03-9. object 클래스

Object Oriented Programming

Nested 클래스

Nested Class

- Nested 클래스는 특정 클래스 내에 선언된 클래스를 지칭

```
class Outer {
    class Nested {
        val name: String = "kkang"
        fun myFun(){
            println("Nested.. myFun...")
        }
    }
}

fun main(args: Array<String>) {
    val obj: Outer.Nested = Outer.Nested()
    println("${obj.name}")
    obj.myFun()
}
```

Nested 클래스

Outer 클래스의 맴버 접근

- 기본으로는 Nested 클래스에서 외부 클래스의 멤버에 접근할 수 없습니다.

```
class Outer {
    var no: Int = 10
    fun outerFun() {
        println("outerFun()...")
    }
    class Nested {
        val name: String = "kkang"
        fun myFun(){
            println("Nested.. myFun...")
            no=20//error
            outerFun()//error
        }
    }
}
```

Nested 클래스

Outer 클래스의 맴버 접근

- Nested 클래스에서는 Outer 클래스의 맴버를 이용하려면 Nested 클래스 선언시 inner 라는 예약어를 추가해 주어야 한다.
- Outer 클래스의 멤버가 private로 선언됐더라도 Nested 클래스에서 이용할 수 있습니다.
- inner를 추가하면 inner가 추가된 클래스는 외부에서 객체로 생성할 수 없습니다.

```
class Outer {
    private var no: Int = 10
    fun outerFun() {
        println("outerFun()...")
    }
    inner class Nested {
        val name: String = "kkang"
        fun myFun(){
            println("Nested.. myFun...")
            no=20
            outerFun()
        }
    }
}

fun main(args: Array<String>) {
    val obj: Outer.Nested = Outer.Nested()//error
    println("${obj.name}")
    obj.myFun()
}
```

Nested 클래스

inner 클래스를 외부에서 이용

```
class Outer {
    private var no: Int = 10
    fun outerFun() {
        println("outerFun()...")
    }
    inner class Nested {
        val name: String = "kkang"
        fun myFun(){
            println("Nested.. myFun...")
            no=20
            outerFun()
        }
    }

    fun createNested(): Nested {
        return Nested()
    }
}

fun main(args: Array<String>) {
    val obj1: Outer.Nested = Outer().Nested()
    val obj2: Outer.Nested = Outer().createNested()
}
```

Object 클래스

object를 이용한 익명 내부 클래스 정의

- object { } 형태로 클래스를 선언
- 클래스명이 없지만 선언과 동시에 객체가 생성
- object 클래스에는 생성자는 추가할수 없다.

```
val obj1=object {
    var no1: Int = 10
    fun myFun() {

    }
}

class Outer {
    val obj2 = object {
        var no2: Int = 0
        fun myFun() {

        }
    }
}
```

Object 클래스

object 클래스의 멤버 이용

- 타입 문제로 불가능

```
class Outer {

    private var no: Int = 0

    val myInner = object {
        val name: String = "kkang"
        fun innerFun(){
            println("innerFun....")
            no++
        }
    }

    fun outerFun(){
        myInner.name//error
        myInner.innerFun()//error
    }
}

fun main(args: Array<String>) {
    val obj=Outer()
    obj.myInner.name//error
    obj.myInner.innerFun()//error
}
```

Object 클래스

object 클래스의 멤버 이용

- object 내에 선언된 멤버를 Outer 에서 사용하려면 객체가 private 으로 선언되어야 한다.

```
class Outer {

    private var no: Int = 0

    private val myInner = object {
        val name: String = "kkang"
        fun innerFun(){
            println("innerFun....")
            no++
        }
    }

    fun outerFun(){
        myInner.name
        myInner.innerFun()
    }
}

fun main(args: Array<String>) {
    val obj=Outer()
    obj.myInner.name//error
    obj.myInner.innerFun()//error
}
```

Object 클래스

타입 명시로 object 이용

- object 클래스를 만들 때 다른 클래스를 상속 받거나 인터페이스를 구현

```
interface SomeInterface {
    fun interfaceFun()
}
open class SomeClass {
    fun someClassFun(){
        println("someClassFun....")
    }
}

class Outer {
    val myInner: SomeClass =object : SomeClass(), SomeInterface {
        override fun interfaceFun() {
            println("interfaceFun....")
        }
    }
}

fun main(args: Array<String>) {
    val obj=Outer()
    obj.myInner.someClassFun()
}
```

Object 클래스

object 선언

- object 클래스명 { }

- 클래스명과 동일한 이름의 객체까지 같이 생성
- object 클래스명 { } 은 객체생성구문

```
class NormalClass {
    fun myFun(){ }
}
object ObjectClass {
    fun myFun() { }
}
fun main(args: Array<String>) {
    val obj1: NormalClass = NormalClass()

    obj1.myFun()

    val obj3: ObjectClass = ObjectClass()//error

    ObjectClass.myFun()
}
```

Object 클래스

companion 예약어

- Java 의 static 효과

```
class Outer {
    companion object NestedClass {
        val no: Int = 0
        fun myFun() { }
    }
    fun myFun(){
        no
        myFun()
    }
}
fun main(args: Array<String>) {
    Outer.NestedClass.no
    Outer.NestedClass.myFun()

    Outer.no
    Outer.myFun()
}
```

감사합니다

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

윌리엄 셰익스피어
William Shakespeare