

< 클래스와 생성자 >

클래스 선언

방법: class 키워드, 본문 내용이 없다면 {} 3 생각 가능

구성요소: 클래스 멤버는 생성자, 변수, 함수, 클래스로 구성
하위클래스

★ 생성자는 constructor 키워드로 선언하는 함수 ★★

객체 생성시 new 키워드 사용 X

생성자 종류 < 주 생성자
보조 생성자

constructor()는 생성자 함수 선언을 위한 것이지만
절대적으로 주 생성자와 관련된 것은 아님.

ex) val user = User("kim")
user.someFun()

객체 생성시 클래스 이름을 사용하여 호출
호출시, 클래스의 constructor() 함수가 자동 호출

주 생성자 선언 class User constructor() {}

주 생성자 선언은 필수 X

한 클래스에 하나의

주 생성자만 가능함.

↓
키워드 생략 class User {}

↓
대개변수 X class User {}

↓

주 생성자의 본문 - init 영역 (초기화 블록)

init 영역은 객체 생성시 자동으로 1번 실행.

init과 constructor() 차이점.

키워드 영역

클래스 내 생성자 메소드

: 클래스에 하나 존재 : 생성자 함수 → 매개변수에 따라 클래스 내에 2개 이상 들수 있다.

★ ★ 생성자의 매개변수로 클래스의 멤버변수 선언

```
class User(name: String, count: Int) {  
    init {  
        println("name : $name, count : $count") // 성공!  
    }  
    fun someFun() {  
        println("name : $name, count : $count") // 오류!  
    }  
}
```

생성자의 매개변수는

init 이나, constructor() 내에서만
사용할 수 있는 지역변수

동일 객체 내 타 함수에서는 해당 매개변수를 사용X

• 생성자의 매개변수를 클래스의 멤버 변수로 선언하는 방법

```
class User(val name: String, val count: Int) {  
    fun someFun() {  
        println("name : $name, count : $count") // 성공!  
    }  
}  
  
fun main() {  
    val user = User("kkang", 10)  
    user.someFun()  
}
```

▶ 실행 결과

name : kkang, count : 10

매개변수 전달할 때, val 이나 var 키워드를 사용하면 ...
init 영역을 통한 별도의 멤버변수 선언이 필요없음.

```

class User1 constructor(_name: String, _count: Int) {
    var name: String = _name
    var count: Int = _count

    fun someFun() {
        println("name: $name, count: $count")
    }
}

class User2 constructor(name: String, count: Int) {
    var name: String
    var count: Int

    // 초기화 블록
    init {
        this.name = name
        this.count = count
        println("다른 초기화 작업")
    }

    fun someFun() {
        println("name: $name, count: $count")
    }
}

```

생성자 매개변수로 멤버변수 초기화
→ init, constructor() 외에서도 접근가능

함수내의 멤버변수는 선언과 동시에
초기화 해주지 않아도 됨.

매개변수명과 멤버변수명이 동일
this. 변수명으로 구분

```

class User3 constructor(var name: String, var count: Int) {
    init {
        println("다른 초기화 작업")
    }

    fun someFun() {
        println("name: $name, count: $count")
    }
}

```

자동으로 매개변수 → 멤버변수
선언 및 초기화가 됨.

```

class User4 constructor(var name: String, var count: Int)

```

→ 매개변수가 있고, var 키워드로 인해 멤버변수 할당 및 초기화 완료.

→ 다른 초기화작업. 함수가 없으면 주 생성자의 constructor 키워드 생략 가능

→ 생성자의 본문 생략 가능.

보조생성자

- "클래스 본문"에서 constructor 키워드로 선언하는 함수.
- 한 클래스에서 여러 개 추가 가능

☆ 보조생성자로 객체를 생성할 때, 클래스 내 주 생성자가 있으면 this() 구문을 이용해 주 생성자 호출.
클래스 선언 헤더부분에... () 함수 호출 여부 확인할 것.

Inheritance

상속과 생성자

class 클래스이름 : 상속받을 클래스이름.

Kotlin에서 기본적으로 상속할 수 없음 → "open" 키워드 필요.

* 상속받은 클래스에서 생성자 호출.

상위클래스의 생성자 호출.

상위클래스의 모든 멤버변수를 포함하고
하위클래스에서는 추가 멤버변수를 추가할 수 있음.

↑
상속받을 부모 클래스 앞에 open 키워드 추가

```
open class Super {  
}  
  
class Sub : Super() {  
}
```

Super 클래스를 상속받으면서 그
클래스의 매개변수가 없는 생성
자를 호출함

Sub 클래스는 Super 라는 상위클래스를 상속받고있음.

Super 는 매개변수가 없는 생성자 함수를 가짐.

Sub에서는 상위클래스의 매개변수가 없는 생성자를 호출

→ 표현을 class Sub : Super()

• 하위 클래스에 보조 생성자만 있는 경우 상위 클래스의 생성자 호출

```
open class Super(name: String) {  
}  
  
class Sub: Super() {  
    constructor(name: String): super(name) {  
    }  
}
```

보조생성자를 사용하는 상속

매개변수가 있는 클래스를 상속받고있음.

그러나, 클래스 선언 (header) 에서 생성자 호출 인공 X ... 주 생성자가 없음.

보조생성자에서 매개변수 명시와 상위클래스 생성자 호출이 필요함.

오버라이딩: 상위클래스에서 선언된 변수나 함수를 같은 이름으로 하위클래스에서 다시 선언하는 것.

• 오버라이딩 예

```
open class Super {  
    open var someData = 10  
    open fun someFun() {  
        println("i am super class function : $someData")  
    }  
}  
  
class Sub: Super() {  
    override var someData = 20  
    override fun someFun() {  
        println("i am sub class function : $someData")  
    }  
}  
  
fun main() {  
    val obj = Sub()  
    obj.someFun()  
}
```

부모의 멤버변수에
모든 open 키워드를
다 써야 하는가?

멤버변수 someData는
var로 선언되어 있어서
자식클래스에서
수정이 가능하지 않은가?

open을 사용했을 때
static 처럼 클래스 자체
전체에 영향을 미치는 것인가?

pairwise 쌍을 이름.

▶ 실행 결과

i am sub class function : 20

접근 제한자 (정보원격과 관련된 부분)

클래스의 멤버를 어느범위까지 이용하게 할 것인가? 결정하는 키워드.

	최상위에서 이용. 클래스에서 못지 않는 변수나 함수	클래스 멤버에서 이용
public	always used.	모든 클래스에서 가능
internal	같은 module 내 사용 (app)	같은 module 내 사용
(+상속개념) protected	클래스로 묶여있지 않으므로 어느곳에서도 사용불가	클래스를 상속받은 하위클래스에서만 사용가능 → 본인에게서도 접근제한
private	파일 내부에서만 사용	클래스 내부에서만 이용.

접근 제한자 사용 예

```
open class Super {  
    var publicData = 10    // 접근 제한자를 생략하면 public이 기본  
    protected var protectedData = 20  
    private var privateData = 30  
}  
  
class Sub: Super() {  
    fun subFun() {  
        publicData++    // 성공!  
        protectedData++    // 성공!  
        privateData++    // 오류!  
    }  
}  
  
fun main() {  
    val obj = Super()  
    obj.publicData++    // 성공!  
    obj.protectedData++    // 오류! ← 상속관계에 있는 하위클래스에서만 제어가능  
    obj.privateData++    // 오류!  
}
```

why 데이터 클래스를 별도로 두었는가? How to use.

데이터 클래스

- 선언: class 키워드 앞에서 "data" 키워드 추가
- equals() : 데이터 클래스 내 데이터를 비교 함수

↔ NonDataClass의 equals()
: 객체 자체를 비교 (객체의 참조, 주소의 동등성 비교)

DataClass의 equals()

: 데이터 클래스 객체는 서로 다르나,
두 객체의 데이터가 같다면 true 반환.
객체 멤버변수 단위에서 값의 동등성을 보는다.

데이터 클래스 주 생성자에
선언한 멤버변수의 데이터가 비교대상

• 데이터 클래스의 equals() 함수

```
data class DataClass(val name: String, val email: String, val age: Int) {  
    lateinit var address: String  
    constructor(name: String, email: String, age: Int, address: String):  
        this(name, email, age) {  
            this.address = address  
        }  
}  
  
fun main() {  
    val obj1 = DataClass("kkang", "a@a.com", 10, "seoul")  
    val obj2 = DataClass("kkang", "a@a.com", 10, "busan")  
    println("obj1.equals(obj2) : ${obj1.equals(obj2)}")  
}
```

주 생성자
init 키워드 내 선언된 멤버변수
비교대상.
주 생성자에 선언된 멤버변수
비교대상.
→

▶ 실행 결과

```
obj1.equals(obj2) : true
```


객체의 데이터를 반환하는 toString() 함수

데이터클래스를 사용하면서... 객체가 가지는 값을 확인해야 할 때 사용.

non data class의 toString() ⇒ 객체가 가지는 값
객체가 저장된 주소, 파일경로.

data class의 toString() ⇒ 클래스 내 값 정보 자체를 반환

• 데이터 클래스의 toString() 함수

```
fun main() {  
    class NonDataClass(val name: String, val email: String, val age: Int)  
    data class DataClass(val name: String, val email: String, val age: Int)  
    val non = NonDataClass("kkang", "a@a.com", 10)  
    val data = DataClass("kkang", "a@a.com", 10)  
    println("non data class toString : ${non.toString()}")  
    println("data class toString : ${data.toString()}")  
}
```

▶ 실행 결과

non data class toString : com.example.test4.ch2.Test2Kt\$main\$NonDataClass@61bbe9ba ← path 정보.
data class toString : DataClass(name=kkang, email=a@a.com, age=10) ← 값 자체 반환

패키지명, 파일이름, 함수이름, 클래스이름, 객체해시코드

Object 클래스

목적: 익명클래스를 만들 목적으로 사용

↳ 왜 익명클래스를 사용할까? 사용사례, 선언하자마자 바로 사용(일회성)

사용방법: 선언과 동시에 객체를 생성 ⇒ object 키워드 사용

• 오브젝트 클래스 사용 예

```
val obj = object : Any {
    var data = 10
    fun some() {
        println("data : $data")
    }
}

fun main() {
    obj.data = 20 // 오류!
    obj.some()   // 오류!
}
```

Any 타입

← 클래스 타입을 명시하지 않았으므로 Any type으로 추론된다.

← ... 그러나 Any type에는 data, some() 이라는 멤버변수가 없다 ⇒ 그렇다면 Any 대신 데이터 타입지정이 필요

• 타입을 지정한 오브젝트 클래스

```
open class Super {
    open var data = 10
    open fun some() {
        println("i am super some() : $data")
    }
}

val obj = object : Super() {
    override var data = 20
    override fun some() {
        println("i am object some() : $data")
    }
}

fun main() {
    obj.data = 30 // 성공!
    obj.some()   // 성공!
}
```

Super 타입

특정클래스를 명시해야함.

익명클래스가 어떤 클래스를 상속받아 선언되도록 해야함. 선언과 동시에 해당 객체가 생성

▶ 실행 결과

```
i am object some() : 30
```

Companion 클래스

· 객체를 생성하지 않고... 멤버변수나 함수를 클래스 이름으로 접근.

· Companion 키워드로 선언 ... 자바에서 static 클래스와 같은 개념.

• 컴패니언 클래스의 멤버 접근

```
class MyClass {
    companion object {
        var data = 10
        fun some() {
            println(data)
        }
    }
}

fun main() {
    MyClass.data = 20 // 성공!
    MyClass.some()   // 성공!
}
```

← 클래스단위에서
공통적인 속성.

obj는 Super 타입의 객체인데, Super 클래스에는 data와 some() 이라는 멤버가 있다.