2장 코틀린 기초

신림프로그래머 최범균

- 기본 요소: 함수와 변수
- 클래스와 프로퍼티
- enum과 when
- while과 for 루프
- 익셉션 처리

함수

```
fun max(a: Int, b: Int): Int {
    return if (a > b) a else b // 블록이 본문
}
fun max2(a: Int, b: Int): Int = if (a > b) a else b // 식이 본문
fun max3(a: Int, b: Int) = if (a > b) a else b // 식이 본문이면 리턴 타입 생략 가능(타입추론)
```

- 함수 선언은 fun 키워드로 시작
- fun 다음에 함수 이름이 위치
- 함수 이름 뒤에 괄호 안에 파라미터 목록
  - 파라미터 이름과 타입은 콜론으로 구분
  - 각 파라미터는 콤마로 구분
- 본문
  - 블록이 본문인 함수: 중괄호로 본문을 둘러쌈
  - 식이 본문인 함수: 줄괄호 대신 등호와 식

변수

```
val question = "나는 누구인가?" // 타입 생략, 컴파일러가 초기화 식을 이용 유추

val answer1 = 30

val answer2: Int = 42 // 타입 지정

val answer3: Int // 초기화 식이 없으면 반드시 타입을 명시
answer3 = 42
```

변경 가능 변수와 변경 불가 변수

- val : 변경 불가능한 참조를 저장하는 변수. 일단 초기화하면 재대입이 불가능
  - 정확히 한 번만 초기화 실행 가능
- var : 변경 가능한 참조. 변수 타입은 고정.

```
// 아래 코드 가능: message를 한 번만 초기화한다는 것을 컴파일러가 알 수 있음
val message: String
if (canPerformOperation()) {
    message = "Success"
} else {
    message = "Failed"
}
```

문자열 템플릿

```
val name = "bk"
println("Hello, $name!") // $ 뒤에 변수 사용
println("Hello, ${name}입니다.") // $ 뒤에 중괄호 사용
println("\$name의 값 = $name") // $ 자체는 \$ 탈출문자 사용
println("max(1, 2) = ${max(1, 2)}") // 중괄호 안에서 식 사용
println("args: ${if (args.isEmpty()) "empty" else args[0] }") // 식에서 큰 따옴표 사용
```

클래스

예제:

```kotlin
// Kotlin
class Person(val name: String) // 코틀린 클래스 기본 가시성: public
```

```java
// Java
public class Person {
    private final String name;

    public Person(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
}
```

프로퍼티

프로퍼티를 기본 언어 기능으로 제공하며 자바의 필드와 접근자 메서드를 완전히 대신함

```
class Person(
  val name: String, // 읽기 전용(val) 프로퍼티
  var isMarried: Boolean // 변경 가능(var) 프로퍼티
)
```

```
Person p = Person("Bob", false)
println(p.name)
println(p.isMarried)
p.isMarried = true
```

프로퍼티의 자바 표현

기본적으로 코틀린은 프로퍼티에 대해 다음을 생성

- 읽기 전용 프로퍼티: 비공개 필드와 필드를 읽는 공개 게터 생성
- 쓸 수 있는 프로퍼티: 비공개 필드와 공개 게터, 공개 세터 생성
- 프로퍼티 이름이 is로 시작할 경우
    - 프로퍼티 이름과 동일한 게터 생성: 예, isMarried())
    - is 대신에 set을 사용하는 세터 생성: 예, setMarried()

지원(backing) 필드: 프로퍼티의 값을 저장하기 위한 비공개 필드

커스텀 접근자

```
class Rectanble(val height: Int, val width: Int) {
    val isSquare: Boolean
        get() { // 프로퍼티 게터 선언, 블록 사용
            return height == width;
        }

    val size: Int
        get() = height * width // 식 사용

}
```

- 프로퍼티에 대한 자세한 내용은 4장에서 닫룸

소스코드 구조

- 파일의 맨 앞에 package 문 사용해서 패키지 지정
- 파일의 모든 선언(클래스, 함수, 프로퍼티 등)이 해당 패키지에 속함
- 디렉토리 구조와 패키지 구조가 일치할 필요 없음 (하지만, 패키지별로 디렉토리 구성이 나음)
- 같은 패키지에 속해 있다면 다른 파일에서 임포트 없이 정의한 선언 사용 가능
- 다른 패키지에서 사용하려면 import 키워드로 사용할 선언을 임포트

enum

- enum 키워드를 사용해서 열거타입 지정

```
enum class Color {
    RED, ORANGE, YELLO, BLUE, VIOLET
}
```

- 프로퍼티와 메서드 선언 가능 (메서드 선언시 마지막 열거 값 뒤에 세미콜론 필요)

```
enum class Color(val r: Int, val g: Int, val b: Int) {
  RED(255, 0, 0), ORANGE(255, 165, 0), YELLOW(255, 255, 0),
  BLUE(0, 0, 255), VIOLET(238, 130, 238);

  fun rgb() = (r * 256 + g) * 256 + b
}

println(Color.BLUE.rgb())
```

when

- 자바의 switch와 유사, when은 식
  - 각 분기에 break 필요 없음

```
fun getWarmth(color: Color) =
    when (color) {
        Color.RED -> "warm"
        Color.ORANGE -> "warm"
        Color.YELLOW -> "warm"
        Color.BLUE -> "cold"
        Color.VIOLET -> "cold"
    }
```

- 여러 매치 패턴을 지정할 수 있음

```
fun getWarmth(color: Color) =
    when (color) {
        Color.RED, Color.ORANGE, Color.YELLOW -> "warm"
        Color.BLUE, Color.VIOLET -> "cold"
    }
```

- 모든 분기 식에 만족하지 않으면 else 분기

```
fun getWarmth(color: Color) =
    when (color) {
        Color.RED -> "very warm"
        Color.ORANGE, Color.YELLOW -> "warm"
        else -> "cold"
    }
```

when 식은 객체의 동등성 사용

```
fun mix(c1: Color, c2: Color) =
        when (setOf(c1, c2)) {
            setOf(RED, YELLOW) -> ORANGE
            setOf(YELLOW, BLUE) -> GREEN
            else -> throw Exception("Dirty color")
        }
```

인자 없는 when 식

```
fun mixOpt(c1: Color, c2: Color) =
        when {
            c1 == RED && c2 == YELLOW -> ORANGE
            c1 == YELLOW && c2 == RED -> ORANGE
            c1 == YELLOW && c2 == BLUE -> GREEN
            else -> throw Exception("Dirty color")
        }
```

- when에 인자가 없으려면, 각 분기의 조건이 불리언 결과를 계산하는 식이어야 함

스마트 캐스트

```
fun eval(e: Expr): Int {
    if (e is Num) {
        val n = e as Num // 실제로는 필요 없음
        return n.value
    }
    if (e is Sum) { // 컴파일러가 캐스트 처리
        return eval(e.left) + eval(e.right)
    }
    throw IllegalArgumentException("Unknown exp")
}
```

```
fun eval(e: Expr): Int {
    when (e) {
      is Num -> e.value // 컴파일러가 캐스트 처리
      is Sum -> eval(e.left) + eval(e.right)
      else ->
        throw IllegalArgumentException("Unknown exp")
}
```

- is 연산자로 변수 타입 검사
- 스마트 캐스트: is 검사 뒤 컴파일러가 캐스팅 수행
  - is 검사 뒤에 변수가 바뀌지 않는 경우에 적용

if와 when의 블록

- 블록의 마지막 식이 if와 when의 결과가 됨

```
fun evalWithLogging(e: Expr): Int =
        when (e) {
            is Num -> {
                println("num: ${e.value}")
                e.value // 결과
            }
            is Sum -> {
                val left = evalWithLogging(e.left)
                val right = evalWithLogging(e.right)
                println("sum: $left + $right")
                left + right // 결과
            }
            else ->
                    throw Exception("Unknown Exp")
        }
```

while 루프

- 자바와 동일

```
while (조건) {
    ...
}

do {
    ...
} while (조건)
```

범위와 수열

- 범위(ClosedRange 인터페이스) : 두 값으로 이뤄진 구간
- 수열(Progression): 범위에 속한 값을 일정한 순서로 이터레이션
- 생성 예
  - 1 rangeTo 10 step 2 또는 1..10 step 2
  - 100 downTo 1 step 2
  - 0 until 10

```
for (i in 1..100) {
    println(i)
}
```

맵, 리스트에 대한 이터레이션

```
val binbaryReps = TreeMap<Char, String>()
...
for ( (key, value) in binbaryReps) { // 맵에 대한 이터레이션
    println("$key = $value")
}
```

```
val list = arrayListOf("10", "11", "1001")
for ((idx, ele) in list.withIndex()) {
    println("$idx: $ele")
}
```

in으로 컬렉션이나 범위 원소 검사

in으로 어떤 값이 범위나 콜렉션에 속하는지 검사

```
fun isLetter(c: Char) = c in 'a'..'z' || c in 'A'..'Z'
fun isNotDigit(c: Char) = c !in '0'..'9'
fun recognize(c: Char) = when(c) {
    in '0'..'9' -> "It's digit!"
    in 'a'..'z', in 'A'..'Z' -> "It's a letter"
    else -> "I don't know."
}
println("Kotlin" in "Java".."Scala") // Comparable 구현 클래스
```

```
println("Kotlin" in setOf("Java", "Scaka", "Kotlin"))
```

- in은 contains 함수

익셉션

발생

```
throw IllegalArgumentException("msg")
```

익셉션 처리

```
fun readNumber(reader: BufferedReader): Int? {
    try {
        val line = reader.readLine()
        return Integer.parseInt(line)
    } catch (e: NumberFormatException) {
        return null
    } finally {
      reader.close()
    }
```

try는 식

```
fun readNumber2(reader: BufferedReader): Int? {
    return try {
        Integer.parseInt(reader.readLine())
    } catch (e: NumberFormatException) {
        null
    } finally {
        reader.close()
    }
}
```