32장. 코틀린 기초 작성법

32.1 코틀린 파일 작성 규칙

32.1.1. 패키지 정의

- 코틀린 파일의 기본 구조는 패키지 선언, import 선언 그리고 파일 내용 순으로 작성
- 가상의 논리적인 패키지명을 사용 가능

```
package com.example.test11_32
import java.text.SimpleDateFormat
import java.util.*
/**
* Created by kkang on 2017-08-08.
*/
class PackageTest {
  fun getNow():String{
    val now=Date()
    val sdf=SimpleDateFormat("yyyy-MM-dd HH:mm")
    return sdf.format(now)
  }
}
```

32.1 코틀린 파일 작성 규칙

- 별도의 import 구문 없이 사용할 수 있는 패키지

- kotlin.*
- kotlin.annotation.*
- kotlin.collections.*
- kotlin.comparisons.* (since 1.1)
- kotlin.io.*
- kotlin.ranges.*
- kotlin.sequences.*
- kotlin.text.*
- java.lang.*
- kotlin.jvm.*

- as 키워드로 이름을 변형하여 사용 가능

```
import java.text.SimpleDateFormat as MySimpleDateFormat
```

32.1 코틀린 파일 작성 규칙

32.1.2. 파일로 작성

- 클래스 단위로 정의하지 않고 일반 파일로 작성하는 방법을 제공

```
package com.example.test11_32
val name = "kkang"
fun sum(count: Int): Int {
    var sum=0
    for (i in 1..count){
        sum += i
    }
    return sum
}
```

```
class MainActivity : AppCompatActivity(), View.OnClickListener {
//...
Log.d("kkang", "name : $name, sum : ${sum(10)}")
//...
}
```

- 파일이 다른 패키지에 있을 때는 변수와 함수를 직접 import해서 사용

```
import some_file.name
import some_file.sum
```

32.2 기초 타입

- 기초 타입 자체가 객체
- Int, Double 등의 클래스로 타입이 표현되며 이렇게 선언한 변수는 그 자체로 객체
- 타입 클래스에 정의된 기초 함수와 프로퍼티(property)를 사용 가능
- 코틀린에서 제공하는 타입은 Int, Double, Float, Long, Short, Byte, Char, Boolean, String, Any

32.2.1. Numbers Type

- 문자(Characters)가 Number Type이 아니며, Number Type에 대한 자동 형 변형(implicit conversions for number)을 제공 하지 않음.

```
val a3: Byte=0b00001011
val a4: Int=123
val a5: Int=0x0F
val a6: Long = 10L
val a7: Double=10.0
val a8: Double=123.5e10
val a9: Float=10.0f
```

- 코틀린에서는 자동 형 변형 (implicit conversions for number)을 제공하지 않음.
- 기초 타입의 형 변형은 toXXX( ) 함수를 이용해 명시적으로 진행

32.2 기초 타입

- toByte(): Byte 타입으로 변형
- toShort(): Short 타입으로 변형
- toInt(): Int 타입으로 변형
- toLong(): Long 타입으로 변형
- toFloat(): Float 타입으로 변형
- toDouble(): Double 타입으로 변형
- toChar(): Char 타입으로 변형

32.2.2. String

- 문자열은 str[i]로 접근할 수 있는 문자(characters)의 집합

```
var str: String = "Hello World!"
Log.d("kkang","str[1] : ${str[1]}")
for (c in str) {
Log.d("kkang", "str result : $c")
}
```

- 큰따옴표로 묶이는 문자열을 'escaped string'이라 표현하며, 큰따옴표 세 개로 묶이는 문자열을 'raw string'이라 표현

```
val str="Hello \n World"
val str2="""Hello
World"""
```

32.2 기초 타입

32.2.3. Characters, Booleans

- Character는 자바와 같이 작은따옴표로 표현
- Boolean은 true, false 값을 가지는 변수의 타입

32.2.4. Arrays

- 배열은 Array로 표현되며 Array는 get ( ), set ( )의 함수와 size 변수 등을 포함
- 배열을 만들려면 라이브러리의 arrayOf ( ) 함수를 이용

```
var array1= arrayOf(1,"Hello",false)
Log.d("kkang","${array1.size} .. ${array1.get(2)}")
```

```
val x1: IntArray = intArrayOf(1, 2, 3)
```

```
var array2= arrayOfNulls<Int>(4)
```

32.2 기초 타입

32.2.5. Any

- 변수를 선언할 때 특정 타입으로 지정하지 않고, 어떤 타입의 데이터도 대입

```
fun getLength(obj : Any) : Int {
if(obj is String) {
return obj.length
}
return 0
}
```

```
fun cases(obj : Any): String{
when(obj) {
1 -> return "One"
"Hello" -> return "Greeting"
is Long -> return "Long"
!is String -> return "Not a string"
else -> return "unknown"
}
}
```

32.2 기초 타입

32.2.6. Nullable과 Null 체크

- null이 대입될 가능성이 있는 곳에는 명시적으로 nullable로 표현
- 타입 뒤에 ? 기호를 이용

```
val a: Int = null //error
val b: Int? = null //ok~
```

```
fun parseInt(str: String): Int? {
return str.toIntOrNull()
}
```

32.2.7 _ 타입 체크 : is 연산자

- is 연산자를 이용하여 타입을 체크

```
fun getStringLength(obj: Any): Int? {
if (obj is String) {
return obj.length
}
return null
}
```

32.3 변수 및 함수 선언

32.3.1. 변수 선언

- 변수를 Assign-once (Read-only) variable과 Mutable variable로 구분하여 선언
- Assign-once variable은 일종의 상수 개념으로 한 번 초깃값을 가지면 더 이상 변경이 불가능한 변수
- Mutable variable은 계속 값이 변경될 수 있는 변수
- val은 변수를 Assign-once variable로 선언한 것이며, var은 Mutable variable로 선언할 때 사용

- 변수 선언 및 초기화: val(혹은 var) 변수명 : 타입 = 값

```
val a: Int = 10
val b = 20
val c: Int
c=20
val d = 30
//d=40 //error...
var e=40
e += 5
```

32.3 변수 및 함수 선언

32.3.2. 함수 선언

- fun이라는 예약어를 사용
- fun 함수명(매개변수명 : 타입) : 반환타입 { }
- 함수 선언에서 의미 있는 반환값이 없을 때는 Unit 키워드를 이용
- Unit은 생략 가능

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

32.4 실행 흐름 제어

32.4.1. If 표현식

- if 문은 표현식(expression)
- if 문을 표현식으로 사용한다는 것은 if 문에 특정 값이 명시되고, 그 값이 if의 수행 결과로 자동 반환

```
val txt = if (a > 10) "hello" else "world"
```

- if 문이 표현식으로 이용될 수 있으므로 { } 블록에 정의할 수 있으며 이때는 마지막 문장이 반환
- if를 표현식으로 사용할 때 주의해야 할 점은 else 문이 꼭 있어야 함.

```
val max = if (a < 10) {
print("hello....")
10+20 // a < 10이면 이 값이 반환
} else {
print("world...")
20+20 // a < 10이 아니면 이 값이 반환
}
```

32.4 실행 흐름 제어

32.4.2. When 표현식

```
val a2=1
when (a2) {
1 -> printLog("a2 == 1")
2 -> printLog("a2 == 2")
else -> {
printLog("a2 is neither 1 nor 2")
}
}
```

- when도 if와 마찬가지로 표현식으로 사용할 수 있으며 이때 else 구문을 꼭 사용

```
val result2= when(a2){
1 -> "1...."
2 -> "2...."
else -> {
printLog("else....")
"hello"
}
}
```

- 콤마(,)를 이용하여 값을 여러 개 표현 가능

```
when (a2) {
0, 1 -> printLog("x == 0 or x == 1")
else -> printLog("otherwise")
}
```

32.4 실행 흐름 제어

- when의 값은 상수뿐 아니라 표현식도 가능

```
when (a2) {
aFun(a2) -> printLog("aFun()....")
else -> printLog("else...")
}
```

- in, !in을 이용하여 collection 값을 체크

```
when (a2) {
in 1..10 -> printLog("a2 is in the range")
!in 10..20 -> printLog("a2 is outside the range")
else -> printLog("none of the above")
}
```

- is, !is를 이용하여 타입을 체크

```
fun hasPrefix(x: Any) = when(x) {
is String -> x.startsWith("prefix")
else -> false
}
```

32.4 실행 흐름 제어

32.4.3. For 반복

```
var sum: Int=0
for(i in 1..10) {
sum += i
}
```

- collection 타입(배열 같은) 데이터의 개수만큼 for 문을 반복

```
val list = listOf("Hello", "World", "!")
for(str in list) {
printLog(str)
}
```

- index 값이 대입되어야 할 때 indices를 이용

```
for (i in list.indices) {
printLog(list[i])
}
```

- withIndex( ) 함수를 이용하여 index와 value를 획득

```
for ((index, value) in list.withIndex()) {
printLog("the element at $index is $value")
}
```

32.4 실행 흐름 제어

32.4.4. While 반복

```
var x=10
while (x > 0) {
x--
}
do {
val y = retrieveData()
} while (y != null)
```

Step by Step 32-1 – 코틀린 기초 문법 테스트

코틀린으로 개발하기

1. 모듈 생성
2. 파일 복사
3. TypeTest 클래스 작성
4. ControlTest 클래스 작성
5. MainActivity.kt 작성
6. 모듈 실행