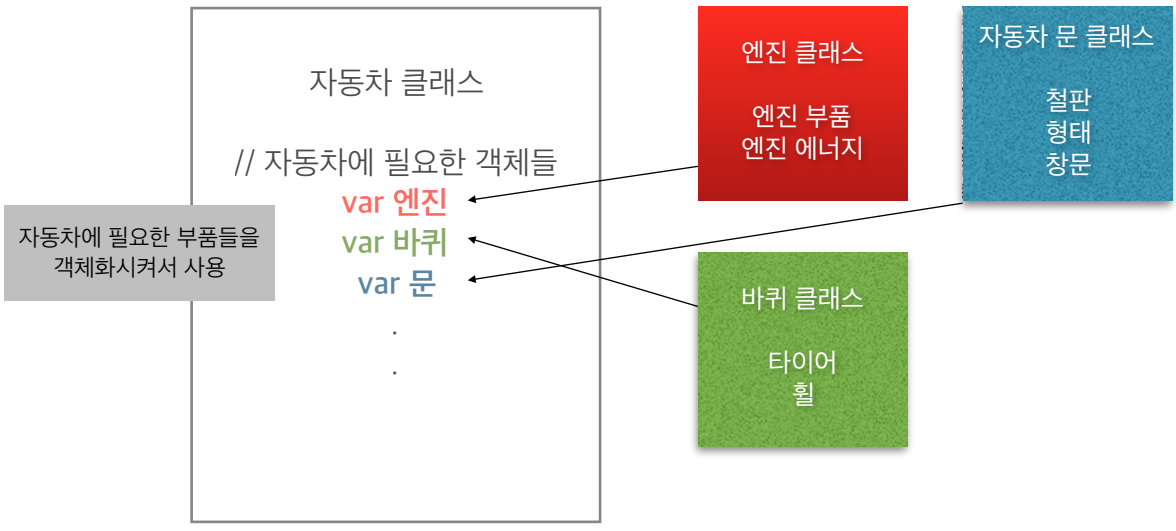
* 새로운 클래스가 기존의 클래스의 자료와 연산을 이용할 수 있게 하는 기능
* 자료 = 속성 / 연산 = 함수 = 메소드 = 행동
* 상속 받았다 = 부모클래스의 변수와 함수를 자식 클래스도 쓸 수 있다.

3. 클래스와 객체

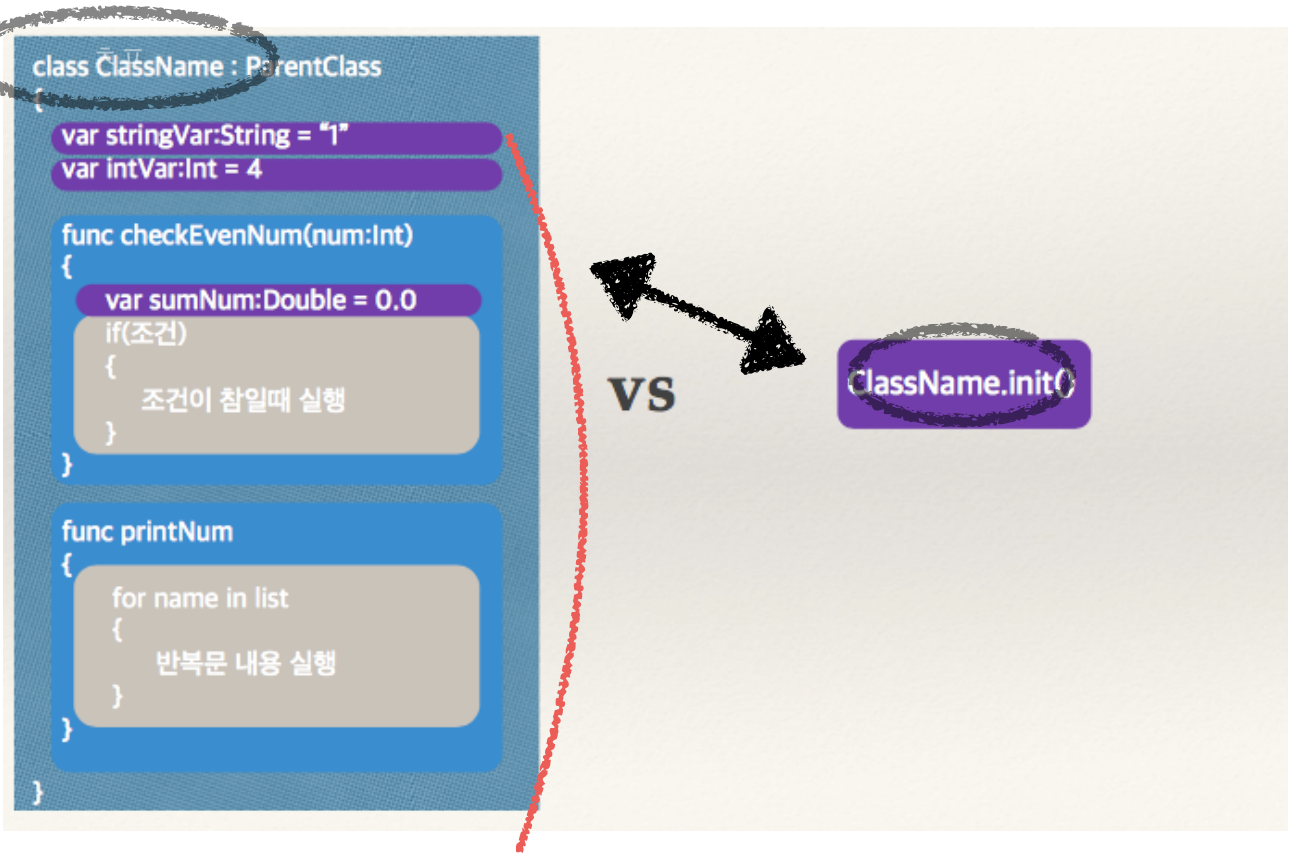
(1) 클래스 & 객체

클래스	객체
특정 기능, 값 등을 하나의 구조로 만들어놓은 객체의 '틀'	클래스를 메모리에 적재시켜 실질적으로 사용하는 상태
붕어빵틀	붕어빵
자동차 도면	엔진, 바퀴, 문 등등… 엔진 (input -> 기름 / output -> 에너지)
클래스는 일정한 도면같은 것이며, 도면 내의 객체들을 이용해 무언가를 만들 수 있는 것이다.	





* 클래스 vs 객체



(2) 객체 만들어 사용하기

* 객체 만들기

* var vc:ClassName = ClassName.init()
  * .init -> 초기화
  * 클래스는 앞에부터 대문자 -> ClassName

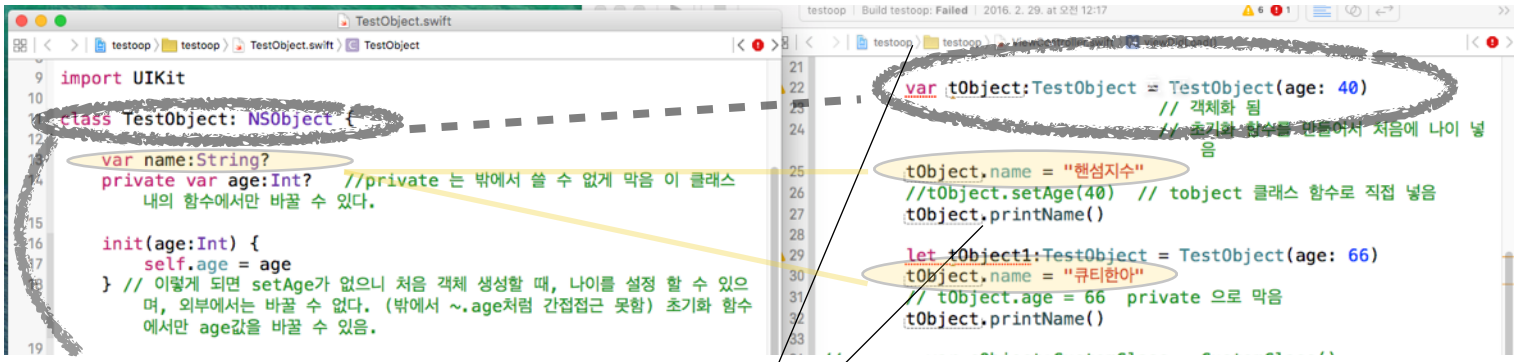
1. 변수 생성: 초기화한 객체를 담을 그릇을 만든다. *여기서 변수 타입은 ClassName이다.
2. 인스터스&초기화(initialize):클래스 내용을 메모리 할당하고 초기화한다.
3. 초기화한 객체를 변수에 넣고 사용한다.

    *해당 객체의 변수, 메소드를 사용하기 위해서는 . 기호를 사용한다.

    classObj.변수

    classObj.메소드()

예시



TestObject 클래스

ViewController에서 TestObject클래스를 tObject로 객체화하여 사용

1. classObject 변수 생성 ㄱ
   * var tObject:TestObject = TestObject()
   ㄴ TestObject를 classObject (위에 객체명: tObject) 로 만든 것 = 객체화 한 것
   -> 초기화 한다 (0로 만든다) -> 2. 인스턴스 초기화

3. 초기화한 TestObject타입의 객체 classObject를 사용
TestObject 에서 정의했었던 변수에 접근할 수 있다.(지금은 객체화된 classObject를 통해 접근하는 것)
classObject.변수 -> . 을 통해 classObj안에 잇는 변수에 접근한다

(3) 초기화

var classObj:Person = Person( )

클래스 이름 ( )
= 객체로 초기화 한 것

classObj.name = “joo”

classObj.age = 23

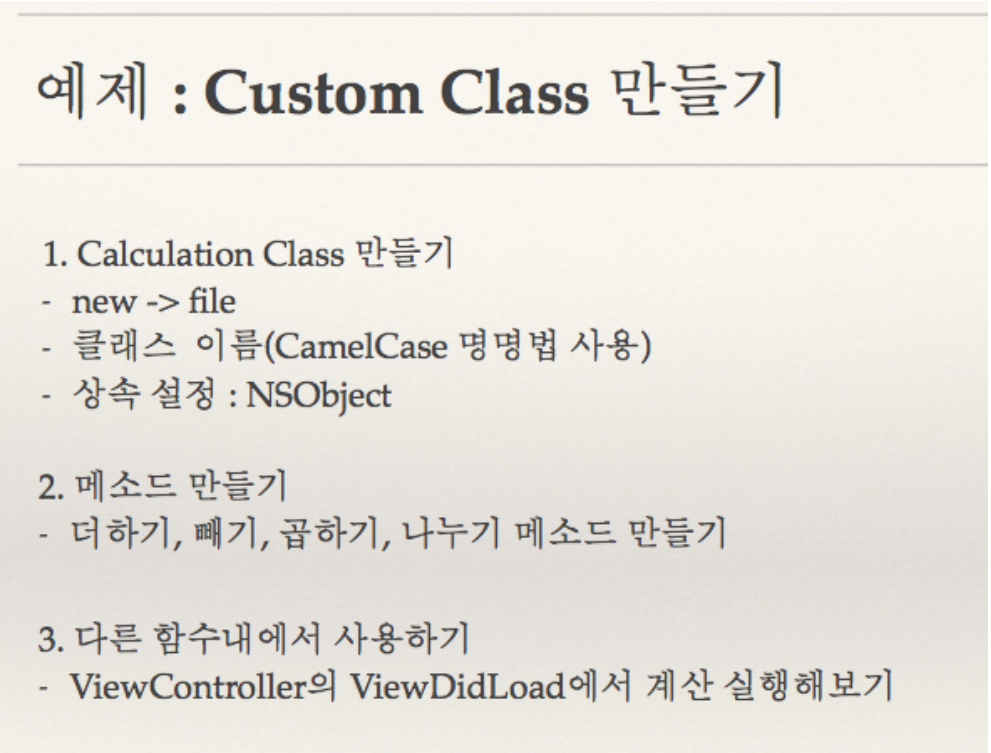
* NSObject를 상속받아서 만든 class이다.
* NSObject는 object-c, swift 통틀어 가장 최상위

```
class Person: NSObject {

    var name:String?
    var age:Int?

    init(name:String, age:Int) {

        self.name = name
        self.age = age

    }
}
```

*객체를 만들 때 필요한 데이터가 있다면?

```
var classObj:Person = Person(name:”Joo”, age:3)
```

init 으로 초기화 함수를 만들어서
객체를 생성한다



< MathClass > —> 위 예시에서의 CalculationCalss입니다.

```
class MathClass: NSObject {

    func add(num1:Int, num2:Int) -> Int {
        return num1+num2
    }

    func minus(num1:Int, num2:Int) -> Int {
        return num1-num2
    }

}
```

파일을 새로 만들어서 **MathClass** 안에 함수를 정의해주었습니다.

<ViewDidLoad에서 사용할 때>

```
// MathClass를 가지고 만든 객체
var math:MathClass = MathClass()

// 객체 math를 통해 MathClass에서 정의했던 함수들을 가져다 쓸 수 있다.
print (math.add(1, num2: 2))
print(math.minus(3, num2: 2))
```

MathClass를 **math**로 객체화 하였습니다.
MathClass타입의 math객체를 통해 MathClass에서 정의되었던 함수를 객체 math를 통해 접근할 수 있습니다.

<해깔릴 수 있는 점>

```
// MathClass를 가지고 만든 객체
// 그러나 얘네는 같은 타입일 뿐, 다른 객체입니다.
var math:MathClass = MathClass()
var math2:MathClass = MathClass()
```

---------- 밴딩머신 관련 예제 답안 ----------

```
//남은돈 변수
var remainCoin:Int = 0
//음료수 데이터
let drinkData:[String:Int] = ["drink1":1000,"drink2":800,"drink3":
    1300,"drink4":900]


//키값을 받아서 음료수 가격 얻음
func getDrinkCostWithKey(key:String) -> Int
{
    //key = 선택된 음료수의 이름
    //전체 음료수 데이터를 순서대로 순회하며 선택된 음료수의 이름과 같은 것이 있는지 확인한다.
    for (drinkName, cost) in drinkData {
        if (key == drinkName)
        {
            return cost
        }
    }

    return 0
}



//돈 계산
func calculationInCoin(cost:Int) -> Bool
{
    if (remainCoin >= cost)
    {
        remainCoin -= cost
        return true
    }
    return false

}
```

```swift
//잔돈 확인
func printRemainCoin(coin:Int) -> String
{
    var remainCoin:Int = coin

    var rc500Count:Int = 0
    var rc100Count:Int = 0
    var rc50Count:Int = 0
    var rc10Count:Int = 0

    if (0<remainCoin/500)
    {
        rc500Count = remainCoin/500
        remainCoin = remainCoin%500
    }
    if (0<remainCoin/100)
    {
        rc100Count = remainCoin/100
        remainCoin = remainCoin%100
    }
    if (0<remainCoin/50)
    {
        rc50Count = remainCoin/50
        remainCoin = remainCoin%50
    }
    if (0<remainCoin/10)
    {
        rc10Count = remainCoin/10
        remainCoin = remainCoin%10
    }

    //예: 500원짜리 1개, 100원짜리2개...
    return "거스름 돈은 500원짜리 \(rc500Count)개, 100원짜리 \(rc100Count)개, 50원짜리 \
        (rc50Count)개, 10원짜리 \(rc10Count)개 입니다."
}
```