Others

코틀린 기초강좌

강사: 배정만

학습요약

- 학습목표: 코틀린의 다양한 기능 학습
- 핵심키워드: 위임, Lazy 프로퍼티, 옵저버블, Null-safety, Operation overloading, Infix, Extensions
- 학습내용: 코틀린의 다양한 기능과 개념에 대해 배웁니다.

델리게이션
(Delegation)

필요할때마다 구현하기 번거로운 기능들을
위임을 통해 구현하는 방식
데이터 저장, 로그 남기기 등

델리게이션 (위임)

```
class Example {
    var p: String by Delegate()
}

class Delegate {
    operator fun getValue(thisRef: Any?, property: KProperty<*>): String {
        return "$thisRef 에 할당된 Example 객체는 '${property.name}'를 Delegate 클래스에 위임했습니다."
    }

    operator fun setValue(thisRef: Any?, property: KProperty<*>, value: String) {
        println("Example 클래스가 생성되어 ${thisRef}에 할당되고, $value 타입의 객체가 멤버변수 '${property.name}'에 할당되었습니다.")
    }
}
```

프로퍼티 위임

- 자주쓰는 프로퍼티 속성관련 델리게이션 라이브러리
  - lazy 프로퍼티
    - 값은 첫 번째 접근시 계산
  - Observable 프로퍼티
    - 리스너는이 프로퍼티 변경 사항에 대한 알림을 받음
  - 프로퍼티를 각각 저장하는 대신 맵 안에 저장

Lazy

```
val lazyValue: String by lazy {
    println("computed!")
    "Hello"
}

fun main() {
    println(lazyValue)
    println(lazyValue)
}
```

```
computed!
Hello
Hello
```

- 첫 번째 접근시 lazy 람다 함수 호출
- 이후에는 값으로 접근 (synchronized)
- LazyThreadSafetyMode.PUBLICATION
- LazyThreadSafetyMode.NONE

Observable

```
import kotlin.properties.Delegates

class User {
    var name: String by Delegates.observable("<no name>") {
        prop, old, new ->
        println("$old -> $new")
    }
}

fun main() {
    val user = User()
    user.name = "first"
    user.name = "second"
}
```

```
<no name> -> first
first -> second
```

- 옵저버 함수는 이전 값과 이후 값을 인수로 받음
- 변수 지정 직후에 호출됨
- 멀티쓰레드//Rx 프로그래밍에서 유용한 함수

Null Safety

```
var a: String = "abc"
a = null // compilation error

var b: String? = "abc"
b = null // ok
print(b)
val l = b.length // error: variable 'b' can be null
```

- Kotlin의 타입 시스템은 코드에서 null 참조의 위험을 제거 함
- 다음은 코틀린 NullException 예
  - NullPointerException()을 명시적으로 호출
    - not-null 단정 연산자 (!!)
  - 초기화 과정에서 발생하는 Null문제
    - 생성자에서 초기화되지 않은 this 사용
    - 자식 클래스에서 초기화하지 않은 오픈 멤버 변수를 부모 클래스에서 호출하는 문제
  - 자바 상호 운용성
    - Java 코드가 Kotlin MutableList<String>에 null을 추가 할 수 있음
      MutableList<String?>이 작업에 사용되어야 함

연산자 오버로딩
Operator overloading

```
data class Point(val x: Int, val y: Int)

operator fun Point.unaryMinus() = Point(-x, -y)

val point = Point(10, 20)

fun main() {
    println(-point)
}
```

```
Point(x=-10, y=-20)
```

- operator 변경자(modifier)로 명시적 선언 후 오버로딩
-

예시> 접두사 연산자

표현식	실행함수
+a	a.unaryPlus()
-a	a.unaryMinus()
!a	a.not()

- a의 타입을 결정
- operator 변경자를 가진 실행함수를 찾음
- 함수가 없거나 모호한 경우 컴파일 오류
- 함수가 존재하고 그 리턴 타입이 T이라면, 표현식 +a는 타입 T 가 됨