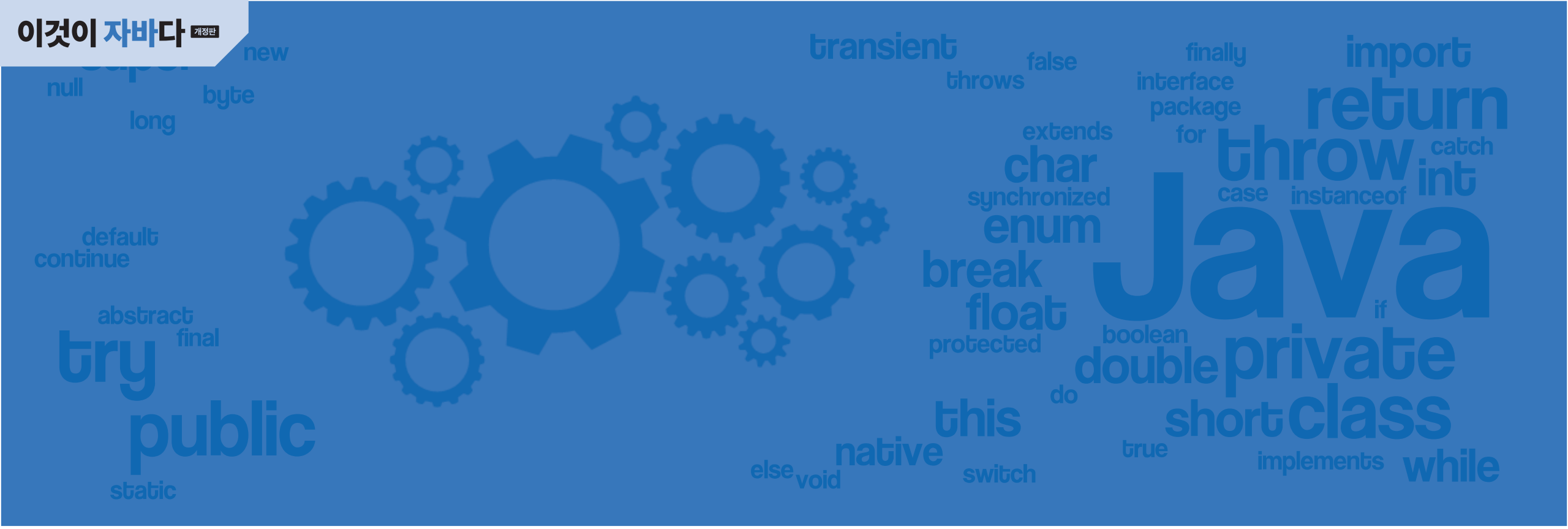


Chapter 03 연산자

Contents

부호 연산자

- 부호 연산자는 변수의 부호를 유지하거나 변경

연산식		설명
+	피연산자	피연산자의 부호 유지
–	피연산자	피연산자의 부호 변경

증감 연산자

- 증감 연산자는 변수의 값을 1 증가시키거나 1 감소시킴

연산식		설명
++	피연산자	피연산자의 값을 1 증가시킴
––	피연산자	피연산자의 값을 1 감소시킴
피연산자	++	다른 연산을 수행한 후에 피연산자의 값을 1 증가시킴
피연산자	––	다른 연산을 수행한 후에 피연산자의 값을 1 감소시킴

산술 연산자

- 더하기(+), 빼기(-), 곱하기(*), 나누기(/), 나머지(%)로 총 5개

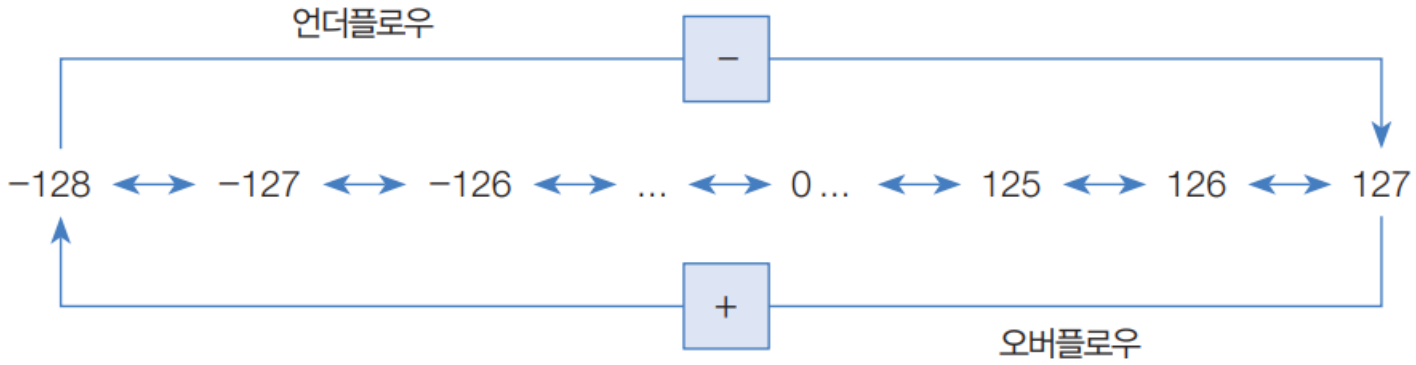
연산식			설명
피연산자	+	피연산자	덧셈 연산
피연산자	–	피연산자	뺄셈 연산
피연산자	*	피연산자	곱셈 연산
피연산자	/	피연산자	나눗셈 연산
피연산자	%	피연산자	나눗셈의 나머지를 산출하는 연산

오버플로우

- 타입이 허용하는 최대값을 벗어나는 것

언더플로우

- 타입이 허용하는 최소값을 벗어나는 것



정수 연산

- 산술 연산을 정확하게 계산하려면 실수 타입을 사용하지 않는 것이 좋음

>>> AccuracyExample1.java

```java
package ch03.sec04;

public class AccuracyExample1 {
  public static void main(String[] args) {
    int apple = 1;
    double pieceUnit = 0.1;
    int number = 7;

    double result = apple - number*pieceUnit;
    System.out.println("사과 1개에서 남은 양: " + result);
  }
}
```

실행 결과

```
사과 1개에서 남은 양: 0.29999999999999993
```

- 정확한 계산이 필요하면 정수 연산으로 변경

나눗셈 연산에서 예외 방지하기

- 나눗셈(/) 또는 나머지(%) 연산에서 좌측 피연산자가 정수이고 우측 피연산자가 0일 경우 ArithmeticException 발생
- 좌측 피연산자가 실수이거나 우측 피연산자가 0.0 또는 0.0f이면 예외가 발생하지 않고 연산의 결과는 Infinity(무한대) 또는 NaN(Not a Number)이 됨

```
5 / 0.0  ➔  Infinity
5 % 0.0  ➔  NaN
```

- Infinity 또는 NaN 상태에서 계속해서 연산을 수행하면 안 됨
- Double.isInfinite()와 Double.isNaN()를 사용해 /와 % 연산의 결과가 Infinity 또는 NaN인지 먼저 확인하고 다음 연산을 수행하는 것이 좋음

비교 연산자

- 비교 연산자는 동등(==, !=) 또는 크기(<, <=, >, >=)를 평가해서 boolean 타입인 true/false를 산출
- 흐름 제어문인 조건문(if), 반복문(for, while)에서 실행 흐름을 제어할 때 주로 사용

구분	연산식			설명
동등 비교	피연산자1	==	피연산자2	두 피연산자의 값이 같은지를 검사
	피연산자1	!=	피연산자2	두 피연산자의 값이 다른지를 검사
크기 비교	피연산자1	〉	피연산자2	피연산자1이 큰지를 검사
	피연산자1	〉=	피연산자2	피연산자1이 크거나 같은지를 검사
	피연산자1	〈	피연산자2	피연산자1이 작은지를 검사
	피연산자1	〈=	피연산자2	피연산자1이 작거나 같은지를 검사

- 문자열을 비교할 때는 동등(==, !=) 연산자 대신 equals()와 !equals()를 사용

논리 연산자

- 논리곱(&&), 논리합(||), 배타적 논리합(^) 그리고 논리 부정(!) 연산을 수행
- 흐름 제어문인 조건문(if), 반복문(for, while) 등에서 주로 이용

구분	연산식			결과	설명
AND (논리곱)	true	&& 또는 &	true	true	피연산자 모두가 true일 경우에만 연산 결과가 true
	true		false	false	
	false		true	false	
	false		false	false	
OR (논리합)	true	|| 또는 |	true	true	피연산자 중 하나만 true이면 연산 결과는 true
	true		false	true	
	false		true	true	
	false		false	false	
XOR (배타적 논리합)	true	^	true	false	피연산자가 하나는 true이고 다른 하나가 false일 경우에만 연산 결과가 true
	true		false	true	
	false		true	true	
	false		false	false	
NOT (논리 부정)		!	true	false	피연산자의 논리값을 바꿈
			false	true	

비트 논리 연산자

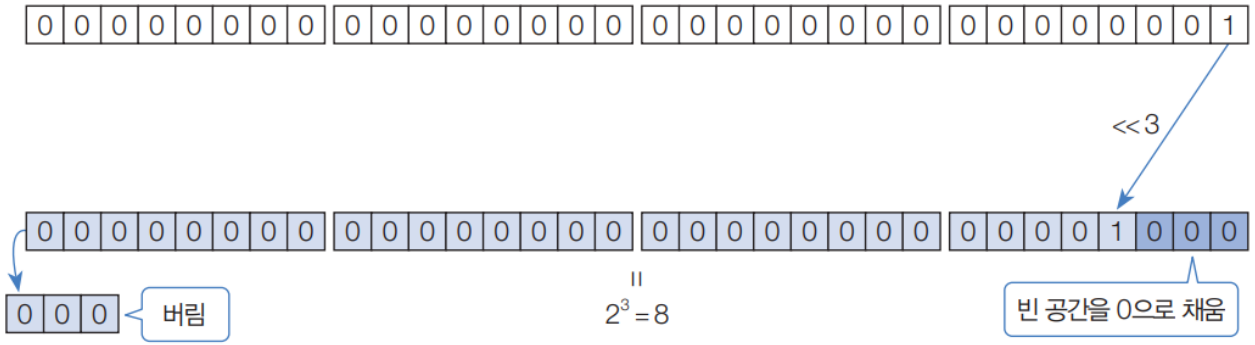
- bit 단위로 논리 연산을 수행. 0과 1이 피연산자가 됨
- byte, short, int, long만 피연산자가 될 수 있고, float, double은 피연산자가 될 수 없음

구분	연산식			결과	설명
AND (논리곱)	1	&	1	1	두 비트 모두 1일 경우에만 연산 결과가 1
	1		0	0	
	0		1	0	
	0		0	0	
OR (논리합)	1	|	1	1	두 비트 중 하나만 1이면 연산 결과는 1
	1		0	1	
	0		1	1	
	0		0	0	
XOR (배타적 논리합)	1	^	1	0	두 비트 중 하나는 1이고 다른 하나가 0일 경우 연산 결과는 1
	1		0	1	
	0		1	1	
	0		0	0	
NOT (논리 부정)		~	1	0	보수
			0	1	

비트 이동 연산자

- 비트를 좌측 또는 우측으로 밀어서 이동시키는 연산을 수행

구분	연산식			설명
이동 (shift)	a	《	b	정수 a의 각 비트를 b만큼 왼쪽으로 이동 오른쪽 빈자리는 0으로 채움 a x 2^b와 동일한 결과가 됨
	a	》	b	정수 a의 각 비트를 b만큼 오른쪽으로 이동 왼쪽 빈자리는 최상위 부호 비트와 같은 값으로 채움 a / 2^b와 동일한 결과가 됨
	a	》》	b	정수 a의 각 비트를 b만큼 오른쪽으로 이동 왼쪽 빈자리는 0으로 채움



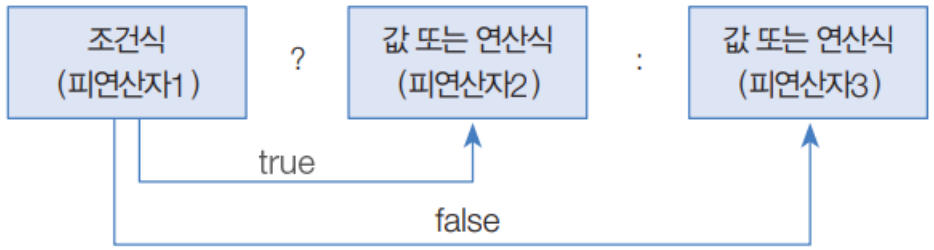
대입 연산자

- 우측 피연산자의 값을 좌측 피연산자인 변수에 대입. 우측 피연산자에는 리터럴 및 변수, 다른 연산식이 올 수 있음
- 단순히 값을 대입하는 단순 대입 연산자와 정해진 연산을 수행한 후 결과를 대입하는 복합 대입 연산자가 있음

구분	연산식			설명
단순 대입 연산자	변수	=	피연산자	우측의 피연산자의 값을 변수에 저장
복합 대입 연산자	변수	+=	피연산자	우측의 피연산자의 값을 변수의 값과 더한 후에 다시 변수에 저장 (변수 = 변수 + 피연산자)
	변수	-=	피연산자	우측의 피연산자의 값을 변수의 값에서 뺀 후에 다시 변수에 저장 (변수 = 변수 – 피연산자)
	변수	*=	피연산자	우측의 피연산자의 값을 변수의 값과 곱한 후에 다시 변수에 저장 (변수 = 변수 * 피연산자)
	변수	/=	피연산자	우측의 피연산자의 값으로 변수의 값을 나눈 후에 다시 변수에 저장 (변수 = 변수 / 피연산자)
	변수	%=	피연산자	우측의 피연산자의 값으로 변수의 값을 나눈 후에 나머지를 변수에 저장 (변수 = 변수 % 피연산자)
	변수	&=	피연산자	우측의 피연산자의 값과 변수의 값을 & 연산 후 결과를 변수에 저장 (변수 = 변수 & 피연산자)
	변수	|=	피연산자	우측의 피연산자의 값과 변수의 값을 | 연산 후 결과를 변수에 저장 (변수 = 변수 | 피연산자)
	변수	^=	피연산자	우측의 피연산자의 값과 변수의 값을 ^ 연산 후 결과를 변수에 저장 (변수 = 변수 ^ 피연산자)
	변수	≪=	피연산자	우측의 피연산자의 값과 변수의 값을 ≪ 연산 후 결과를 변수에 저장 (변수 = 변수 ≪ 피연산자)
	변수	≫=	피연산자	우측의 피연산자의 값과 변수의 값을 ≫ 연산 후 결과를 변수에 저장 (변수 = 변수 ≫ 피연산자)
	변수	≫≫=	피연산자	우측의 피연산자의 값과 변수의 값을 ≫≫ 연산 후 결과를 변수에 저장 (변수 = 변수 ≫≫ 피연산자)

삼항 연산자

- 총 3개의 피연산자를 가짐
- ? 앞의 피연산자는 boolean 변수 또는 조건식. 이 값이 true이면 콜론(:) 앞의 피연산자가 선택되고, false이면 콜론 뒤의 피연산자가 선택됨



연산이 수행되는 순서

- 덧셈(+), 뺄셈(-) 연산자보다는 곱셈(*), 나눗셈(/) 연산자가 우선. &&보다는 >, < 가 우선순위가 높음
- 우선순위가 같은 연산자의 경우 대부분 왼쪽에서부터 오른쪽으로(→) 연산을 수행

연산자	연산 방향	우선순위
증감(++, --), 부호(+, -), 비트(~), 논리(!)	←	높음
산술(*, /, %)	→	
산술(+, -)	→	
쉬프트(<<, >>, >>>)	→	
비교(<, >, <=, >=, instanceof)	→	
비교(==, !=)	→	
논리(&)	→	
논리(^)	→	
논리(|)	→	
논리(&&)	→	
논리(||)	→	
조건(?:)	→	낮음
대입(=, +=, -=, *=, /=, %=, &=, ^=, |=, <<=, >>=, >>>=)	←	


transient
false
finally
import
new
byte
null
long
throws
interface
return
package
extends
for
throw
catch
char
int
synchronized
case
instanceof
enum
default
continue
abstract
final
try
boolean
double
private
public
do
this
short
class
native
true
implements
while
else
void
switch
static

Thank you!