코틀린을 활용한 안드로이드 앱 개발

코틀린 기초문법 이해 – Property, Null Safety

한국기술교육대학교
온라인평생교육원

학습내용

1. 프로퍼티에 대한 이해
2. lateinit, lazy 사용
3. Null Safety,에 대한 이해

학습목표

1. 프로퍼티에 대해 이해할 수 있다.
2. lateinit, lazy 사용 방법을 이해할 수 있다.
3. Null Safety에 대해 이해할 수 있다.

1. 프로퍼티에 대한 이해

1 프로퍼티란?

- 코틀린에서는 Top-Level 변수, 클래스의 멤버 변수를 프로퍼티(Property)라고 함
- getter/setter(accessor)함수가 자동으로 내장된 변수

```
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}")
}
```

```
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}")
}
```

- 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

1. 프로퍼티에 대한 이해

2 사용자 정의 프로퍼티

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

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

- get(), set() 내부에서는 프로퍼티 값을 field로 접근
- var의 경우 set()과 get()을 모두 정의
  - get()이 정의되었다고 하더라도 초기 값이 명시되어야 함
- val의 경우 set()을 정의할 수 없음
  - get()을 정의하였다면 초기 값을 명시하지 않아도 됨

```
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"
}
```

2. lateinit, lazy 사용

1 lateinit

- 프로퍼티 초기화를 미루기 위해 lateinit 이 이용
- lateinit은 var의 경우에만 사용이 가능
- lateinit은 custom getter/setter를 사용하지 않은 프로퍼티에만 사용 가능
- nullable 프로퍼티에는 사용 불가
- 타입은 기초 타입이 아니어야 함

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

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

```
lateinit var data1: String

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

2. lateinit, lazy 사용

2 lazy

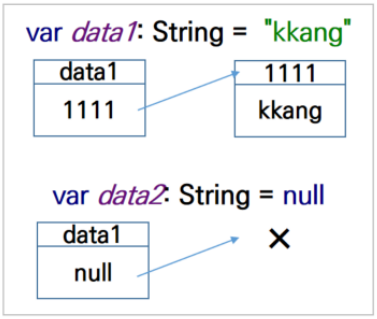
- 초기화를 프로퍼티의 이용 시점으로 미룰 수 있는 방법
- lazy는 일종의 실행 영역
- by lazy { } 로 선언되어 프로퍼티의 초기화를 lazy의 { } 에서 진행 가능
- 프로퍼티 이용 시점에 lazy { } 영역 실행
- val과 같이 사용 가능
- 기초 type에도 사용 가능

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

3. Null Safety에 대한 이해

1 Null Safety란?

- Null이란 프로그램에서 값이 아무것도 대입되지 않은 상태



- Null Safety란 Null에 다양한 처리를 도와줌으로써 Null에 의한 NPE이 발생하지 않는 프로그램을 작성할 수 있게 해준다는 개념

2 Null 허용과 Null 불허 타입으로 구분

```
var data1: String = "kkang"

var data2: String? = null

fun main(args: Array<String>) {
    data1=null//error
}
```

3. Null Safety에 대한 이해

3 Null 체크 연산자 ?.

- 객체가 null이 아니면 멤버에 접근하고 null이면 멤버 접근 없이 null을 리턴하는 연산자

```
fun main(args: Array<String>) {

    var data1: String? = "kkang"

    var length2: Int? = data1?.length
    println(length2)

    data1=null
    length2 = data1?.length
    println(length2)

}
```

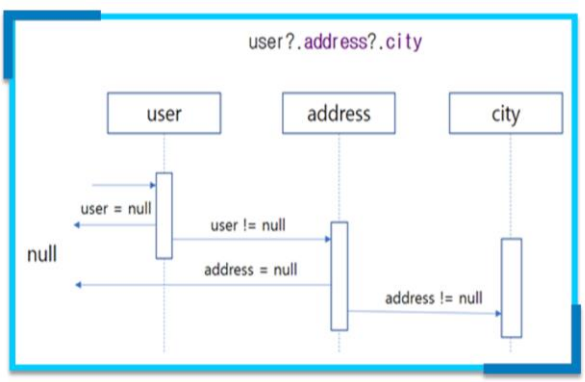
- Null 체크는 객체의 연결 구조에서도 사용이 가능

```
class Address {
    val city: String?="seoul"
}

class User {
    val address: Address? = Address()
}

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

    println(user?.address?.city)
}
```



3. Null Safety에 대한 이해

4 앨비스 연산자 ?:

- Null인 경우에 처리를 명시

```
fun main(args: Array<String>) {
    var data: String? ="kkang"

    var length: Int = if(data != null){
        data.length
    }else {
        -1
    }

    data=null

    length=data?.length ?: -1

    println(length)

    data ?: println("data is null")

}
```

5 예외 발생 연산자 !!

- Null인 경우 Exception을 발생시키기 위한 연산자

```
fun main(args: Array<String>) {
    var data: String? = "kkang"

    data!!.length

    data=null

    data!!.length
}
```

▪ 정리하기

1. 프로퍼티에 대한 이해

프로퍼티(Property)

- 코틀린에서는 Top-Level 변수, 클래스의 멤버 변수를 프로퍼티(Property)라고 부름
- getter/setter(accessor)함수가 자동으로 내장된 변수

2. lateinit, lazy 사용

lateinit

- 프로퍼티 초기화를 미루기 위해 이용
- var의 경우에만 사용이 가능
- custom getter/setter를 사용하지 않은 프로퍼티에만 사용할 수 있음

lazy

- 초기화를 프로퍼티의 이용 시점으로 미루기 위해 lazy { } 가 이용 됨
- val과 같이 사용 가능

3. Null Safety에 대한 이해

- Null 체크 연산자 → ?.
- 앨비스 연산자 → ?:
- 예외 발생 연산자 → !!