5. 제어문

제어문이란 주어진 조건의 평가 결과에 따라서 프로그램의 수행 순서를 제어하거나 문장들의 수행 횟수를 조정하는 문장으로서 조건 제어문으로는 if 와 switch 문을, 반복 제어문으로는 for, while 그리고 do-while 문을, 흐름 제어문으로는 break 와 continue 문을 사용할 수 있다.

5.1 조건 제어문

어떠한 조건을 평가하여 참/거짓의 여부에 따라 수행 문장을 결정하여 처리하는 기능을 지원하는 if 문과 처리 결과 값이 어떠한 값인지의 여부에 따라 수행 문장을 결정하여 처리하는 기능을 지원하는 switch 문이 있다.

■ if 문

if 문은 어떠한 조건을 평가하여 참/거짓의 여부에 따라 수행 문장을 결정하여 처리하려는 경우 사용되는 제어문이다. if 문에는 (조건식) 을 사용하여 조건식의 평가 결과에 따라 문장 또는 문장들의 수행 여부를 결정하게 된다. 조건식은 연산결과가 반드시 boolean 타입이 되는 식이어야만 한다. 수행 문장이 하나인 경우에는 중괄호를 사용한 블록 지정이 선택적이지만 여러 개인 경우에는 필수이다.

```
if (조건식)
    문장1;
```

```
if (score > 60)
    System.out.println("합격입니다.");
```

```
if (조건식) {
    문장1;
      :
    문장n;
}
```

```
if (score > 60) {
    System.out.println("합격입니다.");
    System.out.println("축하합니다.");
}
```

else 절을 사용하여 조건식의 평가 결과에 따른 수행 문장(들)을 나누어 처리할 수도 있다. 즉, 적용하려는 조건이 두 개인 경우에는 if 문에 else 절을 추가로 사용한다.

```
if (조건식)
    문장1;
else
    문장2;
```

```
if (score > 60)
    System.out.println("합격입니다.");
else
    System.out.println("불합격입니다.");
```

```
if (조건식) {
    문장;
        :
} else {
    문장;
        :
}
```

```
if (score > 60) {
    System.out.println("합격입니다.");
    System.out.println("축하합니다.");
} else {
    System.out.println("불합격입니다.");
    System.out.println("다음 기회를…");
}
```

조건이 여러 개인 경우에는 if – else if … – else 구문을 사용한다. 이 때에도 else 절은 생략이 가능하다.

```
if (조건식1)
    문장1;
else if (조건식2)
    문장2;
else if (조건식3)
    문장3;
  :
else
    문장n;
```

```
if (score > 90)
    System.out.print("A");
else if (score > 80)
    System.out.print("B");
else if (score > 70)
    System.out.print("C");
else if (score > 60)
    System.out.print("D");
else
    System.out.print("F");
System.out.println("등급 입니다.");
```

if 문이나 else 절의 수행 문장으로 또 다른 if 문을 사용하는 것이 가능하다. 이러한 구현을 중첩된 if 문 구현이라고 한다. 다음 예제는 if 문의 수행 문장에 또 다른 if 문이 사용된 예를 보여주고 있다.

```
if (score > 90) {
    System.out.print("A");
    if ( score >= 97)
        System.out.print("+");
    else   if ( score >= 94)
        System.out.print("0");
    else
        System.out.print("-");
}
          :
System.out.println("등급 입니다.");
```

중첩된 if 문을 구현할 때의 주의 사항은 else 절이 동일 블록 안에서 가장 가까이 존재하는 if 문과 연결되어 수행된다는 것이다. 다음 두 개의 소스를 분석해 본다.

```
if (month % 2 == 0)
    if (performance == 10)
        System.out.println("인센티브는 10% 입니다.");
else
        System.out.println("홀수 달은 인센티브가 없습니
다.");
```

```
if (month % 2 == 0) {
    if (performance == 10)
        System.out.println("인센티브는 10% 입니다.");
} else
        System.out.println("홀수 달은 인센티브가 없습니
다.");
```

들여쓰기와 계없이 else 절은 가까운 if 문과 한 쌍이 되어 수행한다. 그러므로 밖에 감싸고 있는 if 문과 한 쌍이 되도록 하려면 안에 있는 if 문을 블록을 사용하여 구분한다.

■ switch 문

switch 문도 if 문처럼 주어진 식의 평가 결과에 따라 수행되는 코드를 다르게 할 수 있다. if 문에서는 boolean 타입의 식을 사용하여 참/거짓의 여부에 따라 처리하지만 switch 문에 사용되는 평가식의 경우에는 연산결과가 int 타입이어야 하며 byte, short, char 의 경우에는 int 형으로 내부적 형 변환(자동 형 변환)이 되므로 사용 가능하다.

괄호안에 주어진 식을 계산해서 그 결과와 같은 값을 정의하고 있는 case 절로 이동하여 문장들을 수행하며 break 문을 만나면 switch문 전체를 빠져나간다. 주어진 식의 결과와 일치하는 case 절이 하나도 없으면 default 절의 수행 문장으로 이동하고, default 절도 없으면 switch 문 전체를 빠져나가 다음 문장을 수행한다.

블록대신 break 문을 사용해서 각 case 절의 끝을 구분하며 만일 break 문이 없으면 다음 case 절의 문장들을 계속해서 수행하게 된다. case 절에 사용되는 비교 값도 int 타입이거나 int 로 내부적 형 변환이 될 수 있는 타입이어야 하며 변수의 사용은 불가능하고 상수나 리터럴만 사용할 수 있다. switch 문도 중첩하여 작성하는 것이 가능하다.
다음은 switch 문의 구문과 간단한 예제이다.

```
switch (int 타입의 식) {
    case 상수1:
                    문장1;
                      :
                    [break;]
    case 상수2:
                    문장1;
                      :
                    [break;]
        :
    default:
            문장1;
              :
}
```

```
switch(score / 10) {
    case 10:
    case 9:
        System.out.print("A");
        break;
    case 8:
        System.out.print("B");
        break;
    case 7:
        System.out.print("C");
        break;
    case 6:
        System.out.print("C");
        break;
    default :
        System.out.print("F");
}
System.out.println("등급 입니다.");
```

[break 사용에 따른 switch 문의 수행 흐름]

```
switch(grade) {
    case 'a' :
        System.out.print("A   등급 ");
    case 'b' :
        System.out.print("B   등급 ");
    case 'c' :
        System.out.print("C   등급 ");
    default :
        System.out.print("등급 없음");
}
```

grade 변수의 값이 'a' 이면
A 등급 B 등급 C 등급 등급 없음 이 출력된다.
grade 변수의 값이 'b' 이면
B 등급 C 등급 등급 없음 이 출력된다.
grade 변수의 값이 'c' 이면
C 등급 등급 없음 이 출력된다.
grade 변수의 값이 'a','b','c' 외의 값이면 **등급 없음**이 출력된다.
각 등급별로 해당 문장만 수행되도록 하려면 각 case 절의 마지막에 break 문을 사용해야 한다.

반복문은 이름 그대로 문장 또는 문장들을 반복하기 위한 것으로 Java 언어의 반복문으로는 for 문, while 문, do-while 문 이렇게 세 가지가 사용된다. 사실 do-while 문은 while 문의 변형이기 때문에 반복문은 for문과 while 문 두 가지가 있다고 볼 수 있다.

for 문은 주로 반복 횟수를 적용하여 수행 문장들의 반복을 처리하고자 하는 경우 사용되며 while 문은 주로 주어진 조건이 만족되는 동안 수행 문장들의 반복을 처리하고자 하는 경우 사용된다.

■ for 문

횟수를 적용하는 반복에 사용되는 제어문으로서 for 문에는 초기식, 조건식 그리고 증감식을 원하는 반복 횟수와 사용하려는 값의 변화에 따라 적절하게 정의하여 구현한다. 수행하게 되는 문장이 두 개 이상인 경우 블록({}) 을 지정한다. 다음은 for 문의 구문과 간단한 예제이다.

```
for (초기식; 조건식; 증감식)
    반복하여 수행할 문장;

for (초기식; 조건식; 증감식) {
    반복하여 수행할 문장;
        :
}
```

```
for (int i=1; i<=10; i++)
    System.out.print(i + "   ");

for (int i=1; i<=10; i++) {
    System.out.print(i + "   ");
    sum += i;
}
```

for 문에 사용되는 식들은 모두 생략할 수도 있으며 생략하더라도 세미콜론(;)은 남겨두어야 한다. 이러한 경우에는 무한 반복을 처리하게 된다. 각각의 식들은 다음과 같은 특징을 갖는다.

초기식	for 안에서만 사용되는 변수를 선언하고 초기화하거나 미리 선언되어 있는 변수를 초기화한다. 2 개 이상의 식을 정의할 수도 있으며 여러 개의 식을 정의하는 경우에는 , 로 구분하고 필요치 않으면 생략하는 것도 가능하다.
조건식	연산 결과가 boolean 타입이 되는 연산식만을 사용한다. 하나의 식만 정의할 수 있으므로 적용하고자 하는 조건식이 여러 개라면 논리곱 또는 논리합 연산자로 결합한다.
증감식	계속해서 반복문을 수행할 것인지의 여부를 점검하는데 사용되는 변수의 값을 변경하는 기능을 구현한다. 2 개 이상의 식을 정의할 수도 있으며 여러 개를 정의하는 경우에는 컴마(,) 로 구분하고 생략하는 것도 가능하다.

for 문 구현 시에는 특히 변수의 스코프에 대하여 주의해야 한다. 다음 예제에서 i 변수는 for 문의 초기식에 선언되어 있으므로 for 문 에서만 사용 가능한 스코프가 적용되어 for 문의 블록을 벗어난 수행 문장에서는 i 변수를 사용할 수 없다. **지역 변수는 변수가 선언된 이후부터 해당 블록 안에서만 사용 가능하다.**

```
int sum = 0;
for (int i=1; i<=10; i++) {
    System.out.println(i);
    sum += i;
}
System.out.println("1 부터 "+i+" 까지의 합 : " + sum); //오류
```

i 변수의 스코프

sum 변수의 스코프

위의 소스를 컴파일시 오류가 발생한다. 컴파일 오류를 해결하기 위해서는 i 변수의 선언을 다음과 같이 for 문 전에 해주어야 한다.

```
int sum = 0;
int i = 0;
for (i=1; i<=10; i++) {
    System.out.println(i);
    sum += i;
}
System.out.println("1 부터 "+i+" 까지의 합 : " + sum);
```

▶ 중첩된 for 문 구현

for 문 안에 또 다른 for 문을 구현할 수 있으며 중첩 회수에는 제한이 없다. 다음은 중첩된 for 문을 사용하여 1단부터 9단까지 구구단을 출력하는 예이다.

```
for (int i=1; i<10; i++) {
    for(int j=1;j<10;j++) {
        System.out.print(i + "X" + j + "=" + i*j +"₩t " );
    }
    System.out.println();
}
```

■ while 과 do-while 문

while 과 do-while 문은 어떠한 조건이 만족되는 동안 문장 또는 문장들의 블록을 반복 수행하고자 하는 경우에 사용되는 제어문이다.

while 문은 선 조건 후 수행 구조로서 조건식을 먼저 평가하고 참이면 주어진 문장 또는 문장들의 블록을 수행하는 반복문이며 do-while 문은 선 수행 후 조건 구조로서 주어진 문장 또는 문장들의 블록을 먼저 수행하고 조건식을 평가하여 계속해서 반복문을 수행할 것인지를 결정한다.

for 문이나 while 문은 조건식의 결과에 따라 주어진 문장 또는 문장들의 블록이 한 번도 수행되지 않을 수 있으나 do-while 문은 처음부터 조건이 거짓이라 하더라도 최소한 한 번은 수행되는 특징을 갖는다.

▶ while 문의 구문

while 문은 다음과 같이 조건식과 반복 수행 문장 또는 반복 수행하고자 하는 문장들의 블록으로 구성된다.

```
while (조건식)
    반복하여 수행할 문장;

while (조건식) {
    반복하여 수행할 문장;
        :
}
```

```
while ( (input = (char)System.in.read()) != 0x0d)
{
   if (input >= '₩u0061' && input <= '₩u007a')
      System.out.print((char)(input - '₩u0020'));
   else
      System.out.print(input);
}
```

조건식은 절대로 생략할 수 없으며 무한 루프를 구현하려면 조건식에 true 라는 boolean 타입의 리터럴을 지정한다.

```
while (true)
    반복하여 수행할 문장;
```

▶ do-while 문의 구문

while 문과 동일하게 조건식과 반복 수행 문장 또는 블록으로 구성되는데 while 문과는 다르게 조건식을 뒤에 사용한다. do-while 문의 마지막에는 반드시 세미콜론(;)을 붙여주어야 한다.

```
do
    반복하여 수행할 문장;
while (조건식) ;

do {
   반복하여 수행할 문장;
      :
} while (조건식) ;
```

```
do {
    System.out.print("남성(M), 여성(F)를 입력 : ");
    input = (char)System.in.read();
    System.in.skip(2);
} while (input != 'M' && input != 'F');
if (input == 'M')
    System.out.println("남성이시네요..");
else
    System.out.println("여성이시네요..");
```

5.3 흐름 제어문

반복문의 수행 흐름을 제어하는 흐름 제어문으로는 break 와 continue 가 있으며 break 는 반

복문의 수행을 끝내는 제어문이고 continue 는 반복문 수행 중간에 남은 문장들을 더 이상 수행하지 않고 조건을 점검하는 곳으로 분기하도록 하여 계속해서 다음 반복으로 넘어가도록 하는 제어문이다.

반복문이 중첩 구현되어 있고 반복문 앞에 레이블을 설정한 경우에는 break 와 continue 문 모두 레이블을 선택적으로 사용 가능하다.

■ break 문

break 문은 반복문 수행 중간에 반복문을 끝내고자 하는 경우 사용되며 switch 문에 사용되는 경우와 동일한 효과이다. 반복문이 중첩되게 정의되어 있는 경우 가장 가까이 존재하는 반복문 하나만을 끝내게 되므로 가까이 정의된 반복문 대신 밖의 반복문에 대하여 제어하려는 경우 레이블을 사용한다.

```
while (조건식) {
    문장1
    if (조건식)
        break;
    문장2
}
```

```
while ( (input =(char) System.in.read()) != 0x0d) {
    if (input >= '0' && input <= '9') {
        System.out.println("영문자만    입력해    주세요!");
        break;
    }
    if (input >= '₩u0061' && input <= '₩u007a')
        System.out.print((char)(input - '₩u0020'));
    else
     System.out.print(input);
```

▶ label 을 사용한 break

- break 의 label 에는 반복문에 지정된 label 만 설정 가능하다.
- label 명은 Java 의 식별자 규칙이 적용되며 지정하고자 하는 반복문 앞에 콜론(:) 기호와 함께 정의한다.

```
labelname : while (조건식) {
    문장1
    while (조건식) {
      문장2
      if (조건식)
        break labelname;
      문장3
      문장4
}
```

```
outer : for (int i = 1; i <= 9; i++) {
    inner : for (int j = 1; j <= 9; j++) {
        if (i % 2 == 0)
          break outer;
          System.out.print(i + "X" + j + "=" +
                                  (i*j) + "₩t");
    }
    System.out.println();
  }
  System.out.println("Labeled Break Test");
```

■ continue 문

반복문 수행 중간에 남은 문장들을 더 이상 수행 하지 않고 조건을 점검하는 곳으로 분기하도록 처리하고자 하는 경우 사용한다. 반복문이 중첩되게 정의되어 있는 경우 가장 가까이 존재하는 반복문에 대해서만 수행 흐름을 제어하게 되므로 가까이 정의된 반복문 대신 밖의 반복문에 대하여 제어하려는 경우 break 와 같이 레이블을 사용한다.

```
while (조건식) {
    문장1
    if (조건식)
        continue;
    문장2
}
```

```
while ( (input = (char)System.in.read()) != 0x0d) {
    if (input >= '0' && input <= '9') {
      System.out.println("영문자만 입력해 주세
요!");
      continue;
    }
    if (input >= '₩u0061' && input <= '₩u007a')
       System.out.print((char)(input - '₩u0020'));
    else
     System.out.print(input);
```

▶ label 을 사용한 continue

- continue 의 label 에는 반복문에 지정된 label 만 지정 가능하다.
- label 명은 Java 의 식별자 규칙이 적용되며 지정하고자 하는 반복문 앞에 콜론(:) 기호와

함께 정의한다.

```
labelname : while (조건식) {
    문장1
    while (조건식) {
      문장2
      if (조건식)
        continue labelname;
      문장3
      문장4
}
```

```
outer : for (int i = 1; i <= 9; i++) {
    inner : for (int j = 1; j <= 9; j++) {
        if (i % 2 == 0)
            continue outer;
        System.out.print(i + "X" + j + "=" +
                                (i*j) + "₩t");
    }
    System.out.println();
}
System.out.println("Labeled Continue Test");
```