

# 코틀린 고차함수 & 예외처리

App:ple 코틀린 5차시 | 강의자 최재혁



## 01. 고차함수

## 02. 예외처리

# 코틀린은 함수형 프로그래밍을 지원함

**명령형 프로그래밍**  
어떻게 코드를 짤 것인지를 설명

객체지향 프로그래밍 (Kotlin, Java) : 객체의 집합  
절차지향 프로그래밍 (C, C++) : 순차적인 처리과정

**선언형 프로그래밍**  
무엇을 할 것인지를 설명

함수형 프로그래밍  
순수 함수를 조합하고 소프트웨어를 만드는 방식



작은 문제를 해결하기 위한 함수를 작성해  
**가독성을 높이고 유지보수를 용이하게 함**

함수형 프로그래밍(functional programming)은  
자료 처리를 **수학적 함수의 계산으로 취급**하고  
**가변 데이터를 멀리하는 프로그래밍 패러다임** 중 하나임

## 01. 고차함수

## 02. 예외처리

# (참고) 순수함수

함수 외부의 어떤 상태도 바꾸지 않는다는 의미의 함수로,  
같은 인자에 대해 항상 같은 값을 반환하며 어떤 외부의 상태도 바꾸지 않음



```
fun sum(a : Int, b: Int) : Int = a + b
```

## 01. 고차함수

## 02. 예외처리

# 함수형 프로그래밍에서의 일급 객체

일정 기준을 충족하면 일급 객체(first-class)라고 할 수 있음

일급 객체는 함수의 인자(매개변수)로 전달할 수 있음

일급 객체는 함수의 반환값(return)이 될 수 있음

일급 객체는 변수에 담을 수 있음



Kotlin에서의 함수는  
변수나 데이터 저장 가능,  
다른 고차함수의 매개변수나 반환형으로 사용 가능,  
고차함수를 사용해 새로운 내장 함수를 만들 수 있음

**Kotlin에서의 함수는 1급이라고 할 수 있음**

## 01. 고차함수

## 02. 예외처리

# 고차함수

다른 함수를 인자로 받거나 함수를 반환하는 함수로  
코틀린에서는 람다나 함수 참조를 사용하여 함수를 값으로 표현 가능함

일반 함수를 인자나 반환값으로 사용하는 고차함수

```
private fun sum(a: Int, b: Int): Int = a + b

private fun sumFunction(): Int = sum(40, 2)

fun main() {
    println("sumFunction result = ${sumFunction()}")
}
```

## 01. 고차함수

# 람다

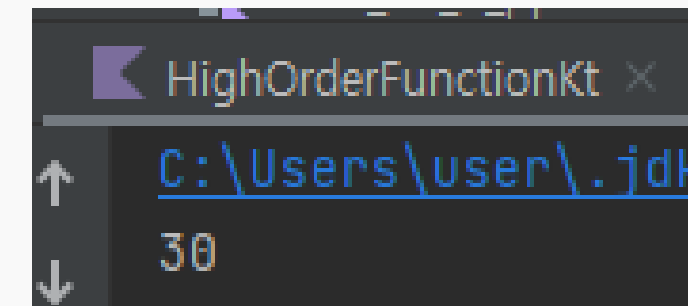
## 02. 예외처리

람다식은 기본적으로 다른 함수에 넘길 수 있는 작은 코드 조각이며, 코틀린 표준 라이브러리는 람다를 많이 사용한다

람다함수는 fun과 함수이름을 명시하지 않고, 축약형으로 선언함  
{ 매개변수 -> 함수내용 }

```
val sum1 = { x1: Int, x2: Int -> x1+x2 }

fun main() {
    val result1 = sum1(10, 20)
    println(result1)
}
```



HighOrderFunctionKt x

C:\Users\user\.jdk

30

변수에 람다식을 적어서 추후에 활용한 경우

## 01. 고차함수

# 람다

## 02. 예외처리

람다식은 기본적으로 다른 함수에 넘길 수 있는 작은 코드 조각이며, 코틀린 표준 라이브러리는 람다를 많이 사용한다

람다함수는 fun과 함수이름을 명시하지 않고, 축약형으로 선언함  
{ 매개변수 -> 함수내용 }

```
fun main() {  
    { println("hello") }()  
    run { println("world") }  
}
```

```
HighOrderFunctionKt  
C:\Users\user\  
hello  
world
```

람다함수를 정의하자마자 함수 호출로 실행한 경우  
( 익명함수 이기에, 추후 다시 사용할 수 없음 )

## 01. 고차함수

## 02. 예외처리

# 고차함수

다른 함수를 인자로 받거나 함수를 반환하는 함수로  
코틀린에서는 람다나 함수 참조를 사용하여 함수를 값으로 표현 가능함

람다식을 인자나 반환값으로 사용하는 고차함수

```
fun main() {  
    val multiply = { x: Int, y: Int -> x * y }  
  
    println("곱셈 결과 = ${multiply(8, 8)}")  
}
```



## 01. 고차함수

# 고차함수

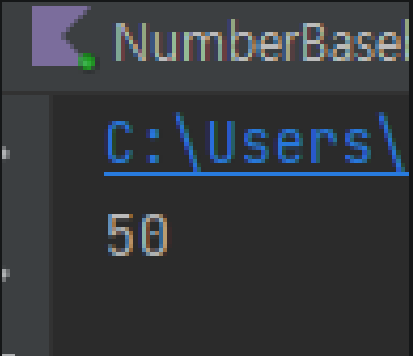
## 02. 예외처리

다른 함수를 인자로 받거나 함수를 반환하는 함수로  
코틀린에서는 람다나 함수 참조를 사용하여 함수를 값으로 표현 가능함

다른 함수를 인자로 사용하거나, 결과값으로 반환하는 함수

```
private fun highOrderFunction(sum: (Int, Int) -> Int, a: Int, b: Int): Int = sum(a, b)

fun main() {
    println(
        highOrderFunction(
            { x, y -> x + y },
            20,
            30
        )
    )
}
```



## 함수형 프로그래밍의 정의와 특징

- 순수함수를 사용해야 한다
- 람다식을 사용할 수 있다
- 고차함수를 사용할 수 있다

## 01. 고차함수

## 02. 예외처리

# try - catch문

```
fun main(){  
    val str : String = "asdfsfd"   
    val num : Int = str.toInt()  
  
    print(num)  
}
```

String을 Int로 변환하여 리턴 하는데,  
.toInt() 는 숫자가 아닌 문자열을 변환할 때 예외가 발생함

```
C:\Users\user\.jdk\openjdk-17.0.2\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Communit  
Exception in thread "main" java.lang.NumberFormatException: Create breakpoint : For input string: "hello, world!"  
    at java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:67)  
    at java.base/java.lang.Integer.parseInt(Integer.java:668)  
    at java.base/java.lang.Integer.parseInt(Integer.java:786)  
    at HighOrderFunctionKt.main(highOrderFunction.kt:16)  
    at HighOrderFunctionKt.main(highOrderFunction.kt)
```

## try - catch문

```
try{  
    // 예외가 발생할 수 있는  
    // 코드를 작성  
} catch (e : Exception) {  
    // 예외를 받아 수행할 일  
}
```

```
fun main() {  
    val str = "hello, world!"  
  
    try {  
        val num = str.toInt()  
        println("숫자 : $num")  
    } catch (e: Exception) {  
        println("숫자가 아닙니다 : $str")  
    }  
}
```

Catch e 부분에서 특정한 예외를 처리해줄 수도 있음  
NullPointerException, NumberFormatException 등등

## 예외 처리 & 고차함수 과제 안내

- ★  $+$ (더하기),  $-$ (빼기),  $*$ (곱하기),  $/$ (나누기)를 입력하여 연산의 종류를 입력 받음
- ★ 2개의 수를 입력 받음 (따로 입력 받아도 상관 X)
- ★ 특정한 예외가 발생할 수 있는데, 예외를 처리 할 수 있도록 한다 ( 모르겠다면 전체 예외로 받아도 상관 X )
- ★ 변수에 람다식을 담아 계산을 수행한 뒤, 출력하도록 한다 ( 입/출력은 어떻게 하든지 상관 없음 )

과제 제출 기한 : 05/20(금) 자정 전까지

과제 제출 폼 : <https://forms.gle/8H1BiYBdxyU7fjV87>