Kotlin : Introduction

Mobile Software
2019 Fall
인천대학교 컴퓨터공학부
홍 윤식 교수

Kotlin : 러시아 Baltic Sea 인근에 있는 섬

Why Kotlin?(1/2)

- Google announces Kotlin for Android
  - https://www.youtube.com/watch?v=d8ALcQiuPWs
  - 2017. 5. 18 google I/O 에서 Google이 Android 공식 언어로 kotlin을 추가
  - Android Studio 3.0부터 kotlin 기본 지원
    - 이전 버전에서도 plug-in만 설치하면 됨

Why Kotlin?(2/2)

- https://kotlinlang.org – official web site
- JetBrains (http://www.jetbrains.com)에서 개발
  - 2011년 최초 공개 → 2016년 정식 버전(1.0) 출시
  - Java와 100% 호환 : Java ↔ Kotlin

Google은 왜 java에서 Kotlin으로 바꿨을까?

James Gosling at Sun Microsystems
Father of Java

Oracle America, Inc. v. Google, Inc.

a current legal case within the USA related to the **nature of computer code and copyright law**.

- *Oracle JDK*
- *Open JDK*
- *Zulu JDK*
- *kotlin*

OpenJDK

What is this? The place to collaborate on an open-source implementation of the Java Platform, Standard Edition, and related projects. (Learn more.)

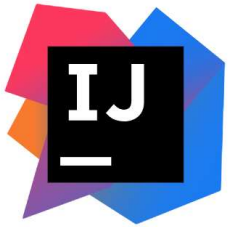
All Zulu Community™ builds of OpenJDK are TCK-tested and free to download and use without restrictions

Kotlin 특징

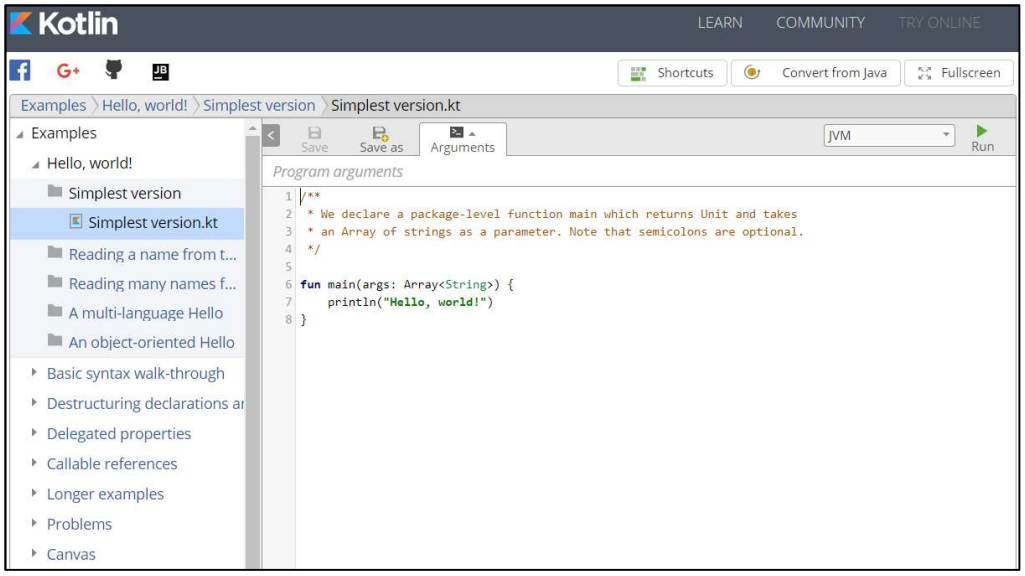
- **Static type**
  - 모든 프로그램 구성 요소의 타입을 compile time에 알 수 있음.
    - 타입 추론 (type inference)
- **Concise and safe**
  - Python, swift 등 최근 언어 추세
- **Multi-paradigm language**
  - 함수형 프로그래밍과 객체지향 프로그래밍이 모두 가능
  - 함수형 프로그래밍
    - **순수 함수**(pure function) : 같은 인자에 대해 항상 같은 결과를 반환
    - **람다 식** (lambda expression) : 이름이 없는 함수를 식으로 표현
    - **고차 함수**(high order function) : 다른 함수를 인자로 사용하거나 함수를 결과로 반환

How to run Kotlin?

- Download IntelliJ IDEA
  - https://www.jetbrains.com/idea/download/#section=windows
  - IntelliJ IDEA (*Community Edition*)

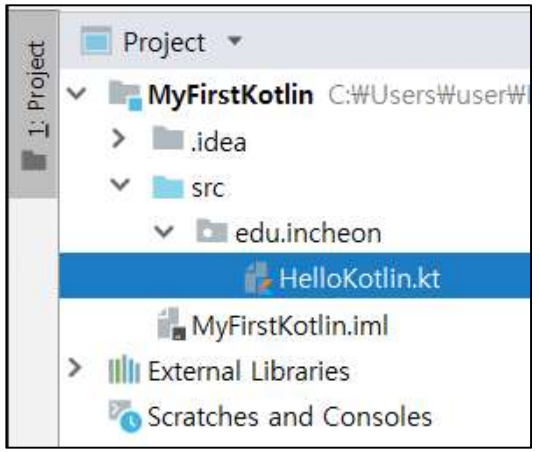


- 웹 브라우저에서 실행
  - https://try.kotl.in



Kotlin 프로그램 구조

- Project > src > package
  - Package
  - Import 문
  - 함수(fun) 및 클래스(class)
    - 파일 확장자는 **.kt**




```kotlin
package edu.incheon

import kotlin.math.PI
import kotlin.math.abs

fun main(){
    println(PI)
    println(abs(-12.6))
}
```

```kotlin
package edu.incheon

import kotlin.math.PI
import kotlin.math.abs

fun main(args: Array<String>){
    println(PI)
    println(abs(-12.6))
}
```

기본 문장: function (1/2)

```
package edu.incheon

fun main(){
    println(max(1, 2))
}

fun max(a: Int, b: Int): Int {
    return if ( a > b ) a else b
}
```

블록(block)이 본문인 함수
→ block : 중괄호({, })로 둘러싸인 문장들

식(expression)이 본문인 함수

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

Return type 생략 – 식이 본문인 함수에서만 가능

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

기본 문장 : function (2/2)

return 값이 있음

return 값이 없음

```
fun main(){
    println(sum(1, 2))
    printSum(5, 6)
}

fun sum(a: Int, b: Int): Int {
    return a + b
}

fun printSum(a: Int, b: Int): Unit {
    println("sum of $a and $b is ${a+b}")
}
```

return type은 유추(infer) 할 수 있으므로 생략

Unit은 생략 가능

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

fun printSum(a: Int, b: Int) {
    println("sum of $a and $b is ${a+b}")
}
```

variables

- **val** (value): **Immutable** variables. read-only variables

```
val a: Int = 1  // immediate assignment
val b = 2   // `Int` type is inferred
val c: Int  // Type required when no initializer is provided
c = 3       // deferred assignment
```

- **var** (variable) : **Mutable** variables. writeable variables

```
var x = 5 // `Int` type is inferred
x += 1
```

Basic Types : Numbers

- Numbers
  - **Double**(64), **Float**(32), **Long**(64), **Int**(32), **Short**(16), **Byte**(8)
    - 괄호 안 숫자는 bit width
  - 상수(literal constants)
    - **Double** : 123.5, 123.5e10
    - **Float** : 123.5**F**, 123.5**f**
    - **Long** : 123**L**
    - **Hexadecimals : 0x**0F
    - **Binaries : 0b**00001011
  - 형 변환(type conversion) 함수
    - **toByte**(), **toShort**(), **toInt**(), **toLong**(), **toFloat**(), **toDouble**(), **toChar**()

Numbers 예

```
fun main(){
    println("Byte : " + Byte.MIN_VALUE + ", " + Byte.MAX_VALUE + ", " + Byte.SIZE_BYTES)
    println("Short : " + Short.MIN_VALUE + ", " + Short.MAX_VALUE + ", " + Short.SIZE_BYTES)
    println("Int : " + Int.MIN_VALUE + ", " + Int.MAX_VALUE + ", " + Int.SIZE_BYTES)
    println("Long : " + Long.MIN_VALUE + ", " + Long.MAX_VALUE + ", " + Long.SIZE_BYTES)
    println("Float : " + Float.MIN_VALUE + ", " + Float.MAX_VALUE)
    println("Double : " + Double.MIN_VALUE + ", " + Double.MAX_VALUE)

    var f: Float = 3.9f
    var i: Int = f.toInt()

    println()
    println("f = $f, i = $i")
}
```

```
Byte : -128, 127, 1
Short : -32768, 32767, 2
Int : -2147483648, 2147483647, 4
Long : -9223372036854775808, 9223372036854775807, 8
Float : 1.4E-45, 3.4028235E38
Double : 4.9E-324, 1.79769313486231570E308

f = 3.9, i = 3
```

Basic Types : Characters

- Characters
  - 문자 자료형은 문자만 대입
  - 숫자를 대입할 경우 컴파일 에러 발생
    - toChar() ➔ 다른 type을 Char로 변환

```
val c : Char = 'A'
```

```
val c : Char = 65
```

```
fun main(){
    val code: Int = 65      // 문자 'A'의 ASCII code
    val han_code = '\uD55C' // unicode '한'

    // ASCII code에 해당하는 문자로 변환
    println(code.toChar() + ", " + (code+1).toChar())
    // unicode 문자 출력
    println(han_code)
}
```

```
fun main(){
    for (i in '0'..'9') {
        print("${decimalValue(i)} ")
    }
    println()
}
```

```
fun decimalValue(c: Char): Int {
    if (c !in '0'..'9')
        throw IllegalArgumentException("out of range")
    return c.toInt() - '0'.toInt()
}
```

Basic Types : Booleans

- 논리
  - 값 : **true**(참), **false**(거짓)
  - 논리 연산자 : &&, ||, !

```
fun main(){
    val foo: Boolean = true
    val bar = false

    println( foo && bar)
    println( foo || bar)
    println( !foo )
}
```

```
false
true
false
```

Basic Types : Strings (1/2)

- String (**문자열**, *string of characters*)
  - String은 **immutable** (read-only).
  - 문자열의 특정 위치의 문자(Char)에 접근하기 위해
    - get() 또는 [ ]와 인덱스 사용

```
fun main(){
    val foo: String = "Lorem ipsum"
    val ch1: Char = foo.get(4)   // 'm' 출력
    val ch2: Char = foo[6]       // 'i' 출력
    var s = "python"

    // foo[0] = 'X'  ---> error
    for (c in foo)
        print(c)
    println()

    // 새 메모리 할당. s는 이 메모리 공간 참조.
    // "python"을 저장하던 메모리 공간은 garbage가 됨.
    s = "Kotlin"
}
```

Basic Types : Strings (2/2)

- Strings
  - **String.format**() : 규격화된 문자열(formatted string) 출력

```
fun main(){
    val pi: Float = PI.toFloat()
    val digit = 10
    val str = "hello"
    val length: Int = 3000

    val lengthStr: String = String.format("Length: %d meters", length)
    println("pi = %.2f, %3d, %s".format(pi, digit, str))
    println(lengthStr)
}
```

```
pi = 3.14,  10, hello
Length: 3000 meters
```

Basic Types : string templates (1/2)

- 앞에서 정의한 변수를 문자열에서 참조할 수 있음
  - 변수 이름 앞에 $를 붙임.

```
fun main(){
    var a = 1
    val s1 = "a is $a"

    a = 2
    val s2 = "${s1.replace("is", "was")}, but now is $a"
    println(s2)

    val s = "abc"
    println("$s.length is ${s.length}")
}
```

```
a was 1, but now is 2
abc.length is 3
```

Basic Types : string templates (2/2)

- 문자열 템플릿(string template) : 문자열 내에 인자를 직접 대입
  - 템플릿 문자열에 포함할 표현식은 중괄호로 구분
  - 문자열 내에 통화 기호($)를 포함해야 하는 경우, escape char를 지원하지 않음.

```
fun main(){
    val expr: String = "Lorem ipsum"
    val price: Int = 1000

    val lengthStr: String = "text length: ${expr.length}"
    val priceStr: String = "price: ${'$'}$price"

    println(lengthStr)
    println(priceStr)
    println("\'Hey\", I have only $price${'$'}")
}
```

```
text length: 11
price: $1000
'Hey", I have only 1000$
```

Safe call (1/2)

- 프로그램 실행 중에 현재 **null** 값을 갖고 있는 변수 또는 객체에 접근하면, 실행 중단!!!
  - **NullPointerException** (NPE) 예외 발생
- 변수(객체)가 null 값을 갖는지 여부는 programmer가 직접 지정해야 함
  - 기본은 **null** 값을 갖지 않음.

```
var str: String
str = null          // 에러 발생
```

  - **null** 값을 허용하도록 하려면

type 뒤에 ?를 붙임

```
fun main(){
    var str: String?
    str = null          // 에러 발생

    if (str.isNullOrBlank())
        println("str is null.")
    else
        println("str is NOT null.")
}
```

Safe call (2/2)

```
fun main(){
    var str: String? = "Hello, Kotlin"
    str = null      // NPE 예외가 발생하도록 str 값을 null로 변경

    println("str: $str length = ${str.length}")
}
```

Error 발생

```
println("str: $str length = ${str?.length}")
```

변수 이름 뒤에 ?를 붙임
➔ safe call

```
val len = if (str != null) str.length else -1
println("str: $str length = ${len}")
```

```
val len = str?.length ?: -1
println("str: $str length = ${len}")
```

Elvis 연산자('?:') 사용

Type checking

```
fun main(){
    var num = 8L
    // Any : 최상위 기본 클래스. 어떤 type으로도 변환 가능
    val str: Any

    typeCheck(num)
    typeCheck(8)

    str = "Hello, Kotlin"
    if (str is String) {
        println("\"$str\" is ${str.javaClass}")
    }
}

fun typeCheck(x: Any) {
    if (x is Int) {
        println("$x is ${x.javaClass}")
    }
    else if (x !is Int) {
        println("$x is NOT Int. The type is ${x.javaClass}")
    }
}
```

```
8 is NOT Int. The type is class java.lang.Long
8 is int
"Hello, Kotlin" is class java.lang.String
```

conditional expressions

```
fun main(){
    val a = 12
    val b =7

    println(maxValue(a, b))
}

fun maxValue(x: Int, y: Int): Int {
    if (x > y) {
        println("$x is chosen")
        return x
    } else {
        println("$y is chosen")
        return y
    }
}
```

```
fun main(){
    val a = 12
    val b =7

    val maxValue = if ( a > b ) {
        println("$a is chosen")
        a
    } else {
        println("$b is chosen")
        b
    }

    println(maxValue)
}
```

block의 마지막 식이 변수 max에 할당

```
fun maxValue(x: Int, y: Int)= if (x > y) x else y
```

when statement

```
fun main(){
    checkValue(1)
    checkValue(3)
    checkValue(5)
}

fun checkValue(x: Int) {
    when (x) {
        1 -> println("x is 1")
        2 -> println("x is 2")
        3 -> println("x is 3")
        else -> {
            println("x is NOT 1,2,3")
        }
    }
}
```

Non-null assertion(!!)

```
fun main(){
    print("Enter the score:")
    val score = readLine()!!.toDouble()
    val grade = decideGrade(score)
    println("Score: $score, Grade: $grade")

}

fun decideGrade(score: Double): Char {
    var grade: Char = 'F'

    when (score) {
        in 90.0..100.0 -> grade = 'A'
        in 80.0..89.0 -> grade = 'B'
        in 70.0..79.0 -> grade = 'C'
        in 60.0..69.0 -> grade = 'D'
        else -> grade = 'F'
    }
    return grade
}
```

```
Enter the score:95
Score: 95.0, Grade: A
```

when에서 enum 클래스 처리

```
enum class Color {  // rainbow 7 colors
    RED, ORANGE, YELLOW, GREEN, BLUE, INDIGO, VIOLET
}

fun main(){
    println(getMnemonics(Color.BLUE))
    println(getWarmth(Color.ORANGE))
}

fun getMnemonics(color: Color) =
    when (color) {
        Color.RED -> "Richard"
        Color.ORANGE -> "Of"
        Color.YELLOW -> "York"
        Color.GREEN -> "Gave"
        Color.BLUE -> "Battle"
        Color.INDIGO -> "In"
        Color.VIOLET -> "Vain"
        else -> "Not a defined color"
    }
```

```
fun getWarmth(color: Color) =
    when(color) {
        Color.RED, Color.ORANGE, Color.YELLOW -> "warm"
        Color.GREEN -> "neutral"
        Color.BLUE, Color.INDIGO, Color.VIOLET -> "cold"
        else -> "Not a defined color"
    }
```

Range and loop

```
fun main(){
    for (i in 1..100) {
        print(fizzBuzz(i))
        if (i % 10 == 0) println()
    }

    for (i in 100 downTo 1 step 2) {
        print(fizzBuzz(i))
        if (i % 10 == 0) println()
    }
}

fun fizzBuzz(i: Int) =
    when {
        i % 15 == 0 -> "FizzBuzz"
        i % 3 == 0 -> "Fizz"
        i % 5 == 0 -> "Buzz"
        else -> "$i"
    }
```

One dimensional arrays (1/3)

```
fun main(){
    // 1차원 배열: 배열 원소의 type에 제한이 없음
    val numbers = arrayOf(1, 2, 3, 4)
    val mixArrays = arrayOf(7, "Kotlin", false)

    // 1차원 배열: 한 가지 type의 원소로만 구성됨.
    val intOnlyArrays = arrayOf<Int>(1, 2, 3, 4)
    val intOnlyArrays2 = intArrayOf(1, 2, 3, 4)

    val i = intOnlyArrays[0]
    val i2 = intOnlyArrays.get(2)

    println("i=$i, i2=${i2}")
}
```

charArrayOf, booleanArrayOf, longArrayOf
shortArrayOf, byteArrayOf, ...

```
fun main(){
    val words: Array<String>
        = arrayOf("Lorem", "ipsum", "dolor", "sit")
    val str: String? = null

    for (str in words) {
        print(str + "\t")
    }
    println()
}
```

One dimensional arrays (2/3)

```
fun main(){
    val words: Array<String>
        = arrayOf("Lorem", "ipsum", "dolor", "sit")
    val intArr: Array<Int> = arrayOf(5, 6, 7, 8)
    val intArr2: IntArray = intArrayOf(1, 2, 3, 4)

    bar(words)
    foo(intArr)
    foo2(intArr2)

}
```

```
fun bar(args: Array<String>) {
    for (s in args) {
        println(s)
    }
}

fun foo(arr: Array<Int>) {
    for (s in arr) {
        println(s)
    }
}

fun foo2(arr: IntArray) {
    for (s in arr) {
        println(s)
    }
}
```

One dimensional arrays (3/3)

```
fun main(){
    val words: Array<String>
        = arrayOf("Lorem", "ipsum", "dolor", "sit")
    val intArr: Array<Int> = arrayOf(5, 6, 7, 8)
    val intArr2: IntArray = intArrayOf(1, 2, 3, 4)

    unified<String>(words)   // bar(words)
    unified<Int>(intArr)     // foo(intArr)
    foo2(intArr2)
}

fun <T> unified(arr: Array<T>){
    for (s in arr) {
        println(s)
    }
}

fun foo2(arr: IntArray) {
    for (s in arr) {
        println(s)
    }
}
```

Two dimensional arrays

```
fun main(){
    val array1 = arrayOf(1, 2, 3)
    val array2 = arrayOf(4, 5, 6)
    val array3 = arrayOf(7, 8, 9)

    val arr2d = arrayOf(array1, array2, array3)

    for (e1 in arr2d) {
        for (e2 in e1) {
            print(e2)
        }
        println()
    }
    println(arr2d[2][1])
}
```

```
123
456
789
8
```