

위 소스 코드는 1급 객체를 만족하는 3가지 조건을 해당 사항에 맞게 번호를 마킹 한 것이다.

**func** up(num: Int) -> Int {

**return** num + 1

}

**func** down(num: Int) -> Int {

**return** num - 1

}

**let** toUp = up

print(up(num:10))

print(toUp(10)) // 일급 객체 1번 조건 사용시 레이블 안넘겨줌

**let** toDown = down

**func** upDown(Fun: (Int) -> Int, value: Int) {

**let** result = Fun(value)

print("결과 = **\(**result**)**")

}

upDown(Fun:toUp, value: 10) *//toUp(10)*

upDown(Fun:toDown, value: 10) *//toDown(10)*

**func** decideFun(x: Bool) -> (Int) -> Int {

*//매개변수형 리턴형이 함수형*

**if** x {

**return** toUp

} **else** {

**return** toDown

}

}

**let** r = decideFun(x:**true**) *// let r = toUp : 함수를 리턴해서 변수에 저장*

print(type(of:r)) *//(Int) -> Int*

print(r(10)) *// toUp(10)*

*// 11*

*// 11*

*// 결과 = 11*

*// 결과 = 9*

*// (Int) -> Int*

*// 11*

**함수를 클로저로 바꿔보기 실습**

**func** add(x: Int, y: Int) -> Int {

**return**(x+y)

}

print(add(x:10, y:20))

**let** add1 = { (x: Int, y: Int) -> Int **in**

**return**(x+y)

}

*//print(add1(x:10, y:20)) //주의 error: extraneous(관련 없는) argument labels 'x:y:' in call*

print(add1(10, 20)) *//OK 1급 객체 1번 조건 클로저도 해당됨*

print(type(of:add1)) *//과제*

*// 함수를 클로저로 바꿀때는 앞에 fun 키워드와 이름을 지우고*

*// 블락을 땡겨 전체로 묶은 다음 in 키워드를 사용해줌*

*// 일종의 함수이니 변수에 넣을수 있음 : 즉 일급 객체이기 때문에 대입 가능*

*// 클로저를 만드는 이유는 한번만 호출될거 같을때 함수를 따로 만들 필요 없이 그 자리에 기능을 넣어주겠다 할때 많이 사용함*

*// 30*

*// 30*

*// (Int, Int) -> Int*

**func** mul(val1: Int, val2: Int) -> Int

{

**return** val1 \* val2

}

**let** result = mul(val1:10, val2:20)

print(result)

**let** multiply = {(val1: Int, val2: Int) -> Int **in**

*//매개변수 리턴형*

**return** val1 \* val2

}*// 여기서 multiply의 자료형은 (Int, Int) -> Int*

**let** result2 = multiply(10, 20) *//상수를 함수처럼 호출,200*

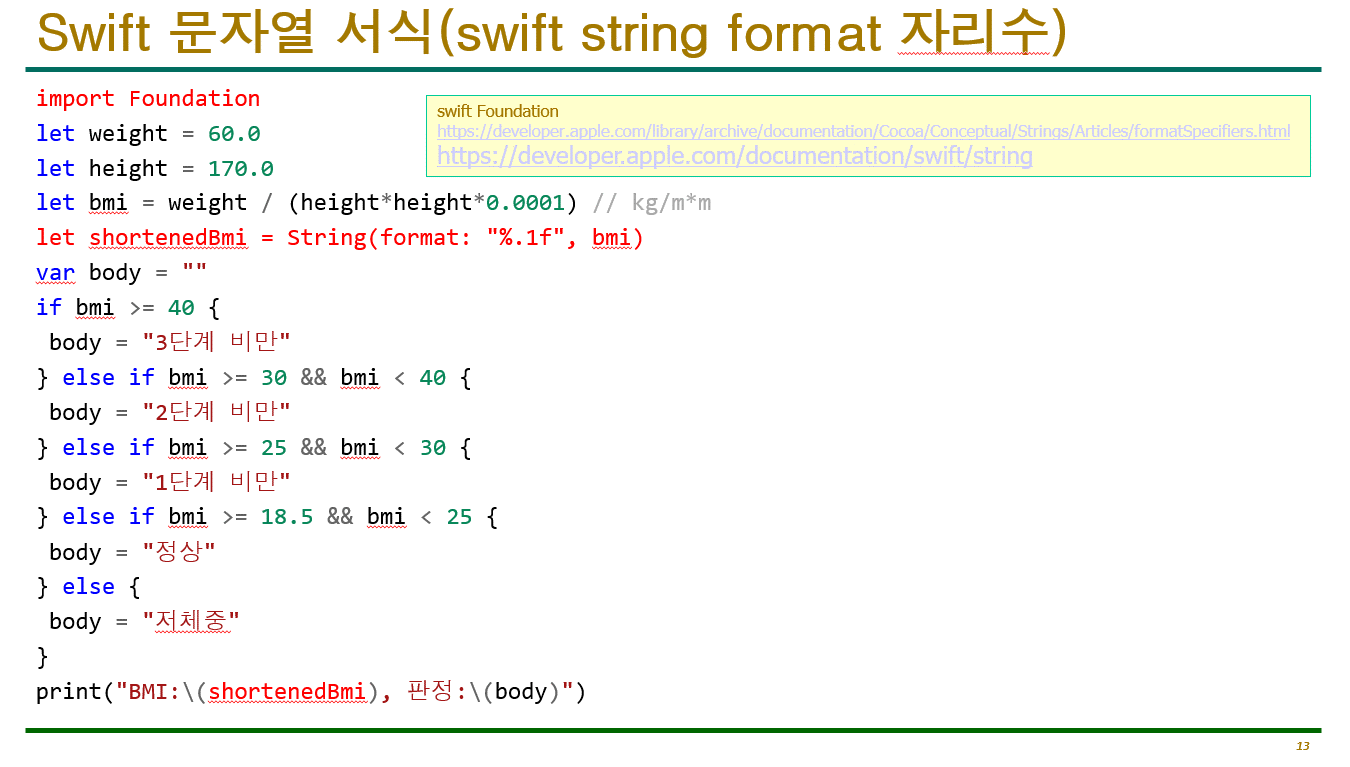
print(result2)

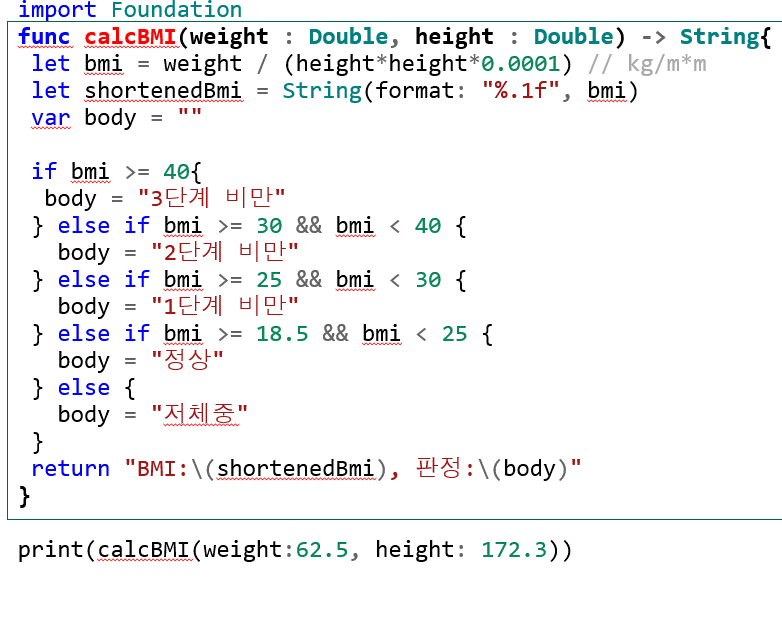
print(type(of:multiply))

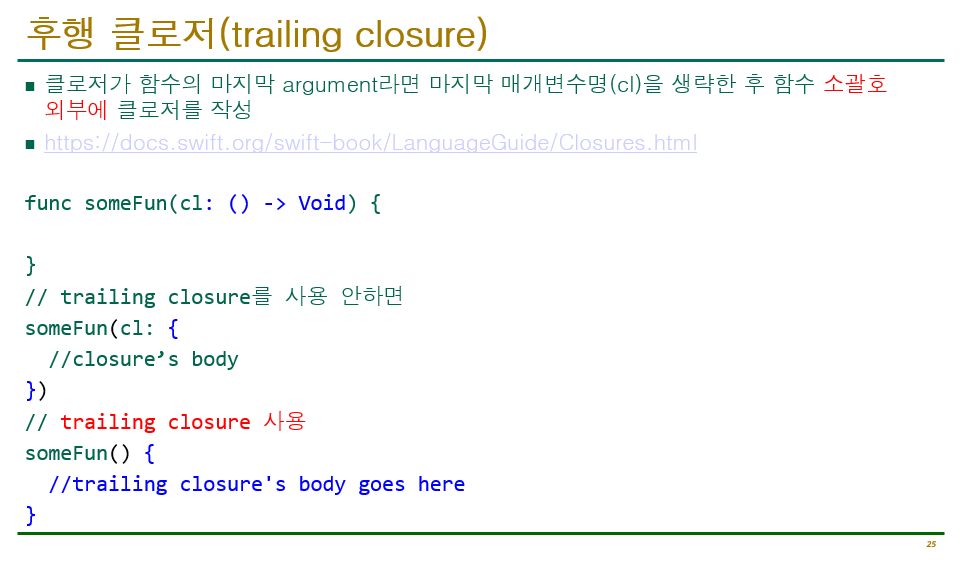
*// 200*

*// 200*

*// (Int, Int) -> Int*

위 코드 아래 함수로 만들 수 있어야함





**위 사진 개념 적기**

*/\**

*아래와 같은 함수를 클로저로 바꿔 보고 다양한 클로저 축약 표현들을 익혀보자*

*\*/*

*// 함수 대신 클로저를 활용하여 상수에 값을 저장하기*

**func** math(x: Int, y: Int, cal: (Int, Int) -> Int) -> Int { *// 일급 시민 함수를 마지막 매개변수로 받기*

**return** cal(x, y)

}

result = math(x: 10, y: 20, cal: add)

print(result)

result = math(x: 10, y: 20, cal: multiply)

print(result)

result = math(x: 10, y: 20, cal: {(val1: Int, val2: Int) -> Int **in**

**return** val1 + val2

}) *//클로저 소스를 매개변수에 직접 작성*

print(result)

*// 클로저가 함수의 마지막 argument라 매개 변수 이름을 생략후 : 후행 클로저로 변환*

result = math(x: 10, y: 20) {(val1: Int, val2: Int) -> Int **in**

**return** val1 + val2

}*//trailing closure*

print(result)

*// 클로저의 다양한 축약 표현들*

result = math(x: 10, y: 20, cal: {(val1: Int, val2: Int) **in** *// -> Int*

**return** val1 + val2

}) *//리턴형 (반환 타입) 생략*

print(result)

result = math(x: 10, y: 20) {(val1: Int, val2: Int) **in** *// -> Int*

**return** val1 + val2

} *//trailing closure, 리턴형 (반환 타입)생략*

print(result)

result = math(x: 10, y: 20, cal: {

**return** $0 + $1 *//$0 + $1 숫자 증가에 따라 순서대로 들어오는 매개변수 순서가 됨*

}) *//매개변수 생략하고 단축인자(shorthand argument name)사용*

print(result)

result = math(x: 10, y: 20) {

**return** $0 + $1

} *//trailing closure, 매개변수 생략하고 단축인자사용*

print(result)

result = math(x: 10, y: 20, cal: {

$0 + $1

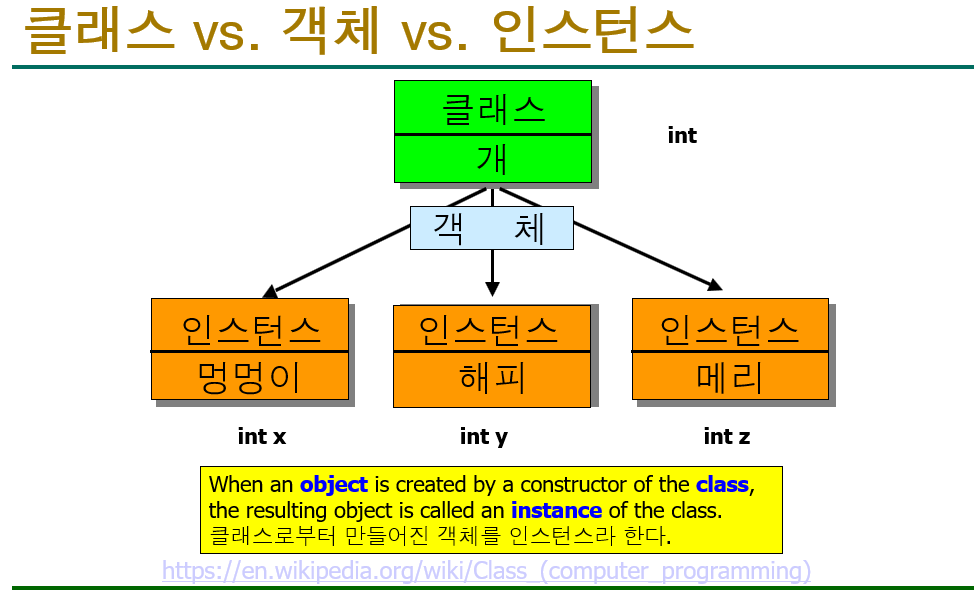
}) *//클로저에 리턴 값이 있으면 마지막 줄을 리턴 하므로 return생략*

print(result)

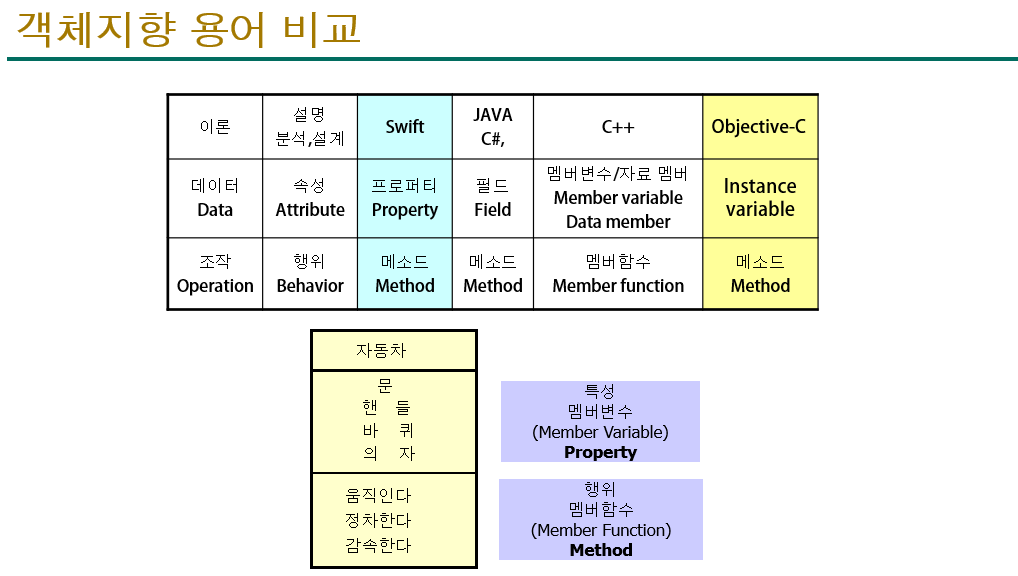
result = math(x: 10, y: 20) { $0 + $1 } *//return 생략*

print(result)

**클래스 vs. 객체 vs. 인스턴스**

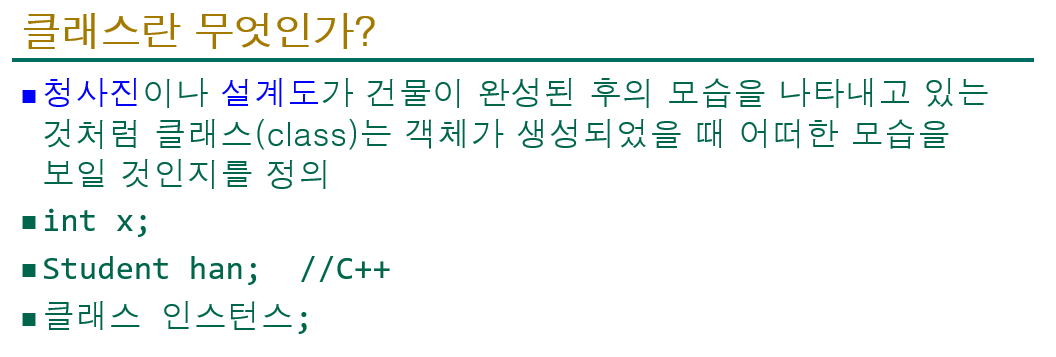


개념 적기



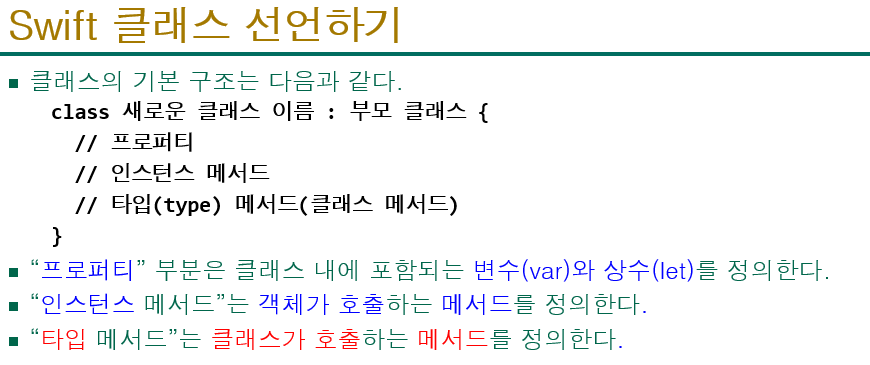
**그렇다면 클래스란 무엇인가?**

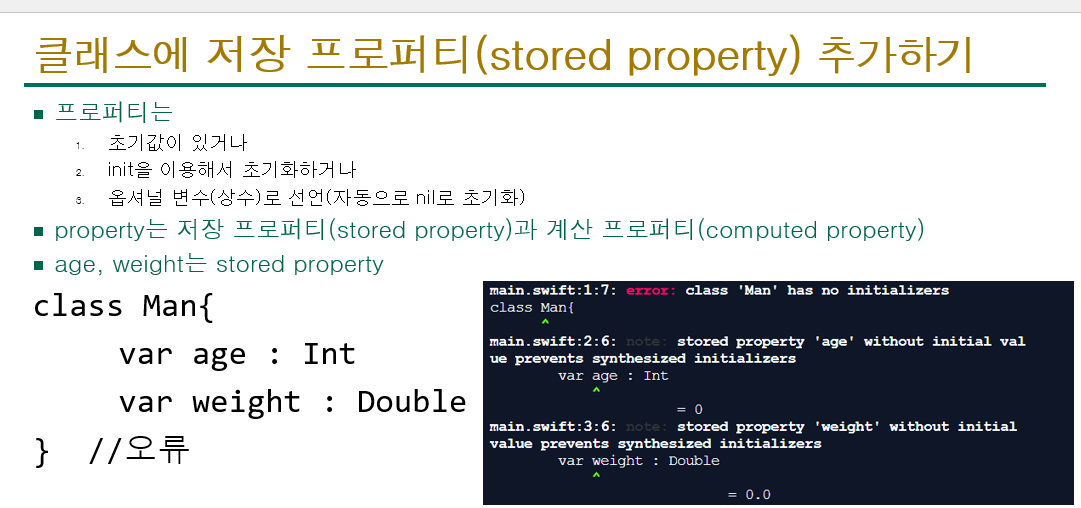
property는 저장 프로퍼티(stored property)과 계산 프로퍼티(computed property)

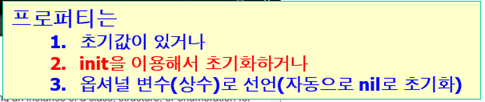


//쉽게 새로운 자료형을 만드는 것이라고 생각하자

**참고 자료 (자바와 문법 비교)**







*//stored property 초기값 주는 방법은 3가지가 있음*

**class** **Man1**{

**var** age : Int = 1 *//stored property는 초기값이 있어야 함*

**var** weight : Double = 3.5 *// 값 직접 주는 방법 1*

}

*// 닐 즉 옵셔널로 줘도 됨 2*

**class** **Man2**{

**var** age : Int? *//stored property는 초기값이 있어야 함, nil*

**var** weight : Double!

}

**3번째 init 을 이용한 초기화 방법 (원래 우리가 사용하고 있지만 숨겨져서 안보였던 것 우리가 정의 해서 초기화 시켜줄 수 있음)**

**인스턴스가 만들어지면서 자동 호출됨 ,init 메서드(생성자)**

*// 인스턴스 만들고 메서드와 프로퍼티 접근*

*// var 인스턴스명 = 클래스명()*

*// 클래스명 다음의 괄호는 눈에 보이지 않는 default initializer를 나타냄*

*// var Kim : Man = Man() //:Man은 생략 가능*

**class** **Man**{

**var** age : Int = 1

**var** weight : Double = 3.5

**func** display(){

print("나이=**\(**age**)**, 몸무게=**\(**weight**)**")

}

*//init() 보이지 않고 숨겨져 있음*

}

**var** kim : Man = Man()

kim.display() *//인스턴스 메서드는 인스턴스가 호출*

print(kim.age)

*// 나이=1, 몸무게=3.5*

*// 1*

*생성자 정의해보자*

**class** **Man**{

**var** age : Int

*// = 1 생성자 있어서 생략 가능 프로퍼티 초기화 3가지 조건 중 하나만 만족 하면 됨*

**var** weight : Double *// = 3.5*

**func** display(){

print("나이=**\(**age**)**, 몸무게=**\(**weight**)**")

}

**init**(yourAge: Int, yourWeight : Double){

age = yourAge //self.age = yourAge 현재 self 생략가능 이름이 다르기 때문에

weight = yourWeight

} *//designated initializer*

}

*//var kim : Man = Man() //오류*

*//init()을 하나라도 직접 만들면 default initializer는 사라짐*

**var** kim : Man = Man(yourAge:10, yourWeight:20.5) *// 자동으로 initializer 호출 됨*

kim.display()

*// 나이=10, 몸무게=20.5*

*매개변수 프로퍼티 이름이 같을 때*

*// 현재 클래스 내 메서드나 프로퍼티를 가리킬 때 메서드나 프로퍼티 앞에 self.을 붙임*

*// 다른 언어의 this와 같음(다른 포인터를 사용하지 않는 언어 ) 포인터 관련해서 this 키워드가 정의 되어 있기 때문에 다른 언어와 차이를 둠*

**class** **Man**{

**var** age : Int = 1

**var** weight : Double = 3.5

**func** display(){

print("나이=**\(**age**)**, 몸무게=**\(**weight**)**")

}

**init**(age: Int, weight : Double){

**self**.age = age *//프로퍼티 = 매개변수*

**self**.weight = weight

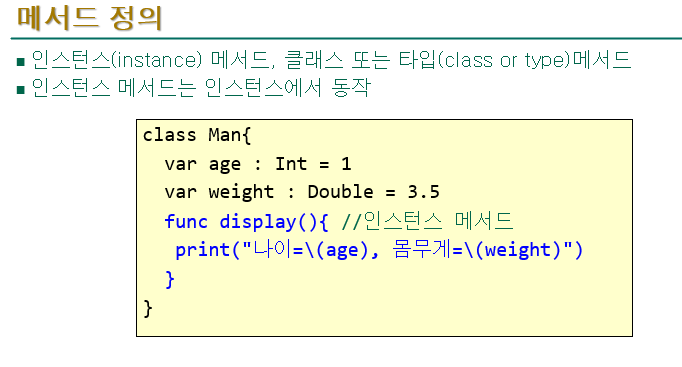
}

}

**var** kim : Man = Man(age:10, weight:20.5)

kim.display()

*// 나이=10, 몸무게=20.5*



자바에 static 메소드처럼 타입 메소드가 있음

이렇게 func 앞에 아무것도 없으면 일반적인 인스턴스 메서드

저 앞에 class or type이 들어가면 타입 메서드임 (클래스가 가지고 노는)

**class** **Man**{

**var** age : Int = 1

**var** weight : Double = 3.5

**func** display(){ *//인스턴스 메서드*

print("나이=**\(**age**)**, 몸무게=**\(**weight**)**")

}

}

**var** kim : Man

print(kim.age)

*// 이러한 에러가 남*

*// 주의 할 점 위와 같이 코드를 작성하면 제대로 인스턴스가 만들어 지지 않음*

*// 인스턴스 만들 때 조심 해야함*

*// main.swift:13:7: error: variable 'kim' used before being initialized*

*// print(kim.age)*

*// ^*

*// main.swift:12:5: note: variable defined here*

*// var kim : Man*

*// ^*

클래스 객체 만들 때 주의

**class** **Man**{

**var** age : Int = 1

**var** weight : Double = 3.5

**func** display(){ *//인스턴스 메서드*

print("나이=**\(**age**)**, 몸무게=**\(**weight**)**")

}

}

**var** x : Int = 3

*//var kim : Man*

**var** kim : Man = Man()

**var** kim2 = Man() *// 이렇게 생략 가능 함*

print(kim.age) *// 1*

*// Swift 변수를 만들 때 는 그냥 저렇게 만들어도 되지만*

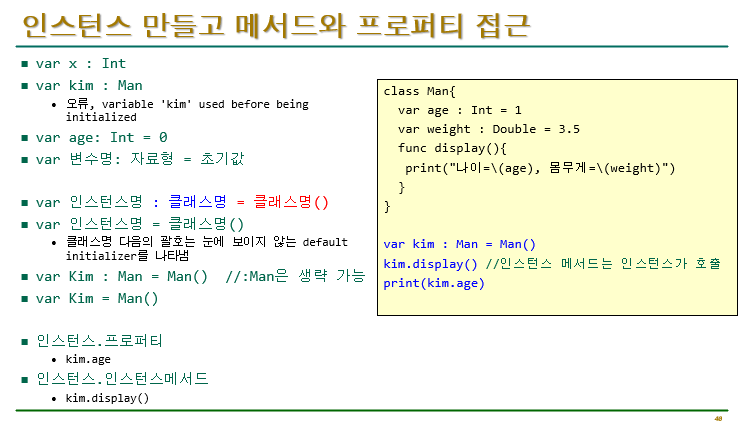
*// 클래스의 인스턴스 객체를 만들 때 는*

*// 반드시 = 클래스 () 이렇게 작성 해야함*

*// initialize 함수때문에 그럼*

*// 자바처럼 클래스를 만들 때 자동으로 생성해 주는 함수인데*

*// Swift에서도 반드시 객체 만들 때 는 initialize 함수 이용해서 만들어야 함*



**타입 메서드 만들기**

**class** **Man** {

**var** age : Int = 1

**var** weight : Double = 3.5

*// 인스턴스 메서드*

**func** display(){

print("나이=**\(**age**)**, 몸무게=**\(**weight**)**")

}

*// 타입 메서드*

**class** **func** cM(){

print("cM은 클래스 메서드입니다.") *// 클래스 메서드는 안에 프로퍼티 age 같은거 가지고 못 놈*

}

**static** **func** scM(){

print("scM은 클래스 메서드(static)")

}

}

**var** kim : Man = Man()

kim.display() *//인스턴스 메서드는 인스턴스가 호출*

Man.cM() *//클래스 메서드는 클래스가 호출*

Man.scM() *//클래스 메서드는 클래스가 호출*

*// class키워드로 만든 클래스 메서드는 자식 클래스에서 override가능 함*

*// 나이=1, 몸무게=3.5*

*// cM은 클래스 메서드입니다.*

*// scM은 클래스 메서드(static)*