

02

## 02-2. 연산자와 제어문

Java Basic Syntax

# 연산자

- 산술 연산자

연산자	예시	설명
+	10 + 2	더하기
-	10 - 2	빼기
*	10 * 2	곱하기
/	10 / 2	나누기
%	10 % 2	나머지 값 구하기

# 연산자

---

- 산술 연산자

- 숫자의 나누기 연산 (/)

- 내부적으로 '버림'에 의한 형변환이 이루어짐

- `10.0 / 3.0` → `3.3333333333333335`

- `10.0 / 3` → `3.3333333333333335`

- `10 / 3.0` → `3.3333333333333335`

- `10 / 3` → `3`

- `(double)10 / 3` → `3.3333333333333335`    *// 10은 내부적으로 double로 자동 형변환됨*

- `10 / (double) 3` → `3.3333333333333335`    *// 3은 내부적으로 double로 자동 형변환됨*

- `(double) (10/3)` → `3.0`    *// (10/3) 전체가 내부적으로 double로 자동 형변환됨*

# 연산자

- 복합 할당 연산자

연산자	예시	설명
+=	a += b	a = a + b
-=	a -= b	a = a - b
*=	a *= b	a = a * b
/=	a /= b	a = a / b
%=	a %= b	a = a % b

# 연산자

---

- 증감 연산자

연산자	예시	설명
++	++a	1 증가하기(전위)
	a++	1 증가하기(후위)
--	--a	1 감소하기(전위)
	a--	1 감소하기(후위)

# 연산자

---

- 증감 연산자

전위 연산: 증감 연산자가 변수 이름 앞에 있을 때

```
int a = 10; // 변수 a를 선언하고 10으로 초기화  
int b = 0;  // 변수 b를 선언하고 0으로 초기화  
b = ++a;    // 변수 a의 값을 1만큼 증가한 후 b변수에 저장. b의 값은 11
```

---

후위 연산: 증감 연산자가 변수 이름 뒤에 있을 때

```
int a = 10; // 변수 a를 선언하고 10으로 초기화  
int b = 0;  // 변수 b를 선언하고 0으로 초기화  
b = a++;    // 변수 a의 값을 b 변수에 저장한 후 a의 값을 1만큼 증가. b의 값은 10
```

---

# 연산자

---

- 비교 연산자
- 비교 연산자는 계산식의 결과가 true(참) 또는 false(거짓)의 논리 타입으로서, "~인지 아닌지"를 판단할 때 사용

연산자	예시	설명
>	a > b	a가 b보다 큰 값인지 판단
<	a < b	a가 b보다 작은 값인지 판단
>=	a >= b	a가 b보다 크거나 같은 값인지 판단
<=	a <= b	a가 b보다 작거나 같은 값인지 판단
=	a = b	a와 b가 같은 값인지 판단
!=	a != b	a와 b가 다른 값인지 판단

# 연산자

---

- 논리 연산자
- 여러 개의 조건을 하나의 명령문으로 만들 때 사용하는 연산자

a	b	a && b	a    b	!a
False	False	False	False	True
False	True	False	True	
True	False	False	True	False
True	True	True	True	



# 연산자

---

- &와 && 연산자의 차이
  - &는 &로 연결된 모든 조건식을 항상 실행
  - &&는 실행된 조건식의 결과값에 따라 다음에 조건식의 실행 여부를 결정

gender= 'F' && balance >=1000000000

①false

②조건식 검사 종료함. 결과값은 false

gender= 'F' & balance >=1000000000

①false

②true

&

③false

# 연산자

---

- 비트 논리 연산자
  - AND(&), OR(|), XOR(^), NOT(~)
- 비트 이동 연산자
  - 우측으로 n비트 이동(>>n)
  - 좌측으로 n비트 이동(<<n)

<b>A</b>	<b>0000 1111</b>
<b>B</b>	<b>1111 0000</b>
<b>A &amp; B</b>	<b>0000 0000</b>
<b>A   B</b>	<b>1111 1111</b>
<b>A ^ B</b>	<b>1111 1111</b>
<b>~A</b>	<b>1111 0000</b>
<b>A&gt;&gt;3</b>	<b>0000 0001</b>
<b>A&lt;&lt;3</b>	<b>0111 1000</b>

# 연산자

---

## 【삼항 연산자】

변수 = 조건 ? 명령1 : 명령2

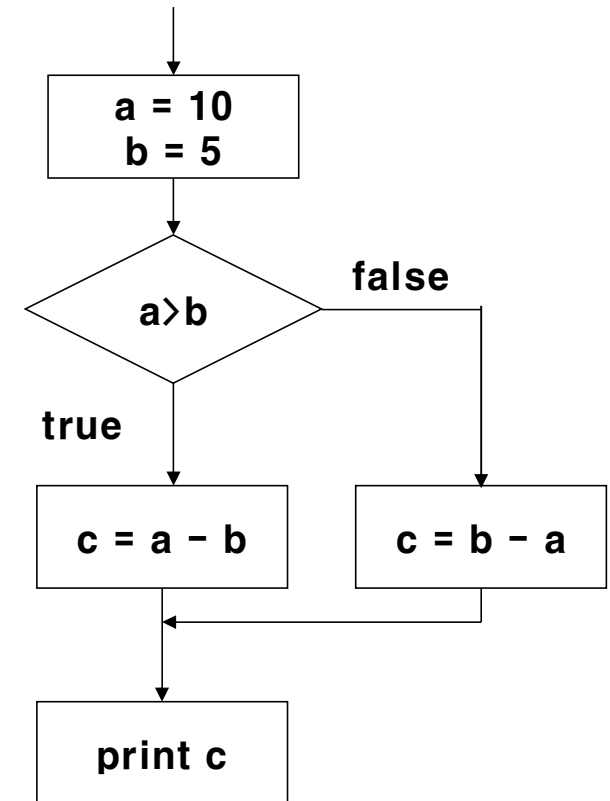
- 조건: true나 false 값 또는 true와 false의 결괏값이 나오는 비교 연산식
- 명령1: 조건이 true일 때 실행하는 명령
- 명령2: 조건이 false일 때 실행하는 명령

# 조건문 – if

---

- 기본 구문

```
if (condition) statement;  
if (condition) {  
    statement1;  
    statement2;  
}  
if (condition) {  
    statement1;  
    statement2;  
} else {  
    statement3;  
}
```



## 조건문 – if

---

- 유형 I: 단순 if

```
if (year % 4 == 0) {  
    System.out.println (year + “년은 윤년입니다.”);  
}
```

- 유형 II: If-else

```
if (year % 4 == 0) {  
    System.out.println (year + “년은 윤년입니다.”);  
} else {  
    System.out.println (year + “년은 윤년이 아닙니다.”);  
}
```

## 조건문 – if

---

- 유형 III: 중첩 if

```
if (year % 4 != 0) {  
    System.out.println (year + "년은 윤년이 아닙니다.");  
} else if (year % 100 != 0) {  
    System.out.println (year + "년은 윤년입니다.");  
} else if (year % 400 != 0) {  
    System.out.println (year + "년은 윤년이 아닙니다.");  
} else {  
    System.out.println (year + "년은 윤년입니다.");  
}
```

# 조건문 – switch-case

---

- 기본구문

```
switch(expression) {  
    case value1:  
        statement1;  
        break;  
    case value2:  
        statement2;  
        break;  
    default:  
        statement3;  
        break;  
}
```

- expression의 값에 해당되는 case를 수행
- 해당되는 값이 없을 경우 default 절을 수행
- switch-case 문에서의 break는 switch 문의 끝으로 프로그램의 흐름을 이동시킴

## 조건문 – switch-case

---

### ▪ *switch*

```
switch (month) {  
    case 2:  
        days = 28;  
        break;  
    case 4:  
    case 6:  
    case 9:  
    case 11:  
        days = 30;  
        break;  
    default:  
        days = 31;  
        break;  
}
```

### ▪ *if*

```
if (month == 2) {  
    days = 28;  
} else if ((month == 4) || (month ==  
6) || (month == 9) || (month == 11))  
{  
    days = 30;  
} else {  
    days = 31;  
}
```



## 조건문 – switch-case

---

- switch-case 문에서의 break을 사용하지 않을 경우 다음 case 문이 계속해서 수행 됨

```
switch (month) {  
    case 12:  
        system.out.println("한 해를 마무리하는 12월입니다.");  
    case 11: case 1: case 2:  
        system.out.println("여전히 겨울입니다.");  
}
```

- month = 12:

“한 해를 마무리하는 12월 입니다.”

“여전히 겨울입니다.”

- month = 11:

“여전히 겨울입니다.”

- month = 3:

아무 동작도 하지 않고 switch를 빠져나옴 (why? )

- 그러나 바람직하지 않은 프로그램 스타일
- switch 조건문에서는 앞 장치럼 default를 포함해서 가능한 모든 경우에 대한 case 절과 break문을 정의할 것

# while 반복문

---

- 기본구문(Syntax):

```
while (condition) {statements}
```

- condition 값은 if문에서와 같이 boolean 값을 가짐
- condition 값이 참인 동안 해당되는 구문을 반복해서 수행함
- 초기 condition 값이 false인 경우 한번도 수행하지 않음
- condition 값이 false가 되면 while 문을 빠져 나옴
  - condition 값이 계속 true이면 무한히 반복함
  - 의도적으로 무한 Loop를 실행시키는 경우도 있음

```
// 1에서 10까지 숫자를 출력함
int x = 1;
while (x <= 10){
    System.out.println(x);
    x = x + 1;
}
```

## 반복문 – do-while

---

- do-while 반복문
  - while 반복문과의 차이점 → do 블록을 우선 한번 수행하고 반복 여부를 확인함
  - 적어도 한번은 수행되어야 하는 반복 구문에 사용
  - while 문에서 확인할 조건이 반복 구문 내에서 할당되는 경우

```
char ch;  
do {  
    ch = getCharFromFile;  
    process input;  
} while (ch != end-of-file);
```

# 반복문 - for

- 세 가지 제어 요소

```
for( 초기화; 조건검사; 증감연산 )
```

```
{
```

```
    statement1;
```

```
    statement2;
```

```
    .....
```

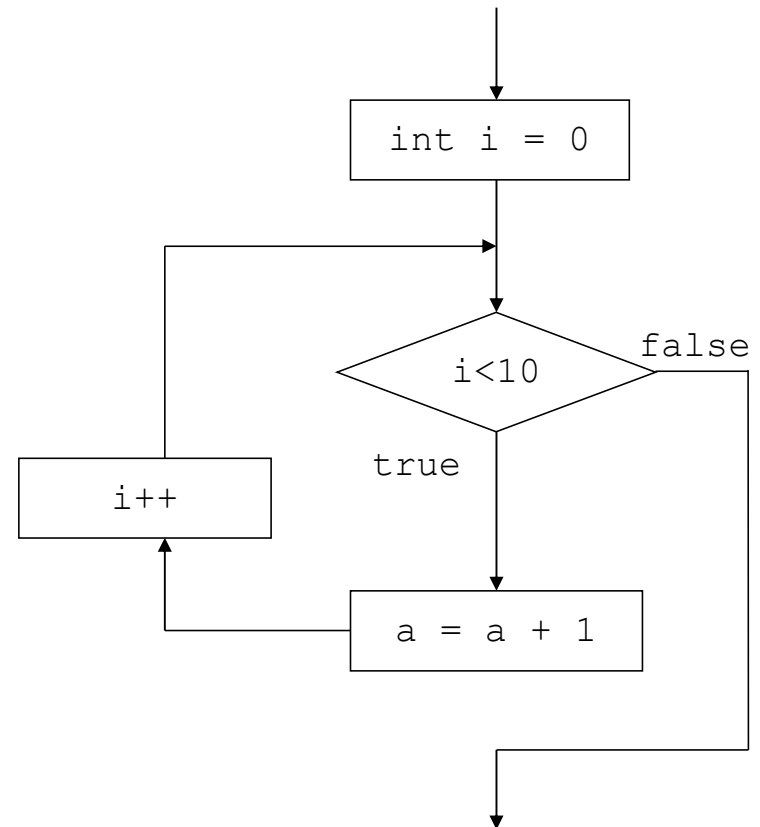
```
}
```

```
for( int i=0; i<10; i++ )
```

```
{
```

```
    a = a + 1;
```

```
}
```



# 반복문 - for

---

- for 반복문
  - while 반복문을 간략화하여 사용 가능

```
Initialize;  
while (test) {  
    Statements;  
    next;  
}
```

```
int x = 1;  
while (x <= 10){  
    System.out.println(x);  
    x = x + 1;  
}
```

```
for (initialize; test; next) {  
    Statements;  
}
```

```
for (int x = 1; x <= 10; x = x + 1)  
    System.out.println(x);
```

## 반복문 – break/continue

---

- switch-case 문에서의 break (*Review*)
  - switch 문의 끝으로 프로그램의 흐름을 이동시킴
  - break 문을 사용하지 않을 경우 다음 case 문이 계속해서 수행 됨

```
switch (month) {  
    case 2:  
        days = 28;  
        break;  
    case 4:  
    case 6:  
    case 9:  
    case 11:  
        days = 30;  
        break;  
    default:  
        days = 31;  
        break;  
}
```

```
switch (month) {  
    case 12:  
        system.out.println("한 해를 마무리하는 12월입  
        니다.");  
    case 11: case 1: case 2:  
        system.out.println("여전히 겨울입니다.");  
}
```

- 바람직하지 않은 프로그램 스타일
- switch 조건문에서는 default를 포함해서 가능한 모든 경우에 대한 case 절과 break문을 정의할 것

# 반복문 – break/continue

---

- break
  - 반복문을 종료하고 반복문을 벗어나 다음 문장을 실행
  - 중첩된 반복문에서 한 단계씩 반복문을 벗어나
  - 몇몇 반복문은 하나 이상의 반복 종료 시점을 가질 수 있음

```
while (true) { // 무한 반복
    ch = getCharFromFile;
    if (ch == end-of-file) {
        break;
    }
    process input;    ...
}
```

```
for (int i = 0; i < 10; i++) {
    ch = getCharFromFile
    if (ch == end-of-file) {
        break;
    }
    process(s);
}
```

## 반복문 – break/continue

---

### □ continue

- 반복문 내에서 **continue**를 만나면 **continue** 이후의 문장은 실행하지 않음
- 반복문의 검사 위치로 돌아가 반복문을 계속 수행
- 반복문에만 적용됨 → **Switch** 문에는 적용되지 않음
- 반복 블록을 벗어나지 않고 블록의 가장 마지막으로 이동하여 다시 반복 조건을 검사하도록 함




## 반복문 – break/continue

---

- Labeled Break and Continue
  - break, continue 는 자신을 감싸고 있는 가장 가까운 루프문을 대상으로 한다.
  - 루프문이 중첩되어 있는 경우 break 와 continue 에 의해 특정 루프문이 제어되게 하려면 label 을 활용해야 한다.

```
label:
for (int; testExpresison, update) {
    // codes
    for (int; testExpression; update) {
        // codes
        if (condition to break) {
            break label;
        }
        // codes
    }
    // codes
}
```

A diagram illustrating the effect of the 'break label;' statement. A blue line originates from the 'break label;' line within the inner 'for' loop, extends to the left, and then turns downwards and to the right, ending with an arrowhead pointing to the 'label:' line, which is the start of the outer 'for' loop. This visualizes how the break statement exits the inner loop and jumps to the beginning of the labeled outer loop.



# 감사합니다

단단히 마음먹고 떠난 사람은  
산꼭대기에 도착할 수 있다.  
산은 올라가는 사람에게만 정복된다.



윌리엄 셰익스피어  
William Shakespeare