5월11일 복습

강사 - Innova Lee(이상훈)

학생- jonghyeon Yoo(유종현)

if문

```java
//if문의 기본
int num = 3, num1 = 4;
//    * 조건에는 다음과 같은 케이스들이 존재한다.
//      >, <, >=, <=, ==, !=
//      A > B: A가 B 보다 크면 참(1) 아니면 거짓(0)
//      A < B: A가 B 보다 작으면 참(1) 아니면 거짓(0)
//      A >= B: A가 B 보다 크거나 같으면 참(1) 아니면 거짓(0)
//      A <= B: A가 B 보다 작거나 같으면 참(1) 아니면 거짓(0)
//      A == B: A와 B가 같다면 참(1) 아니면 거짓(0)
//      A != B: A와 B가 같지 않다면 참(1) 아니면 거짓(0)
// 3. 조건이 만족하였을 경우 동작시킬 코드를 중괄호 내부에 작성한다.
if (num < num1) {
    System.out.printf("참 %d < %d\n", num, num1);
}else {
    System.out.printf("거짓 %d > %d\n", num, num1);
}
```

```java
//if문 if -> else if -> else
System.out.println("두개의 숫자를 입력받아 비교해보기");
Scanner scan = new Scanner(System.in);

System.out.print("첫 번째 숫자 입력 : ");
int num2 = scan.nextInt();
System.out.print("두 번째 숫자 입력 : ");
int num3 = scan.nextInt();
// 문제: 같은 숫자를 넣으면 원하는 동작을 하지 않는다.
// 원인: 현재 조건은 둘 중 하나가 무조건 큰 경우만 보고 있음
// 즉 두 숫자가 같은 경우를 전혀 고려하고 있지 않다.
// 그러므로 이 부분에 대한 대응이 필요하다!

// ex) num1 = 7, num2 = 7인 경우
//      num1 > num2는 조건이 만족되지 않는다.
//      거짓이므로 무조건 else로 가서 else 쪽 print가 동작해 잘못된 결과를 얻산하게
if (num2 < num3) {
    System.out.printf("출력 : %d < %d\n", num2, num3);
} else if (num2 > num3){ //위의 if조건 만족이 안됐다면 아래의 if 조건을 실행
    System.out.printf("출력 : %d > %d\n", num2, num3);
} else { // 위의 if 조건이 만족되지 않았으면 마지막 else 조건이 성립.
    System.out.printf("출력 :%d 는 %d 값이 동일하다.\n", num2, num3);
}
```

if문 키보드/ string문자열 입력

```java
//if문 키보드 입력
Scanner scan2 = new Scanner(System.in);
// 1. Scanner 는 클래스라는 개념

// 2. new는 Heap(힙) 메모리에 동적할당하는 개념으로 클래스 객체를 만들때 활용

// 3. System.in 시스템 입력이라는 의미로 사용
//    키보드나 마우스 같은 입력 장치라고 생각하면 되는데 키보드라고 생각하면 됨.
System.out.print("아무 숫자나 입력해보세요: ");
// scan2 <<<--- 키보드에 대한 제어를 수행
// scan.nextInt() scan2 변수 내부에 있는 nextInt()를 실행한다.
// 키보드 입력으로 들어온 값을 숫자로 바꿈.

// ex) 시나리오: 키보드에 숫자 35를 입력하고 엔터를 침
// scan.nextInt()에서 키보드 입력 35를 받아서 int 형 35를 만듦
// 그리고 변수 num에 변환된 int형 숫자 35를 저장한다.

// 결론: scan.nextInt()는 그냥 키보드 입력을 숫자로 만들어준다.
int num4 = scan.nextInt();

System.out.println("당신이 입력한 숫자는 = " + num4);
```

```java
// string 문자열 입력
System.out.println("문장도 입력이 된다는데요 ?");
Scanner scan3 = new Scanner(System.in);


System.out.print("아무 문장이나 입력하시오: ");
// String은 클래스다.
// 그러나 우선은 문장을 표현할 수 있는 데이터타입이라고 기억하도록 하자!
// 또한 scan.nextLine()은 문장 입력을 받을 수 있다.
// 결국 키보드로 입력된 문장을 str 변수에 저장하는 역할을 한다.
// 결론: 문장 입력을 받고 싶다면 scan.nextLine()을 사용하자!
String str = scan.nextLine();
System.out.println("당신이 입력한 문장은 = " + str);
```

while문 (전위/후위 연산)

```java
while(true) { // 가장 기본적인 while 루프의 형태
    System.out.println("안녕 숫가락으로 n번 맞으면 죽을수도 있데");
}
//while 무한루프

// while 루프를 작성하는 방법
// 1. 먼저 while을 적고 소괄호를 열고 닫고 중괄호를 열고 닫는다.
// 2. 소괄호 내부에는 if 문과 마찬가지로 조건을 적는다.
// 3. 조건이 만족하는 동안 동작시킬 코드를 중괄호 내부에 적는다.

// 현재 예에서는 조건이 항상 true(참) 이므로
// 무한 반복을 하게 됨.

// false를 직접 집어넣을 경우: unreachable statement가 나오면서
// 루프 내부에 도달할 수 없음을 알리는 에러 메시지가 나타남

// 모든 제어문(if, for, while 등)에서 사용하는 조건은
// 반드시 결과가 true 혹은 false 로 나타나게 됨
```

```java
int i = 0;
System.out.println("i++은 다음 라인에서 연산이 진행됩니다.");
// i++에서 ++은 더하기 1과 동일한 개념
// i++ < 10은 i 값을 1 더하고 10하고 비교하라는 뜻으로 볼 수 있음
// i < 10에 대한 조건 검사를 먼저하고
// while 문의 중괄호(본문)를 실행하기 직전에 값이 증가함.
// 이와 같이 다음 라인에서 연산되는 것을 후위 연산이라고 부른다.
while(i++ < 10) { // 전위연산자
    // 여기서 출력을 하고 다시 while의 조건을 검사하러 올라간다.
    System.out.println(i);
}

System.out.println("++i는 해당 라인에서 바로 연산이 진행됩니다.");
i = 0;
// 이와 같이 해당 라인에서 바로 연산되는 것을 전위 연산이라고 부른다.
while(++i < 10) { //후위연산
    System.out.println(i);
}
System.out.println("난 0부터 출력하고 싶다면");
i = 0;
while(i < 10) {
    System.out.println(i++);
}
```

관계 연산자

```java
int num1 = 3, num2 = 7;
int num3 = 21, num4 = 24;

// 관계 연산자 AND (&&) 단축키 shift + 7
// AND 의 특성 ===> 양쪽 조건이 모두 참인 경우에만 참
//                  한쪽이라도 거짓이 있다면 거짓으로 판정
if ((num3 % num1 == 0) && (num3 % num2 == 0)) {
    System.out.printf("%d는 %d의 배수이며 %d의 배수이다.\n", num3, num1, num2);
}

// 표기법 ===> 엔터 위에 원 표시가 있다.
//            Shift + 원 표시를 누르면 파이프 기호가 생성된다.
// 관계 연산자 OR (||)
// OR 의 특성 ===> 양쪽 중 하나라도 참이라면 참
//                 양쪽 모두가 거짓인 경우에만 거짓으로 판정
if ((num4 % num1 == 0) || (num4 % num2 == 0)) {
    System.out.printf("%d는 %d의 배수 혹은 %d의 배수이다.\n", num4, num1, num2);
}

// 관계 연산자 NOT (!)
// 어떤 상황이던 어떤 조건이던 무조건 그 반대의 역할을 수행함
// 부정의 부정은 긍정, 긍정의 부정은 부정
if (!(num4 % num1 == 0) || (num4 % num2 == 0)) {
    System.out.printf("%d는 %d의 배수 혹은 %d의 배수이다.\n", num4, num1, num2);
} else {
    System.out.println("출력이 안되네 ;;;");
}
```

```
21는 3의 배수이며 7의 배수이다.
24는 3의 배수 혹은 7의 배수이다.
출력이 안되네 ;;;

Process finished with exit code 0
```

관계연산자

```java
public class Test3 {
    public static void main(String[] args) {
        int num1 = 3, num2 = 4;

        // AND 연산과 OR 연산의 특성 때문에 발생한 논리적 오류!
        // AND는 하나라도 거짓이면 거짓(false)
        // 어 ? 이미 앞의 케이스가 거짓인데 뒤의 내용을 확인할 필요가 있을까 ?
        // 그래서 뒤의 ++ 코드를 실행하지 않음
        if ((num1 % 3 == 1) && (num2++ % 5 == 0)) {
            System.out.println("이 조건은 실행되지 않습니다.");
        }

        // OR 는 하나라도 참이면 참(true)
        // 어 ? 이미 맨 앞의 케이스가 참인데 뒤에거 볼 필요가 있어 ? 어차피 참인데 ???
        // 그래서 뒤의 ++ 코드를 실행하지 않음
        if ((num1 % 3 == 0) || (num2++ % 6 == 0)) {
            System.out.println("이 조건은 실행됩니다.");
        }

        System.out.printf("num1 = %d, num2 = %d\n", num1, num2);
    }
}
```

```
이 조건은 실행됩니다.
num1 = 3, num2 = 4

Process finished with exit code 0
```

```java
// Q: 위와 같은 케이스를 어디에서 많이 사용할 수 있을까요 ?
// A:
// 전체 데이터가 1000개 있다고 가정합니다.
// 케이스 A에 해당하는 데이터는 150개 있습니다.
// 케이스 B에 해당하는 데이터는 600개 있습니다.
// 우리가 찾아야 하는 것은 케이스 A와 케이스 B를 동시에 만족해야함
// 아무것도 아닌 케이스는 250개
// 이런 상황에서 가장 효율적인 코드를 작성한다면 어떻게 작성해야할까요 ?

// if (case A && case B): 1번
// 1000번 검사중 case A를 만족하는 것은 150개가 나옴
// 그러므로 뒤의 case B에 대한 검사를 150번 추가로 함

// if (case B && case A): 2번
// 1000번 검사중 case B를 만족하는 것은 600개가 나옴
// 그러므로 뒤의 case A에 대한 검사 또한 600번 해야함

// 결론: 관계 연산자 AND 는 if 연산에서 사용할 때
//       전체에서 적용되는 횟수가 가장 적은 녀석이 맨 앞에 배치될 때 가장 큰 효율을 얻을 수 있다.
```