05장. 기본구문 - 반복문

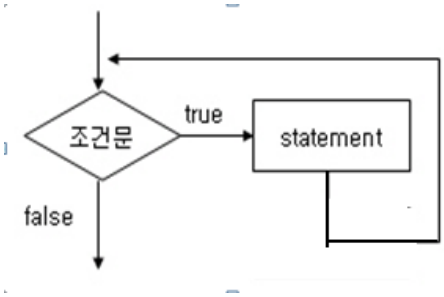
1. while문

while문은 반복을 몇 번해야 할지 알 수 없는 경우에 사용된다. 즉, 반복횟수를 알 수 없는 경우에 사용된다. while문은 조건문을 비교해서 조건을 만족하는 경우에는 문장(statement)을 수행하고 조건을 만족하지 않으면 while문을 빠져나온다. 이때 수행되는 문장안에는 반드시 반복횟수를 제어하는 변수를 가지고 있어야 한다. 그래야 수행되는 횟수를 제어할 수 있다.

[형식]

```
while (조건식) {
  실행 문 1;
  실행 문 2;
  ...
}
```

[순서도]



while 문에서는 조건식을 평가 true 인 경우에는 「{」에서 「}」의 블록 내에 기술 된 문장을 실행한다. 조건식은 관계 연산자 및 논리 연산자 등을 사용하며 블록의 처리가 끝나면 다시 조건식을 평가한다. 또한 true 인 경우에는 블록 문장을 실행, false 인 경우에는 while 문을 종료한다.

반복 실행 문이 하나 인 경우에는 "{"와 "}"를 생략하여 다음과 같이 설명 할 수 있습니다.

```
while (조건식)
  실행 문;
```

[프로그램 예]

```
int i = 0;
while (i <2) {
  System.out.println ( "i ="+ i);
  i ++;
}
```

이 경우 다음과 같이 처리된다.

```
1) i = 0;
2) 조건식을 평가. i 는 2 보다 작기 때문에 반복 실행
3) "i = 0"을 화면에 출력
4) i + +;
5) 조건식을 평가. i 는 2 보다 작기 때문에 반복 실행
6) "i = 1"을 화면에 출력
7) i ++;
8) 조건식을 평가. i 는 2 보다 작지 않은 때문에 반복을 종료
```

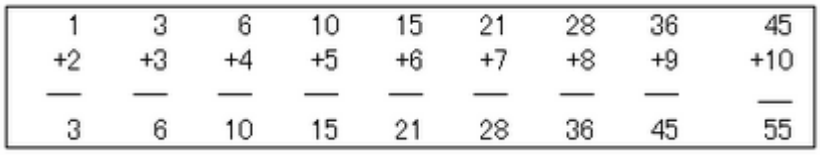
while 문은 먼저 조건식이 평가된다. 만약 조건식이 갑자기 false 가 되었을 경우에는 반복은 실행되지 않는다.조건식이 true 인 경우에는 while 문장의 "{"부터 "}"까지 블록 내의 문장을 순서대로 수행한다.

블록의 마지막 문장을 실행하면 첫 번째 반복이 완료된다. 그리고 while 문장의 처음으로 돌아 다시 조건식을 평가한다. true 라면 다시 블록 내의 문장을 실행하고 false 인 경우에는 while 문 다음 문장으로 처리가 이동된다.

연산식을 이용하여 반복구문을 이용하면 간단하게 결과를 얻을 수 있다. while 을 이용한 반복프로그램으로 1 부터 10 까지의 정수를 합계 한 값을 계산하는 프로그램을 작성하는 구문을 살펴 보자.

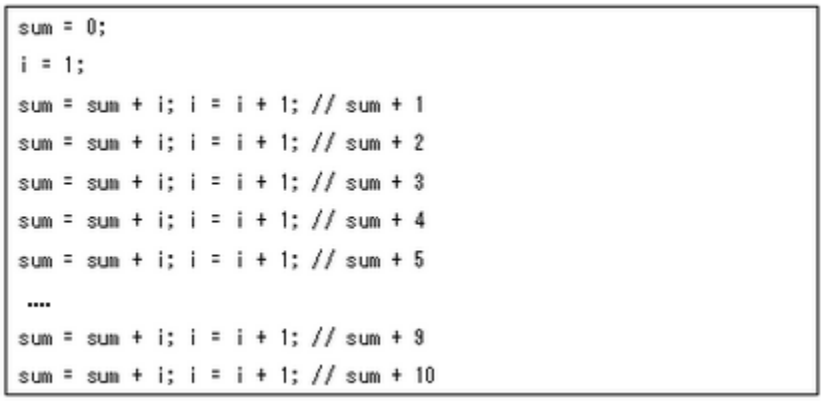
단순하게 생각하면 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 라는 식의 계산 결과를 int 형 변수 x 에 대입하여 x 의 값을 표시하는 프로그램이 떠오른다. 숫자를 하나씩 더해 가면 다음과 같이 된다. 1 +2 의 계산 결과 3 은 2 의 다음 3 을 더하면 6 이다.

1 + 2 + 3 의 합은 6 이며 3 다음 4 를 더하면 10 이된다. 이것이 반복되면 1 부터 10 까지의 정수를 합계 한 값을 계산 할 수 있다.



1	3	6	10	15	21	28	36	45
+2	+3	+4	+5	+6	+7	+8	+9	+10
3	6	10	15	21	28	36	45	55

값을 저장하기 위해 int 형의 변수 sum 을 제공한다. 그러면 1 에서 10 까지의 정수를 합계하는 프로그램은 그림과 같다.. sum = 0;처럼 먼저 sum 에 0 을 대입하는 작업을 하고 계산을 시작하는 것이 이루어진다. .또 하나의 포인트로 계산 도중에 sum = sum + 1;처럼 sum 을 이용한 식의 값을 sum 에 대입하고 있다는 처리한다.



```
sum = 0;
i = 1;
sum = sum + i; i = i + 1; // sum + 1
sum = sum + i; i = i + 1; // sum + 2
sum = sum + i; i = i + 1; // sum + 3
sum = sum + i; i = i + 1; // sum + 4
sum = sum + i; i = i + 1; // sum + 5
 ....
sum = sum + i; i = i + 1; // sum + 9
sum = sum + i; i = i + 1; // sum + 10
```

다음과 같이 코드를 구현하게 되면 반복연산이 간단하게 이루어진다.

```
public class ch05_1 {

   public static void main(String[] args) {
      int sum=0;
      int i=1;
      while(i<=10){
         sum=sum+i;
         System.out.println(sum);
         i++;
      }

   }

}
```

출력결과는 다음과 같다.

```
1
3
6
10
15
21
28
36
45
55
```

2. do-while 문

do-while 문 while 문과 유사하나 일단 한번 수행을 한 후에 조건을 비교한다. 즉 조건이 맞지 않더라도 반드시 한번은 while 내의 문장을 수행한다. do~while 문에서는 우선 「{」에서 「}」의 블록 내에 기술 된 문장을 실행한다. 그리고 조건식을 평가 true 인 경우에는 다시 블록을 수행하지만 false 인 경우에는 do~while 문을 종료하고 다음 문장으로 명령처리를 이동한다.

주의해야 할 점은 마지막에 세미콜론을 붙일 필요가 있다는 것이다.

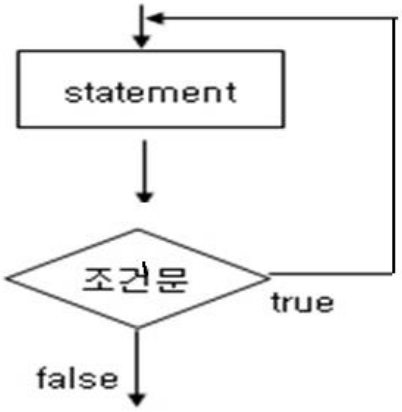
[형식]

```
do {
  실행 문 1;
  실행 문 2;
  ...
} while (조건식);
```

반복 실행 문이 하나 인 경우에는 "{"와 "}"를 생략하여 다음과 같이 선언할 수 있다.

```
do
  실행 문;
while (조건식);
```

[흐름도]



while 문과 비슷하지만, 다른 점은 코드 블록이 적어도 한 번은 실행에서 조건문을 평가한다는 점이 다르다.

```
public class ch05_2 {

   public static void main(String[] args) {
      int i = 0;
      do {
         System.out.println(i);
         i++;
      } while (i < 3);

   }

}
```

실행 결과는 다음과 같다.

0

1

2

3. for 문

조건에 의한 일정한 문장을 반복 수행하는 for문은 반복을 수행할 횟수가 결정된 경우의 프로그램에 주로 사용되는 제어문이다.
배열과 같이 반복해야 하는 횟수가 결정된 형태를 제어할 때 주로 사용된다.
초기값은 for문 수행시 단 한번만 수행된다. 조건문은 루프 탈출조건이라고도 불리며 for문안의 문장(statement)을 수행하기 전에 수행해서 조건을 만족하면 문장을 수행한다. 연산식은 for문 안의 문장을 수행하고 나서 수행된다. 반복횟수만큼 반복한다.

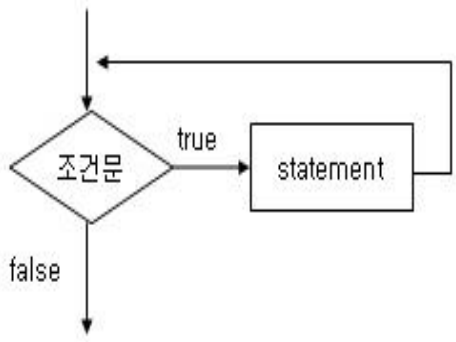
[형식]

```
for ( 변수 초기화; 조건문; 연산식 ) {
     조건문이 true 일 때 실행되는 코드
}
```

for문의 흐름은 다음과 같이 구현된다.

초기값 -> 조건문 (true)->명령 -> 연산식
-> 조건문 (true)->-> 명령 -> 연산식
-> 조건문 (true)->-> 명령 -> 연산식
-> 조건문 (false)->for문 종료

[흐름도]



[프로그램 예]
다음은 for문의 프로그램을 구현한 예이다.

```
int sum = 0;
int count = 0;

for (; count <2;) {
```

```
  sum + = 2;
  count ++;
}

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

이 예에서는 조건식 "count <2"true 가되는 동안 반복한다. 반복을 한 번 실행하기 위해 조건을 변화시키지 않으면 무한 반복을 하고 버리기 때문에 이번 예제에서는 반복에서 실행되는 문장의 하나로서 변수 "count"값을 1 씩 증가 시키고 있다. 내부적으로 다음과 같이 문장이 실행된다.

```
 1) sum = 0;
 2) count = 0;
 3) 조건식을 평가. count 는 2 보다 작기 때문에 반복 실행
 4) sum + = 2;
 5) count + +;
 6) 조건식을 평가. count 는 2 보다 작기 때문에 반복 실행
 7) sum + = 2;
 8) count + +;
 9) 조건식을 평가. count 는 2 보다 작지 않은 때문에 반복을 종료
10) System.out.println (sum);
```

이렇게 for 문은 먼저 조건식이 평가된다. 만약 조건식이 갑자기 false 가되었을 경우에는 반복은 실행되지 않는다.조건식이 true 인 경우에는 for 문장의 "{"부터 "}"까지 블록 내의 문장을 순서대로 수행하고 있다.

블록의 마지막 문장을 실행하면 첫 번째 반복이 완료된다. 그리고 for 문의 처음으로 돌아 다시 조건식을 평가한다. true 라면 다시 블록 내의 문장을 실행하고 false 인 경우에는 for 문 다음 문장으로 처리가 이동한다.

for 문은 이와 같이 조건식의 평가와 블록의 처리의 실행 반복을 위한 것입니다.

for 문은 조건식 외에 초기화 식 변화 식을 작성할 수 있다. 초기화 식으로 설명하는 식은 for 문에서 먼저 조건식이 평가되기 전에 한 번만 실행된다. 용도로는 for 문의 조건식에 사용되는 변수의 선언과 초기화에 이용되는 경우가 많다.

위의 프로그램을 다음과 같이 변경할 수 있다.

```
int sum = 0;

for (int count = 0; count <2; count ++) {
  sum + = 2;
}

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

for 문의 다양한 구문 예를 살펴 보자.

Exam01) Hello World!를 세 번 출력

```
public class ch05_3 {

   public static void main(String[] args) {

      for(int i=1;i<=3 ;i++) // i가 1에서 3보다 같거나 작을 동안 1씩 증가
      {
        System.out.println("Hello World!"); //명령 실행
      }
   }

}
```

[결과]

```
Hello World!
Hello World!
Hello World!
```

Exam02) 1 ~ 10 까지 출력

```
public class ch05_4 {

   public static void main(String[] args) {
      int i = 0;
      for (i = 1; i <= 10; i++) { // 초기값 1에서 10까지 1씩 증가하면서 출력
         System.out.printf("%5d", i);
      }

   }

}
```

[실행결과]

1 2 3 4 5 6 7 8 9 10

Exam03) 10 ~ 1 까지 출력

```
public class ch05_5 {

   public static void main(String[] args) {

      int i = 0;
      for (i = 10; i >= 1; i--) // 초기값 10이 1보다 크거나 작을 동안 1씩 감소
      {
         System.out.printf("%5d", i);
      }
   }

}
```

[실행결과]

10 9 8 7 6 5 4 3 2 1

Exam04) 10 20 30 40 50 60 70 80 90 100 출력

```
public class ch05_6 {

   public static void main(String[] args) {
      int i = 0;
      for (i = 10; i <= 100; i += 10) { // 초기값 10에서 100보다 같거나 작을동안 10씩 증가
         System.out.printf("%5d", i);
      }
   }
}
```

[실행결과]

10 20 30 40 50 60 70 80 90 100

Exam05) 0 ~50 까지 5의 배수를 출력

```
public class ch05_7 {

   public static void main(String[] args) {
      for (int i = 0; i <= 50; i += 5) {
         System.out.printf("%5d", i);
      }
   }
}
```

[실행결과]

5 10 15 20 25 30 35 40 45 50

for 문은 초기화 식, 조건식, 연산식등의 3 가지 식을 사용하지만, 초기화 식 연산 식에는 여러 수식을 작성할 수 있다. 여러 수식을 작성하려면 수식과 수식 사이를 쉼표 (,)로 구분하여 선언한다.

```
for ( 초기화 식 1 초기화 식 2 .. ; 조건식; 연산식 1, 연산  식 2, .. ) {
  실행 문;
}
```

※ 조건식은 관계 연산자와 논리 연산자 복잡한 조건식을 작성할 수 있다.

다음은 초기식과 연산식을 2 개 선언한 구문이다. i 는 1~9 까지 출력하고 j 는 9 에서 1 까지 출력되는 구문이다.

```
int i, j;

for (i = 1, j = 9; i <10; i++, j--) {
  System.out.println ( "i ="+ i + ", j ="+ j);
}
```

초기화 식으로 변수 'i'와 'j'의 초기 값을 설정한다. 또한 변화 식에서 변수 "i"는 1 만 증가하고 변수 "j"는 1 만큼 감소시키고 있다. 이렇게 하나의 for 문에 여러 변수의 초기화와 연산식을 선언할 수 있다.

단, 변수 선언을 수반하는 경우는 주의한다. 다음과 같은 기술은 컴파일 에러가 발생한다.

```
for (int i = 1, int j = 9; i <10; i++, j--) {
  System.out.println ( "i ="+ i + ", j ="+ j);
}
```

동일한 데이터 형식의 경우라면 "int i = 1, j = 9"이라고 할 수 있다. 단지 "int i = 1, int j = 9"이라고 할 수 없으며, 다른 데이터 형식의 변수에 대해 "int i = 1, double d = 1.0"과 같이 기술 할 수 없다. 여러 초기화 식을 사용하는 경우는 for 문 밖에서 변수 선언을 실시한 초기화 만 선언한다.

4. 중첩 for

자바의 모든 기본 구문은 중첩해서 사용할 수 있다. 그 중에서 for 문에서 실행되는 블록 안에는 다양한 문장을 작성할 수 있으므로 다른 for 문을 기술 할 수 있다.

[형식]

```
for ( 변수 초기화; 조건문; 연산식 ) {
        for ( 변수 초기화; 조건문; 연산식 ) {
                조건문이 true 일 때 실행되는 코드
                }
```

```
    }
```

구문에 따라 중첩 반복을 사용하게 된다. 예를 들어, 2 차원 배열의 경우 모든 요소를 검색하려고 하면 다음과 같이 이중 루프가 필요하다. 카운터 i와 j의 루프가 중첩되어 각각 4회, 6회 반복한다.

```
public class cho5_8 {

   public static void main(String[] args) {
      for (int i = 0; i <= 3; i++) {
         for (int j = 0; j <= 5; j++) {
            System.out.print("(" + i + "" + j + ")");
         }
         System.out.println(" ");
      }

   }
}
```

[실행결과]

(00)(01)(02)(03)(04)(05)

(10)(11)(12)(13)(14)(15)

(20)(21)(22)(23)(24)(25)

(30)(31)(32)(33)(34)(35)

실행순서와 함께 다음 코드를 살펴 보자.

```
for (int i = 0; i <2; i++) {
  for (int j = 0; j <2; j++) {
    System.out.println ( "i ="+ i + ", j ="+ j);
  }
}
```

이 예에서는 외부의 for 문 블록 내에 다른 for 문을 기술하고 있다. 내부 실행은 다음과 같다

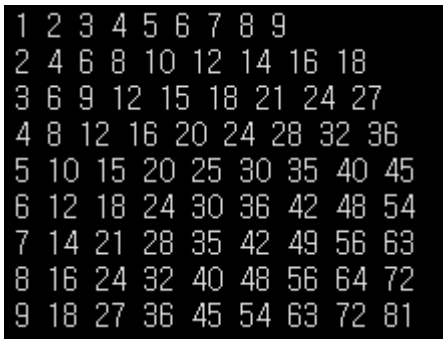
```
 1) 변수 "i"를 선언하고 0 으로 초기화
 2) 외부의 조건식을 평가 반복을 실행하는
 3) 변수 "j"를 선언하고 0 으로 초기화
 4) 내부의 조건식을 평가 반복을 실행하는
 5) "i = 0, j = 0"을 출력
 6) 변화 식에서 변수 "j"의 값을 1 로
 7) 내부의 조건식을 평가 반복을 실행하는
 8) "i = 0, j = 1"을 출력
 9) 변화 식에서 변수 "j"의 값을 2 로
10) 내부의 조건식을 평가 안쪽의 for 문을 종료
```

```
11) 변화 식에서 변수 "i"의 값을 1 로

12) 외부의 조건식을 평가 반복을 실행하는
13) 변수 "j"를 선언하고 0 으로 초기화
14) 내부의 조건식을 평가 반복을 실행하는
15) "i = 1, j = 0"을 출력
16) 변화 식에서 변수 "j"의 값을 1 로
17) 내부의 조건식을 평가 반복을 실행하는
18) "i = 1, j = 1"을 출력
19) 변화 식에서 변수 "j"의 값을 2 로
20) 내부의 조건식을 평가 안쪽의 for 문을 종료
21) 변화 식에서 변수 "i"의 값을 2 로

22) 외부의 조건식을 평가 바깥 쪽 for 문을 종료
```

2) ~11)까지 외부 for 문장의 첫 번째 반복, 12)에서 21)까지 외부 for 문장의 두 번째 반복된다. 그리고 바깥 쪽 for 문이 한 번 돌 때마다 안쪽의 for 문이 반복하게 된다.

다음과 같이 곱을 구하는 프로그램을 구현하는 구문이다.

[실행결과]



```
public class ch05_9 {

   public static void main(String[] args) {
      for (int i = 0; i < 9; i++) {
         for (int j = 0; j < 9; j++) {
            int k = (i + 1) * (j + 1);
            if (k < 10) {
               System.out.print(" " + (i + 1) * (j + 1));
            } else {
               System.out.print("" + (i + 1) * (j + 1));
            }
         }
         System.out.println("");
      }
   }
}
```

카운터 i 와 j 의 곱셈 결과 k 를 출력하고 있을 뿐이다. 카운터 변수는 초기값을 0 으로하기 때문에, 1 - 9 까지 표현하려면 1 씩 증가한다.표시를 마련하기 위해 k <10 을 조건으로 숫자 앞에 삽입하는 공간의 개수를 조정한다.

5. 제어 흐름문

자바의 기본 구문의 흐름을 제어하는 키워드로 다음과 같이 구분하여 사용한다.
반복 또는 구문에 사용되는 키워드로 break, continue, label 등이 있으며 메소드를 종료하는 키워드인 return 도 있다.

>> break : switch, loop 를 벗어나는 구문

break 문은 for 문, while 문, do..while 문, switch 문 블록에서 사용되는 break 문이 실행되면 블록을 빠져 다음 작업에 옮깁니다. break 문장의 형식은 다음과 같이 되어 있다.

```
break;
```

for 문은 조건식이 true 사이 블록의 처리를 반복 실행되지만 break 문이 실행되면 for 문을 종료하고 다음 작업으로 이동한다.

```
for (초기화 식; 조건식; 연산식) {
  ...
  if (조건식) {
    break;
  }
  ...
}
```

if 문 등과 함께 break 문을 기술한다. (if 문 등과 함께 않으면 반드시 1 번째 반복 과정에서 break 문이 실행되고 for 문이 종료 해 버린다.).

>> continue : loop 구문을 진행 하는 구문

continue 문은 break 문장과 같이 for 문, while 문, do~while 문장 블록에서 사용되며 continue 문이 실행되면 실행 된 위치에서 블록의 마지막까지 남은 처리를 건너뛴다. continue 문장의 형식은 다음과 같이 되어 있다.

```
continue;
```

다음의 구문을 살펴봅시다.

```
for (int i = 0; i <10; i + +) {
  if (i == 3) {
    continue;
  }
  System.out.println (i); --------------→ 0 1 2 4 5 6 7 8 9 가 출력된다.
}
```

위에서 변수 "i"의 값이 3 시에 continue 문이 실행됩니다. continue 문이 실행되면 블록 내에서 그 이후에 작성된 모든 작업을하지 않고 다음 반복 조건 확인으로 이동한다. 즉, 위의 경우 변수 "i"의 값이 3 때만 "System.out.println (i);"가 실행되지 않는다.

또한 for 문의 경우는 변화식이 별도 준비되어 있기 때문에 걱정하지 않아도 괜찮지 만, while 문 등에서는 조건식의 값을 변화시키는 과정을 포함하여 스킵시켜 버리지 않게 조심 한다.

예를 들어 다음과 같이 기술 해 버리면 변수 "i"가 3 이되면 변수 "i"의 값을 변화시키는 과정도 생략 해 버리기 때문에 다음 반복 되어도 변수 "i"값이 변화하지 않고 무한 루프가 되어 버린다.

```
int i = 0;

while (i <10) {
  if (i == 3) {
    continue;
  }
  i++;
  System.out.println (i); --------------→ 0 1 2 출력 되고 무한 루프 상태가 된다.
}
```

또한 break 문과 마찬가지로 반복 처리가 다중으로 되어있는 경우에는 continue 문을 포함한 가장 안쪽의 반복을 생략하고 다음 반복으로 넘어간다.

>>label : loop구문에서 블록의 외부에 제어를 옮기는 키워드

```
public class ch05_10 {

   public static void main(String[] args) {
      escape: for (int i = 0; i <= 9; i++) {
         System.out.print(i + ":");
         for (int j = 0; j <= 9; j++) {
            System.out.print("" + i * j);
            if (i * j >= 20) {// 곱이 20 이상이되면
               break escape; // 레이블에 지정된 계층에 탈출
            }
         } // j 루프 종료
         System.out.println("");
      } // i 루프 종료
   }
}
```

[실행결과]

```
0 : 0 0 0 0 0 0 0 0 0 0
1 : 0 1 2 3 4 5 6 7 8 9
2 : 0 2 4 6 8 10 12 14 16 18
3 : 0 3 6 9 12 15 18 21
```