03-2. 프로퍼티

Object Oriented Programming

프로퍼티

프로퍼티란?

- 코틀린에서는 클래스의 변수를 프로퍼티(Property)라 부른다.
- var, val로 선언되는 변수들이 프로퍼티 이다.
- getter/setter(accessor)함수가 자동으로 내장

```
class User {
    var name: String = "kkang"
    val age: Int = 20
}

fun main(args: Array<String>) {
    val user=User()

    user.name="kim"
    println("name : ${user.name}")
    println("age : ${user.age}")
}
```

프로퍼티

프로퍼티란?

- var로 선언한 변수는 값의 획득 및 변경이 가능함으로 get(), set() 이 추가
- val로 선언한 변수는 값의 변경이 불가능함으로 get() 만 추가
- get(), set() 의 field는 예약어로 프로퍼티에 저장된 값 자체를 지칭

```
class User {
    var name: String = "kkang"
        get() = field
        set(value) {field=value}

    val age: Int = 20
        get() = field
}

fun main(args: Array<String>) {
    val user=User()

    user.name="kim"
    println("name : ${user.name}")
    println("age : ${user.age}")
}
```

프로퍼티

사용자 정의 프로퍼티

- 개발자가 특정 작업(유효성 검증 등)에 프로퍼티 값을 이용하고자 한다면 getter/setter 함수를 직접 정의

```
//custom property
class User1 {
    var greeting: String = "Hello"
        set(value) {
            field = "Hello" + value
        }
        get() = field.toUpperCase()

    var age: Int=0
        set(value)  {
            if(value >0 ){
                field=value
            }else {
                field=0
            }
        }
}

fun main(args: Array<String>) {
    val user1=User1()
    user1.greeting="kkang"
    println(user1.greeting)
    user1.age=-1
    println("age : ${user1.age}")
}
```

프로퍼티

사용자 정의 프로퍼티

- get(), set() 내부에서는 프로퍼티 값을 field로 접근한다.
- var의 경우는 set() 과 get() 을 모두 정의할수 있지만 val 의 경우는 set() 을 정의할수 없다.
- val 의 경우 get() 을 정의하였다면 초기 값을 명시하지 안아도 된다.
- var의 경우는 get() 이 정의되었다고 하더라도 초기 값이 명시되어야 한다.

```
class User2 {
    //val 의 set() 사용시 에러
    val name: String = "kkang"
        get() = field.toUpperCase()
        set(value) { field = "Hello" + value}//error

    //val의 초기값 생략 가능
    val age: Int
        get() = 10

    //var의 경우 초기값 생략 불가능
    var phone: String//error
        get() = "01000000"
}
```

프로퍼티

주생성자와 프로퍼티

- 프로퍼티로 이용되는 것은 Top-Level 변수와 클래스 내부의 변수

```
//Top-Lvel Variable
var myVal: String="hello"
    get() = field.toUpperCase()
    set(value){
        field = "hello" + value
    }

class User1(val name: String){
    //Class Member Variable
    var age: Int=0
        set(value) {
            if(value>0) field=value
            else field=0
        }

    fun myFun(){
        //함수의 local variable은 프로퍼티가 아니다.
      var no=0
//            get() = field * 10//error
    }
}
```

프로퍼티

주생성자와 프로퍼티

- 주생성자의 var, val 이 추가된 변수에 Custom get(), set() 이 추가 불가

```
class User2(var name: String){
    var myName: String = name
        get() = field.toUpperCase()
        set(value) {
            field="Hello"+value
        }
}

fun main(args: Array<String>) {
    val user2=User2("kkang")
    user2.name="lee"
    user2.myName="kim"
    println("name : ${user2.name}") //lee
    println("myName : ${user2.myName}") //HELLOKIM
}
```

프로퍼티

주생성자와 프로퍼티

- 일반적으로 매개변수 값을 대입하는 클래스의 프로퍼티명은 매개변수명과 똑같이 작성

```
class User3(name: String){
    var name: String = name
        get() = field.toUpperCase()
        set(value) {
            field="Hello"+value
        }
}
fun main(args: Array<String>) {
    val user3=User3("kkang")
    user3.name="kim"
    println("name : ${user3.name}")
}
```

프로퍼티 초기화

초기화 블록에서 초기화

- 클래스의 프로퍼티를 선언하면서 동시에 초깃값을 대입하지 않고, 초기화 블록에서 프로퍼티를 초기화해 사용할 수 있습니다

```
class User {
    var data: String
    val data2: Int

    init {
        data="kkang"
        data2=10
    }
}
```

프로퍼티 초기화

null 허용으로 선언

- 프로퍼티를 null 허용으로 선언해 null로 초기화하는 방법

```
class User {
    val name1: String="kkang"
    var name2: String?=null
    val name3: String?=null
    var age: Int? = null

    constructor(name2: String, name3: String, age: Int){
        this.name2=name2
        this.name3=name3//error
        this.age=age
    }
}
```

프로퍼티 초기화

늦은 초기화

- 프로퍼티를 null 허용으로 선언하지 않고 프로퍼티 초기화를 미루는 방법도 있습니다. 이 방법은 lateinit 예약어를 이용하며, '늦은 초기화'라고 부릅니다.

```
class User1 {
    lateinit var lateData: String
}

fun main(args: Array<String>) {
    //lateinit
    val user=User1()
    user.lateData="hello"
    println(user.lateData)
}
```

프로퍼티 초기화

늦은 초기화

- lateinit은 var의 경우에만 사용이 가능하다.
- lateinit은 주생성자에서는 사용할수 없다.
- lateinit 은 custom getter/setter를 사용하지 않은 프로퍼티에만 사용할수 있다.
- nullable 프로퍼티에는 사용할수 없다.
- 타입은 기초 타입이 아니어야 한다.

```
lateinit var data1: String

class User2(lateinit var data: String) {//error
    lateinit val data2: String//error
    lateinit var data3: String?//error
    lateinit var data4: Int//error
    lateinit var data5: String//ok
}
```

프로퍼티 초기화

---

isInitialized

- lateinit으로 선언된 변수가 초기화 되었는지 확인
- 클래스 내부에 선언된 lateinit 을 클래스 밖에서 사용할 때 초기화 되었는지 직접 확인하는 방법은 없다.

```
lateinit var data1: String

class Test {
    lateinit var data2: String

    fun some() {
        //클래스 내부에 선언된 lateinit 이 최기화 되었는지 확인하는 방법
        println(::data2.isInitialized)
    }

    fun isData2Initialized(): Boolean {
        return ::data2.isInitialized
    }
}
```

프로퍼티 초기화

초기화 미루기

- 초기화를 프로퍼티의 이용 시점으로 미룰수 있는 방법
- lazy는 일종의 실행 영역
- by lazy { } 로 선언되어 프로퍼티의 초기화를 lazy의 { } 에서 진행

- 호출 시점에 초기화 진행
- val 과 같이 사용 가능
- 클래스 바디 , Top-Level 에서 사용가능
- primary type에도 사용 가능

프로퍼티 초기화

초기화 미루기

```
val someData: String by lazy {
    println("i am someData lazy...")
    "hello"
}
class User1 {
    val name: String by lazy {
        println("i am name lazy...")
        "kkang"
    }
    val age: Int by lazy {
        println("i am age lazy...")
        10
    }
    init {
        println("i am init...")
    }
    constructor(){
        println("i am constructor...")
    }
}
fun main(args: Array<String>) {
    //lazy 초기화
   val user1=User1()
    println("name use before...")
    println("name : ${user1.name}")
    println("name use after....")
    println("age use before...")
    println("age : ${user1.age}")
    println("age use after....")
}
```

실행결과

```
i am init...
i am constructor...
name use before...
i am name lazy...
name : kkang
name use after....
age use before...
i am age lazy...
age : 10
age use after....
```

프로퍼티 변경 감지

```
class User {
    var name: String by Delegates.observable("nonValue", {props, old, new ->
        println("old : $old ... new : $new")
    })
}

fun main(args: Array<String>) {
    val user=User()
    println(user.name)
    user.name="kkang"
    user.name="kim"
    println(user.name)
}
```

감사합니다

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

윌리엄 셰익스피어
William Shakespeare