05

04-b. 확장

다양한 기법

확장의 원리

확장이란?

- 클래스 내부에 선언된 함수와 프로퍼티 이외에 다른 함수나 프로퍼티를 추가
- 일반적으로 클래스 확장은 상속 기법이용

```
open class Super {
    val superData: Int = 10
    fun superFun() {
        println("superFun....")
    }
}

class Sub: Super() {
    val subData: Int = 20
    fun subFun() {
        println("subFun....")
    }
}
```

확장의 원리

확장이란?

- 상속을 받지 않고 이미 선언된 클래스에 함수나 프로퍼티를 추가한후 이용하는 방법

```
class Super {
    val superData: Int = 10
    fun superFun() {
        println("superFun....")
    }
}

val Super.subData: Int
    get() = 20

fun Super.subFun() {
    println("subFun.....")
}

fun main(args: Array<String>) {
    val obj: Super = Super()
    println("superData : ${obj.superData}, subData : ${obj.subData}")
    obj.superFun()
    obj.subFun()
}
```

확장의 원리

정적 등록에 의한 실행

- 확장 함수는 기존 클래스내에 정적으로 추가되지는 않는다.
- 실행 시점에 확장 클래스(Receiver Type) 만 보고 실행

```
class Sub: Super() {
    var data: Int = 20
    override fun superFun() {
        println("Sub .. superFun.... ${data}")
    }
    fun some1(data: Int) {
        this.data=data
        superFun()
        super.superFun()
    }
}
//실행시에 판단하는 것임으로
fun Sub.some2(data: Int){
    this.data = data
    superFun()
    super.superFun()//error
}
fun main(args: Array<String>) {
    val obj: Sub = Sub()
    obj.some1(10)
    obj.some2(100)
}
```

확장의 원리

정적 등록에 의한 실행

- 정적 등록임으로 OOP의 다형성도 불가
- 상속을 통한 다형성을 구현

```
open class Super {
    open fun sayHello(){
        println("Super1.. sayHello()")
    }
}

class Sub : Super() {
    override fun sayHello(){
        println("Sub1.. sayHello()")
    }
}

fun some(obj: Super){
    obj.sayHello()
}
fun main(args: Array<String>) {
    some(Sub())
}
```

실행결과

```
Sub.. sayHello()
```

확장의 원리

정적 등록에 의한 실행

- 확장에 의한 다형성을 구현

```
open class Super

class Sub : Super()

fun Super.sayHello(){
    println("Super..sayHello()")
}
fun Sub.sayHello(){
    println("Sub..sayHello()")
}

fun some(obj: Super){
    obj.sayHello()
}

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

실행결과

```
Super..sayHello()
```

확장의 원리

확장 클래스에 동일 이름의 함수가 있는 경우

```
class Test {
    fun sayHello(){
        println("Test.. sayHello()")
    }
}

fun Test.sayHello(){
    println("Test extension.. sayHello()")
}


fun main(args: Array<String>) {
    val test=Test()
    test.sayHello()
}
```

실행결과

```
Test.. sayHello()
```

프로퍼티와 컴패니언 오브젝트 확장

프로퍼티 확장

- 프로퍼티 확장이 가능
- getter에 의해서만 초기화 가능

```
class Test {
    val classData: Int = 0
}

//val Test.extensionData1: Int = 0//error

val Test.extensionData2: Int
    get() = 10

fun main(args: Array<String>) {
    val obj=Test()
    println("classData ${obj.classData} ... extensionData2 : ${obj.extensionData2}")
}
```

String 을 내 마음대로

isReverseEqual()

- 문자열이 회문(앞뒤가 같은 문자열)인지 확인
- 반환: Boolean, 예: "level" → true, "hello" → false
- 힌트 : String.reversed() 로 문자열 거꾸로 변경후 원본 문자열과 비교

toTitleCase()

- 각 단어의 첫 글자를 대문자로 변환
- 반환: String, 예: "hello world" → "Hello World"
- 힌트 : 스페이스( )로 자른후(split()) joinToString(" ", { }) 이용, { } 를 먼저 실행, 문자열 변형, 각 문자열을 " "로 결합

  String.replaceFirstChar { } – 문자열의 첫글자를 { } 로직대로 변환, { }의 매개변수가 첫번째 문자(Char)

  Char.isLowerCase(), Char.titlecase() – 대문자 변환

maskEmail()

- 이메일 주소를 마스킹 처리
- @ 앞 3글자 이후를 ***로 변경, 반환: String
- 예: "gildonghong@example.com" → gil*****@example.com
- 힌트 : split("@"), String.take(3) – 문자열 앞 3글자 획득

```
fun main() {
    println("level".isReverseEqual())            // true
    println("hello".isReverseEqual())            // false
    println("hello world".toTitleCase())        // Hello World
    println("gildonghong@example.com".maskEmail()) //gil*****@example.com
}
```

감사합니다

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

윌리엄 셰익스피어
William Shakespeare