💙

Lec 05

코틀린에서 조건문 을 다루는 방법

0. 복습

- 키워드

1. var (변수 이름) = (값) / val (변수 이름) = (값) ⇒ 타입 추론

   (변수 이름) : (변수 타입) ⇒ 타입 명시

   (변수 이름) : (변수 타입)? ⇒ nullable(null가능)

2. 객체 생성시 new를 사용하지 않음

   ex) val person = Person( )

3. Safe Call(nullable한 변수에 사용 가능)

   (변수명)**?.**( )

4. Elvis 연산자(Safe Call과 같이 사용)

   (변수명)?:(값) → 변수가 null일 때 (값) 반환

5. null 아님 단언!!

   (변수명)!! → 변수에 null이 들어왔을 때, NPE 발생

6. 자동 타입 변환 X

   → 타입 변환 방법 : (변수) = (값).**to(타입)**

7. 강제 타입 변환

   Java ⇒ (변수) = (타입)(값);

   Kotlin ⇒ (변수) = (값) **as 타입**

8. 타입캐스팅

   Java → instanceof / Kotlin → is

타입 캐스팅	value **is** type	value **!is** type	value **as** type	value as? type
value가 type이면	true	false	type으로 타입 캐스팅	type으로 타입캐스팅

타입 캐스팅	value **is** type	value **!is** type	value **as** type	value as? type
value가 type이 아니면	false	true	예외 발생	null
value가 null이면	X	X	X	null

9. 특이한 타입

Java	Kotlin
Object	Any
void	Unit
x	Nothing

```
fun 함수명 (매개변수 : 타입) : (특이한 타입) {
        // 위와 같은 형식으로 사용
}
```

- Object(java), Any(kotlin) ⇒ 최상위 클래스
- Any : 생략 가능
- Nothing : 함수가 정상적으로 끝나지 않음을 표현

10. ${변수/식/값} / $값

    ⇒ 문자열 출력

    ⇒ 보편적으로 ${변수/식/값} 사용

11. “””(여러 줄의 문자열)”””
12. 객체 비교할 때

    → **compareTo( )** 자동 호출

13. 동일성 / 동등성

	Java	Kotlin
동일성 비교	==	===
동등성 비교	equals	==

14. in / !in
    - 컬렉션이나 범위에 포함되어 있다 / 포함되어 있지 않다

---

1. if문

- Java와 차이 없음

```
if(조건) {

}
```

- Java와 Kotlin 비교
  - Java

```
private String getPassOrFail(int score) {
    if (score >= 50) {
        return "P";
    } else {
        return "F";
    }
}
```

- Kotlin

```
fun getPassOrFail(score : Int) : String {
    if (score >= 50) {
        return "P"
    } else {
        return "F"
    }
}
```

2. Expression & Statement

- Statement, Expression의 예시

```
int score = 30 + 40;
// 70이라는 하나의 결과가 나옴
// Expression이면서 Statement
```

```
String grade = if(score >= 50) {
    "P";
} else {
        "F";
}
// if문은 하나의 값으로 취급하지 않음 -> 에러 발생
// Statement
```

```
String grade = score >= 50 ? "P" : "F";
// 3항연산자는 하나의 값으로 취급
// Expression이면서 Statement
```

- if - else if - else
  - Java와 동일

- 비교

```
// Java
private String getGrade(int score) {
    if (score >= 90) {
        return "A";
    } else if (score >= 80){
        return "B";
    } else if (score >= 70){
        return "C";
    } else {
        return "D";
    }
}
```

```
// Kotlin
fun getGrade(score: Int): String {
    return if (score >= 90) {
        "A"
    } else if (score >= 80) {
        "B"
    } else if (score >= 70) {
        "C"
    } else {
        "D"
    }
}
```

- 어떠한 값이 특정 범위에 포함되어 있는지, 포함되어 있지 않은지
  - in a..b

```
if (score in 0..100) {

}
// in 뒤에 오는 값의 범위 안에 변수가 포함되는지 확인
// a..b : a와 b 사이
```

---

3. switch와 when

▼ Java와 Kotlin 비교

- Java

```
private String getGradeWithSwitch(int score) {
    switch (score / 10) {
        case 9:
            return "A";
        case 8:
            return "B";
        case 7:
            return "C";
        default :
            return "D";
    }
}
```

- Kotlin

```
// 방법 1
fun getGradeWithWhen(score: Int): String{
    return when (score / 10) {
        9 -> "A"
        8 -> "B"
        7 -> "C"
        else -> "D"
    }
}
```

```
// 방법 2
fun getGradeWithWhen(score: Int): String{
    return when (score) {
        in 90..99 -> "A"
        in 80..89 -> "B"
        in 70..79 -> "C"
        else -> "D"
    }
}
```

- when의 형태

```
when (값){
    조건부 -> 어떠한 구문
    조건부 -> 어떠한 구문
```

```
    else -> 어떠한 구문
}
```

- 조건부
  - 어떠한 expression이라도 들어갈 수 있음
  - 여러개의 조건을 동시에 검사할 수 있음(, 로 구분)
- 값
  - 값이 없을 수도 있음 → early return 처럼 동작
- when은 Enum Class 혹은 Sealed Class와 함께 사용할 때 좋음

📌 **총정리**
- if / if-else / if-else if-else 모두 Java와 문법 동일
- 단, Kotlin에서는 Expression으로 취급됨
- 때문에 Kotlin에는 삼항 연산자가 없음
- Java의 switch ⇒ Kotlin의 when