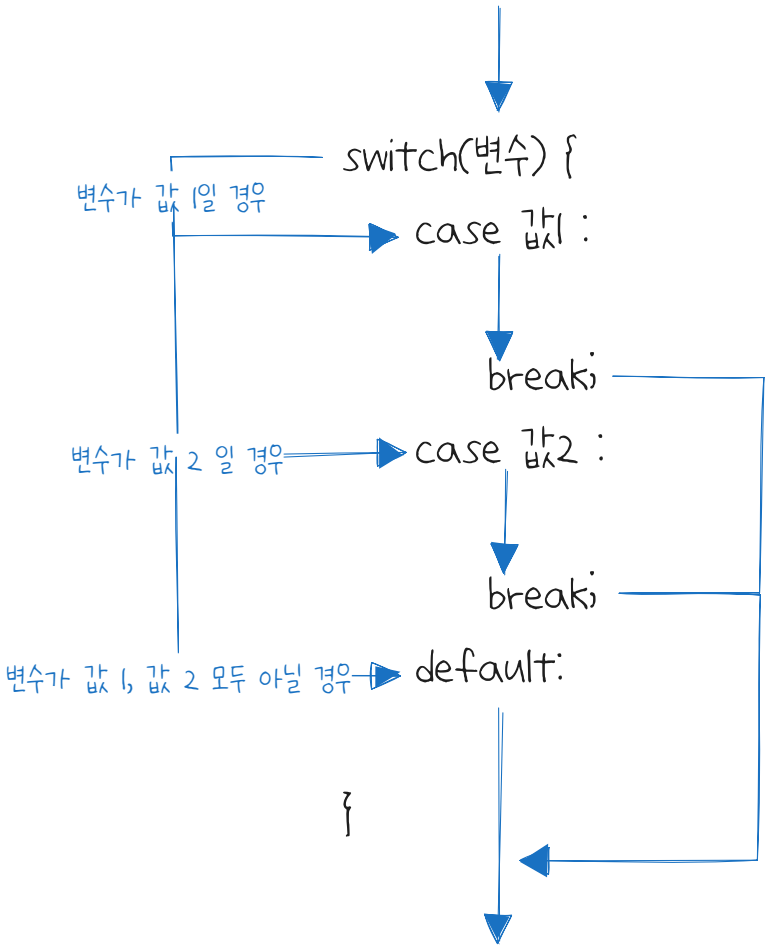
# 3rd-week

## switch 문

> switch 문의 괄호에는 `정수타입(byte, char, short, int, long)`과 `문자열 타입(String)` 변수를 사용할 수 있다.




- 만약 변수와 동일한 값을 갖는 case가 없으면 default를 실행시킨다.(default는 필요없다면 생략 가능하다.)
- break는 다음 case를 실행하지 않고 switch 문을 빠져나가기 위해 필요하다. 만약 case 끝에 break가 없다면 case가 연달아 실행되는데, 이때는 case 값과 상관없이 실행된다.

```
int num = 9;

switch(num) {
        case 8:
                System.out.println("8입니다.");
        case 9:
                System.out.println("9입니다.");
        case 10:
                System.out.println("10입니다.");
        default:
                System.out.println("초기값입니다.");
}
```

```
>>9입니다.
>>10입니다.
>>초기값입니다.
```

## Switch Expressions

- Java 12 이후 부터는 Expressions(표현식)을 사용할 수 있다.

```
char grade = 'B';

switch(grade) {
        case 'A', 'a' -> {
                System.out.println("우수 회원입니다.");
        }
        case 'B', 'b' -> {
                System.out.println("일반 회원입니다.");
        }
        default -> {
                System.out.println("손님입니다.");
        }
}

// 중괄호 안에 실행문이 하나만 있을 경우에는 중괄호를 생략할 수 있다.
switch(grade) {
        case 'A', 'a' -> System.out.println("우수 회원입니다.");
        case 'B', 'b' -> System.out.println("일반 회원입니다.");
        default -> System.out.println("손님입니다.");
}
```

- Switch Expressions을 변수에 바로 대입 가능
  - 중괄호를 사용하는 경우에 `yield`(Java 13부터 사용가능) 키워드로 값을 지정
    → 이 경우 `default`가 반드시 존재해야한다.

```
타입 변수 = switch(grade) {
    case "값 1" -> 변수값;
    case "값 2" -> {
        ...;
        yield 변수값;
    }
    default -> 변수값;
};
```

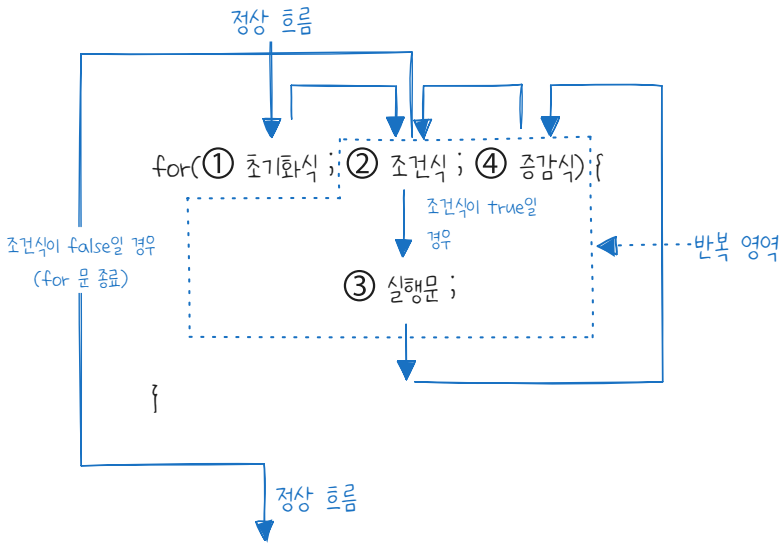
```
String grade = "B";

//Java 12부터 가능
int score = switch(grade) {
 case "A" -> 100;
```

```
 case "B" -> {
         int result = 100 - 20;
         // Java 13부터 가능
         yield result;
 }
 default -> 60;
};
System.out.println("score: : + score);
>>80
```

여기서 `yield`는 switch문의 끝내는 `break`의 역할과 값을 반환하는 `return`의 역할을 해준다.

---

# for문




[순서]

- ①초기화식이 제일 먼저 실행된다.
- 그 이후 ②조건식을 평가해서 false이면 실행문을 건너뛰고 for문을 종료한다.
- ②조건식이 true이면 ③실행문을 실행시키고, ④증감식 실행 후 ②조건식을 평가한다.
- ②조건식이 true인지 false인지에 따라 위에 동작을 반복한다.

```
for(int i=1; i<=10; i++){
        System.out.print(i + " ");
}
>>1 2 3 4 5 6 7 8 9 10
```

- 초기화식과 증감식은 둘 이상 있을 수 있다. 이 경우 `,`로 구분해서 작성한다.

```
for ( int i = 0, j = 100 ; i <= 50 && j >= 50 ; i++, j++) {···}
      초기화식              조건식              증감식
```

- 초기화식에서 선언된 변수는 for 문 블록 안에서만 사용되는 로컬 변수이므로 for 문을 벗어나서도 사용하고 싶다면 초기화식에서 변수를 선언하지 말고 for 문 이전에 선언해야 한다.

```
int i;
for(i = 1; i<=100;i++){…}
System.out.print("최종 i값 : " + i);
```

- 초기화식에서 부동 소수점을 쓰는 float타입을 사용하지 말아야 한다.
  → 부동 소수점 방식의 float타입은 연산 과정에서 정확히 0.1을 표현하지 못하기 때문에 증감식에서 x에 더해지는 실제 값은 0.1보다 약간 클 수 있다. 따라서 아래 코드의 최종 반복 횟수는 9번이 된다.

```
for(floate x=0.1f; x<=1.0f; x+=0.1f){
        System.out.print(x + " ");
}
>> 0.1 0.2 0.3 0.4 0.5 0.6 0.7000005 0.800001 0.90001
```

## 중첩 for 문

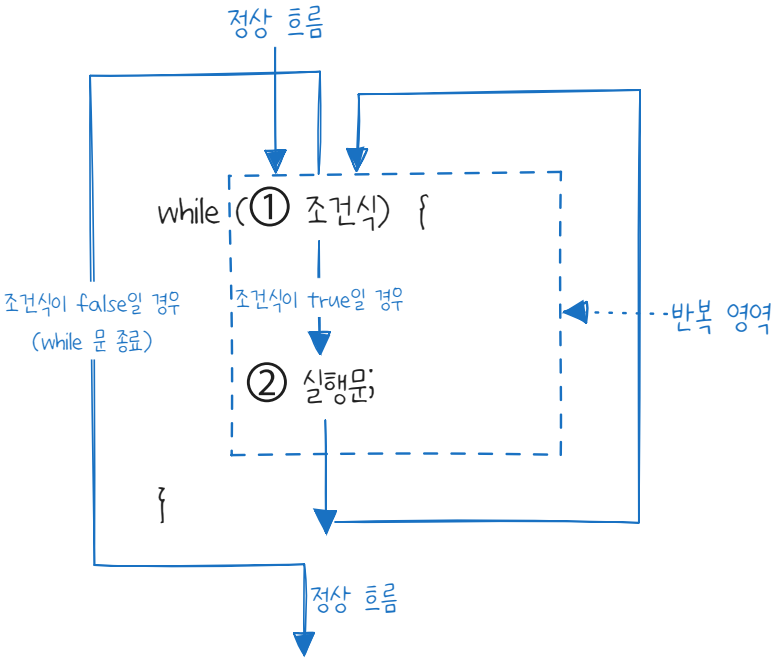
for문은 for문 안에 for문을 내포할 수 있다.

```
for(int m=2; m<=9; m++){
        System.out.println("*** "+m+"단 ***");
        for(int n=1; n<=9; n++){
                System.out.println(m+" x "+n+" = "+(m * n));
        }
}
>>*** 2단 ***
>>2 x 1 = 2
>>2 x 2 = 4
>>...
```

---

## while 문

> 조건식이 true일 경우에 계속해서 반복하고, false가 되면 반복을 멈추고 while문을 종료한다.



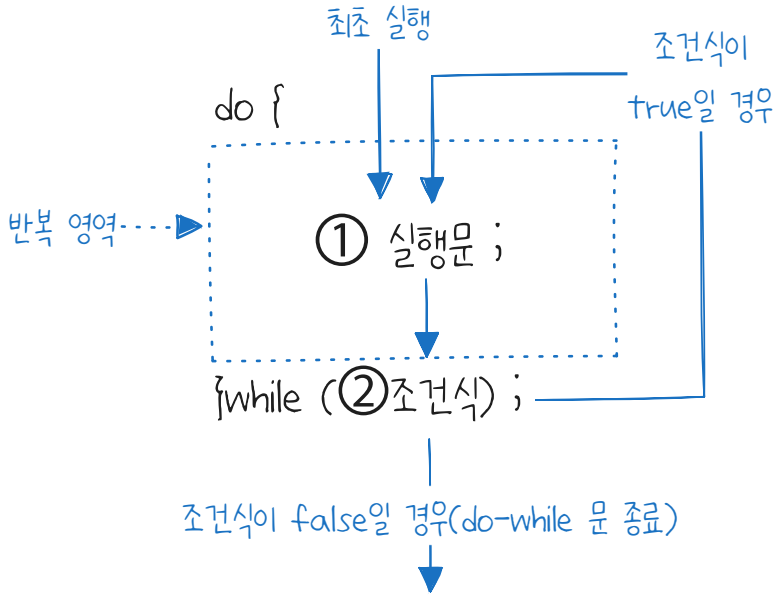

[순서]

- 처음 실행 시 ①조건식을 평가한다. 평가 결과가 true이면 ②실행문을 실행한다.
- ②실행문이 모두 실행되면 조건식으로 되돌아가서 ①조건식을 다시 평가한다.
- 다시 조건식이 true라면 ②→①로 진행하고 false라면 while문을 종료한다.

```
int i = 1;
while(i<=10){
        System.out.print(i + " ");
        i++;
}
>> 1 2 3 4 5 6 7 8 9 10
```

---

# do-while 문




[순서]

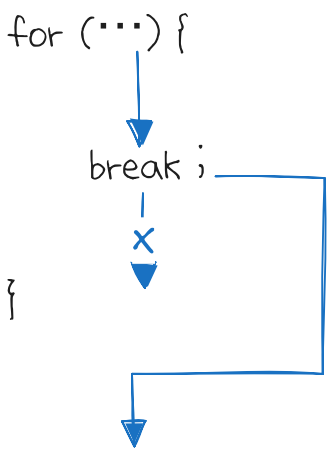
- ①실행문을 우선 실행한다. ①실행문이 모두 실행되면 ②조건식을 평가한다.
- ②조건식의 결과가 true이면 ①→②와 같이 반복 실행하고, 조건식의 결과가 false이면 do-while문을 종료한다.

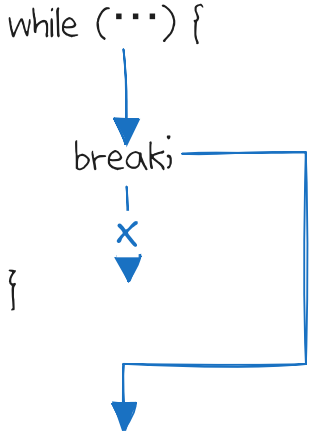
```
Scanner sc = new Scanner(System.in);

String inputString;

do {
        System.out.print(">>");
        inputString = sc.nextLine();
        System.out.println(inputString);
} while (!inputString.equals("q"));

System.out.println("프로그램 종료");
```

---

# break

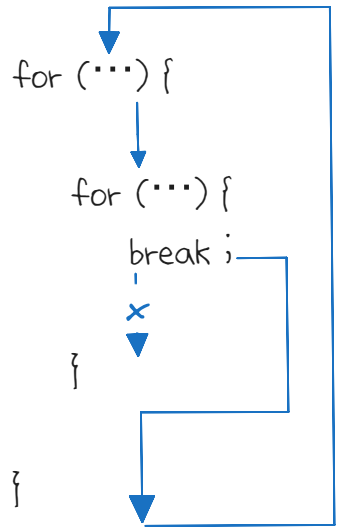
break은 반복문인 for 문, while 문, do-while 문을 실행 중지하거나 조건문인 switch문을 종료할 때 사용한다.

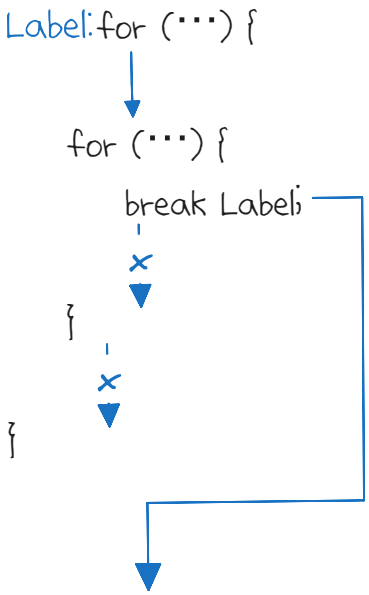



```
while(true) {
        int num = (int)(Math.random() * 6) + 1;
        System.out.println(num);
        if(num == 6){
                break;
        }
}
System.out.println("프로그램 종료");
>>4
>>2
>>6
>>프로그램 종료
```

- 반복문이 중첩되어 있을 경우 break문은 가장 가까운 반복문만 종료하고 바깥쪽 반복문은 종료시키지 않는다. 바깥쪽 반복문까지 종료시키려면 바깥쪽 반복문에 이름(레이블)을 붙이고 `break 이름;`을 사용하면 된다.




- 레이블을 사용하지 않은 break;

```
for(char upper='A'; upper<='Z'; upper++) {
        for(char lower='a'; lower <= 'z'; lower++) {
                System.out.println(upper + "-" + lower);
                if(lower =='c'){
```

```
                        break;
                }
        }
}
System.out.println("프로그램 종료");
>>A-a
>>A-b
>>A-c
>>B-a
>>B-b
>>B-c
(...생략...)
>>Y-a
>>Y-b
>>Y-c
>>Z-a
>>Z-b
>>Z-c
>>프로그램 종료
```

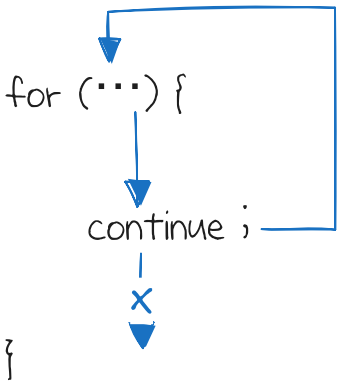
- 레이블을 사용한 break;

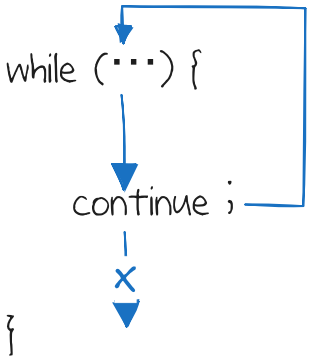
```
Outter: for(char upper='A'; upper<='Z'; upper++) {
        for(char lower='a'; lower <= 'z'; lower++) {
                System.out.println(upper + "-" + lower);
                if(lower =='c'){
                        break Outter;
                }
        }
}
System.out.println("프로그램 종료");
>>A-a
>>A-b
>>A-c
>>프로그램 종료
```

---

# continue문

> continue문은 반복문인 for문, while 문, do-while 문에서만 사용되고, continue문이 실행되면 continue 이후의 실행문을 실행시키지 않고 바로 조건식으로 이동한다.




- 주로 if문과 같이 사용되며 특정 조건이 만족하면 continue를 실행시켜 뒤에 실행문을 실행시키지 않고 바로 조건식으로 넘어간다.

```
for(int i=1;i<=10;i++){
        if(i%2 != 0){
                continue;
        }
        System.out.print(i + " ");
}
>>2 4 6 8 10
```

---

# break vs continue vs return

| `break` | `continue` | `return` |
|---|---|---|
| - 반복문인 for 문, while 문, do-while 문을 실행 중지하거나 조건문인 if문, switch문을 종료할 때 사용 | -반복문인 for문, while 문, do-while 문에서만 사용 | - 메소드에 사용 |
| - 조건문이나 반복문을 종료시킴<br>- 중첩 for문의 경우 가장 가까운 for문만 중지시키고 바깥 for문은 그래도 반복 유지됨. | - `continue` 이후의 실행문은 실행하지 않고 바로 증감식 실행. 증감식이 없는 경우 바로 조건식을 평가함. | - 현재 `메소드` 를 즉시 종료. return 뒤에 값이 있으면 그 값을 메소드의 결과로 반환 |