

04장 프로그램의 흐름 제어

Copyright Note

- Book: Do it! 코틀린 프로그래밍 / 이지스퍼블리싱
 - Author: 황영덕 (Youngdeok Hwang; sean.ydhwang@gmail.com)
 - Update Date: 10-July-2019
 - Issue Date: 25-Nov-2017
 - Slide Revision #: rev02
 - Homepage: acaroom.net
 - Distributor: 이지스퍼블리싱 (담당: 박현규)
-
- Copyright© 2019 by acaroom.net All rights reserved.
 - All slides cannot be modified and copied without permission.
 - 모든 슬라이드의 내용은 허가없이 변경, 복사될 수 없습니다.

04 프로그램의 흐름 제어

04-1 조건문

04-2 반복문

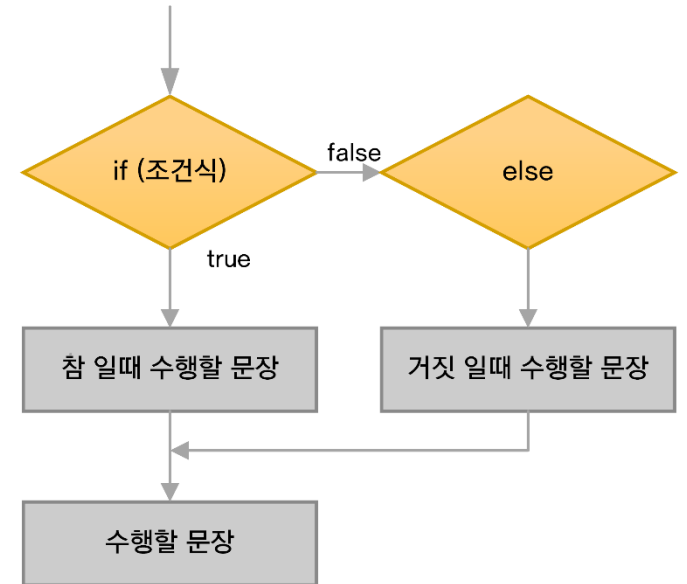
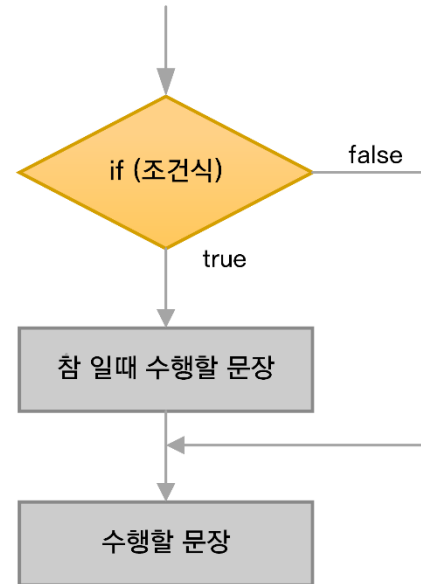
04-3 흐름의 중단과 반환

if문과 if~else문

❖ if문, if~else문

```
if (조건식) {  
    수행할 문장 // 조건식이 참인 경우에만 수행  
    ...  
}
```

```
if (조건식) {  
    수행할 문장 // 조건식이 참인 경우에만 수행  
} else {  
    수행할 문장 // 조건식이 거짓인 경우에 수행  
}
```



```
// if 문  
var max = a  
if (a < b)  
    max = b // 수행할 문장이 한 줄이면 중괄호를 생략할 수 있음
```

if문과 if~else문

❖ if~else문의 간략화

```
var max: Int
if (a > b)
    max = a
else
    max = b
```

```
val max = if (a > b) a else b
```

코딩해 보세요! 조건식의 표현식 사용해 보기 - IfCondition.kt

❖ 블록과 함께 사용하는 경우

```
package chap04.section1

fun main() {

    val a = 12
    val b = 7

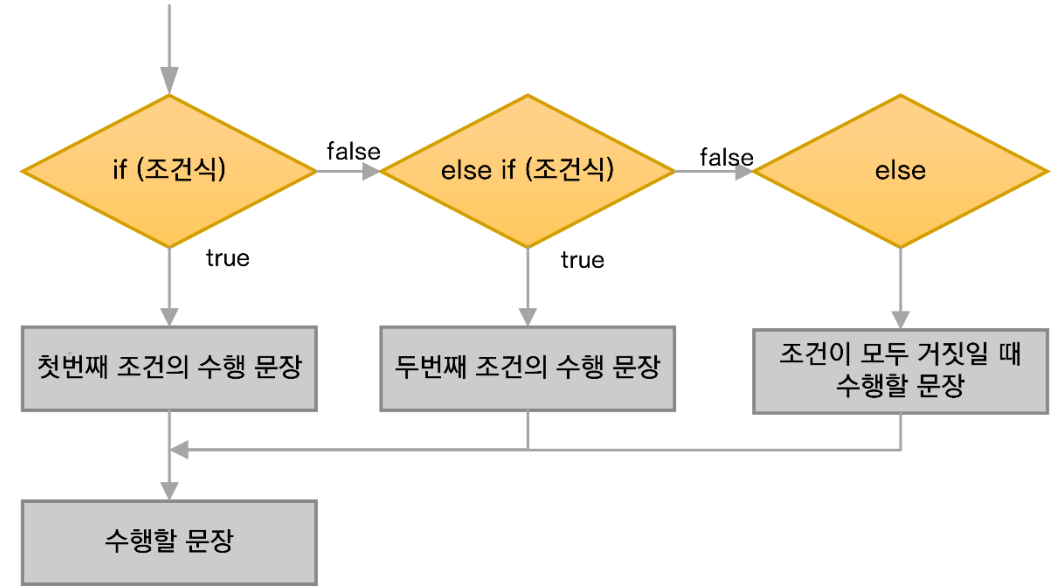
    // 블록과 함께 사용
    val max = if (a > b) {
        println("a 선택")
        a // 마지막 식인 a가 반환되어 max에 할당
    }
    else {
        println("b 선택")
        b // 마지막 식인 b가 반환되 max에 할당
    }

    println(max)
}
```

else if문으로 조건문 중첩하기

❖ else if는 필요한 만큼 조합할 수 있다.

- 마지막은 else로 구성



```
val number = 0
val result = if (number > 0)
    "양수 값"
else if (number < 0)
    "음수 값"
else
    "0"
```

코딩해 보세요! else if를 사용한 등급 판별

❖ IfElseIfCondition.kt

```
package chap04.section1

fun main() {

    print("Enter the score: ")
    val score = readLine()!!.toDouble() // 콘솔로부터 입력 받음
    var grade: Char = 'F'

    if (score >= 90.0) {
        grade = 'A'
    } else if (score >= 80.0 && score <= 89.9) {
        grade = 'B'
    } else if (score >= 70.0 && score <= 79.9) {
        grade = 'C'
    }

    println("Score: $score, Grade: $grade")
}
```


in 연산자와 범위 연산자로 조건식 간략하게 만들기

❖ 비교 연산자와 논리 연산자의 복합

- ...} else if (score >= 80.0 && score <= 89.9) {...
- 비교 연산자(>=, <=)와 논리곱 연산자(&&)가 사용

❖ 범위(range) 연산자

- 변수명 in 시작값..마지막값
- score in 80..89이면 score 범위에 80부터 89까지 포함

in 연산자와 범위 연산자로 조건식 간략하게 만들기

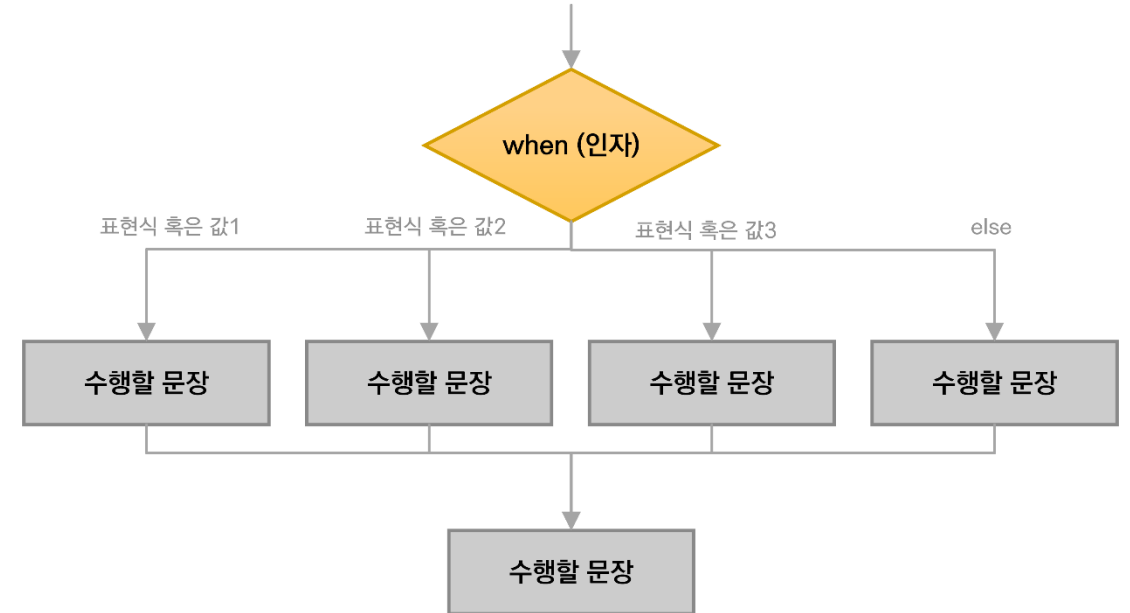
❖ 범위 연산자 이용 예

```
...  
    if (score >= 90) {  
        grade = 'A'  
    } else if (score in 80.0..89.9) {  
        grade = 'B'  
    } else if (score in 70.0..79.9) {  
        grade = 'C'  
    }  
...
```

when문으로 다양한 조건 처리

❖ 인자를 사용하는 when문

```
when (인자) {  
    인자에 일치하는 값 혹은 표현식 -> 수행할 문장  
    인자에 일치하는 범위 -> 수행할 문장  
    ...  
    else -> 문장  
}
```



```
when (x) {  
    1 -> print("x == 1")  
    2 -> print("x == 2")  
    else -> { // 블록 구문 사용 가능  
        print("x는 1, 2가 아닙니다.")  
    }  
}
```

when문으로 다양한 조건 처리

❖ 인자를 사용하는 when문 (Cont.)

- 일치되는 여러 조건

```
when (x) {  
    0, 1 -> print("x == 0 or x == 1")  
    else -> print("기타")  
}
```

- 함수의 반환값 사용하기

```
when (x) {  
    parseInt(s) -> print("일치함!")  
    else -> print("기타")  
}
```

when문으로 다양한 조건 처리

❖ 인자를 사용하는 when문 (Cont.)

- in 연산자와 범위 지정자 사용

```
when (x) {  
    in 1..10 -> print("x는 1 이상 10 이하입니다.")  
    !in 10..20 -> print("x는 10 이상 20 이하의 범위에 포함되지 않습니다.")  
    else -> print("x는 어떤 범위에도 없습니다.")  
}
```

- is 키워드 함께 사용하기

```
val str = "안녕하세요."  
val result = when(str) {  
    is String -> "문자열입니다."  
    else -> false  
}
```

코딩해 보세요! when문을 이용한 점수 등급 구하기 - WhenArgs.kt

```
package chap04.section1

fun main() {

    print("Enter the score: ")
    val score = readLine()!!.toDouble()
    var grade: Char = 'F'

    when(score) {
        in 90.0..100.0 -> grade = 'A'
        in 80.0..89.9 -> grade = 'B'
        in 70.0..79.9 -> grade = 'C'
        !in 70.0..100.0 -> grade = 'F'
    }
    println("Score: $score, Grade: $grade")
}
```

when문으로 다양한 조건 처리

❖ 인자가 없는 when

```
when {  
    조건[혹은 표현식] -> 실행문  
    ...  
}
```

- 특정 인자에 제한하지 않고 다양한 조건을 구성

코딩해 보세요! 인자가 없는 when문 사용하기 - WhenNoArgs.kt

```
package chap04.section1

fun main() {

    print("Enter the score:")
    var score = readLine()!!.toDouble()
    var grade: Char = 'F'

    // 인수 없는 when의 사용
    when {
        score >= 90.0 -> grade = 'A' // 인자 있는 when과 다르게 조건식을 구성할 수 있음
        score in 80.0..89.9 -> grade = 'B'
        score in 70.0..79.9 -> grade = 'C'
        score < 70.0 -> grade = 'F'
    }
    println("Score: $score, Grade: $grade")
}
```


코딩해 보세요! 다양한 자료형의 인자 받기 - WhenAnyCase.kt

```
package chap04.section1

class MyClass

fun main() {
    cases("Hello") // ② String
    cases(1) // Int
    cases(System.currentTimeMillis()) // Long
    cases(MyClass()) // 객체
}

fun cases(obj: Any) { // ①
    when (obj) {
        1 -> println("Int: $obj")
        "Hello" -> println("String: $obj")
        is Long -> println("Long: $obj")
        !is String -> println("Not a String")
        else -> println("Unknown")
    }
}
```

04 프로그램의 흐름 제어

04-1 조건문

04-2 반복문

04-3 흐름의 중단과 반환

for문

❖ for 문의 선언

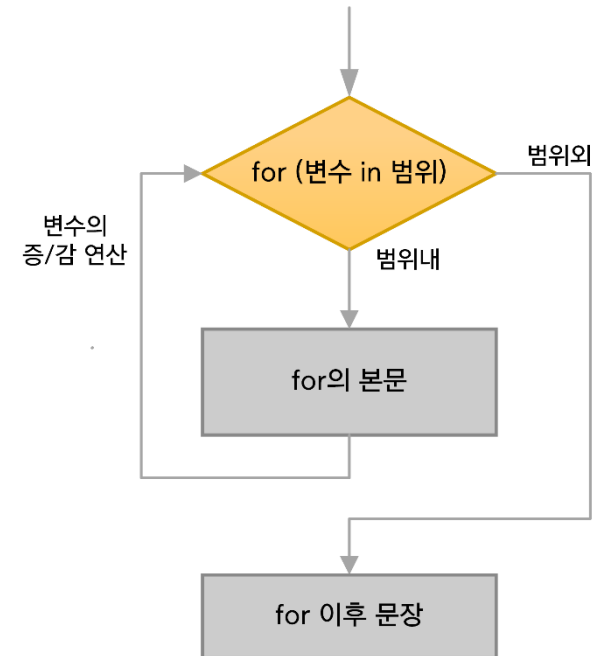
```
for (요소 변수 in 컬렉션 혹은 범위) { 반복할 본문 }
```

```
for (x in 1..5) { // 코틀린의 in과 범위 지정을 활용한 루프  
    println(x) // 본문  
}
```

```
for (x in 1..5) println(x) // 한줄에 표현하는 경우
```



for (int i = 1; i <= 5; i++) { ... } // 에러!
코틀린에서는 자바와 같은 세미콜론 표현식을 사용하지
않음



코딩해 보세요! 1부터 10까지 더하는 예제 작성

❖ ForSum.kt

```
package chap04.section2

fun main() {

    var sum = 0

    for (x in 1..10) sum += x

    println("sum: $sum")
}
```

하행, 상행 및 다양한 반복 방법

❖ 하행 반복 - downTo

- 5, 4, 3, 2, 1

```
for (i in 5 downTo 1) print(i)
```

```
for (i in 5..1) print(i) // 잘못된 사용! 아무것도 출력되지 않는다
```

❖ 필요한 단계 증가 - step

- 1, 3, 5

```
for (i in 1..5 step 2) print(i)
```

❖ 혼합 사용

- 5, 3, 1

```
for (i in 5 downTo 1 step 2) print(i)
```

for문을 활용한 삼각형 출력하기

❖ 의사 코드(Pseudo-code)

```
n: 줄 수 입력
반복 (line: 1→n만큼) {
    반복 (space: 1→(n-line)만큼) { 공백 출력 }
    반복 (star: 1→(2*line-1)만큼) { 별표 출력 }
    개행
}
```

코딩해 보세요! 반복문을 이용해 삼각형 출력하기

❖ ForTriangle.kt

```
package chap04.section2

fun main() {

    print("Enter the lines: ")
    val n = readLine()!!.toInt() // 콘솔로부터 입력받음

    for (line in 1..n) {
        for (space in 1..(n - line)) print(" ") // 공백 출력
        for (star in 1..(2 * line - 1)) print("*") // 별표 출력
        println() // 개행
    }
}
```

```
Enter the lines: 5
  *
 ***
*****
*****
*****
```

코딩해 보세요! 짝수와 홀수의 합 구하기

❖ ForOddSum.kt

```
package chap04.section2

fun main() {

    var total: Int = 0

    for (num in 1..100 step 2) total += num
    println("Odd total: $total")

    for (num in 0..99 step 2) total += num
    println("Even total: $total")

}
```

```
Odd total: 2500
Even total: 4950
```


while문

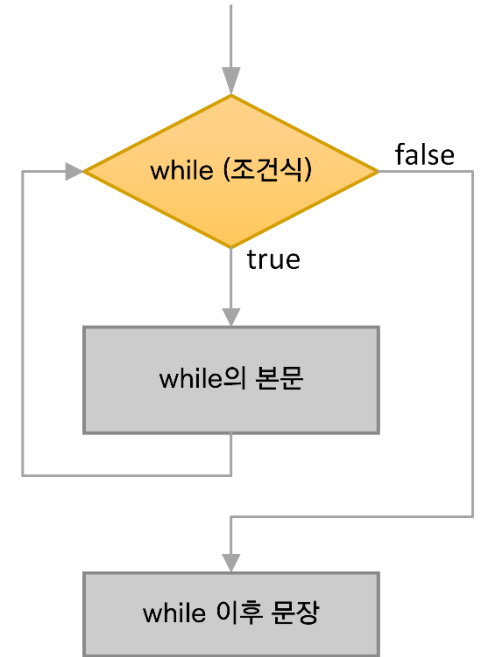
❖ 선언 사용

```
while (조건식) { // 조건식이 true인 동안 본문의 무한 반복
    본문
    ....
}
```

```
var i = 1
while (i <= 5) {
    println("$i")
    ++i // 계속 반복하다 보면 조건식이 5 이상으로 넘어갈 때 false가 되어 탈출
}
```

■ 데몬 프로그램의 사용 예

```
while (true) {
    temp = 온도 검사
    if ( temp > 한계 온도) { 경고 발생 }
    ...
}
```



코딩해 보세요! while문으로 활용한 팩토리얼 계산하기

❖ WhileFactorial.kt

```
package chap04.section2

fun main() {

    print("Enter the number: ")
    var number = readLine()!!.toInt()
    var factorial: Long = 1

    while (number > 0) { // n × ... × 4 × 3 × 2 × 1
        factorial *= number
        --number
    }

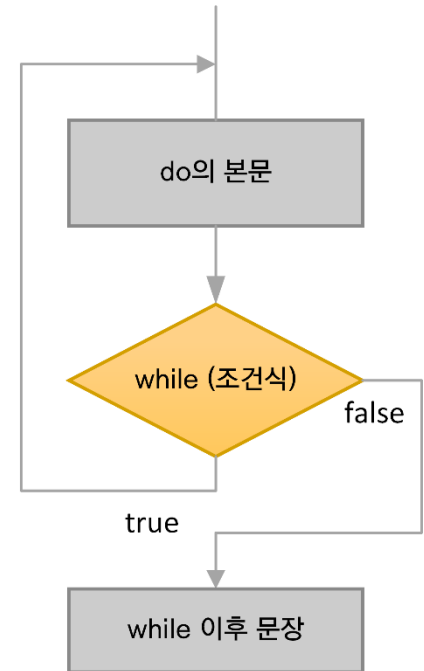
    println("Factorial: $factorial")
}
```

```
Enter the number: 5
Factorial: 120
```

do~while문

❖ 선언

```
do {  
    본문  
} while (조건식)
```



코딩해 보세요! do~while 사용해 보기

❖ DoWhileLoop.kt

```
package chap04.section2

fun main() {

    do {
        print("Enter an integer: ")
        val input = readLine()!!.toInt()

        for (i in 0..(input-1)) {
            for (j in 0..(input-1)) print((i + j) % input + 1)
            println()
        }
    } while (input != 0)
}
```

```
Enter an integer: 5
12345
23451
34512
45123
51234
Enter an integer: 0
```

04 프로그램의 흐름 제어

04-1 조건문

04-2 반복문

04-3 흐름의 중단과 반환

흐름 제어

❖ 흐름 제어 관련 요약

흐름 제어문

- **return**: 함수에서 결과값을 반환하거나 지정된 라벨로 이동
- **break**: for나 while의 조건식에 상관없이 반복문을 끝냄
- **continue**: for나 while의 반복문의 본문을 모두 수행하지 않고 다시 조건으로 넘어감

예외 처리문

- **try {...} catch {...}**: try 블록의 본문을 수행하는 도중 예외가 발생하면 catch 블록의 본문을 실행
- **try {...} catch {...} finally {...}**: 예외가 발생해도 finally 블록 본문은 항상 실행

return의 사용

❖ return으로 값 반환하기

```
fun add(a: Int, b: Int): Int {  
    return a + b  
    println("이 코드는 실행되지 않습니다.") // 여기에 도달하지 않음  
}
```

❖ return으로 Unit 반환하기

```
// 1. Unit을 명시적으로 반환  
fun hello(name: String): Unit {  
    println(name)  
    return Unit  
}
```

```
// 2. Unit 이름을 생략한 반환  
fun hello(name: String): Unit {  
    println(name)  
    return  
}
```

```
// 3. return문 자체를 생략  
fun hello(name: String) {  
    println(name)  
}
```

return의 사용

❖ 람다식에서 return 사용하기 - InlineLambdaReturn.kt

```
fun main() {  
    retFunc()  
}  
  
inline fun inlineLambda(a: Int, b: Int, out: (Int, Int) -> Unit) {  
    out(a, b)  
}  
  
fun retFunc() {  
    println("start of retFunc") // ①  
    inlineLambda(13, 3) { a, b -> // ②  
        val result = a + b  
        if(result > 10) return // ③ 10보다 크면 이 함수를 빠져 나감  
        println("result: $result") // ④ 10보다 크면 이 문장에 도달하지 못함  
    }  
    println("end of retFunc") // ⑤  
}
```


람다식에서 라벨과 함께 return 사용하기

❖ 람다식에서 라벨 사용

```
람다식 함수명 라벨이름@ {  
    ...  
    retrun@라벨이름  
}
```

```
...  
fun inlineLambda(a: Int, b: Int, out: (Int, Int) -> Unit) { // inline이 제거됨  
    out(a, b)  
}  
  
fun retFunc() {  
    println("start of retFunc")  
    inlineLambda(13, 3) lit@{ a, b -> // ① 람다식 블록의 시작 부분에 라벨을 지정함  
        val result = a + b  
        if(result > 10) return@lit // ② 라벨을 사용한 블록의 끝부분으로 반환  
        println("result: $result")  
    } // ③  
    println("end of retFunc") // ④ 이 부분이 실행됨  
}
```

NoInlineLambdaReturn.kt

람다식에서 라벨과 함께 return 사용하기

❖ 암묵적 라벨

```
...  
fun retFunc() {  
    println("start of retFunc")  
    inlineLambda(13, 3) { a, b ->  
        val result = a + b  
        if(result > 10) return@inlineLambda  
        println("result: $result")  
    }  
    println("end of retFunc")  
}  
...
```

익명 함수를 사용한 반환

❖ 익명 함수의 사용

```
fun retFunc() {  
    println("start of retFunc")  
    inlineLambda(13, 3, fun (a, b) {  
        val result = a + b  
        if(result > 10) return  
        println("result: $result")  
    }) // inlineLambda()함수의 끝  
    println("end of retFunc")  
}
```

람다식과 익명 함수 사용

❖ 람다식 방법

```
...
val getMessage = lambda@ { num: Int ->
    if(num !in 1..100) {
        return@lambda "Error" // 레이블을 통한 반환
    }
    "Sucess" // 마지막 식이 반환
}
...
```

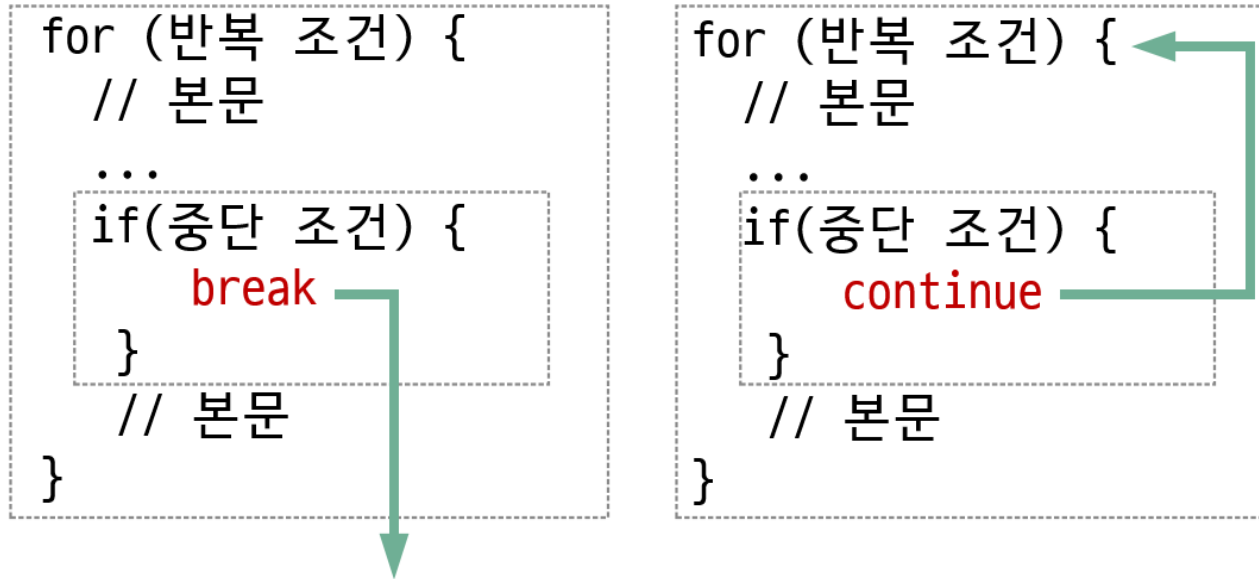
❖ 익명 함수 방법

```
...
val getMessage = fun(num: Int): String {
    if(num !in 1..100) {
        return "Error"
    }
    return "Success"
}
...
val result = getMessge(99)
```

break와 continue

❖ break와 continue의 사용

- for나 while, do...while문 루프를 빠져 나옴



코딩해 보세요! 조건에 따른 break 사용하기

❖ NormalBreakContinue.kt

```
package chap04.section3

fun main() {
    for(i in 1..5) {
        if (i==3) break
        print(i)
    }
    println() //개행 문자
    println("outside")
}
```

```
12
outside
```

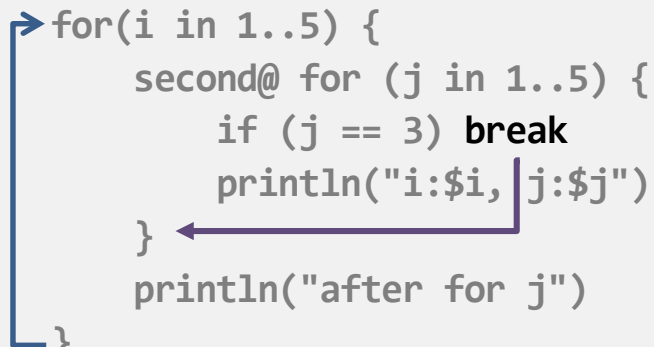
- break를 continue로 바꾸면?

```
1245
outside
```

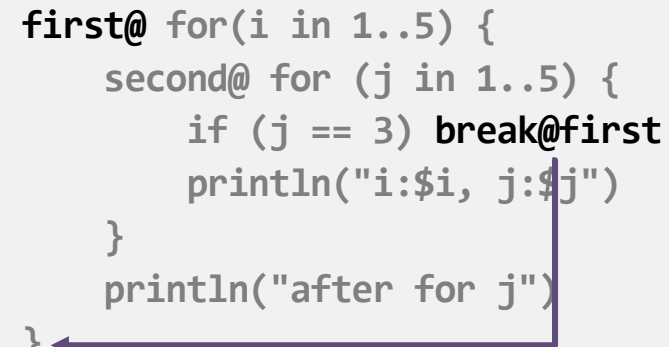
break와 라벨 함께 사용하기

❖ 라벨 없는 break와 라벨을 사용한 break

```
fun labelBreak() {  
    println("labelBreak")  
    for(i in 1..5) {  
        second@ for (j in 1..5) {  
            if (j == 3) break  
            println("i:$i, j:$j")  
        }  
        println("after for j")  
    }  
    println("after for i")  
}
```



```
fun labelBreak() {  
    println("labelBreak")  
    first@ for(i in 1..5) {  
        second@ for (j in 1..5) {  
            if (j == 3) break@first  
            println("i:$i, j:$j")  
        }  
        println("after for j")  
    }  
    println("after for i")  
}
```



- continue를 사용할 때도 생각해 보자.

예외 처리

❖ 예외(exception)

- 실행 도중의 잠재적인 오류까지 검사할 수 없기 때문에 정상적으로 실행이 되다가 비정상적으로 프로그램이 종료되는 경우
 - 운영체제의 문제 (잘못된 시스템 호출의 문제)
 - 입력값의 문제 (존재하지 않는 파일 혹은, 숫자 입력란에 문자 입력 등)
 - 받아들일 수 없는 연산 (0으로 나누기 등)
 - 메모리의 할당 실패 및 부족
 - 컴퓨터 기계 자체의 문제 (전원 문제, 망가진 기억 장치 등)

예외 처리

❖ 예외를 대비하기 위한 구문

```
try {  
    예외 발생 가능성 있는 문장  
} catch (e: 예외처리 클래스명) {  
    예외를 처리하기 위한 문장  
} finally {  
    반드시 실행되어야 하는 문장  
}
```

- 반드시 실행해야 할 작업이 없다면 finally 블록은 생략하고 try~catch 블록만으로 코드를 구성할 수 있다.

코딩해 보세요! 0으로 나누었을 때 예외 처리하기

❖ TryCatch.kt

```
package chap04.section3

fun main() {
    val a = 6
    val b = 0
    val c : Int

    try {
        c = a/b // 0으로 나눔
    } catch (e : Exception){
        println("Exception is handled.")
    } finally {
        println("finally 블록은 반드시 항상 실행됨")
    }
}
```

```
Exception is handled.
finally 블록은 반드시 항상 실행됨
```

특정 예외 처리

❖ 산술 연산에 대한 예외를 따로 특정해서 잡을 때

```
...  
} catch (e : ArithmeticException){  
    println("Exception is handled. ${e.message}")  
}
```

Exception is handled. / by zero
finally 블록은 반드시 항상 실행됨

■ 스택의 추적

```
...  
} catch (e : Exception){  
    e.printStackTrace()  
}  
...
```

finally 블록은 반드시 항상 실행됨
java.lang.ArithmeticException: / by zero
at com.acaroom.kotlin.chap04.section3.TryCatchKt.main(TryCatch.kt:9)

예외 발생시키기

❖ 특정 조건에 따른 예외 발생

```
throw Exception(message: String)
```

코딩해 보세요! throw를 사용해 예외 발생시키기

❖ ThrowExceptionTest.kt

```
package chap04.section3

fun main() {

    var amount = 600

    try {
        amount -= 100
        checkAmount(amount)
    } catch (e : Exception){
        println(e.message)
    }
    println("amount: $amount")
}

fun checkAmount(amount : Int){
    if (amount < 1000)
        throw Exception("잔고가 $amount 으로 1000 이하입니다.")
}
```

사용자 정의 예외

❖ 사용자 예외 정의

```
class <사용자 예외 클래스명>(message: String) : Exception(message)
```

```
class InvalidNameException(message: String) : Exception(message) // ① 사용자 예외 클래스

fun main() {
    var name = "Kildong123" // ② 숫자가 포함된 이름
    try {
        validateName(name)
    } catch (e : InvalidNameException){ // ④ 숫자가 포함된 예외 처리
        println(e.message)
    } catch (e : Exception){ // ⑤ 기타 예외 처리
        println(e.message)
    }
}

fun validateName(name : String){
    if(name.matches(Regex(".*\\d+.*"))){ // ③ 이름에 숫자가 포함되어 있으면 예외 던지기
        throw InvalidNameException("Your name : $name : contains numerals.")
    }
}
```