Chapter 03.코틀린 고급 - 09.스코프 함수

스코프 함수

1. 스코프 함수란

- 코틀린의 표준 라이브러리에는 객체의 컨텍스트 내에서 코드 블록을 실행하기 위해서만 존재하는 몇가지 함수가 포함되어 있는데 이를 `스코프 함수` 라고 부른다
- 스코프 함수를 제대로 사용하면 불필요한 변수 선언이 없어지며 코드를 더 간결하고 읽기 쉽게 만든다
- 스코프 함수의 코드 블록 내부에서는 `변수명을 사용하지 않고도` 객체에 접근할 수 있는데 그 이유는 수신자 객체에 접근할 수 있기 때문이다
- 수신자 객체는 람다식 내부에서 사용할 수 있는 객체의 참조이다
- 스코프 함수를 사용하면 수신자 객체에 대한 참조로 `this` 또는 `it` 를 사용한다

---

2. 스코프 함수 선택 방법

- 코틀린은 총 5개의 유용한 스코프 함수를 제공하며 각 스코프 함수들은 본질적으로 유사한 기능을 제공하는데 각각 어떻게 다른지 살펴보자
- 스코프 함수 선택표

함수명	수신자 객체 참조 방법	반환 값	확장 함수 여부
let	it	함수의 결과	O
run	this	함수의 결과	O
with	this	함수의 결과	X
apply	this	컨텍스트 객체	O
also	it	컨텍스트 객체	O

2.1. let

- null이 아닌 경우 사용될 로직을 작성하고 새로운 결과를 반환하고 싶을때 사용

```
fun main() {

    val str: String? = null

    str?.let {
        println(it)
    }
    // 아무것도 출력되지 않음
}
```

- null이 아닌 경우 실행

```
fun main() {

    val str: String? = "안녕"

    str?.let {
```

```
        println(it)
    }
    // 안녕
}
```

- 함수의 결과를 반환 (let 함수 내부의 마지막 코드가 결과로 반환)

```
fun main() {

    val str: String? = "안녕"

    val result = str?.let {
        println(it)

        1234 // let 함수 마지막 코드가 결과로 반환
    }
    println(result)
    // 1234
}
```

- let을 쓸때 호출이 중첩되면 코드가 복잡해지므로 이런 경우엔 if를 사용하는 편이 낫다

```
val str: String? = "안녕"
val result = str?.let {
    println(it)

    val abc : String? = "abc"

    abc?.let {

        val def : String? = "def"
        def?.let {
            println("abcdef가 null이 아님")
        }
    }
    1234
}
println(result)
```

- if 식으로 변경

```
val str: String? = "안녕"
val result = str?.let {
    println(it)

    val abc : String? = "abc"
    val def : String? = "def"
    if (!abc.isNullOrEmpty() && !def.isNullOrEmpty()) {
        println("abcdef가 null이 아님")
    }

    1234
}
println(result)
```

2.2. run

- 수신 객체의 프로퍼티를 구성하거나 새로운 결과를 반환하고 싶을때

```
class DatabaseClient {
    var url: String? = null
    var username: String? = null
    var password: String? = null

    // DB에 접속하고 Boolean 결과를 반환
    fun connect(): Boolean {
        println("DB 접속 중 ...")
        Thread.sleep(1000)
        println("DB 접속 완료")
        return true
    }
}

fun main() {

    val config = DatabaseClient()
    config.url = "localhost:3306"
```

```
    config.username = "mysql"
    config.password = "1234"
    val connected = config.connect()

    println(connected)
}
// DB 접속 중 ...
// DB 접속 완료
// true
```

- run을 사용

```
class DatabaseClient {
    var url: String? = null
    var username: String? = null
    var password: String? = null

    // DB에 접속하고 Boolean 결과를 반환
    fun connect(): Boolean {
        println("DB 접속 중 ...")
        Thread.sleep(1000)
        println("DB 접속 완료")
        return true
    }
}

fun main() {
    val connected = DatabaseClient().run {
        url = "localhost:3306"
        username = "mysql"
        password = "1234"
        connect()
    }
    println(connected)
}
```

- let을 사용할 순 있으나 it을 사용해야하기 때문에 불편

```
val connected = DatabaseClient().let {
        it.url = "localhost:3306"
        it.username = "mysql"
        it.password = "1234"
        it.connect()
```

```
    }
    println(connected)
```

2.3. with

- 결과 반환없이 내부에서 수신 객체를 이용해 다른 함수를 호출하고 싶을때 사용

```
val str  = "안녕하세요"
with (str) {
    println("length = $length")
}
```

- let이나 run과 같이 함수의 결과가 반환된다

```
val str = "안녕하세요"
val length = with(str) {
    length
}
println(length)
// 5
```

- 다른 스코프 함수와 다른 점은 with는 확장 함수가 아니다

- run으로 작성한 코드를 with로 변경

```
val connected = with(DatabaseClient()) {
    url = "localhost:3306"
    username = "mysql"
    password = "1234"
    connect()
```

```
  }
println(connected)
```

2.4. apply

- 수신 객체의 프로퍼티를 구성하고 수신 객체를 그대로 결과로 반환하고 싶을때 사용

```
val client: DatabaseClient = DatabaseClient().apply {
    url = "localhost:3306"
    username = "mysql"
    password = "1234"
    connect()
}
```

- 앞서 공부한 let, run, with는 함수의 결과가 반환타입으로 변환되는데 반해서 apply는 수신 객체 그대로 반환된다

2.5. also

- 부수 작업을 수행하고 전달받은 수신 객체를 그대로 결과로 반환하고 싶을때 사용

- also를 쓰지 않은 경우

```
class User(val name: String, val password: String) {

    fun validate() {
        if (name.isNotEmpty() && password.isNotEmpty()) {
            println("검증 성공!")
```

```
        } else {
            println("검증 실패!")
        }
    }

}

fun main() {
  val user: User = User(name = "tony", password = "1234")
  user.validate()
}
```

- also를 쓴 경우

```
fun main() {
  User(name = "tony", password = "1234").also {
    it.validate()
  }
}
```

---

3. 스코프 함수 사용시 유의할 점

- 이와 같은 스코프 함수는 모두 기능이 유사하기 때문에 실무에선 섞어쓰는 경우도 많다
- this는 키워드이다 키워드는 사전에 정의된 예약어이기 때문에 다른 의미로 사용할 수 없지만 it은 특정 용도에서만 작동하는 소프트 키워드이기 때문에 다른 용도로 사용할 수 있다

```
val this: String? = null // 컴파일 오류
val it: String? = null // 작동
```

- 중첩으로 사용하는 경우 this, it에 대해 혼동하기 쉽다

- 또한 중첩 함수내에서 외부 함수에 대한 접근을 하려면 it은 자기 자신의 참조기 때문에 불가능하다

```
fun main() {

    val hello = "hello"
    val hi = "hi"

    hello.let {
        println(it.length)

        hi.let {
            println(it.length)
        }

    }
}
```

- it 대신 변수를 선언한다

```
hello.let { a ->
        println(a.length)

        hi.let { b ->
            println(a.length)
            println(b.length)
        }

    }
```