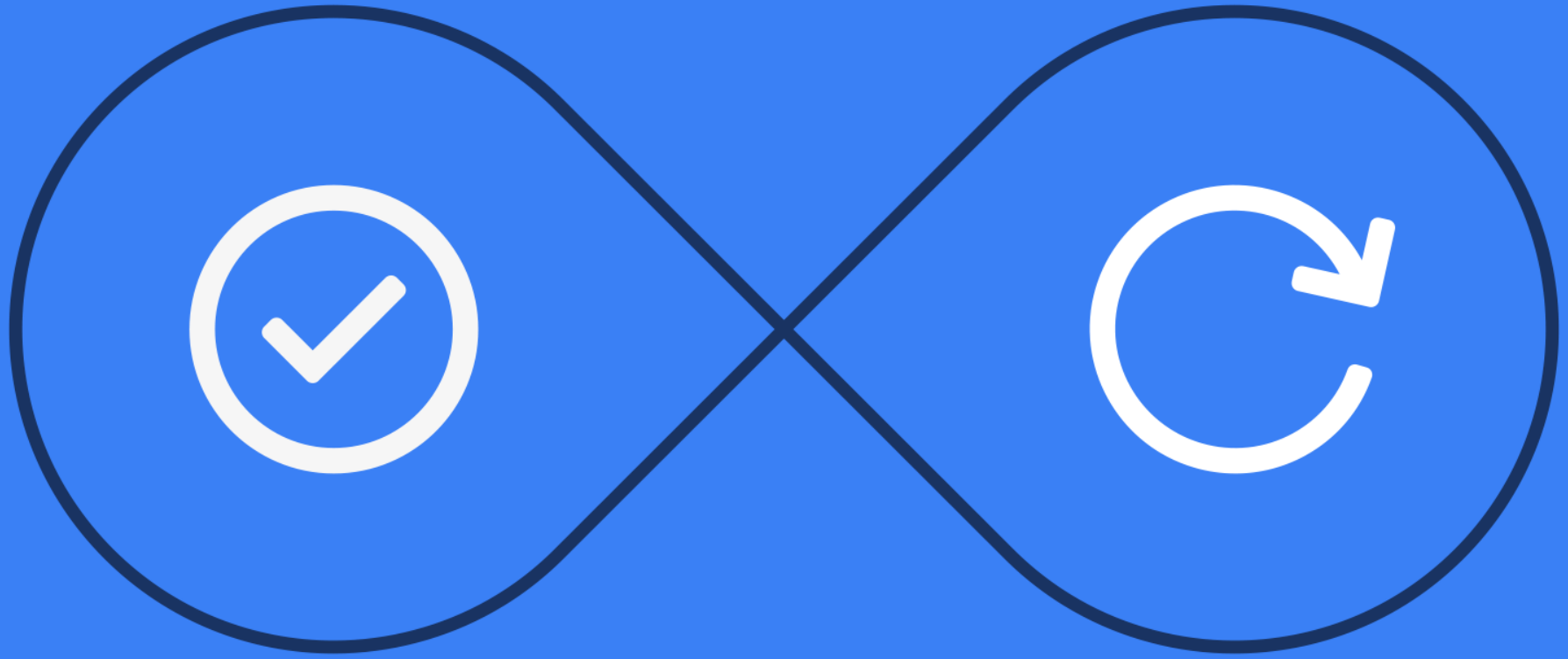
Java 기초

반복문



반복문

반복문

반복문

- 반복문
  - 같은 로직을 계속 반복하는 문장을 의미
  - 주변에는 반복되는 로직이 많으며 이런 로직을 반복할 수 있도록 도와줌으로써 로직의 중복을 막고 코드를 간결하게 쓸 수 있다.
  - java에서는 while문과 for문이라는 반복문을 제공하고 있으며 상당히 보편적으로 많이 사용되는 반복문이다.
  - 역시나 Java에서 잘 사용해야 하는 명령문 중 하나이다.

While

- while

  while(조건식 & 논리값){ 조건식/값이 참일 때 반복되는 로직; }

  - Java의 반복문 중 하나의 형태
  - while문 옆의 괄호 안에는 조건식 혹은 논리값이 들어간다.
  - 만약 논리값 true를 괄호 안에 넣는다면 해당 로직은 무한 반복한다.
  - 무한 반복을 쉽게 할 수 있으며 일정 이상 반복하면 멈추는 로직을 만들어야만 한다.
  - 특정 조건이 참일 동안 계속해서 반복을 하는 로직을 만들 때 사용한다.

while

while문 예제

```
Run | Debug
public static void main(String[] args) {
    // while문 기본 문법 예제
    int a = 0;
    while (a < 3) {
        System.out.println("안녕하세요 반갑습니다.");
        a++;
    }
}
```

While

- while
  - while문은 조건식의 문장이 참일 동안 반복하는 문장이기 때문에 반드시 거짓이 될 조건을 걸어놔야 한다.
  - 그렇지 않으면 무한 반복을 하게 된다.
  - 보통 이런 무한반복을 피하기 위해 while문 밖에 카운트 변수를 만들어 놓고 while문 안에서 증감혹은 다른 연산을 반복시킨다.
  - 보통 메서드 중에서 반복의 끝인지 아닌지를 판별하는 메서드가 있는데 이럴 때도 while문을 사용하는 경우가 있다.

While

while문 예제

```
// 증감연산자를 잘못 입력할 경우 while문이 무한으로
// 반복하는 경우가 생긴다. 보통 본인의 예상과는 다른 결과가
// 나오기 때문에 나오는 현상으로 상당히 빈번하게 발생한다.
// 또한 무한 반복되는 문장의 경우 코드상에서는 에러가 발견되지
// 않으므로 결과를 쉽게 예측하기 어렵다.
int b = 5;
while(b > 1){
    System.out.println("이 문장은 무한 반복됩니다.");
    b++;
}
```

While

- whlie문 읽기
  - while문을 읽기 위해서는 카운트 되는 변수가 얼마나 카운팅이 되며 해당 카운트를 통해 while문이 얼마나 돌지를 예측할 수 있어야 한다.
  - 하지만 사람의 머리로는 while문의 로직을 예측하기 힘들며 그렇기 때문에 처음에는 일일히 노트에 적어가며 실제 돌아가는 값을 예측하는 시뮬레이션을 해보고 코드를 돌려 자신의 예측과 맞는지 틀린지를 대조하는 연습을 계속 해봐야 한다.
  - 경력자의 경우 이런 훈련이 많이 되어있어서 어떻게 돌아가는지를 한방에 예측 가능한 사람들이 많으며 이런 로직적인 부분에서 크게 머리를 쓰지 않는다.

While

- while문 읽기 예제

```
Run | Debug
public static void main(String[] args) {
    // while문 읽기 연습
    // while문을 처음에 눈으로 읽으려고 하면 잘 적응이 되지 않을것이다.
    // while문을 읽는 연습은 처음에는 순차적으로 노트에 써보며 한 루프
    // 루프의 변경 사항을 적어보는 것이 가장 좋다.
    // 이후 자신이 조금 숙달되었다 싶으면 눈으로 읽고 예측하는 것이 가능하다.

    int a = 1;
    while(a <= 5){
        System.out.println(a);
        a++;
    }
}
```

끝에서 다시 비교로 이동

a = 1

1번째 루프

출력 => 1
a 값 1 증가. 후 비교

a = 2

2번째 루프

출력 => 2
a 값 1 증가. 후 비교

a = 3

3번째 루프

출력 => 3
a 값 1 증가. 후 비교

a = 4

4번째 루프

출력 => 4
a 값 1 증가. 후 비교

a = 5

5번째 루프

출력 => 5
a 값 1 증가. 후 비교

a = 6

루프 종료

While

- while문 읽기 예제

```
Run | Debug
public static void main(String[] args) {
    // while문 읽기 예제
    // 일반적인 증감 연산 뿐만이 아니라 다양한 산술 연산을 통해
    // 카운트 값을 변형시키는 경우는 더 복잡할 수 있다.
    int a = 1;
    while(a < 40){
        System.out.println(a);
        a = a * 2;
    }
}
```

끝에서 다시 비교로 이동

a = 1

1번째 루프

출력 => 1

a 값 2 곱. 후 비교

a = 2

2번째 루프

출력 => 2

a 값 2 곱. 후 비교

a = 4

3번째 루프

출력 => 4

a 값 2 곱. 후 비교

.....

a = 32

6번째 루프

출력 => 32

a 값 2곱. 후 비교

a = 64

루프 종료

While

- break;
  - break문은 switch ~ case 문장을 빠져나갈 때 썼던 명령어다.
  - 하지만 break문은 switch ~ case 문장 뿐만이 아니라 while같은 반복문에도 사용이 가능하다.
  - 기능은 switch ~ case문과 동일하며 while문 안에 break문을 만날 경우 해당 로직을 종료하고 while문의 밖으로 빠져나와 while문 아래의 다음 로직을 실행시킨다.

While

break

```
Run | Debug
public static void main(String[] args) {
    // while break 문 예제
    int a = 1;
    while(true){
        System.out.println(a);
        // break 문을 아래와 같이 씀으로써
        // 실제 a라는 변수의 값을 while문에서 조건으로
        // 비교하는 것 처럼 만들 수 있다.
        if(a==5)break;
        a++;
    }
    // break로 빠져 나오기 전 a 값이 6으로 변해서
    // 올라왔을거라 생각하는 사람들이 많지만
    // break문은 아래있는 로직을 전부 생략해버리고
    // 다음 로직을 태우기 때문에 a는 6으로 되지 않고
    // 5가 되어 출력된다.
    System.out.println(a);  // 5
}
```

While

- Continue
  - Continue는 아래 로직을 스킵하고 다시 반복문의 비교를 하고자 할 경우 사용한다.
  - 이 경우는 break와의 공통점이라면 아래 로직을 스킵하는 것이 공통일 수 있고 차이라면 break문은 반복문을 아예 빠져나가지만 continue문은 다시 루프의 처음으로 돌아온다.
  - 문제는 continue문 아래에 증감연산이 존재할 경우 무한 루프를 탈 수가 있다.
  - continue는 해당 로직의 처음으로 가는데 수가 변하지 않으면 while에 걸어놓았던 조건식을 무한정 만족하게 되기 때문이다.

While

Continue 예제

```java
Run | Debug
public static void main(String[] args) {
    // continue 문 예제
    // continue 문 아래쪽에 증감 연산자를 놓을 경우 증감연산이
    // 일어나지 않는다. 즉 다시말해서 증감연산 제어를 하기 위해
    // continue 문 보다 증감연산이 위치해야 한다.

    // 잘못된 로직의 예
    int a = 1;
    while(a <= 5){
        System.out.println(a);
        // continue문 위에 증감연산을 놓아야 증감이
        // 효과적으로 일어난다.
        a++;
        if(a==3)continue;
        // continue문 때문에 증감연산이 일어나지 않는다.
        // 즉 a = 3 부터 a는 더 이상 증가하지 않는다.
        // a++;
    }
}
```

While

Continue 예제

```
Run | Debug
public static void main(String[] args) {
    // 5 를 생략한 1부터 10까지의 출력 예제
    int a = 0;
    while(a < 10){
        a++;
        if(a==5)continue;
        System.out.println(a);
    }
}
```

While

- Continue, Break 문
  - Continue문과 Break문은 로직을 강제로 제어하기 위해 사용하는 경우가 많다.
  - 상황에 따라서 어쩔 수 없이 쓰는 경우가 있으나 너무 남발할 시 로직을 읽는데 상당히 힘들 수 있다.
  - 상황에 따라 적절히 사용하는 것이 관건. 그 외에는 최대한 while문의 조건식을 통해 빠져나가도록 코딩하는 것을 추천한다.

do while

- do while

  do{ 조건식/값이 참일 때 반복되는 로직; } while(조건식 & 논리값)

  - while문에서 변형된 형태
  - 반복하는 시작 지점에는 do라는 문장이 붙고 while이 뒤에 붙는다.
  - 조건이 아래에 붙어 있어 일단 로직을 우선 실행 후에 조건 비교를 하는 것이 특징이다.
  - 만약 무조건 로직이 한번은 실행되어야 한다면 do while문을 쓰는것이 좋다.

do while

- do while 예제

```
Run | Debug
public static void main(String[] args) {
    // do while 예제
    int a = 0;
    do{
        System.out.println(a);
        a++;
    }while(a < 3);
}
```

끝에서 다시 비교로 이동

a = 0

1번째 루프

출력 => 0
a 값 1 증가.

a = 1 후 비교

2번째 루프

출력 => 1
a 값 1 증가.

a = 2 후 비교

3번째 루프

출력 => 2
a 값 1 증가.

a = 3 후 비교

루프 종료

For

- while 한계점
  - 기존의 while문은 카운트 변수를 밖에 선언해서 while문 안에서 증가하는 형식으로 갔었다.
  - 중간에 break와 continue를 쓰게 되면 밖의 카운트 변수의 결과가 달라지는 현상을 볼 수 있었다.
  - 이러한 카운트 변수의 관리가 쉽지 않았고 또한 로직이 복잡하면 복잡해 질 수록 while문은 카운트 변수와의 가독성이 급격하게 내려가는 단점이 존재한다.

For

- for

  for(초기변수;조건식;증감식){ 조건식이 참일 때 반복되는 로직; }

  - while문의 비교문에 확장해서 카운트 변수와 증감 값을 같이 관리할 수 있는 반복문이 생겼으며 그것이 For문이다.
  - for문은 while문과 다르게 카운트 변수와 증감 값을 함께 관리할 수 있도록 설계된 반복문이다.
  - 반드시 카운트 변수를 두고 관리를 해야하는 반복문 이라면 while문 보다는 for문을 쓰는것이 더 좋을 수 있다.

For

for 문 예제

```
Run | Debug
public static void main(String[] args) {
    // for문 예제
    for (int i = 0; i < 5; i++) {
        System.out.println(i);
    }
}
```

```
0
1
2
3
4
```

For

- for
  - for문의 괄호 안에는 순차적으로 초기변수, 조건식, 증감식이 들어간다.
  - 초기변수와 조건식, 증감식 사이에는 이들을 구분해주기 위한 세미콜론(;)이 사이에 들어가며 반드시 for문 안에는 두개의 세미콜론(;)이 존재해야 한다.
  - 실제 초기변수는 외부에서 선언한 것을 쓸 수도 있으며 이때 초기변수는 생략이 가능하다.
  - 조건식도 생략이 가능한데 이때 조건식의 값은 무조건 true라 인식하고 무한 반복을 하도록 유도한다.
  - 증감식은 증감연산자(++, --) 외에도 대입 연산자를 통해 복잡한 증감연산도 가능하며 증감식도 생략이 가능하다.

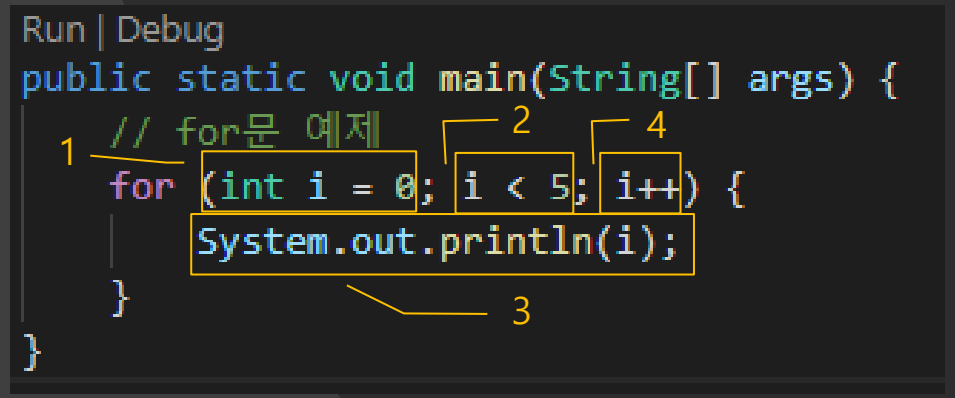
For

for문 예제

```java
// for문의 세부 활용법
// for문의 3개의 칸에는 각각 초기변수;조건식;증감식
// 이 위치할 수 있으며 이 3가지는 생략이 가능하다.

// 초기값 생략이 가능하다.
// 하지만 반드시 for문에 초기값이 필요한 경우라면
// while문 처럼 초기값을 상위에 선언해 놓고
// 사용이 가능하다.
int i = 0;
for( ; i<5 ; i++){
    System.out.println(i);
}

// 조건식 생략이 가능하다.
// 조건식 생략 시 조건의 결과는 무조건 true라
// 판단하고 적절하게 break문을 통해 끊는 것이
// 중요하다.
for (int j = 0; ; i++) {
    if(j==5)break;
    System.out.println(j);
}

// 증감식 생략이 가능하다.
// 증감식 생략 시 while문 처럼 따로 증감식을 넣어
// 내부에서 돌 수 있도록 하거나 다른 로직을 걸어
// for문을 무한 루프에서 벗어날 수 있도록 해야 한다.
for (int k = 0; k < 5; ) {
    System.out.println(k);
    k++;
}
```

For

- for문 읽기
  - for문은 다음 순서를 통해 실행된다
    - 1. 초기값 설정
    - 2. 비교문 실행
    - 3. 로직 실행
    - 4. 증감값 실행
    - 5. 위의 2부터 다시 반복



For

- for문 읽기

```
Run | Debug
public static void main(String[] args) {
    // for문 예제
    for (int i = 0; i < 5; i++) {
        System.out.println(i);
    }
}
```

i = 0

1번째 루프

출력 => 0

i 값 1 증가.

i = 1

2번째 루프

출력 => 1

i 값 1 증가.

i = 2

3번째 루프

출력 => 2

i 값 1 증가.

i = 3

4번째 루프

출력 => 3

i 값 1 증가.

i = 4

5번째 루프

출력 => 4

i 값 1 증가.

i = 5

루프 종료

For

- for문 읽기

```
Run | Debug
public static void main(String[] args) {
    // for문 증감식 확장
    for (int i = 64; i > 4; i/=2) {
        System.out.println(i);
    }
}
```

i = 64

1번째 루프

출력 => 64
i 값 2 분.

i = 32

2번째 루프

출력 => 32
i 값 2 분.

i = 16

3번째 루프

출력 => 16
i 값 2 분.

i = 8

4번째 루프

출력 => 8
i 값 2 분.

i = 4

루프 종료

For

- Break, Continue
  - while 문에서 Break와 Continue를 쓰듯이 For문에서도 Break와 Continue를 쓰는 것이 가능하다.
  - while 문과는 달리 for문에서는 증감연산을 ()안에 넣어 관리하기 때문에 Continue나 Break문을 통해서 실제 연산이 무시되는 것을 고려하지 않아도 된다.
  - while문 보다 관리가 쉽기 때문에 Break문과 Continue문 사용이 용이하다.

For

Break, Continue 예제

```
Run | Debug
public static void main(String[] args) {
    // for문에서 break 사용하기
    // 아까 무한 반복되는 for문의 경우 break문을 통해서
    // 중간에 끊어 줄 수 있다.
    for (int i = 0; ; i++) {
        if(i==5)break;
        System.out.print(i+"\t");
    }
    System.out.println();
    // for 문에서 continue 사용하기
    // while문 하고 달리 for문에서는 continue라는 문장
    // 자체를 좀 더 자유롭게 사용이 가능하다.
    for (int i = 0; i < 10; i++) {
        if(i==5)continue;
        System.out.print(i+"\t");
    }
    System.out.println();
}
```

```
0       1       2       3       4
0       1       2       3       4       6       7       8       9
```

다중반복문

- 다중 반복문
  - 반복문 안에 반복문이 있는 형태
  - 반복문은 안에 다수개의 반복문을 넣을 수 있고 중첩해서 넣을 수 있는 반복문은 제한이 없다.
  - 하지만 반복문 안에 반복문을 넣게 되면 가독성이 매우 떨어지고 속도가 상당히 떨어지는 단점이 있으므로 되도록이면 2중 반복문까지 넣는 것을 원칙으로 한다.
  - 처음 접할 시 가장 읽기 어려운 반복문의 형태 중 하나.
  - 하지만 자주 쓰이는 형태이므로 최대한 숙달을 시켜야 한다.

다중반복문

- while문 에서의 다중 반복문

```
while(조건식 & 논리값){
        while(조건식 & 논리값){
                조건식/값이 참일 때 반복되는 로직;
        }
}
```

- for문 에서의 다중 반복문

```
for(초기변수;조건식;증감식){
        for(초기변수;조건식;증감식){
                조건식이 참일 때 반복되는 로직;
        }
}
```

다중 반복문

- while문 읽기

```java
Run | Debug
public static void main(String[] args) {
    // while문을 통한 다중 반복문 예제
    int a = 1;
    while(a <= 5){
        int b = 1;
        while (b <= 5) {
            System.out.println("a : "+a+"  b : "+b);
            b++;
        }
        a++;
    }
}
```

a = 1

상위 1번째 루프

b = 1

하위 1번째 루프

출력 a : 1 b : 1

b 값 1 증가. 후 비교

b = 2

하위 2번째 루프

출력 a : 1 b : 2

b 값 1 증가. 후 비교

b = 3

하위 3번째 루프

출력 a : 1 b : 3

b 값 1 증가. 후 비교

b = 4

하위 4번째 루프

출력 a : 1 b : 4

b 값 1 증가. 후 비교

b = 5

하위 5번째 루프

출력 a : 1 b : 5

b 값 1 증가. 후 비교

b = 6

루프 종료

a 값 1 증가

다중 반복문

- while문 읽기

```
Run | Debug
public static void main(String[] args) {
    // while문을 통한 다중 반복문 예제
    int a = 1;
    while(a <= 5){
        int b = 1;
        while (b <= 5) {
            System.out.println("a : "+a+"  b : "+b);
            b++;
        }
        a++;
    }
}
```

a = 2

상위 2번째 루프

b = 1 — 재초기화

하위 1번째 루프

출력 a : 2 b : 1

b 값 1 증가. 후 비교

b = 2

하위 2번째 루프

출력 a : 2 b : 2

b 값 1 증가. 후 비교

b = 3

하위 3번째 루프

출력 a : 2 b : 3

b 값 1 증가. 후 비교

b = 4

하위 4번째 루프

출력 a : 2 b : 4

b 값 1 증가. 후 비교

b = 5

하위 5번째 루프

출력 a : 2 b : 5

b 값 1 증가. 후 비교

b = 6

루프 종료

a 값 1 증가

다중 반복문

- while문 읽기

```
Run | Debug
public static void main(String[] args) {
    // while문을 통한 다중 반복문 예제
    int a = 1;
    while(a <= 5){
        int b = 1;
        while (b <= 5) {
            System.out.println("a : "+a+"  b : "+b);
            b++;
        }
        a++;
    }
}
```

a = 5

상위 5번째 루프

b = 1

하위 1번째 루프

출력 a : 5 b : 1

b 값 1 증가. 후 비교

b = 2

하위 2번째 루프

출력 a : 5 b : 2

b 값 1 증가. 후 비교

b = 3

.....

b = 5

하위 5번째 루프

출력 a : 1 b : 5

b 값 1 증가. 후 비교

b = 6

루프 종료

a 값 1 증가

a = 6

상위 루프 종료

```
a : 1  b : 1
a : 1  b : 2
a : 1  b : 3
a : 1  b : 4
a : 1  b : 5
a : 2  b : 1
a : 2  b : 2
a : 2  b : 3
a : 2  b : 4
a : 2  b : 5
a : 3  b : 1
a : 3  b : 2
a : 3  b : 3
a : 3  b : 4
a : 3  b : 5
a : 4  b : 1
a : 4  b : 2
a : 4  b : 3
a : 4  b : 4
a : 4  b : 5
a : 5  b : 1
a : 5  b : 2
a : 5  b : 3
a : 5  b : 4
a : 5  b : 5
```

다중 반복문

- for문 읽기

```
Run | Debug
public static void main(String[] args) {
    // for문을 통한 다중 반복문 예제
    for (int a = 1; a <= 5; a++) {
        for (int b = 1; b <= 5; b++) {
            System.out.println("a : "+a+"  b : "+b);
        }
    }
}
```

a = 1

상위 1번째 루프

b = 1

하위 1번째 루프

출력 a : 1 b : 1

b 값 1 증가. 후 비교

b = 2

하위 2번째 루프

출력 a : 1 b : 2

b 값 1 증가. 후 비교

b = 3

하위 3번째 루프

출력 a : 1 b : 3

b 값 1 증가. 후 비교

b = 4

하위 4번째 루프

출력 a : 1 b : 4

b 값 1 증가. 후 비교

b = 5

하위 5번째 루프

출력 a : 1 b : 5

b 값 1 증가. 후 비교

b = 6

루프 종료

a 값 1 증가

다중 반복문

- for문 읽기

```
Run | Debug
public static void main(String[] args) {
    // for문을 통한 다중 반복문 예제
    for (int a = 1; a <= 5; a++) {
        for (int b = 1; b <= 5; b++) {
            System.out.println("a : "+a+"  b : "+b);
        }
    }
}
```

a = 2

상위 2번째 루프

b = 1 — 재초기화

하위 1번째 루프

출력 a : 2 b : 1

b 값 1 증가. 후 비교

b = 2

하위 2번째 루프

출력 a : 2 b : 2

b 값 1 증가. 후 비교

b = 3

하위 3번째 루프

출력 a : 2 b : 3

b 값 1 증가. 후 비교

b = 4

하위 4번째 루프

출력 a : 2 b : 4

b 값 1 증가. 후 비교

b = 5

하위 5번째 루프

출력 a : 2 b : 5

b 값 1 증가. 후 비교

b = 6

루프 종료

a 값 1 증가

다중 반복문

- for문 읽기

```
Run | Debug
public static void main(String[] args) {
    // for문을 통한 다중 반복문 예제
    for (int a = 1; a <= 5; a++) {
        for (int b = 1; b <= 5; b++) {
            System.out.println("a : "+a+"  b : "+b);
        }
    }
}
```

a = 5

상위 5번째 루프

b = 1

하위 1번째 루프

출력 a : 5 b : 1

b 값 1 증가. 후 비교

b = 2

하위 2번째 루프

출력 a : 5 b : 2

b 값 1 증가. 후 비교

b = 3

.....

b = 5

하위 5번째 루프

출력 a : 1 b : 5

b 값 1 증가. 후 비교

b = 6

루프 종료

a 값 1 증가

a = 6

상위 루프 종료

```
a : 1  b : 1
a : 1  b : 2
a : 1  b : 3
a : 1  b : 4
a : 1  b : 5
a : 2  b : 1
a : 2  b : 2
a : 2  b : 3
a : 2  b : 4
a : 2  b : 5
a : 3  b : 2
a : 3  b : 3
a : 3  b : 4
a : 3  b : 5
a : 4  b : 1
a : 4  b : 2
a : 2  b : 4
a : 2  b : 5
a : 3  b : 1
a : 3  b : 2
a : 3  b : 3
a : 3  b : 4
a : 3  b : 5
a : 4  b : 1
a : 4  b : 2
a : 4  b : 3
a : 4  b : 4
a : 4  b : 5
a : 5  b : 1
a : 5  b : 2
a : 5  b : 3
a : 5  b : 4
a : 5  b : 5
```

변수의 생존 범위

- 변수의 생존 범위
  - 이전시간에 우리는 브레이스({})를 기준으로 변수가 생존하고 소멸하는 것을 봤었다.
  - 하지만 우리가 조건문과 반복문을 쓰면서 실제 if문에도 브레이스가 쓰이는 걸 보았고 반복문에도 브레이스가 쓰이는 것을 보았다.
  - 조건문과 반복문에 쓰이는 브레이스도 일종의 구역으로 인식되며 그 안에 선언된 변수 또한 생존 소멸범위가 해당 브레이스 내에서 지정된다.

변수의 생존 범위

if문의 변수 생존범위

```java
// if문에서의 변수 생존 범위
if(true){
    int a = 1;
    if(true){
        int b = 2;
        if(true){
            int c = 3;
            System.out.println(a);
            System.out.println(b);
            System.out.println(c);
        }
        System.out.println(a);
        System.out.println(b);
        // System.out.println(c); // 에러
    }
    System.out.println(a);
    // System.out.println(b); // 에러
    // System.out.println(c); // 에러
}
```

```java
// for문의 생존 범위
for (int i = 0; i < 1; i++) {
    for (int j = 0; j < 1; j++) {
        for (int k = 0; k < 1; k++) {
            System.out.println(i);
            System.out.println(j);
            System.out.println(k);
        }
        System.out.println(i);
        System.out.println(j);
        // System.out.println(k); // 에러
    }
    System.out.println(i);
    // System.out.println(j); // 에러
    // System.out.println(k); // 에러
}
```