011주차14그루비의제어문

그루비의 제어문

Table of Contents

분기문

if문

다음은 그루비에서 if문 사용하기 이다.
특별한 부분은 없다. assert true로 실행될 코드 블록을 나타냈고, 실행되지 않는 블록은 assert false 로 표시했다.

```
// if문 사용하기

if( true ) assert true
else       assert false

if ( 1) {
    assert true
}else {
    assert false
}

if ( 'non-empty' ) assert true
else if (['x'])       assert false
else                  assert false

if ( 0 )    assert false
else if([]) assert false
else        assert true
```

?조건 연산자
간단한 한 줄짜리 검사에 사용하는 조건연산자 ?:를 제공한다.
다만 그루비에서는 모든 것이 객체이기 때문에 가운데와 마지막 표션식이 완전히 다른 자료형이어도 괜찮다.

```
// 조건 연산자
def result = (1==1) ? 'ok' : 'failed'
assert result == 'ok'

result = 'some string' ? 10 : ['x']
assert result == 10
```

switch

자바의 switch 문은 굉장히 제한적이다. 오직 int만 쓸 수 있고 byte, char, short를 사용하면 int로 자동으로 변환된다.

그러나 그루비에서는 자바의 switch와 형태가 똑같으나, 자바의 상수를 이용한 case와 달리, 주어진 대상이 여러 case를 활용이 가능하다.

```
switch ( 10 ){
    case 0  : assert false ; break // 기존 int형
    case 0..9  : assert false ; break // 범위
    case [8,9,11]  : assert false ; break // 콜렉션
```

```
    case Float  : assert false ; break // 자료형으로 분류
    case {it%3==0}  : assert false ; break //  클로저로 분류
    case ~/../  : assert true ; break // 정규 표현식으로 분류.
    default : assert false ; break
}
```

반복문

while

while문은 자바와 비슷하게 동작한다. 단 한 가지 차이점이 있다면이미 알고 있듯이 그루비의 조건식이 더 강력하는 점이다.
간단하게 요약해서 말하면 while문은 주어진 조건을 테스트해서 참이면 반복문의 본체(body)를 실행한다. 그리고 조건을 다시 테스트해서 이과정을 반복한다. 조건이 참이 아닐때에만 while문을 빠져나간다.

```
// whiel문 예제

def list = [1,2,3]
while ( list ) {
    list.remove(0)
}
assert list == []

while (list.size() < 3) list << list.size() +1
assert list == [1,2,3]
```

for 루프문

그루비는 for문을 간단한 형태로 만들기 했기때문에 자바와 그루비 간의 가장 큰자이점이 생겨났다.
그루비의 for문은 다음과 같다

```
for ( variable in iterable ) { body }
```

그루비의 for 문은 iterable의 요소들에 되풀이해서 실행된다. iterable에는 범위, 컬렉션, 맵, 배열, iterator, 열거형(enumeration)등이 사용된다.

전체적인 예제를 통해 보자

```
for (int i = 0; i < 5; i++) {
}

// iterate 범위 루프문
def x = 0
for ( i in 0..9 ) {
    x += i
}
assert x == 45

// iterate 배열 루프문
x = 0
for ( i in [0, 1, 2, 3, 4] ) {
    x += i
}
assert x == 10

// iterate Array 루프문
array = (0..4).toArray()
x = 0
for ( i in array ) {
    x += i
}
assert x == 10

// iterate Map 루프문
def map = ['abc':1, 'def':2, 'xyz':3]
x = 0
for ( e in map ) {
    x += e.value
}
assert x == 6

// iterate Map의 values 값의 루프문
x = 0
for ( v in map.values() ) {
    x += v
}
assert x == 6

// iterate 문자열 루프문
def text = "abc"
def list = []
for (c in text) {
    list.add(c)
}
assert list == ["a", "b", "c"]
```

블록이나 메서드에서 나가기

평범하게 나가기 - return, break, continue

return, break, continue 문은 대부분 자바와 비슷하게 동작한다. 한가지 차이점은 메서드나 클로저의 마지막 문장에서 return 키워드를 생략할 수 있다는 점이다. 생략이 될때는 마지막 표현식의 값이 리턴된다. 리턴이 void로 명시된 메서드는 리턴하지 않으나 클로저일경우 무조건 어떤 값을 리턴한다.(클로저에서 void 일경우 null을 리턴한다.)

```
// 단순한 break와 continue

def a = 1
while (true) {
    a++
    break
}
assert a == 2

for ( i in 0..10 ){
    if( i==0) continue
    a++
    if( i >0 ) break
}
assert a==3
```

익셉션 - throw/try-catch-finally

익셉션은 자바와 문법도 똑같고 처리하는 방식도 같다. 자바처럼 try-catch-finally를 모두 명시하거나 try-catch또는 try-finally만 사용해도 된다. 유일하게 자바와 다른점은 메서드 선언에서 익셉션을 생략할수 있다.(필수 익셉션도 생략가능)

```
// throw, try, catch, finally

def myMethod() {
    throw new IllegalArgumentException()
}

def log = []
try {
    myMethod()
} catch ( Exception e ) {
    log << e.toString()
} finally {
    log << 'finally'
}
assert log.size() == 2
```