03-3. 상속

Object Oriented Programming

상속

Any 클래스

- 클래스를 선언할 때 코드에 명시적으로 상위 클래스를 선언하지 않으면 기본으로 Any의 서브 클래스
- Any 클래스는 자바의 java.lang.Object 클래스와는 다릅니다. Any 클래스는 equals ( ), toString ( ) 이외의 다른 멤버들은 제공하지 않습니다.

```
class Shape {
    var x: Int = 0
    var y: Int = 0
    var name: String = "Rect"

    fun draw() {
        println("draw $name : location : $x, $y")
    }
}

fun main(args: Array<String>) {
    val obj1: Any = Shape()
    val obj2: Any = Shape()
    val obj3 = obj1
    println("obj1.equals(obj2) is ${obj1.equals(obj2)}")
    println("obj1.equals(obj3) is ${obj1.equals(obj3)}")
}
```

상속

상속을 통한 클래스 정의

- 코틀린에서 클래스는 개발자가 명시적으로 선언하지 않아도 기본은 final
- open 예약어로 선언한 클래스만 상속 가능

```
open class Shape {
    var x: Int = 0
        set(value) {
            if(value < 0) field = 0
            else field = value
        }

    var y: Int = 0
        set(value) {
            if(value < 0) field = 0
            else field = value
        }

    lateinit var name: String

    fun print() {
        println("$name : location : $x, $y")
    }
}
```

상속

상속을 통한 클래스 정의

- 코틀린에서 상속 관계 표현은 콜론(:)을 이용합니다. 클래스 선언 부분에 상위 클래스를 콜론 오른쪽에 명시해 상속 관계를 표현합니다.

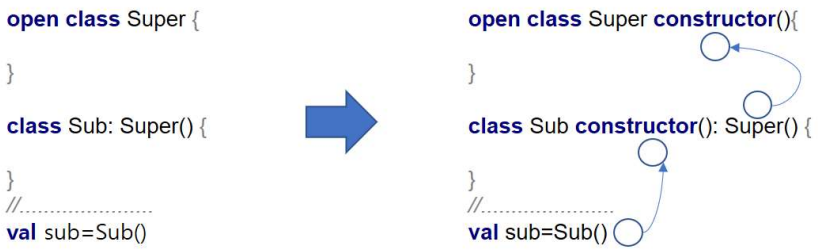
```
class Rect: Shape()  {
    var width: Int = 0
        set(value) {
            if(value < 0) field = 0
            else field = value
        }
    var height: Int = 0
        set(value) {
            if(value < 0) field = 0
            else field = value
        }
}

class Circle: Shape() {
    var r: Int = 0
        set(value) {
            if(value < 0) field = 0
            else field = value
        }
}
```

상속과 생성자

상위 클래스 생성자 호출

- 하위 객체 생성시 어떤 식으로든 상위 클래스의 생성자는 무조건 실행되어야 한다

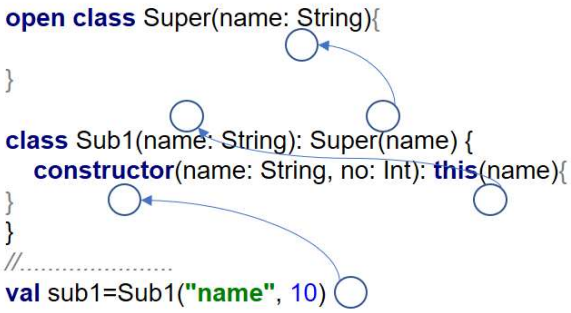
```
open class Super {

}

class Sub: Super() {

}
//.....................
val sub=Sub()
```



상속과 생성자

상위 클래스 생성자 호출

- 하위 클래스에 주생성자가 선언된 경우



상속과 생성자

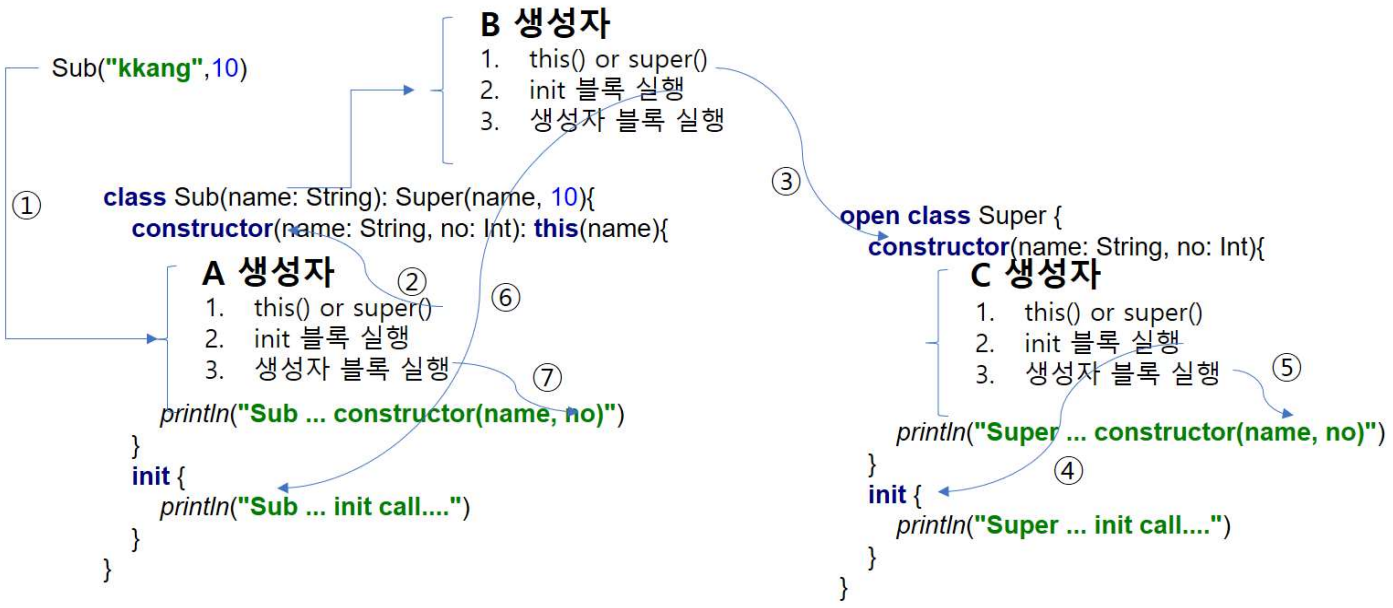
상하위간 생성자의 수행 흐름

```
open class Super {
    constructor(name: String, no: Int){
        println("Super ... constructor(name, no)")
    }
    init {
        println("Super ... init call....")
    }
}
class Sub(name: String): Super(name, 10){
    constructor(name: String, no: Int): this(name){
        println("Sub ... constructor(name, no) call")
    }
    init {
        println("Sub ... init call....")
    }
}

fun main(args: Array<String>) {

    Sub("kkang")
    println("..................................")
    Sub("kkang",10)
}
```

상속과 생성자

상하위간 생성자의 수행 흐름

1. this() 혹은 super() 에 의한 다른 생성자 호출
2. init 블록 호출
3. 생성자의 { } 영역 실행



상속과 캐스팅

기초 타입의 캐스팅

- 기초데이터 타입의 캐스팅은 자동 형변환이 안되고 함수에 의해서만 형변환이 가능

```
val data1: Int = 10
val data2: Double = data1.toDouble()
```

스마트 캐스팅

- is 예약어 이용시

```
fun smartCast(data: Any): Int{
    if(data is Int) return data*data
    else return 0
}

fun main(args: Array<String>) {
    println("result : ${smartCast(10)}")
    println("result : ${smartCast(10.0)}")
}
```

상속과 캐스팅

상속관계에서 스마트 캐스팅

- 하위 타입의 객체를 상위 타입에 대입

```
open class Super

class Sub1: Super()

//.....................
val obj1: Super = Sub1()
```

상속과 캐스팅

as 를 이용한 캐스팅

- as 를 이용한 캐스팅은 상속관계에 의한 객체의 명시적 캐스팅
- 하위타입->상위타입->하위타입

```
val obj3: Super = Sub1()
val obj4: Sub1 = obj3 as Sub1
obj4.superFun()
obj4.subFun1()
```

- 상위타입->하위타입

```
val obj5: Sub1 = Super() as Sub1 //런타임 에러
obj5.subFun1()
```

- 하위타입->하위타입

```
val obj6: Sub2 = Sub1() as Sub2// 런타임 에러
```

상속과 캐스팅

null 허용 객체의 캐스팅 as?

- as? 는 캐스팅 대상의 객체가 정상적인 객체이면 캐스팅을 진행하고 만약 Null이 대입되어 있으면 캐스팅을 진행하지 않고 Null을 리턴

```
val obj7: Super? = Sub1()
val obj8: Sub1 = obj7 as Sub1
```

```
val obj7: Super? = null
val obj8: Sub1 = obj7 as Sub1//런다임 에러
```

```
val obj7: Super? = null
val obj8: Sub1? = obj7 as? Sub1
```

감사합니다

단단히 마음먹고 떠난 사람은
산꼭대기에 도착할 수 있다.
산은 올라가는 사람에게만 정복된다.

윌리엄 셰익스피어
William Shakespeare