세상의 속도를
따라잡고 싶다면

Do it!

깡샘의 안드로이드 앱 프로그래밍 with 코틀린

이지스퍼블리싱(주)

05

코틀린의 유용한 기법

05-1 람다 함수와 고차함수

05-2 널 안전성

이지스퍼블리싱(주)

05-1 람다 함수와 고차함수

람다함수

- 람다 함수는 익명 함수 정의 기법
- 람다 함수 선언과 호출
  - 람다 함수는 fun 키워드를 이용하지 않으며 함수 이름이 없습니다.
  - 람다 함수는 { }로 표현합니다.
  - { } 안에 화살표(->)가 있으며 화살표 왼쪽은 매개변수, 오른쪽은 함수 본문입니다.
  - 함수의 반환값은 함수 본문의 마지막 표현식입니다.

• 함수 선언 형식

```
fun 함수명(매개변수) { 함수 본문 }
```

• 람다 함수 선언 형식

```
{ 매개변수 -> 함수 본문 }
```

• 일반 함수 선언

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

• 람다 함수 선언

```
val sum = {no1: Int, no2: Int -> no1 + no2}
```

05-1 람다 함수와 고차함수

- 매개변수 없는 람다 함수
  - 화살표 왼쪽이 매개변수를 정의하는 부분인데 매개변수가 없을 경우 비워 두거나 화살표까지 생략 가능

• 매개변수가 없는 람다 함수

```
{-> println("function call")}
```

• 화살표를 생략한 람다 함수

```
{println("function call")}
```

05-1 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
```

05-1 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
```

05-1 람다 함수와 고차함수

함수 타입과 고차 함수

- 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

05-1 람다 함수와 고차함수

- 타입 별칭 — typealias
  - typealias는 타입의 별칭을 선언하는 키워드

• 타입 별칭 선언과 사용

```
typealias MyInt = Int
fun main() {
    val data1: Int = 10
    val data2: MyInt = 10
}
```

• 함수 타입 별칭

```
typealias MyFunType = (Int, Int) -> Boolean

fun main() {
    val someFun: MyFunType = {no1: Int, no2: Int ->
        no1 > no2
    }
    println(someFun(10, 20))
    println(someFun(20, 10))
}
```

▶ 실행 결과

```
false
true
```

05-1 람다 함수와 고차함수

- 매개변수 타입 생략
  - 매개변수의 타입을 유추할 수 있다면 타입 선언을 생략할 수 있습니다.

• 매개변수 타입을 생략한 함수 선언

```
typealias MyFunType = (Int, Int) -> Boolean
val someFun: MyFunType = {no1, no2 ->
    no1 > no2
}
```

• 매개변수 타입 선언 생략 예

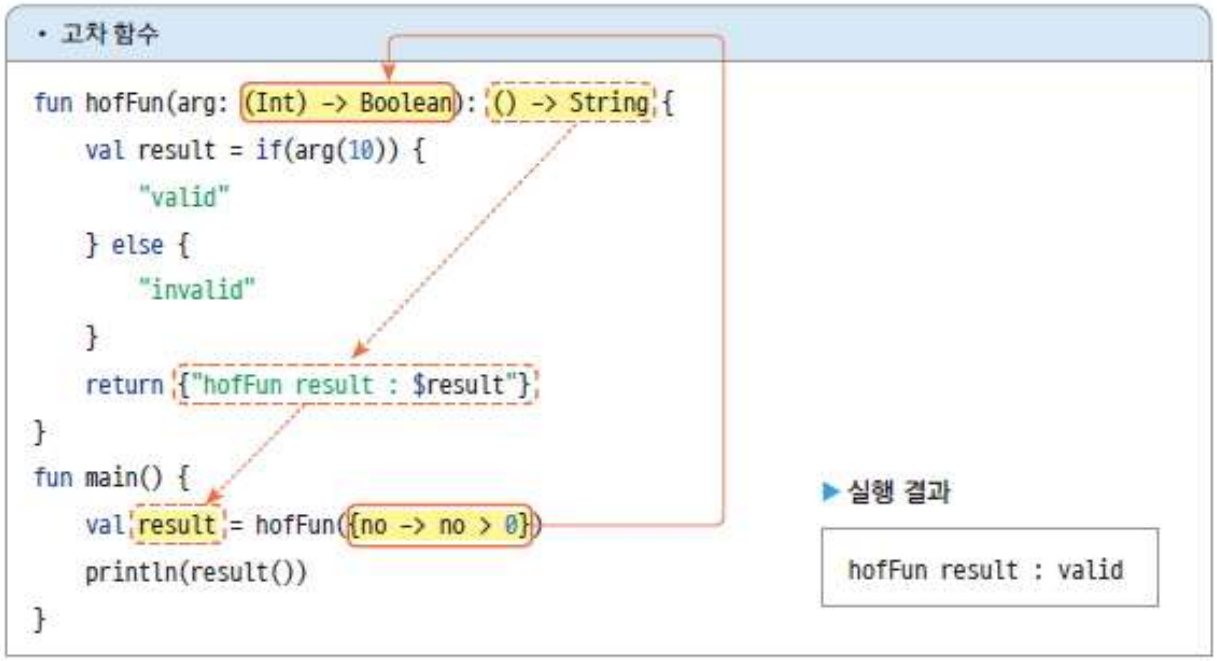
```
val someFun: (Int, Int) -> Boolean = {no1, no2 ->
    no1 > no2
}
```

• 변수 선언 시 타입 생략

```
val someFun = {no1: Int, no2: Int ->
    no1 > no2
}
```

05-1 람다 함수와 고차함수

- 고차 함수
  - 고차 함수란 함수를 매개변수로 전달받거나 반환하는 함수를 의미



05-2 널 안전성

널 안전성이란?

- 널(null)이란 객체가 선언되었지만 초기화되지 않은 상태를 의미
- 널인 상태의 객체를 이용하면 널 포인트 예외(NullPointException)가 발생
- 널 안정성이란 널 포인트 예외가 발생하지 않도록 코드를 작성하는 것

• 널 안전성을 개발자가 고려한 코드

```
fun main() {
    var data: String? = null
    val length = if (data == null) {
        0
    } else {
        data.length
    }
    println("data length : $length")

}
```

▶ 실행 결과

```
data length : 0
```

05-2 널 안전성

- 프로그래밍 언어가 널 안전성을 지원한다는 것은 객체가 널인 상황에서 널 포인터 예외가 발생하지 않도록 연산자를 비롯해 여러 기법을 제공한다는 의미

• 코틀린이 제공하는 널 안전성 연산자를 이용한 코드

```
fun main() {
    var data: String? = null
    println("data length : ${data?.length ?: 0}")
}
```

▶ 실행 결과

```
data length : 0
```

05-2 널 안전성

널 안전성 연산자

- 널 허용 — ? 연산자
  - 코틀린에서는 변수 타입을 널 허용과 널 불허로 구분

• 널 허용과 널 불허

```
var data1: String = "kkang"
data1 = null    // 오류!

var data2: String? = "kkang"
data2 = null    // 성공!
```

05-2 널 안전성

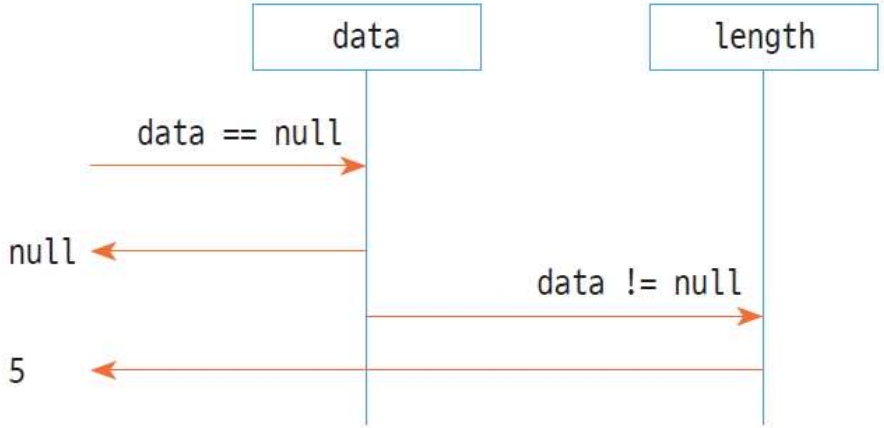
- 널 안전성 호출 — ?. 연산자
  - 널 허용으로 선언한 변수의 멤버에 접근할 때는 반드시 ?. 연산자를 이용해야 합니다.

• 널 포인트 예외 오류

```
var data: String? = "kkang"
var length = data.length    // 오류!
```

• 널 안전성 호출 연산자 사용

```
var data: String? = "kkang"
var length = data?.length    // 성공!
```



05-2 널 안전성

- 엘비스 — ?: 연산자
  - 널일 때 대입해야 하는 값이나 실행해야 하는 구문이 있는 경우 이용

• 엘비스 연산자 사용

```
fun main() {
    var data: String? = "kkang"
    println("data = $data : ${data?.length ?: -1}")
    data = null
    println("data = $data : ${data?.length ?: -1}")
}
```

▶ 실행 결과

```
data = kkang : 5
data = null : -1
```

05-2 널 안전성

- 예외 발생 — !! 연산자
  - 객체가 널일 때 예외를 일으키는 연산자

• 예외 발생 연산자

```
fun some(data: String?): Int {
    return data!!.length
}
fun main() {
    println(some("kkang"))
    println(some(null))
}
```

▶ 실행 결과

```
5
Exception in thread "main" java.lang.NullPointerException
```

감사합니다

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

윌리엄 셰익스피어
William Shakespeare