Kotlin 람다함수
참고 자료

2022.09
이양민
컴퓨터AI공학부

람다 함수 구조

함수 타입 선언

- 함수 타입 선언
  - 함수 타입이란 함수를 선언할 때 나타내는 매개변수와 반환 타입을 의미

• 일반 함수 선언

```
fun some(no1: Int, no2: Int): Int {
    return no1 + no2
}
```

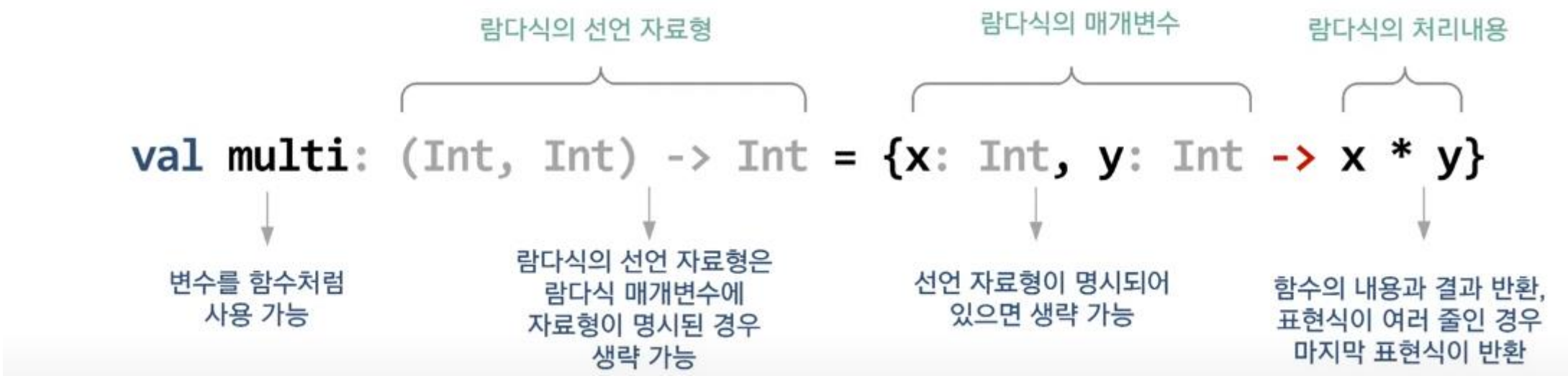
• 함수 타입을 이용해 함수를 변수에 대입

```
val some: (Int, Int) -> Int = { no1: Int, no2: Int -> no1 + no2 }
```

함수 타입 함수 내용

람다식의 구성

- 변수에 지정된 람다식



✦ 표현식이 2줄 이상일 때

```
val multi2: (Int, Int) -> Int = { x: Int, y: Int ->
    println("x * y")
    x * y // 마지막 표현식이 반환됨
}
```

✦ 자료형의 생략

```
val multi: (Int, Int) -> Int = {x: Int, y: Int -> x * y} // 생략되지 않은 전체 표현
```

✦ 반환 자료형이 없거나 매개변수가 하나 있을 때

```
val greet: ()->Unit = { println("Hello World!") }
val square: (Int)->Int = { x -> x * x }
```

- 반환 자료형이 없으면 화살표 표기법도 사라짐(반환 타입이 없는 경우 Unit으로 표기)

✦ 람다식 안에 람다식이 있는 경우?

```
val nestedLambda: ()->()->Unit = { { println("nested") } }
```

- 작성은 가능한 형태지만 자주 사용하지 않음

✦ 선언부의 자료형 생략

```
val greet = { println("Hello World!") } // 추론 가능
val square = { x: Int -> x * x } // 선언 부분을 생략하려면 x의 자료형을 명시해야 함
val nestedLambda = { { println("nested") } } // 추론 가능
```

람다 함수와 고차함수

- 매개변수가 1개인 람다 함수
  - 람다 함수의 매개변수가 1개일 때는 매개변수를 선언하지 않아도 it 키워드로 매개변수를 이용할 수 있습니다.

• 매개변수가 1개인 람다 함수

```
fun main() {
    val some = {no: Int -> println(no)}
    some(10)
}
```

▶ 실행 결과

```
10
```

• 매개변수가 1개인 람다 함수에 it 키워드 사용

```
fun main() {
    val some: (Int) -> Unit = {println(it)}
    some(10)
}
```

▶ 실행 결과

```
10
```

람다 함수와 고차함수

- 람다 함수의 반환
  - 람다 함수에서는 return 문을 사용할 수 없습니다.
  - 람다 함수의 반환값은 본문에서 마지막 줄의 실행 결과입니다.

• 람다 함수에서 return 문 사용 오류

```
val some = {no1: Int, no2: Int -> return no1 * no2}     // 오류!
```

• 람다 함수의 반환문

```
fun main() {
    val some = {no1: Int, no2: Int ->
        println("in lambda function")
        no1 * no2
    }
    println("result : ${some(10, 20)}")
}
```

▶ 실행 결과

```
in lambda function
result : 200
```

람다 함수와 고차함수

- 고차 함수
  - 고차 함수란 함수를 매개변수로 전달받거나 반환하는 함수를 의미
  - 일반적으로는 람다식을 매개변수로 받거나 결과로 리턴하는 함수를 의미함
  - 코드 조각을 람다로 만들면 코드 중복 제거(고차 함수 목적)

```
fun calculator(a: Int, b: Int, operation: (Int, Int)->Int) = operation(a, b)

fun main(args: Array<String>?) {
    val sum = {x: Int, y: Int -> x+y}
    println( calculator(1, 2, sum) )
    println( calculator(4, 3, {a: Int, b: Int -> a-b}) )
}
```

람다 함수와 고차함수

- 고차 함수: 매개변수 및 리턴 모두 사용하는 고차함수 예제

• 고차 함수

```
fun hofFun(arg: (Int) -> Boolean): () -> String {
    val result = if(arg(10)) {
        "valid"
    } else {
        "invalid"
    }
    return {"hofFun result : $result"}
}
fun main() {
    val result = hofFun({no -> no > 0})
    println(result())
}
```

▶ 실행 결과

```
hofFun result : valid
```

매개변수에 람다식이 있는 경우:고차함수 호출 방법 예제

```
package chap03.section3

fun main() {

    var result: Int

    // 람다식을 매개변수와 인자로 사용한 함수
    result = highOrder({ x, y -> x + y}, 10, 20)
    println(result)
}

fun highOrder(sum: (Int, Int) -> Int, a: Int, b: Int): Int {
    return sum(a, b)
}
```

Call by Value

```
package chap03.section3

fun main() {
    val result = callByValue(lambda()) // 람다식 함수를 호출
    println(result)
}

fun callByValue(b: Boolean): Boolean { // 일반 변수 자료형으로 선언된 매개변수
    println("callByValue function")
    return b
}

val lambda: () -> Boolean = {  // 람다 표현식이 두 줄이다
    println("lambda function")
    true                // 마지막 표현식 문장의 결과가 반환
}
```

Call by Name

```
package chap03.section3

fun main() {
    val result = callByName(otherLambda) // 람다식 이름으로 호출
    println(result)
}

fun callByName(b: () -> Boolean): Boolean { // 람다식 함수 자료형으로 선언된 매개변수
    println("callByName function")
    return b()
}

val otherLambda: () -> Boolean = {
    println("otherLambda function")
    true
}
```

다른 함수의 참조에 의한 호출

```
fun sum(x: Int, y: Int) = x + y
```

```
funcParam(3, 2, sum) // 오류! sum은 람다식이 아님
...
fun funcParam(a: Int, b: Int, c: (Int, Int) -> Int): Int {
    return c(a, b)
}
```

```
funcParam(3, 2, ::sum)
```

안드로이드 기반: 매개변수 없음

```kotlin
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    Log.d(TAG,  msg: "MainActivity - onCreate() called")

    someFunction()
    someLambda.invoke()
}

// 매개변수 X, 반환 X
fun someFunction () {
    Log.d(TAG,  msg: "someFunction: () called")
}

// 매개변수 X, 반환 X
val someLambda : () -> Unit = {
    Log.d(TAG,  msg: "someLambda() called")
}
```

안드로이드 기반: 매개변수 있음

- 선언 및 구현

```
fun someFunctionWithParam (title: String) {
    Log.d(TAG,  msg: "someFunctionWithParam: () called $title")
}

// 매개변수 X, 반환 X
val someLambda : () -> Unit = {
    Log.d(TAG,  msg: "someLambda() called")
}

//    val someLambdaWithParam : (String) -> Unit = {
//        Log.d(TAG, "someLambda() called $it")
//    }

val someLambdaWithParam : (String) -> Unit = { userInput ->
    Log.d(TAG,  msg: "someLambda() called $userInput")
}
```

- 호출

```
someFunctionWithParam( title: "하하하")
someLambdaWithParam("호호호")
}
```

교재 해석

- dp.init()이 호출되면 안의 인자로 기본 작업 진행, 특별한 숫자가 할당 안되었으므로 현재날짜로 설정됨
- { } 내부 해석
  - View, year, monthOfYear, dayOfMonth가 인자로 들어옴(dp에 대한 이벤트 처리 시 자동으로 들어옴)
  - 10~16까지 작업을 진행하지만 람다는 별도 리턴 지정 없어도 됨
  - 두번째 링크의 30분~33분 정도 확인)

예제 8-5 Kotlin 코드 2

```
1           ~~~ 생략([예제 8-4]와 동일) ~~~
2           btnWrite = findViewById<Button>(R.id.btnWrite)
3
4           var cal = Calendar.getInstance()
5           var cYear = cal.get(Calendar.YEAR)
6           var cMonth = cal.get(Calendar.MONTH)
7           var cDay = cal.get(Calendar.DAY_OF_MONTH)
8
9           dp.init(cYear, cMonth, cDay) { view, year, monthOfYear, dayOfMonth ->
10              fileName = (Integer.toString(year) + "_"
11                      + Integer.toString(monthOfYear + 1) + "_"
12                      + Integer.toString(dayOfMonth) + ".txt")
13              var str = readDiary(fileName)
14              edtDiary.setText(str)
15              btnWrite.isEnabled = true
16          }
17      }
18
19      fun readDiary(fName: String) : String? {
20          return null
21      }
22  }
```

참고 링크

- https://www.youtube.com/watch?v=z8prdZgk4kA&ab_channel=Acaroom

- https://www.youtube.com/watch?v=wD7b7-VYBoU&ab_channel=%EA%B0%9C%EB%B0%9C%ED%95%98%EB%8A%94%EC%A0%95%EB%8C%80%EB%A6%AC