Chapter 02.코틀린 기초 - 04.흐름제어

흐름제어

코틀린에서 제공하는 흐름제어 문법

- if...else 식
- when 식
- for 루프
- While 루프

1. if...else

- if...else 문법

```
if (조건) {
  // if의 조건이 참인 경우 실행
} else {
  // if의 조건이 거짓인 경우 실행
}
```

- if else 사용

```
val job = "Software Engineer"

if (job == "Software Engineer") {
    println("개발자임")
} else {
    println("개발자아님")
}
```

- 코틀린의 if...else는 표현식이다

```
val age : Int = 18

val str = if (age > 18) {
  "성인"
} else {
  "아이"
}

println(str)
```

- `표현식` 또는 `식`은 그 자체로 값을 만들어 낼 수 있는 문법을 말한다
- `구문` 또는 `문`은 그 자체로 값을 만들지 못한다

- 자바의 if...else는 구문이므로 값을 반환할 수 없다

```
int age = 18;

String str = "";
if (age > 18) {
  str = "성인";
} else {
  str = "아이";
}

System.out.println(str);
```

- 코틀린은 삼항 연산자가 없다. if..else가 표현식이므로 불필요하다

```
val a = 1
val b = 2

val c = if (b > a) b else a
```

---

2. when

- when 문법

```
when (대상변수) {
  조건 -> 참인 경우 실행
  조건 -> 참인 경우 실행
  조건 -> 참인 경우 실행
  else -> 참인 조건이 없을 경우 실행
}
```

- 자바의 switch문과 유사하다

```
int day = 2;
String result = "";
switch (day) {
    case 1:
        result = "월요일";
        break;
    case 2:
        result = "화요일";
        break;
    case 3:
        result = "수요일";
        break;
    case 4:
        result = "목요일";
        break;
    case 5:
        result = "금요일";
        break;
```

```
    case 6:
        result = "토요일";
        break;
    case 7:
        result = "일요일";
        break;
    default:
        result = "에러";
        break;
}

System.out.println(result);
```

- 기존 자바의 switch문은 말그대로 표현식이 아니어서 값을 반환하지 못했다 (최신 자바는 switch식 지원)
- 실수로 break를 빼먹어서 버그가 발생하는 경우가 잦았다

- 위의 자바 코드를 코틀린의 when식으로 변환한 코드

```
val day = 2

val result = when (day) {
  1 -> "월요일"
  2 -> "화요일"
  3 -> "수요일"
  4 -> "목요일"
  5 -> "금요일"
  6 -> "토요일"
  7 -> "일요일"
  else -> "에러"
}

println(result)
// 화요일
```

- else를 생략할 수 있다

```
enum class Color {
    RED, GREEN, BLUE
}

fun getColor() = Color.RED
```

```
fun main() {

    when (getColor()) {
        Color.RED -> println("red")
        Color.GREEN -> println("green")
        Color.BLUE -> println("blue")
        // 모든 경우의 수를 커버하므로 'else' 필요 없음
    }

    when (getColor()) {
        Color.RED -> println("red")
        else -> println("not red")
        //GREEN, BLUE의 경우가 남아 있으므로 else가 필요
    }
}
```

- 여러개의 조건을 콤마로 구분해 한줄에서 정의할 수 있다

```
fun getNumber() = 2

fun main() {

    when (getNumber()) {
        0, 1 -> print("number == 0 or number == 1")
        else -> print("0과 1이 아닌 경우")
    }
    // "0과 1이 아닌 경우"
}
```

---

3. for loop

- 기본적으로 자바의 `foreach`와 유사
- 범위 연산자 `..` 를 사용해 for loop 돌리기

```
for (i in 0..3) {
  println(i)
}
// 0
// 1
// 2
// 3
```

- `until` 을 사용해 반복한다 (뒤에 온 숫자는 포함하지 않는다)

```
for (i in 0 until 3) {
    println(i)
}
// 0
// 1
// 2
```

- `step` 에 들어온 값 만큼 증가시킨다

```
for (i in 0..6 step 2) {
    println(i)
}
// 0
// 2
// 4
// 6
```

- `downTo`를 사용해 반복하면서 값을 감소시킨다

```
for (i in 3 downTo 1) {
    println(i)
}
// 3
// 2
// 1
```

- 전달받은 배열을 반복

```
val numbers = arrayOf(1, 2, 3)

for (i in numbers) {
    println(i)
}
// 1
// 2
// 3
```

---

4. while loop

- 자바의 while문과 동일
- 조건을 확인하고 참이면 코드 블록을 실행한 후 다시 조건을 확인

```
var x = 5

while (x > 0) {
    println(x)
    x--
}
// 5
// 4
// 3
// 2
// 1
```