

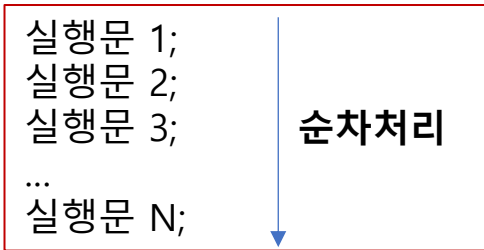
제어문과 제어키워드

제어문과 제어키워드 - 제어문

☞ 제어문의 개념

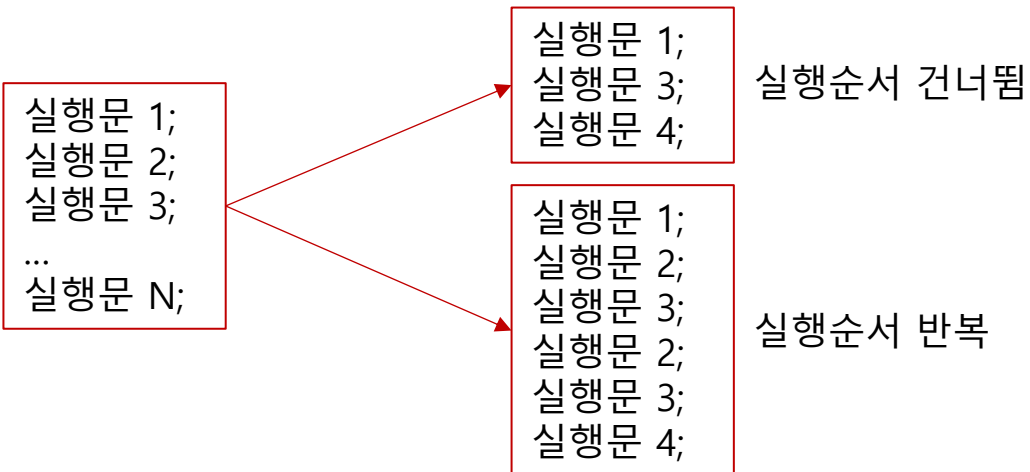
- 일반적인 프로그램 처리 방식

1



- 프로그램 실행 순서 제어 → 제어문

2

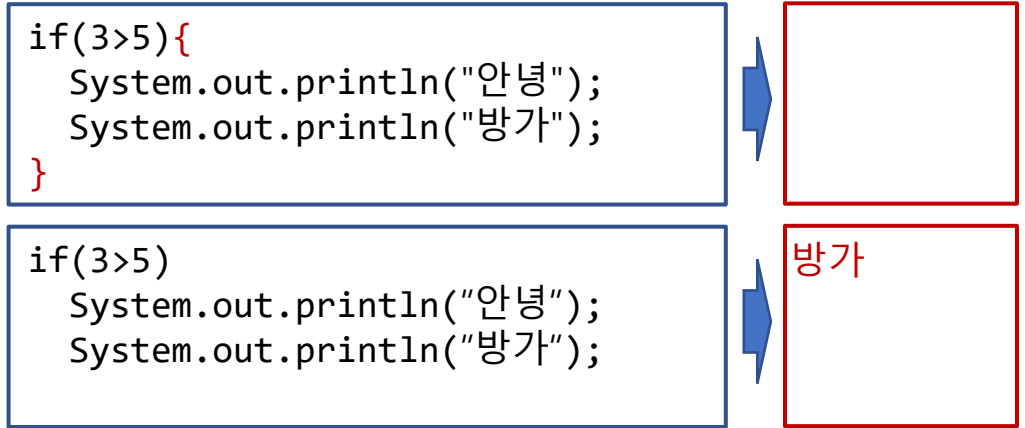


TIP

4

- 모든 제어문(5종류)은 중괄호({ })를 가짐
- 단, 중괄호 안의 **실행문이 1개인 경우 생략 가능**
(컴파일러에 의해서 자동으로 삽입)

5



3

- 제어문

- 1. if
 - 2. switch
 - 3. for
 - 4. while
 - 5. do-while
- ← 선택제어문 (if, switch)
- ← 반복제어문 (for, while, do-while)

- 제어 키워드

- 1. break
- 2. continue

선택제어문

제어문과 제어키워드 - 제어문

TIP

👉 **if** 제어문 (선택제어문)

- TYPE 2는 삼항연산자와 상호변환 가능
- TYPE 3은 조건식을 위에서 부터 순차적으로 검사 후 **처음 참(true)이 나오는 블록만 실행 후 if문 탈출** (TYPE2가 중복되어 있는 개념)

TYPE 1

1 `if(조건식) {` → boolean(true, false) 타입만 가능
실행내용; → 조건식이 true일 때만 실행
`}`

TYPE 2

2 `if(조건식) {` → boolean(true, false) 타입만 가능
실행내용; → 조건식이 true일 때 실행
`}`
`else {` → 생략가능 (생략시 TYPE 1)
실행내용; → 조건식이 false일 때 실행
`}`

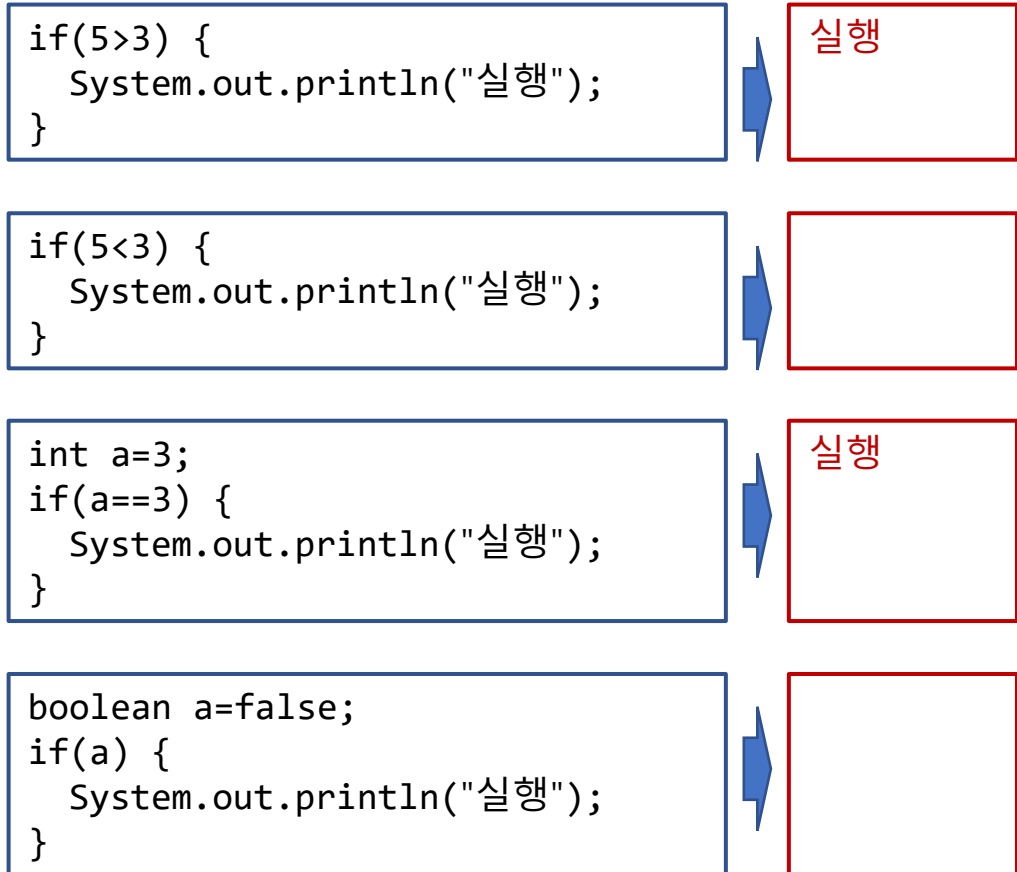
TYPE 3

3 `if(조건식1) {`
실행내용; → 조건식1이 true일 때 실행
(실행 후 제어문 탈출)
`}`
`else if(조건식2) {` → 생략가능 (생략시 TYPE 2)
실행내용; → 조건식2가 true일 때 실행
(실행 후 제어문 탈출)
`}`
`else if(조건식3) {`
실행내용; → 조건식3이 true일 때 실행
(실행 후 제어문 탈출)
`}`
...
`else {`
실행내용; → 모든 조건식이 false일 때 실행
`}`

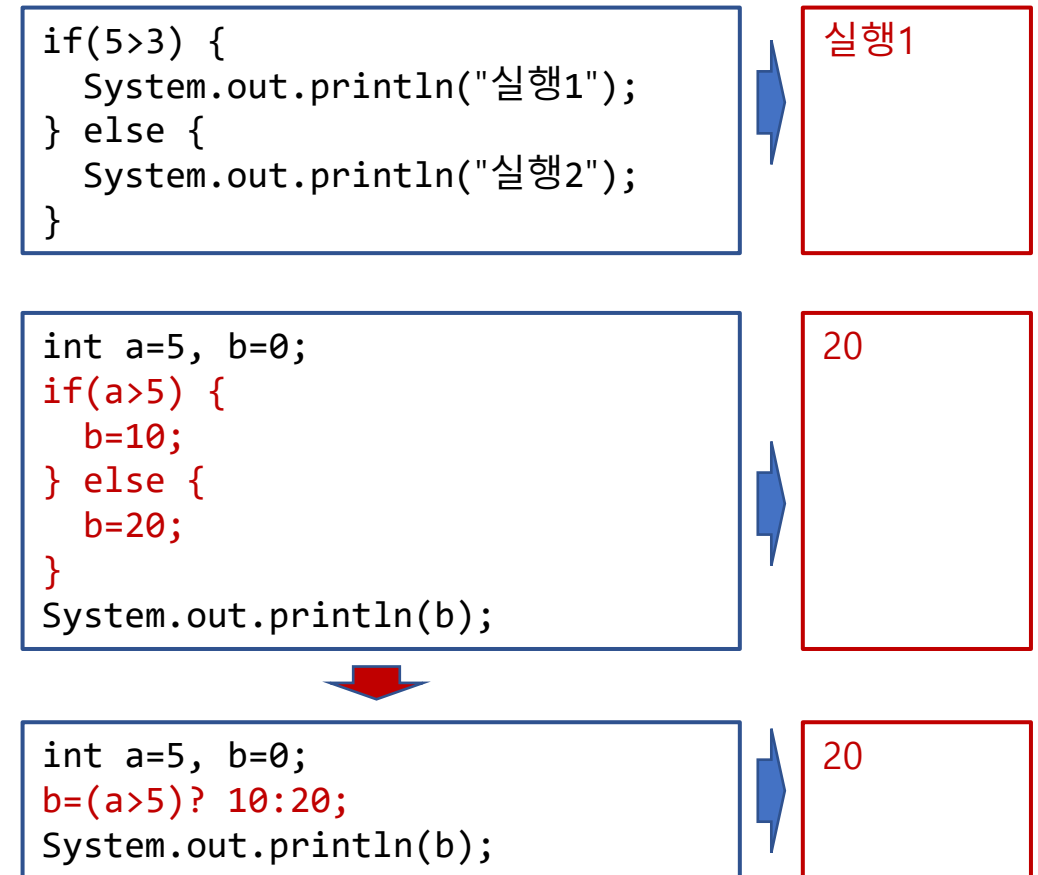
제어문과 제어키워드 - 제어문

☞ if 제어문 (선택제어문)

1 TYPE 1



2 TYPE 2



제어문과 제어키워드 - 제어문

☞ **if** 제어문 (선택제어문)

TYPE 3

1

```
int a=85;
if(a>=90) {
    System.out.println("A학점");
}
else if(a>=80) {
    System.out.println("B학점");
}
else if(a>=70) {
    System.out.println("C학점");
}
else {
    System.out.println("F학점");
}
```

B학점

TIP

4 - (70<=a<80)과 같은 표현은 불가능
(70<=a<80) → (a>=70 && a<80)

2

```
int a=85;
if(a>=70) {
    System.out.println("C학점");
}
else if(a>=80) {
    System.out.println("B학점");
}
else if(a>=90) {
    System.out.println("A학점");
}
else {
    System.out.println("F학점");
}
```

C학점

3

```
int a=85;
if(a>=70 && a<80) {
    System.out.println("C학점");
}
else if(a>=80 && a<90) {
    System.out.println("B학점");
}
else if(a>=90) {
    System.out.println("A학점");
}
else {
    System.out.println("F학점");
}
```

B학점

제어문과 제어키워드 - 제어문

👉 **switch** 제어문 (선택제어문)

TIP

2

- 0개 이상의 case 절과 0이나 1개의 default 절로 구성
- 연산식: 정수, 문자, 문자열



1

기본 문법 구조

```
switch(점프위치변수) { → 정수, 문자, 문자열
  case 위치값1: → 점프위치변수=위치값1이면 이 위치로 점프
                실행내용;
  case 위치값2: → 점프위치변수=위치값2이면 이 위치로 점프
                실행내용;
                ...
                case 구문은 콜론(:)으로 끝남
  case 위치값n: → 점프위치변수=위치값n이면 이 위치로 점프
                실행내용;
  default: → 일치하는 위치값이 없는 경우 이 위치로 점프
            실행내용;
            생략가능(if-else에서 else에 해당)
}
```

콜론(:)의 개념적 의미

3

- Java 프로그램에서 콜론(:)은 이점표의 역할 (점프위치 표시)
- 삼항연산자, case, 레이블 및 람다식
- switch문 탈출을 위해서는 **break** 키워드 사용

4

TIP

- switch문은 if문과 상호 변환 가능

제어문과 제어키워드 - 제어문

👉 **switch** 제어문 (선택제어문)

1

switch의 역할

- 특정 위치로 점프시키는 역할

```
int a=2;
switch(a){
case 1:
    System.out.println("A");

case 2:
    System.out.println("B");

case 3:
    System.out.println("C");

default:
    System.out.println("D");
}
```



```
B
C
D
```

2

3

break의 의미

- if문을 제외한 가장 가까운 중괄호({ })를 탈출

4

```
int a=2;
switch(a){
case 1:
    System.out.println("A");
    break;
case 2:
    System.out.println("B");
    break;
case 3:
    System.out.println("C");
    break;
default:
    System.out.println("D");
}
```



```
B
```

제어문과 제어키워드 - 제어문

👉 **switch** 제어문 (선택제어문)

의도적 break 미사용

- 하나의 실행문에 break을 사용하지 않고 여러 개의 case를 연결하는 경우 코드가 간결해질 수도 있다.

1

```
int a=8;
switch(a){
case 10:
    System.out.println("Pass"); break;
case 9:
    System.out.println("Pass"); break;
case 8:
    System.out.println("Pass"); break;
case 7:
    System.out.println("Pass"); break;
default:
    System.out.println("Fail");
}
```

Pass

3

```
int a=8;
switch(a){
case 10:
case 9:
case 8:
case 7:
    System.out.println("Pass"); break;
default:
    System.out.println("Fail");
}
```

Pass

제어문과 제어키워드 - 제어문

👉 **switch** 제어문 (선택제어문)

1

TIP

- switch문은 if문으로 변환 가능

2

```
int a=8;
switch(a){
case 10:
case 9:
    System.out.println("A");
    break;
case 8:
    System.out.println("B");
    break;
case 7:
    System.out.println("C");
    break;
default:
    System.out.println("D");
}
```



B

3

```
int a=8;
if(a>=9){
    System.out.println("A");
}
else if(a==8){
    System.out.println("B");
}
else if(a==7){
    System.out.println("C");
}
else {
    System.out.println("D");
}
```



B

if 문으로 변환

switch 문으로 변환

Quiz

- switch과 if문 사이의 성능 차이가 있을까?

The End

반복제어문

제어문과 제어키워드 - 제어문

👉 **for** 제어문 (반복문)

1

TIP

- for 문의 경우 반복 횟수가 정해진 경우 주로 사용 (for문을 보면 반복횟수를 알 수 있어야 함)

기본 문법 구조

2

for문이 시작될 때 딱 1번 실행

for문 안으로 들어가는 유일한 출입구 (true인 경우 진입, false이면 for문 종료)

for문의 닫힌괄호(})이 후에 실행

```
for( 초기식; 조건식; 증감식 ) {  
    실행내용; → 조건식이 true인 동안 실행  
}
```

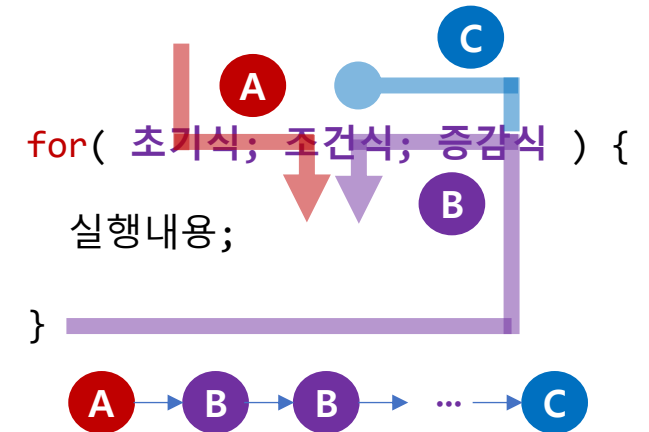
컴파일러는 문법적으로 괄호안에 세미콜론이 2개인지만 체크

4

for문의 실행 순서

- Phase1. 초기식 →
- Phase2. 조건식(참) → 실행문 → 증감식 →
- Phase3. 조건식(참) → 실행문 → 증감식 →
- ...
- PhaseN. 조건식(거짓) → 종료

3



제어문과 제어키워드 - 제어문

👉 **for** 제어문 (반복문)

TIP

- for 문에서 조건식이 비워져 있는 경우 compile는 조건식 **true**를 입력 (항상 참)
- 초기식과 증감식에 쉼표를 사용하여 여러 개 구성 가능 (예, **for(int i=0, j=0; (i+j)<10; i++, j++)** { })
- 무한루프도 break를 이용하여 탈출 가능

2

특수한 형태

조건식을 생략하면 컴파일러는 자동으로 true 입력 (무한루프)

1

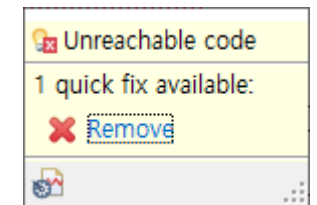
```
for( 초기식; ; 증감식 ) {  
    실행내용;  
}
```

```
for( ; ; ) {  
    실행내용;  
}
```

```
for( ; ; ) {  
    실행내용;  
}  
실행내용;
```

```
for( ;false ; ) {  
    실행내용;  
}
```

오류 발생
(도달할 수 없는 코드)



제어문과 제어키워드 - 제어문

☞ **for** 제어문 (반복문)

1

```
int i;
for(i=0; i<3; i++){
    System.out.println(i);
}
```

0
1
2



```
for(int i=0; i<3; i++){
    System.out.println(i);
}
```

0
1
2

2

```
for(int i=0; i<100; i++){
    System.out.println(i);
}
```

0
1
...
99

3

```
for(int i=10; i>0; i--){
    System.out.println(i);
}
```

10
9
...
1

```
for(int i=0; i<10; i+=2){
    System.out.println(i);
}
```

0
2
...
8

```
for(int i=0, j=0; i<10; i++, j++){
    System.out.println(i+j);
}
```

0
2
...
18

제어문과 제어키워드 - 제어문

☞ **for** 제어문 (반복문)

1

```
for(int i=0; ; i++){  
    실행문;  
}
```



무한루프

```
for(; ;){  
    실행문;  
}
```



무한루프

2

```
for(int i=0; ; i++){  
    if(i>10){  
        break;  
    }  
    System.out.println(i+" ");  
}
```



0
1
2
3
...
10

제어문과 제어키워드 - 제어문

👉 **while** 제어문 (반복문)

TIP

4 - while 문과 for문은 상호 변환 가능

기본 문법 구조

1

필수문법은 아니지만 일반적으로 사용

```
초기식;
while(조건식) {
    실행내용;
    증감식;
}
```

while문 안으로 들어가는 유일한 출입구 (생략불가능) (true인 경우 진입, false이면 while문 종료)

실행내용; → 조건식이 true인 동안 실행

TIP

2

- while 문에서는 조건식을 생략할 수 없음
(cf. for문의 경우 조건식 생략 시 컴파일러가 true입력)

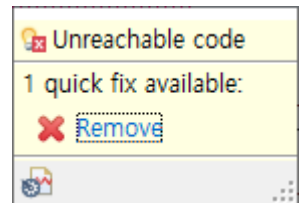
특수한 형태

3

```
while(true) {
    실행내용;
}
실행내용;

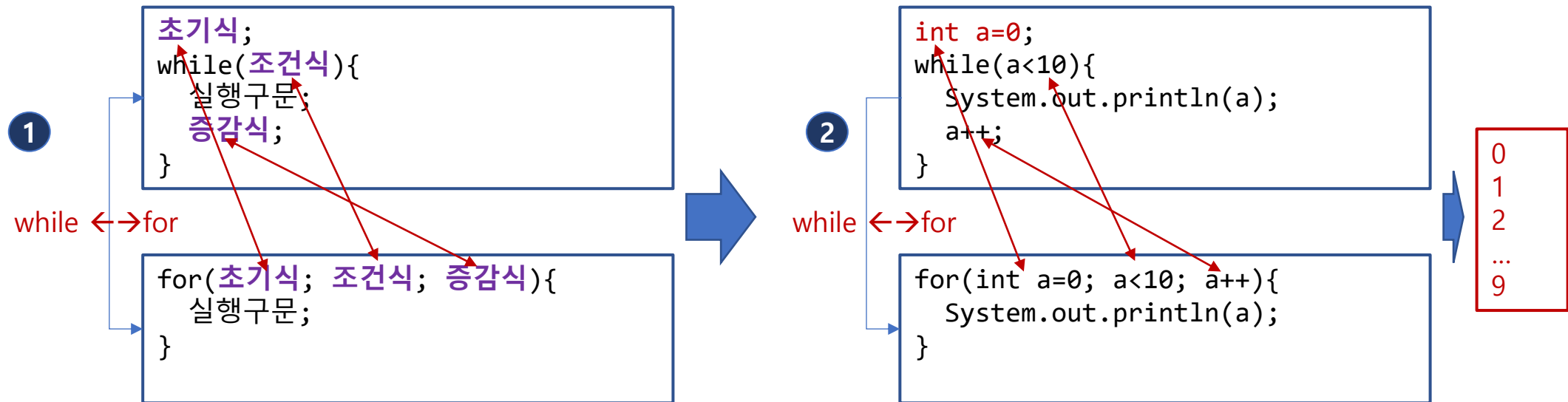
while(false) {
    실행내용;
}
```

오류발생
(도달할 수 없는 코드)



제어문과 제어키워드 - 제어문

☞ **while** 제어문 (반복문)



TIP

3

- 무한루프가 아닌 경우 while 문에는 for문과 같이 초기식, 조건식, 증감식이 모두 포함되어야 함

제어문과 제어키워드 - 제어문

☞ while 제어문 (반복문)

TIP

- while 문의 경우 반복 횟수가 정해지지 않고 **특정 조건까지 반복**하고자 할 때 주로 사용

```
int num=0, sum=0;
while(sum<100){
    sum+=num;
    num++;
}
```

TIP

- 초기식을 while 중괄호 안에 넣으면 원하지 않는 **무한루프**가 될 수 있음

```
int a=0;
while(a<10){
    a=0; //초기식
    System.out.println("A");
    a++; //증감식
}
```

3

```
while(true){
    실행문;
}
```

무한루프

```
int a=0;
while(a<10){
    System.out.println(a);
}
```

무한루프

```
while(false){
    실행문;
}
```

문법오류
(unreachable code)

```
int a=0;
while(true){
    if(a>10){
        break;
    }
    System.out.println(a);
    a++;
}
```

0
1
2
3
4
5
...
10

제어문과 제어키워드 - 제어문

👉 **do-while** 제어문 (반복문)

TIP

3

- do-while 문과 while문은 최초 실행 순서만 상이함

TIP

4

- do-while 문은 제어문 중 유일하게 세미콜론(;)으로 마침 (중괄호({ })로 끝나지 않기 때문)

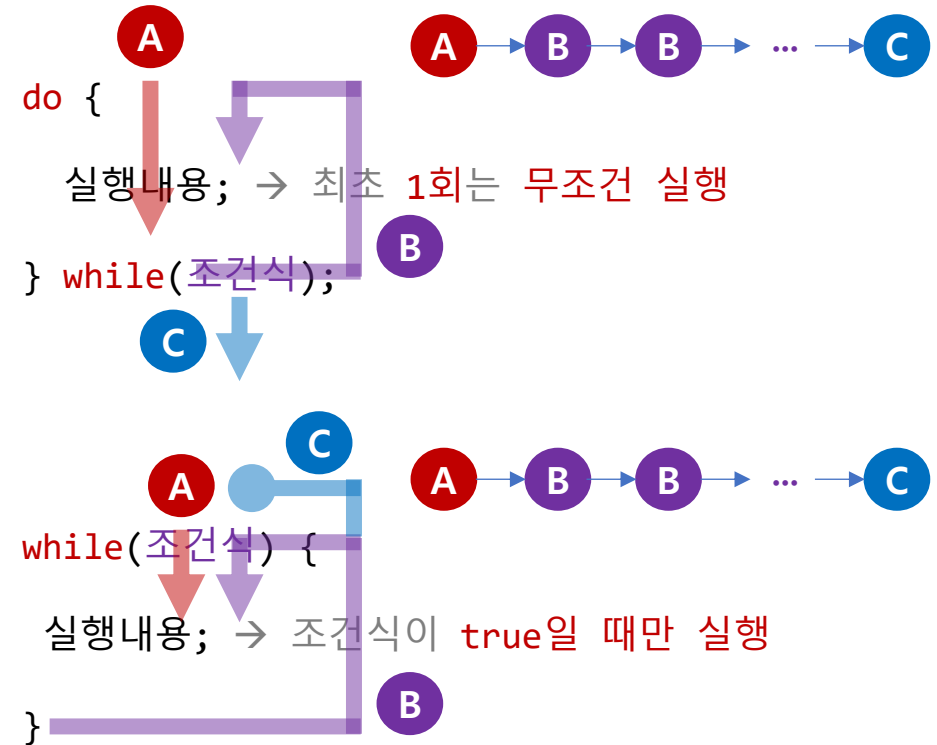
기본 문법 구조

1

```
초기식; → 필수문법은 아니지만 일반적으로 사용
do {
  실행내용; → 최초 1회는 무조건 실행
  증감식;
} while(조건식); → 문법 구조상 중괄호가 없기 때문에 세미콜론(;)으로 끝남
```

while vs. do-while 실행순서 비교

2



제어문과 제어키워드 - 제어문

☞ **do-while** 제어문 (반복문)

1

```
int a=0;
do{
  System.out.println(a);
  a++;
} while(a<10);
```



```
0
1
2
...
9
```

```
int a=0;
do{
  System.out.println(a);
  a++;
} while(a<0);
```



```
0
```

2

```
int a=0;
while(a<10){
  System.out.println(a);
  a++;
}
```



```
0
1
2
...
9
```

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



3

TIP

do-while 문과 while문은 반복횟수가 1회 이상일때는 동일한 결과

제어문과 제어키워드 - 제어문

☞ 제어문의 **중복**

if-if

```
int a=5, b=3;
if(a>5){
    if(b<2){
        실행문1;
    }
    else{
        실행문2;
    }
}
```

1

switch-for

```
switch(a){
case 1:
    for(int k=0; k<10; k++){
        ...
    }
    break;
case 2:
    ...
}
```

2

for-for

```
for(int i=0; i<10; i++){
    for(int j=0; j<5; j++){
        System.out.println(i);
        System.out.println(j);
        if(i==j){
            ...
        }
    }
}
```

3

The End

제어키워드

제어문과 제어키워드 - 제어키워드

👉 break

TIP

- 1 - break은 if문을 제외한 가장 가까운 중괄호({ })를 탈출

break을 만나면 for문을 탈출

2

```
for(int i=0; i<10; i++){
    System.out.println(i);
    break;
}
```

0

if(조건문) 없이 break을 사용하는 경우는 거의 없음

3

```
for(int i=0; i<10; i++){
    if(i==5){
        break;
    }
    System.out.println(i);
}
```

0
1
2
3
4

break으로 탈출하는 중괄호

4

```
for(int i=0; i<10; i++){
    for(int j=0; j<10; j++){
        if(j==3)
            break;
        System.out.println(i+", "+j);
    }
}
```

0, 0
0, 1
0, 2
1, 0
1, 1
1, 2
...
9, 2

break으로 탈출하는 중괄호

5

Quiz

- 두개의 중괄호를 한번에 탈출하고 싶다면?

제어문과 제어키워드 - 제어키워드

👉 break

- 1 Quiz
- 두개의 중괄호를 한번에 탈출하고 싶다면?

2

```
for(int i=0; i<10; i++){
  for(int j=0; j<10; j++){
    if(j==3)
      break;
    System.out.println(i+", "+j);
  }
}
```

break으로 탈출하는 중괄호

0, 0
0, 1
0, 2
1, 0
1, 1
1, 2
...
9, 2

3

```
for(int i=0; i<10; i++){
  for(int j=0; j<10; j++){
    if(j==3){
      i=100;
      break;
    }
    System.out.println(i+", "+j);
  }
}
```

break으로 탈출하는 중괄호는 동일

0, 0
0, 1
0, 2

제어문과 제어키워드 - 제어키워드

👉 break

break + LABEL(레이블)

TIP

2

- 레이블명 다음에는 **콜론(:)**이 오며 개념적으로 콜론은 점프할 위치를 의미

1

레이블이 표시된 반복문 종료

```
out: for(int i=0; i<10; i++){  
    for(int j=0; j<10; j++){  
        if(j==3)  
            break out;  
        System.out.println(i+", "+j);  
    }  
}
```

break out으로 탈출하는 중괄호

0, 0
0, 1
0, 2

제어문과 제어키워드 - 제어키워드

👉 continue

TIP

1 - continue는 if문을 제외한 가장 가까운 닫힌 중괄호(})로 대체되는 개념

for문의 닫힌 중괄호인 것처럼 동작

2

```
for(int i=0; i<10; i++){
  continue;
  System.out.println(i); //오류
}
```



Unreachable code

if(조건문) 없이 continue를 사용하는 경우는 거의 없음

3

```
for(int i=0; i<10; i++){
  if(i==5){
    continue;
  }
  System.out.println(i);
}
```

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

continue가 대체하는 중괄호

4

```
for(int i=0; i<5; i++){
  for(int j=0; j<5; j++){
    if(j==3){
      continue;
    }
    System.out.println(i+", "+j);
  } ← continue가 대체하는 중괄호
}
```

- 0, 0
- 0, 1
- 0, 2
- 0, 4
- 1, 0
- ...
- 4, 2
- 4, 4

```
POS1: for(int i=0; i<5; i++){
  for(int j=0; j<5; j++){
    if(j==3){
      continue POS1;
    }
    System.out.println(i+", "+j);
  } ← continue가 대체하는 중괄호
}
```

- 0, 0
- 0, 1
- 0, 2
- 1, 0
- 1, 1
- ...
- 4, 1
- 4, 2

The End