

Android 기초 프로그래밍 – Kotlin 심화 문법

메소드, 예외처리, 람다함수, 고차함수, 널 안전성



부산대학교 정보·의생명공학대학
정보컴퓨터공학부



이론

강의 목표와 구성

❖ Kotlin 심화 프로그래밍

- 예외처리
- 람다함수
- 고차함수
- 확장함수
- 오버로딩
- 오버라이딩
- 널안전성

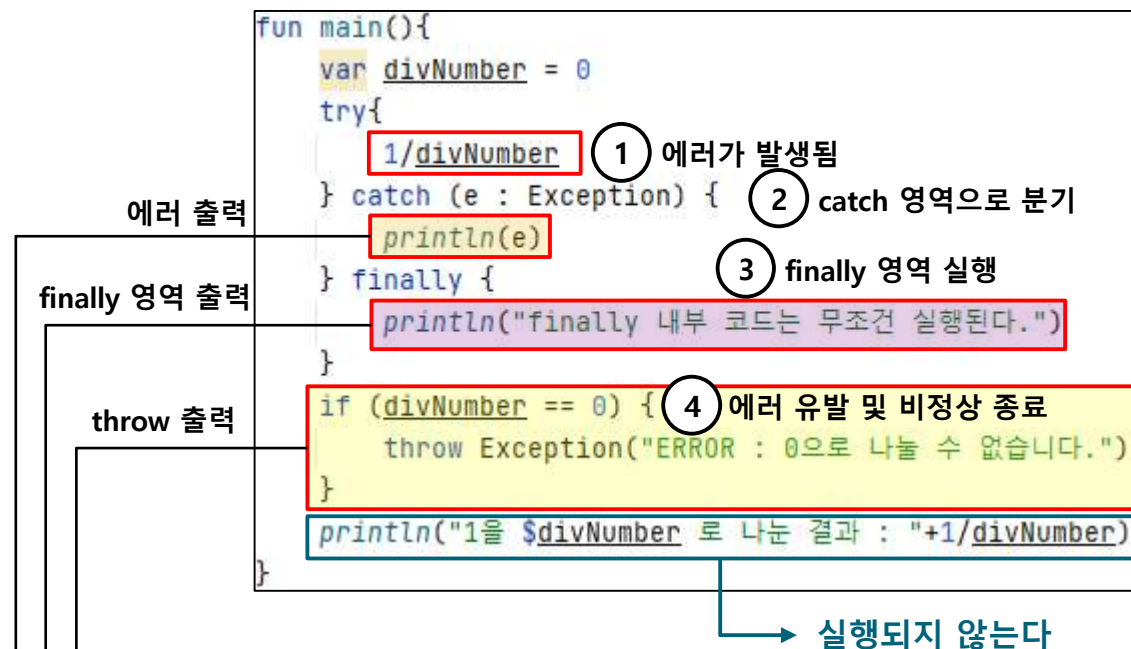
예외처리

❖ 예외처리란?

- 코드 실행 시 발생하는 에러를 수습 및 방지하기 위한 처리
 - try-catch, throw 두 가지 방법으로 예외처리가 가능하다.
- **try-catch** 예외처리
 - **try** : 코드 실행을 시도하며, 해당 구역 에러 발생을 감지
 - **catch** : try 구역의 에러 발생시 분기되어 해당 코드가 수행됨
 - **finally** : try-catch와 상관없이 무조건 수행되는 코드
- **throw** 예외처리
 - 조건에 맞을 경우, 정의된 에러를 발생시킴

❖ try-catch 와 throw 예외처리의 차이점

- **try-catch**는 에러에 취약한 영역에서 발생하는 에러를 감지하여 프로그램을 **정상 종료** 시키는 것이 목적이다.
- **throw**는 코드로 **에러를 유발**시키는 것이 목적이다.
 - 이후 조치를 하지 않으면 프로그램이 **비정상 종료**된다.



```
"C:\Program Files\Android\Android Studio\jre\bin\java.exe" ...  
java.lang.ArithmeticException: / by zero  
finally 내부 코드는 무조건 실행된다.  
Exception in thread "main" java.lang.Exception: Create breakpoint : ERROR : 0으로 나눌 수 없습니다.  
    at com.example.myapplication.UserKt.main(User.kt:31)  
    at com.example.myapplication.UserKt.main(User.kt)  
  
Process finished with exit code 1
```

람다함수

❖ 람다함수 이란?

- 람다함수는 **익명함수 기법**으로 **함수를 간단하게 정의할 때 사용**
- 람다함수 정의

val 함수이름 : (매개변수 타입) -> 리턴타입 = { 매개변수 -> 함수본문 }

'=' 이후 매개변수를 명시했다면, 생략가능

※ 매개변수가 없으면 생략 후 코드만 적는 것 가능

마지막 줄의 실행결과가 **return**값!

❖ 람다함수의 특징

- 람다함수는 함수형 프로그래밍 개념으로 **함수를 파라미터로 전달**할 수 있다.
- 재귀반복 문제의 성능 개선에 효과적**이다.

```
fun main(){  
    fun normal_function(a:Int, b:Int):Int{  
        println("----일반함수----")  
        print("$a 와 $b 의 곱 : ")  
        return a*b  
    }  
    println(normal_function( a: 10, b: 5))  
    val lamda_function:(Int,Int)->Int={  
        a:Int,b:Int->  
        println("----람다함수----")  
        print("$a 와 $b 의 곱 : ")  
        a*b ^lambda  
    }  
    println(lamda_function(10,5))  
}
```

일반 함수

람다 함수

파라미터 가이드라인 제공

파라미터 가이드라인 미제공

```
"C:\Program Files\Android\Android  
----일반함수----  
10 와 5 의 곱 : 50  
----람다함수----  
10 와 5 의 곱 : 50  
Process finished with exit code 0
```

일반 함수

람다 함수

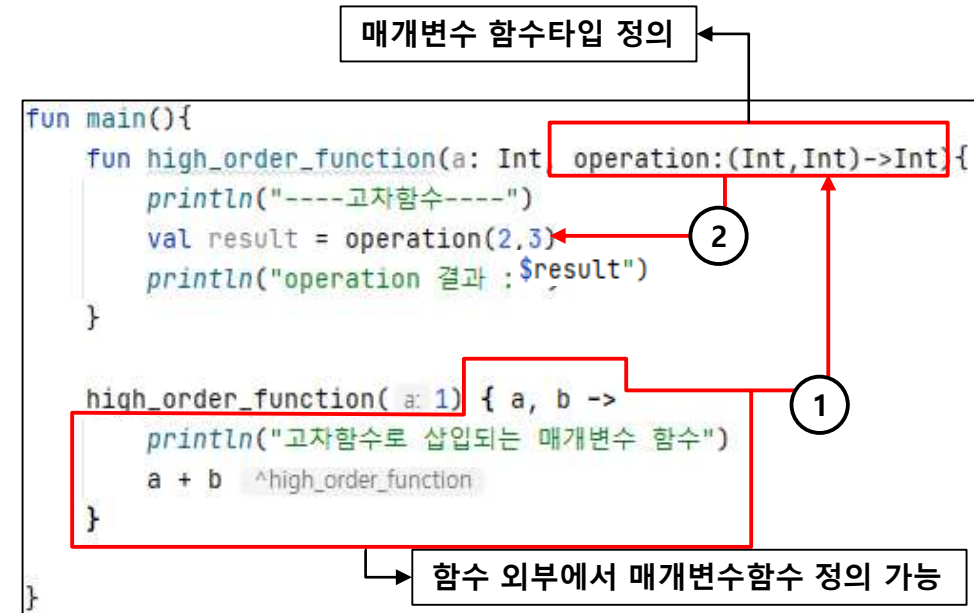
고차함수

❖ 고차함수란?

- 다른 함수를 매개변수로 받거나 함수를 반환하는 함수
 - Kotlin에서는 람다나 함수 참조를 사용해 함수를 값으로 표현할 수 있다.

❖ (Type 1) 매개변수로 함수를 받는 고차함수

- 우측의 high_order_function는 함수를 인자로 받는 고차함수이며, 매개변수 함수 operation은 외부에서 정의한다.



```
"C:\Program Files\Android\Android
----고차함수----
고차함수로 삽입되는 매개변수 함수
operation 결과 : 5

Process finished with exit code 0
```

고차함수

❖ (Type 2) 함수를 반환하는 고차함수

- 고차함수는 함수를 리턴 값으로 반환하는 것이 가능하다.

```
enum class Delivery {STANDARD, EXPEDITED}
class Order(val itemCount: Int)

fun getShippingCostCaculator(
    delivery: Delivery
): (Order) -> Double {
    if (delivery == Delivery.EXPEDITED) {
        return {order -> 6+2.1*order.itemCount}
    }
    return {order->61.2*order.itemCount}
}

fun main() {
    val calculator = getShippingCostCaculator(Delivery.EXPEDITED)

    println("배달 비용은 : ${calculator(Order( itemCount: 3))}")
}
```

Delivery 나열 클래스 타입을 가지는 매개변수

리턴 타입으로 람다함수를 가진다

앞서 설정한 람다함수 리턴 타입으로 반환
즉, 매개변수 -> Double 타입으로 반환

var calculator = { order: Order -> 6 + 2.1 * order.itemCount }

```
"C:\Program Files\Android\Android
Shipping costs 12.3
```

```
Process finished with exit code 0
```

확장함수와 오버로딩

❖ 확장함수 란?

- 클래스 밖에서도 함수(메소드)를 추가할 수 있음
- 어떤 클래스가 존재하는데 이 **클래스를 직접 수정할 수 없을 때**, 기존 클래스는 그대로 두고 클래스 주변에서 새로운 함수나 프로퍼티를 추가하여 클래스의 크기를 확장하는 함수
- 오른쪽의 예제에서 볼 수 있듯, Adder클래스 외부에서도 **fun Adder.add** 함수를 추가함

❖ 오버로딩(Overloading)

- **동일한 이름**을 갖는 함수를 **매개변수만 다르게** 하여, 여러 개의 함수를 정의하는 것
 - 동일한 기능을 하는 함수들에 대해서 파라미터에 따라서, 다양한 이름으로 정의할 필요가 없어짐

오버로딩

확장함수

확장함수
호출

확장함수
호출 결과

```
class Adder() {  
    fun add(a: Int, b: Int): Int {  
        return a + b  
    }  
    fun add(a: Int, b: Int, c: Int): Int {  
        return a + b + c  
    }  
}  
  
fun Adder.add(a: Int, b: Int, c: Int, d: Int): Int {  
    return a + b + c + d  
}  
  
fun main() {  
    var adder = Adder()  
  
    println(adder.add(a: 1, b: 2))  
    println(adder.add(a: 1, b: 2, c: 3))  
    println(adder.add(a: 1, b: 2, c: 3, d: 4))  
}
```

```
"C:\Program Files\Android\Android  
3  
6  
10  
Process finished with exit code 0
```


오버라이딩

❖ 오버라이딩(Overriding)이란?

- 부모 클래스의 메서드를 **자식 클래스에서 재정의해서 사용**
- (**override fun 메소드**)를 사용하여 함수를 재정의
 - 부모클래스의 drink 함수를 오버라이딩하여 재정의한다.

```
open class Beverage(){
    open fun drink(money: Int): Int{
        println("슈퍼클래스의 drink함수")
        return money
    }
}
class Cola() : Beverage(){
    var price = 1500
    override fun drink(money: Int): Int{
        println("콜라를 마십니다.")
        return money - this.price
    }
}
fun main() {
    var sup = Beverage()
    var sub1 = Cola()
    println("money : ${sup.drink( money: 2000)}")
    println("money : ${sub1.drink( money: 2000)}")
}
```

오버라이딩

자식 클래스
객체 생성

오버라이딩
함수 호출

오버라이딩
함수 결과

```
"C:\Program Files\Android\Android
슈퍼클래스의 drink함수
money : 2000
콜라를 마십니다.
money : 500
Process finished with exit code 0
```

❖ 오버라이딩의 장점

- **필요한 부분에 대해서만 상속 받고 그 외의 부분은 수정이 가능**함
으로써, 클래스 선언으로 중복되는 코드를 줄일 수 있다.
- 클래스간 유지 보수를 용이하게 하고, 코드의 간결성을 보장한다.

널 안전성

❖ 널 안전성이란?

- **NullPointerException**에서 안전을 보장함
 - 기본적으로 Kotlin은 Null을 허용하지 않지만, **변수의 자료형 뒤에 '?'를 붙여 예외적으로 NULL을 허용**할 수 있다.

변수 a의 컴파일 에러



NULL 허용된 변수 a

```
fun main() {  
    var a : String = "NPE Test"  
    a = null  
}
```

Null can not be a value of a non-null type String

Add 'toString()' call Alt+Shift+Enter More actions... Alt+Enter

```
fun main() {  
    var a : String? = "NPE Test"  
    a = null  
}
```

❖ 널 안전성 예제

- NULL이 허용된 변수는 **세이프콜(?.)**과 **엘비스 연산자(?:)**를 사용하여 NULL을 검사하고 사용할 수 있다.
 - **엘비스 연산자(?:)**는 변수가 NULL이라면 왼쪽, 아니면 오른쪽을 선택한다.
 - **세이프콜(?.)**는 변수 값이 NULL이라면 뒤의 메소드는 실행하지 않는다.

세이프 콜

```
fun main() {  
    var a : String? = null  
    println(a?.length)  
    var b : String? = null  
    var l = (b?.length)?: -1  
    println("길이 : $l")  
}
```

엘비스 연산자

```
"C:\Program Files\Android\Android  
null  
길이 : -1  
  
Process finished with exit code 0
```

실습

실습 목표와 구성

1. 기초(따라하기)

- 변형 클래스
- 예외 처리

2. 응용(로직구현)

- 클래스 만들기(맥주와 와인)

3. 심화(완성하기)

- 소문자에서 대문자로 변환하는 함수

4. 심화(과제물)

- 자판기 만들기

기초(따라하기) – 예제 1

❖ 변형 클래스

- (line 4) 슈퍼클래스 선언
- (line 5~7) 슈퍼클래스의 drink 메소드 선언
- (line 11) 세금과 세금계산 함수를 고차함수로 받으며, Beverage 클래스의 자식 클래스 Cola 클래스를 생성
- (line 12) 콜라의 가격
- (line 13) 외부에서 정의된 고차함수를 사용하여 계산된 tax
- (line 14~16) 슈퍼클래스에서 선언된 drink 함수를 오버라이딩하여 재정의
- (line 20~22) Cola 클래스 외부에서 메소드를 오버로딩하여 reset_cost 함수 정의
- (line 26~28) 매개변수 고차함수 정의
- (line 29) 잔돈 출력
- (line 31) Cola의 price를 600 -> 300으로 변경
- (line 32) 변경된 잔돈 출력

```
4  open class Beverage() {
5      open fun drink(money: Int): Int {
6          println("슈퍼클래스의 drink함수")
7          return money
8      }
9  }
10
11  class Cola(var taxRate: Double, operation: (Double) -> Double) : Beverage() {
12      var price = 600
13      var tax = operation(taxRate)
14      override fun drink(money: Int): Int {
15          println("콜라를 마십니다.")
16          return money - this.price - this.tax.toInt()
17      }
18  }
19
20  fun Cola.reset_cost(cost: Int) {
21      price = cost
22  }
23
24  fun main() {
25
26      var sub1 = Cola( taxRate: 10.0){taxRate->
27          33.0*(taxRate/100.0)
28      }
29      println("잔돈 : ${sub1.drink( money: 2000)}")
30
31      sub1.reset_cost( cost: 300)
32      println("잔돈 : ${sub1.drink( money: 2000)}")
33  }
```

```
"C:\Program Files\Android\Android
콜라를 마십니다.
잔돈 : 1397
콜라를 마십니다.
잔돈 : 1697

Process finished with exit code 0
```

기초(따라하기) – 예제 2

❖ 예외 처리

- (line 7) 에러 메시지를 반환하는 클래스 선언
- (line 9) score을 인자로 받아 String grade를 반환하는 함수 선언
- (line 10~11) score가 100초과거나 0미만일 경우, ScoreInvalidException클래스를 사용하여 원하는 에러 메시지 출력
- (line 14~19) score의 범위마다 해당 grade를 return
- (line 25~26) readLine()뒤에 붙은 '!!'은 NULL을 받지 않을 것이라고 미리 말해주는 표시
- (line 30) readLine()으로 읽어온 값은 String 형식이며 이를 .toInt()로 변환할 때, 숫자 String 값이 아니면 NumberFormatException 에러 반환
- (line 32) gradeFun 함수 내에서 throw한 ScoreInvalidException 에러를 받았을 때 실행

정상

```
시험점수를 입력하세요.  
95  
95  
95 : A등급  
프로그램 종료
```

NumberFormatException

```
시험점수를 입력하세요.  
#sdfbnjhvd  
형 변환이 불가능합니다.  
프로그램 종료
```

ScoreInvalidException

```
시험점수를 입력하세요.  
130  
130  
0~100의 범위를 넘었습니다.  
프로그램 종료
```

```
7 class ScoreInvalidException(message: String):Exception(message)
8
9 fun gradeFunc(score:Int):String{
10     if(score>100||score<0){
11         throw ScoreInvalidException("0~100의 범위를 넘었습니다.")
12     }
13
14     return when(score){
15         in 90..100 -> "A"
16         in 80..90 -> "B"
17         in 70..80 -> "C"
18         in 60..70 -> "D"
19         else -> "F"
20     }
21 }
22
23 fun main(){
24     try{
25         println("시험점수를 입력하세요.")
26         val score = readLine()!!.toInt()
27
28         val grade = gradeFunc(score)
29         println("${score} : ${grade}등급")
30     }catch(e:NumberFormatException){
31         println("형 변환이 불가능합니다.")
32     }catch(e:ScoreInvalidException){
33         println(e.message)
34     }finally{
35         println("프로그램 종료")
36     }
37 }
```

에러를 반환
하는 클래스

ScoreInvalidException를
throw하면 try문 밖에서
catch

응용(로직구현) – 예제 3

❖ 클래스 만들기(맥주와 와인)

- (line 23~24) 맥주와 와인 타입 나열클래스 정의
- (line 25~26) 맥주와 와인 클래스 정의
- (line 28) 맥주 이름과 맥주 타입, 가격을 출력하는 함수 작성
- (line 31) 맥주의 가격을 바꾸는 함수 작성
- (line 34) 와인은 유로가격으로 받는데, 유로에서 원화로 환산된 가격을 출력하는 함수 작성
 - 원화 : 유로 = 1350 : 1 의 비율로 출력 하시오
- (line 41~42) 맥주 클래스 생성
- (line 44~47) 변경된 맥주 가격 출력
- (line 49~50) 와인 클래스 생성
- (line 52~53) wine1에 대한 원화 가격

```
23  enum class BeerType{LIGHT_LAGER, LIGHT_HYBRID, BOCK, AMBER_HYBRID, FRUIT}
24  enum class WineType{WHITE, ROSE, RED, SPARKLING, DESSERT}
25  class Beer(var name:String, var beerType:BeerType, var cost:Int)
26  class Wine(var name:String, var wineType:WineType, var cost:Int)
27
28  fun Beer.print(){
31  fun Beer.change_price
34  fun Wine.euro_to_won
39  fun main() {
40
41      var beer1 = Beer( name: "Hite", BeerType.FRUIT, cost: 200)
42      var beer2 = Beer( name: "Cass", BeerType.LIGHT_HYBRID, cost: 200)
43
44      beer1.change_price( price: 600)
45      beer1.print()
46      beer2.change_price( price: 600)
47      beer2.print()
48
49      var wine1 = Wine( name: "Cabernet", WineType.RED, cost: 10)
50      var wine2 = Wine( name: "Chardonnay", WineType.WHITE, cost: 12)
51
52      wine1.euro_to_won { a,b->
53          a*b
54      }
55
56  }
```

```
"C:\Program Files\Android\Android Studio\jre\bin\java.exe" ...
맥주이름 : Hite , 맥주타입 : FRUIT, 맥주가격 : 600
맥주이름 : Cass , 맥주타입 : LIGHT_HYBRID, 맥주가격 : 600
유로 : 10 , 원화 : 13500

Process finished with exit code 0
```

심화(완성하기) – 예제 4

❖ 소문자에서 대문자로 변환하는 함수

- 소문자 to 대문자로 변환하여 반환하는 **change()** 함수를 만들어야 하며, **오직 소문자알파벳만** 받아올 수 있다.

Hint

```
var a = 'a'  
println( a.toInt() )
```

```
fun main(){  
    var a = "abcd"  
    println(change(a))  
  
    var b = "EfgH"  
    println(change(b))  
  
    var c = "!@%$"   
    println(change(c))  
}
```

```
"C:\Program Files\Android\Android  
ABCD  
error with = EH  
error with = !@%$  
  
Process finished with exit code 0
```


심화(과제물) – 예제 5

❖ 자판기 만들기

- 다음과 같은 **출력**이 나오도록 **자판기**를 만들어야 한다. **Scanner**로 **메뉴 및 돈을 입력한다.**
- 자판기에 필요한 함수는 다음과 같다.
 - `getChange` // 잔돈 반환하기
 - `getCoin` // 돈 넣기
 - `getMenu` // 메뉴 선택하고 반환하기
 - `getPrice` // 선택한 메뉴의 가격정보 가져오기
- **NULL 값이 입력되는 경우를 고려**해야한다.

❖ (추가) Scanner를 통한 입력 받아오는 방법

- 아래와 같은 방법으로 `inputString` 변수에 입력을 받아올 수 있다.

```
val sc: Scanner = Scanner(System.`in`)
var inputString = sc.nextLine()
```

메뉴

아무것도 선택 되지 않을 시 null 반환 및 재선택

1~5의 메뉴 선택 시, 코인 페이지 돌입

잔돈 반환

```
"C:\Program Files\Android\Android Studi
===== MENU =====
| (1) 참깨라면 - 1,000원 |
| (2) 햄버거 - 1,500원 |
| (3) 콜라 - 800원 |
| (4) 핫바 - 1,200원 |
| (5) 초코우유 - 1,500원 |
Choose menu:
아무것도 선택하지 않았습니다.
다시 선택해주세요.
===== MENU =====
| (1) 참깨라면 - 1,000원 |
| (2) 햄버거 - 1,500원 |
| (3) 콜라 - 800원 |
| (4) 핫바 - 1,200원 |
| (5) 초코우유 - 1,500원 |
Choose menu:
#$$!
아무것도 선택하지 않았습니다.
다시 선택해주세요.
===== MENU =====
| (1) 참깨라면 - 1,000원 |
| (2) 햄버거 - 1,500원 |
| (3) 콜라 - 800원 |
| (4) 핫바 - 1,200원 |
| (5) 초코우유 - 1,500원 |
Choose menu:
1
참깨라면이 선택되었습니다.
Insert coin
2000
2000 원을 넣었습니다.
감사합니다. 잔돈반환:1000
Process finished with exit code 0
```

심화(과제물) – 예제 5

❖ 자판기 만들기

- Menu선택이나 Insert coin 시에 잘못된 선택을 했을 때, **null을 반환**하여 처리해야 한다.
- 금액이 부족할 시, null을 반환한다.

돈을 넣지 않았을 때,
null 반환 및 에러 출력

금액이 부족할 때,
null 반환 및 종료

```
===== MENU =====
| (1) 참깨라면   - 1,000원 |
| (2) 햄버거     - 1,500원 |
| (3) 콜라       -   800원 |
| (4) 핫바       - 1,200원 |
| (5) 초코우유   - 1,500원 |
Choose menu:
2
햄버거가 선택되었습니다.
Insert coin
%@GNVC
돈을 넣지 않았습니다.
다시 돈을 넣어주세요.
Insert coin
100
100 원을 넣었습니다.
현금이 부족합니다.

Process finished with exit code 0
```