3장 함수 정의와 호출

신림프로그래머 최범균

- 콥렉션 생성
- 이름 붙인 인자, 디폴트 파라미터 값, 중위 호출
- 확장 함수와 확장 프로퍼티
- 최상위 및 로컬 함수와 프로퍼티

콜렉션 생성

주요 메서드

- val set = hashSetOf(1, 2, 3) // HashSet
- val list = arrayListOf(1, 2, 3) // ArrayList
- val map = hashMapOf(1 to "one", 2 to "two") // HashMap

자체 콜렉션이 아닌 자바 콜렉션 사용: 자바 코드와 상호작용 용이

추가 확장 확수 제공: 예, list.last(), set.max() 등

이름 붙인 인자

```
fun <T> joinToString(
    collection: Collection<T>,
    separator: String,
    prefix: String,
    postfix: String
): String {
   ...
}
```

이름 없는 인자

```
joinToString(collection, " ", " ", ".")
```

이름 붙인 인자

```
joinToString(collection, separatpr = "",
        prefix = " ", postfix = ".")
```

- 인자 중 하나라도 이름을 명시하면, 그 뒤에 오는 모든 인자는 이름을 명시해야 함

디폴트 파라미터 값

```
fun <T> joinToString(
    collection: Collection<T>,
    separator: String = ", ",
    prefix: String = "",
    postfix: String = ""
): String {
    ...
}
```

```
joinToString(collection, ", ", "", "")
joinToString(collection)
joinToString(collection, "; ")

joinToString(collection,
             postfix = ";", prefix = "# ")
```

- 이름 없는 인자: 일부를 생략하면 뒷부분의 인자들 생략
- 이름 붙인 인자: 지정하고 싶은 인자를 이름을 붙여 지정

@JvmOverloads: 맨 마지막 파라미터로부터 파라미터를 하니씩 생략한 오버로딩한 자바 메서드 추가

최상위 함수

```kotlin
// join.kt
package chap03

// 특정 클래스나 오브젝트에 속하지 않은
// 최상위 함수
fun <T> joinToString(
        collection: Collection<T>,
        separator: String = ", ",
        prefix: String = "",
        postfix: String = ""
): String { ... }
```

```kotlin
import chap03.*

fun main(args: Array<String>) {
    // 최상위 함수 사용
    joinToString(arrayListOf(3, 4, 5),
                 prefix = "#")
}
```

- 최상위 함수는 자바의 정적 메서드로 변환
  - 소스파일 이름을 자바 클래스 이름으로 사용
  - JoinKT.joinToString(....)

최상위 프로퍼티

```kotlin
// join.kt
var opCount = 0

fun performOperation() {
    opCount++
}

// 자바의 getter
val WINDOW_LINE_SEPARATOR = "\r\n" // JoinKt.getWINDOW_LINE_SEPARATOR()

// const 수식어: 자바의 public static final
const val UNIX_LINE_SEPARATOR = "\n" // JoinKt.UNIX_LINE_SEPARATOR
```

- const는 기본 타입과 String에만 가능

확장 함수

어떤 클래스의 멤버 메서드인 것처럼 호출할 수 있지만, 그 클래스의 밖에 선언된 함수

```
fun String.lastChar(): Char = this.get(this.length - 1)

    ^수신객체타입                    ^수신객체

println("Kotlin".lastChar())
```

- 함수 이름 앞에 확장할 클래스 이름을 붙여서 정의
  - 수신 객체 타입: 확장을 정의할 클래스의 타입
  - 수신 객체: 그 클래스에 속한 인스턴스
- 확장 함수 안에서 수신 객체의 메서드나 프로퍼티 사용 가능
  - private 멤버나 protected 멤버만 접근 불가
- 확장 함수보다 멤버 함수가 우선함

확장 함수와 임포트

- 확장 함수를 사용하려면 임포트해야 함

```
import strings.lastchar

val c = "Kotlin".lastChar()
```

- as로 확장 함수를 다른 이름으로 임포트 가능(이름이 충돌할 경우 주로 사용)

```
import strings.lastChar as last

val c = "Kotlin".last()
```

확장 함수는 실제로 정적 메서드 호출

```kotlin
// strings.kt
fun Collecion<String>.join(separator: String, prefix: String, postfix: String)

listOf("1", "2", "3").join(",", "#", "")
```

```kotlin
List<String> strs = ...
StringsKt.join(strs, ",", "#", ""); // 수신 객체가 첫 번째 인자
```

확장 함수는 변수의 정적 타입 기준으로 결정 (오버라이딩 안 됨)

```
fun View.showOff() = ...
fun Button.showOff() = ...

val view:View = Button()
view.showOff() // view 변수의 정적 타입인 View 기준으로 확장함수(View.showOff()) 결정
```

확장 프로퍼티

- 기존 클래스 객체에 대한 프로퍼티 형식 구문으로 사용할 수 있는 API 추가

```
val String.first: Char
    get() = get(0)

var StringBuilder.lastChar: Char
    get() = get(length - 1)
    set(ch: Char) {
        this.setCharAt(length - 1, ch)
    }
```

- 상태를 저장할 방법이 없지만, 프로퍼티 문법으로 더 짧게 코드를 작성할 수 있는 편리함
- 지원 필드가 없으므로
  - 최소한 게터는 정의(기본 게터 구현 제공 불가)
  - 초기화 코드 쓸 수 없음

가변 길이 인자

```
fun listOf<T>(vararg values: T): List<T> { ... }
```

```
fun main(args: Array<String>) {
    val list = listOf(1, 2, 3, 4)

    // 배열은 펼침연산자(*)로 전달
    val arglist = listOf("args: ", *args)
    // listOf(args) --> List<Array<String>>이 됨
}
```

중위 호출 (infix 수식어)

```
infix fun Any.to(other: Any) = Pair(this, other)

1.to("One")

1 to "One" // 중위 호출
```

- infix 수식어로 중위 호출 함수 지정

분해 선언(destructing declaration)

```
val (number, name) = Pair(1, "one")
val (_, name) = Pair(1, "one") // 사용하지 않는 변수

for ((index, element) in collection.withIndex()) { // IndexedValue<Int>

}

map.mapValues { (key, value) -> "$value!" } // 람다에서의 분해 선언
map.mapValues { (_, value) -> "$value!" }
```

- 분해 선언은 실제로 componentN() 함수 호출을 사용
  - componentN 함수에는 operator 수식어를 붙임

문자열 분리

```
// 문자열 구분자
"12.345-6.A".split(".")
"12.345-6.A".split(".", "-")
"12.345-6.A".split('.', '-')

// 구분자를 정규표현식으로 지정
"12.345-6.A".split("\\.|-".toRegex())

path.substringBeforeLast("/")
path.substringAfterLast("/")
```

3중 따옴표 문자열

```
"""(.+)/(.+)\.(.+)""" // 역슬래시 등 이스케이프 필요 없음

"""$name $('$')""" // 문자열 템플릿 가능, $ 문자 자체는 문자열 템플릿 안에 표시

val kotlinLogo = """|  //
                   .| //
                   .|/ \""".trimMargin(".") // .과 그 앞 문자를 제거
```

- 3중 따옴표 문자열에서는 역슬래시 등 어떤 문자도 이스케이프할 필요가 없음
- 문자열 템플릿 가능

정규식과 분해 선언

```
val regex = """(.+)/(.+)\.(.+)""".toRegex()
val matchResult: MatchResult? = regex.matchEntire("/aaa/bbb/cc.jpg")
if (matchResult != null) {
    // MatchResult.Destructured는 component1() ~ component10() 정의
    val (directory, filename, extenstion) = matchResult.destructured
}
```

로컬 함수

```
fun saveUser(user: User) {
    // 로컬 함수
    fun validate(value: String, fieldName: String) {
        if (value.isEmpty()) {
            // user.id : 자신이 속한 바깥 함수의 파라미터와 변수 사용할 수 있음
            throw IllegalArgumentException("Can't save user ${user.id}: " +
                    "empty $fieldName")
        }
    }

    validate(user.name, "Name")
    validate(user.address, "Address")
}
```