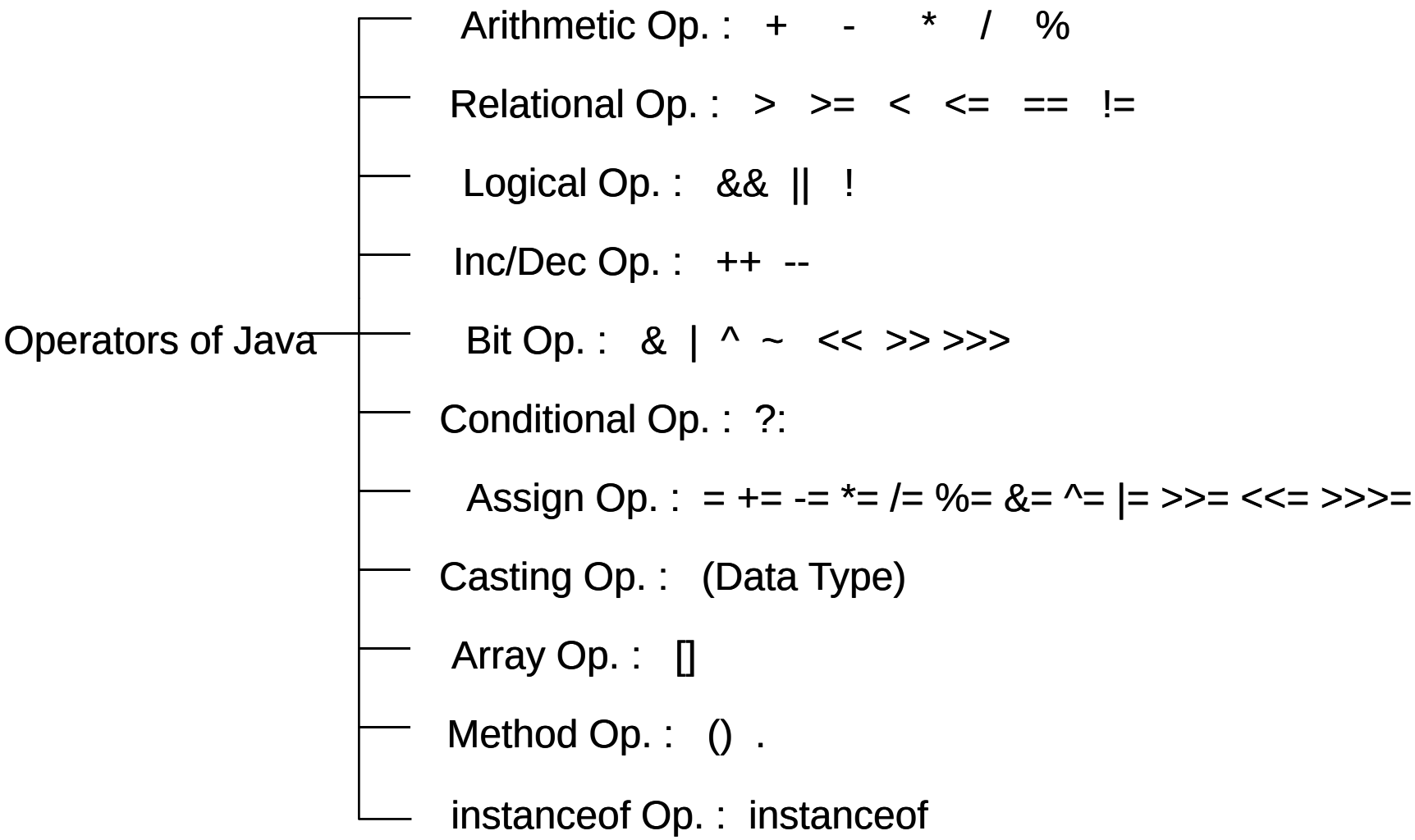
Operator dan Assignment

Macam-macam Operator



Arithmetic Operator

- Operator untuk operasi arithmetic
  - Single term operator : +, -
  - Binary term operator : +, -, *, /, %

```
x  =  -5 ;
x  =  -(-5) ;
x  =  -(3-5) ;
```

Kondisi Arithmetic Error

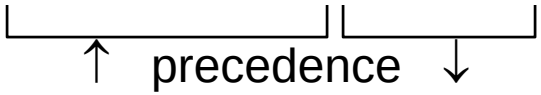
- Tipe data (char, byte, short, int, and long) dibagi dengan 0 akan menyebabkan ArithmeticException.
  - 1. int x = 2;
  - 2. int y =0;
  - 3. int z = x/y;
- Untuk tipe data float dan double pembagian dengan nol tidak membangkitkan error tapi menghasilkan infinity (IPOSITIVE_INFINITY or NEGATIVE_INFINITY).
- Mencari akar dari bilangan negatif (float atau double) akan membangkitkan nilai NaN (Not a Number), tapi bukan exception.

Nan

- In general, an NaN value indicates that the calculation has no meaningful result in ordinary arithmetic.
- Two NaN values are defined in the java.lang package: Float.NaN, and Double.NaN.
- Because NaN means not a sensible value, all the following value of the double variable x, including NaN:
  - x < Double.NaN
  - x <= Double.NaN
  - x > Double.NaN
  - x >= Double.NaN
  - x == Double.NaN
- As a corollary to the preceding example, consider
  - double x = 7.0/0.0;
  - x != Double.NaN (return true)

Operator relasi

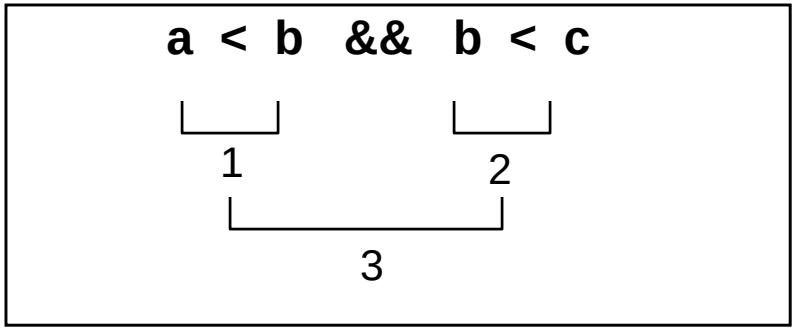
- Operator relasi disebut juga operator perbandingan, membandingkan dua operand dan mengembalikan nilai true atau false.
- Operand dapat sembarang nilai numerik.
- Ekspresi yang melibatkan operator relasi
  - for, while, ...
- Operator
  - <, <=, >, >=, ==, !=




```
a > b + c    ===>a > (b + c)
b == x < y  ===> b == (x < y)
```

Operator Kondisional

- Conditional Logical Relationship dari dua operand.
- Operator
  - ! , && , ||



Operator Kondisional

The Short Circuit Logical Operators

Operator	Name	Usage	Outcome
&&	Short-circuit logical AND	op1 && op2	true if op1 and op2 are both true, otherwise false. Conditionally evaluates op2.
||	Short-circuit logical OR	op1 || op2	true if either op1 or op2 is true, otherwise false. Conditionally evaluates op2.

- Untuk operasi AND, jika satu operand adalah false, maka hasil sudah pasti false, tanpa harus menguji operand lainnya, karena F && X = F
- Untuk operasi OR, jika satu operand adalah true, maka hasil sudah pasti true, tanpa harus menguji operand lainnya, karena T || X = T

- Jadi,
  - false && X = false
  - true || X = true

```
public class BooleanAnd {
    public static void main(String args[]) {
        int a=5, b=7;
        if ((a<2) & (b++<10)) b+=2;
        System.out.println(b);
    }
}
```

Hasil eksekusi :

```
8
```

```
public class ShortCircuitBooleanAnd {
    public static void main(String args[]) {
        int a=5, b=7;
        if ((a<2) && (b++<10)) b+=2;
        System.out.println(b);
    }
}
```

Hasil eksekusi :

```
7
```

```
public class BooleanOr {
    public static void main(String args[]) {
        int a=5, b=7;
        if ((a>2) | (b++<10)) b+=2;
        System.out.println(b);
    }
}
```

Hasil eksekusi :

```
10
```

Contoh:

```
int x = 2;  int y = 3;
if ((y == x++) | (x < ++y)) {
  System.out.println("x = " + x + " y = " + y) ;
 }
```

- Output
- x = 3 y = 4

```
public class ShortCircuitBooleanOr {
    public static void main(String args[]) {
        int a=5, b=7;
        if ((a>2) || (b++<10)) b+=2;
        System.out.println(b);
    }
}
```

Hasil eksekusi :

```
9
```

Operator Increment & Decrement

- Operator
  - ++, --
  - Prefix operator

```
n = 1;
x = ++n;       // x=2, n=2
```

  - Postfix operator

```
n = 1;
x = n++;       // x=1, n=2
```

  - Tidak dapat digunakan pada ekspresi aritmatika, hanya pada variabel saja

```
(a + b)++    // error
```

  - Tidak dapat digunakan untuk tipe real

Operator Bitwise

- Operator
  - &, |, ^, ~,
  - <<, >>, >>>
  - Operand harus bertipe integer
  - Precedence

Operator	Precedence
~ << >> >>> & ^ |	(H) ↕ (L)

Operator Bitwise

- Digunakan untuk memanipulasi bit

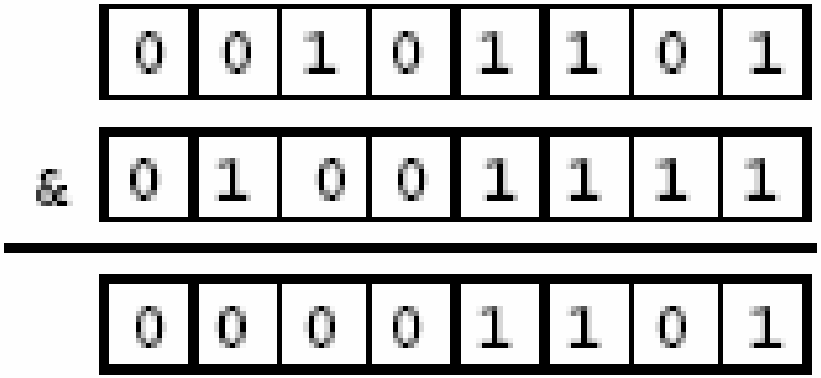
Operator	Use	Operation
&	op1 & op2	AND
|	op1 | op2	OR
^	op1 ^ op2	XOR
~	~op	Bitwise inversion
!	!op	NOT (Boolean inversion)

Operator Bitwise

- Bitwise AND
  - 1001_2 & 0011_2 = 0001_2
  - To extract the special area in variable by masking that area
- Bit OR
  - 1001_2 | 0011_2 = 1011_2
- Exclusive AND
  - 1001_2 ^ 0011_2 = 1010_2
- 1's Complement
  - ~ 00001010_2 = 11110101_2

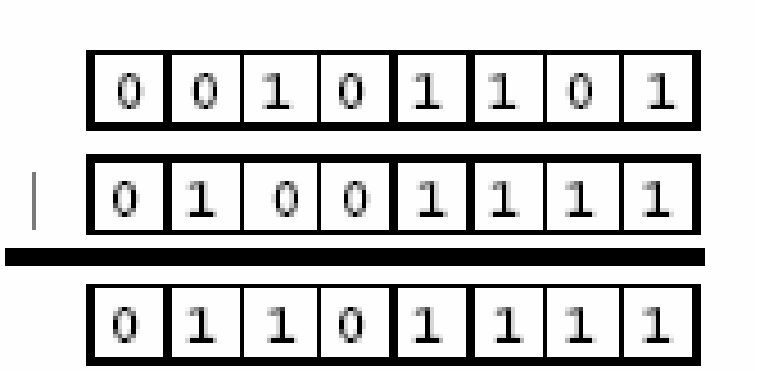
Operator Bitwise :&

op1	op2	op1 & op2
0	0	0
0	1	0
1	0	0
1	1	1



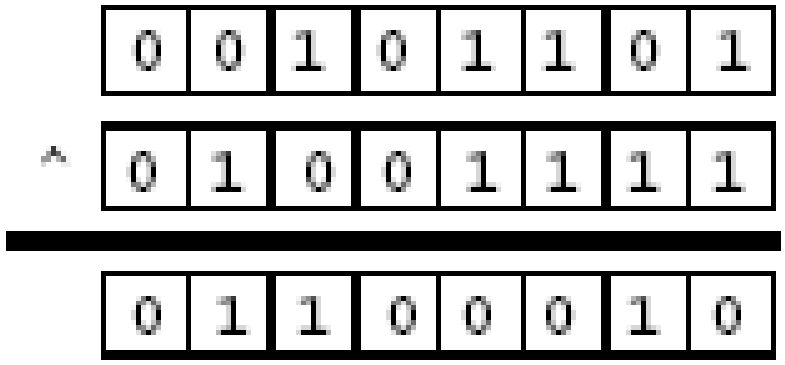
Operator Bitwise : |

op1	op2	op1 | op2
0	0	0
0	1	1
1	0	1
1	1	1



Operator Bitwise : ^

op1	op2	op1^op2
0	0	0
0	1	1
1	0	1
1	1	0



```
public class And {
    public static void main(String args[]) {
        int i;
        i = 6 & 13;
        System.out.println("Hasil operasi & = " + i);
    }
}
```

Hasil eksekusi :

```
Hasil operasi &  = 4
```

Susunan bit dari nilai 6 : 0000 0000 0000 0000 0000 0000 0000 0110
Susunan bit dari nilai 13 : 0000 0000 0000 0000 0000 0000 0000 1101
---------- &
Bit hasil operasi & : 0000 0000 0000 0000 0000 0000 0000 0100 → 4

```
public class Or {
    public static void main(String args[]) {
        int i;
        i = 5 | 9;
        System.out.println("Hasil operasi & = " + i);
    }
}
```

Hasil eksekusi :

```
Hasil operasi | = 13
```

```
Susunan bit dari nilai 5   :    0000 0000 0000 0000 0000 0000 0000 0101
Susunan bit dari nilai 9   :    0000 0000 0000 0000 0000 0000 0000 1001
                                ---------------------------------------|
Bit hasil operasi |  :         0000 0000 0000 0000 0000 0000 0000 1101  → 13
```

```
public class Or {
    public static void main(String args[]) {
        int i;
        i = 5 ^ 9;
        System.out.println("Hasil operasi & = " + i);
    }
}
```

Hasil operasi ^ = 3

```
Susunan bit dari nilai 5   :    0000 0000 0000 0000 0000 0000 0000 0101
Susunan bit dari nilai 9   :    0000 0000 0000 0000 0000 0000 0000 1001
                                ---------------------------------------^
Bit hasil operasi ^ :           0000 0000 0000 0000 0000 0000 0000 1100  → 12
```

Mencari nilai biner suatu bil negatif

- Cara:
  1. Tulis biner bilangan positifnya
  2. Dikurangi dengan 1
  2. ~(hasil)
- Misal: Bagaimana representasi biner untuk bilangan –5 ?

0000 …. 0000 0101 → 5

1

0000 …. 0000 0100 _

1111 …. 1111 1011 → -5

Mencari bilangan desimal dari bilangan biner negatif

- Cara:
  1. Lakukan negasi terhadap bilangan biner tersebut
  2. Ditambah dengan 1
- Misal : 1111 …. 1111 1011

1111 …. 1111 1011 → berapa?

0000 …. 0000 0100

1

0000 … 0000 0101 --> -5

Operator Bitwise

- Bitwise Shift Operator
  - Shift left(<<)

    $x << y = x * 2^y$

  - Shift right(>>)

    $x >> y = x / 2^y$

  - Unsigned shift right(>>>)
    - Give this operator because Java does not support unsigned integer.

Operator Bitwise

The Shift Operators

- **Shift operator:**
  - << : left shift
  - >> : sign right shift
  - >>> : unsigned right shift
- **Fundamentals of Shifting**
  - moving the bit pattern left or right.
  - applied to arguments of integral types only.
- **Pada operator << dan >>>: Nilai bit yang baru adalah 0**
- **Pada operator >> : Nilai bit yang baru tergantung pada bit pada posisi terkiri yang akan digeser, jika nilainya :**
  - **1 → negatif, maka nilai baru adalah 1**
  - **0 → positif, maka nilai baru adalah 0**

Operator Bitwise

The Shift Operators

The basic mechanisms of shifting

Original data			**192**			
in binary		00000000	00000000	00000000	11000000	
Shifted left 1 bit	0	00000000	00000000	00000001	1000000?	
Shifted right 1 bit		?0000000	00000000	00000000	01100000	0
Shifted left 4 bits	0000	00000000	00000000	00001100	0000????	
Original data			**–192**			
in binary		11111111	11111111	11111111	01000000	
Shifted left 1 bit	1	11111111	11111111	11111110	1000000?	
Shifted right 1 bit		?1111111	11111111	11111111	00100000	0

Operator Bitwise

The Shift Operators >>

Signed right shift of positive and negative numbers

Original data	192			
in binary	00000000	00000000	00000000	11000000
Shifted right 1 bit	00000000	00000000	00000000	01100000
Shifted right 7 bits	00000000	00000000	00000000	00000001
Original data	–192			
in binary	11111111	11111111	11111111	01000000
Shifted right 1 bit	11111111	11111111	11111111	10100000
Shifted right 7 bits	11111111	11111111	11111111	11111110

Shifting positive and negative numbers right

Original data		192		
in binary	00000000	00000000	00000000	11000000
Shifted right 1 bit = 96 = 192 / 2	00000000	00000000	00000000	01100000
Shifted right 4 bits = 12 = 192 / 16 = 192 / 2^4	00000000	00000000	00000000	00001100
Original data		–192		
in binary	11111111	11111111	11111111	01000000
Shifted right 1 bit = –96 = –192 / 2	11111111	11111111	11111111	10100000
Shifted right 4 bits = –12 = –192 / 16 = –192 / 2^4	11111111	11111111	11111111	11110100

```
public class RightShift {
    public static void main(String args[]) {
        int i=7;
        i=i >> 2;
        System.out.println(i);
    }
}
```

Hasil eksekusi :

```
1
```

Susunan bit 7 : 0000 0000 0000 0000 0000 0000 0000 0111

Geser ke kanan 2 kali : 0000 0000 0000 0000 0000 0000 0000 0001 → 1

```
public class UnsignedRightShift {
    public static void main(String args[]) {
        int i = -1;
        i = i >>> 30;
        System.out.println(i);
    }
}
```

Hasil eksekusi :

```
3
```

Susunan bit -1 : 1111 1111 1111 1111 1111 1111 1111 1111

Geser ke kanan 30 kali : 0000 0000 0000 0000 0000 0000 0000 0011 → 3

```
public class LeftShift {
    public static void main(String args[]) {
        int i=3;
        i=i << 2;
        System.out.println(i);
    }
}
```

Hasil eksekusi :

```
12
```

Susunan bit 3 : 0000 0000 0000 0000 0000 0000 0000 0011

Geser ke kanan 2 kali : 0000 0000 0000 0000 0000 0000 0000 1100 → 12

The Conditional Operator

- Operator
  - Expr1 ? Expr2 : Expr3 (3 Terms Operator)

```
max = x > y ? x : y ;
```

```
if (x > y)  max = x;
else  max = y;
```

```
m = a > b ? (c > a ? c : a) : (c > b ? c : b) ;
```

Assignment Operators

Expr **1** = Expr **1** op Expr2 ⟷ Expr1 op= Expr 2

- Operator
  - Arithmetic operator : + - * / %
  - Bitwise operator : & | ^ << >> >>>

sum = sum + i ; ⟷ sum += i ;

x = x * y + 1; ✗ x *= y + 1;

x = x * (y+1)

Shortcut Assignment Operators

x = x + y ;

- Dapat disingkat x += y;

Operator	Use	Equivalent To
+=	op1 += op2	op1 = op1 + op2
-=	op1 -= op2	op1 = op1 - op2
*=	op1 *= op2	op1 = op1 * op2
/=	op1 /= op2	op1 = op1 / op2
%=	op1 %= op2	op1 = op1 % op2
&=	op1 &= op2	op1 = op1 & op2
|=	op1 |= op2	op1 = op1 | op2
^=	op1 ^= op2	op1 = op1 ^ op2
<<=	op1 <<= op2	o1 = op1 << op2
>>=	op1 >>= op2	o1 = op1 >> op2
>>>=	op1 >>>= op2	o1 = op1 >>> op2

Cast Operator

- Data Type Casting Operator

(Data Type) 식

  - Cast operator : (,)

```
(int) 3.75          ===>  3
(float) 3           ===>  3.0
(float) (1 / 2)     ===>  0.0
(float)1/2          ===>  0.5
```

Operator Unary : cast → (type)

- Casting digunakan untuk melakukan konversi tipe secara eksplisit ke dalam type baru yang ada dalam tanda ().
- Akan dilakukan pengecekan tipe terlebih dahulu.
- Contoh:

```
int keliling = (int) (Math.PI * diameter);
```

Operator Precedence

Operator	Association	Precedence
() [] .	Left Assoc. (High)	
! ~ ++ -- + - (Data Type)	Left Assoc.	
* / %	Left Assoc.	
+ -	Left Assoc.	
<< >> >>>	Left Assoc.	
< <= > >= instance	Left Assoc.	
== !=	Left Assoc.	
&	Left Assoc.	
^	Left Assoc.	
|	Left Assoc.	
&&	Left Assoc.	
||	Left Assoc.	
? :	Left Assoc.	
= += -= *= /= %= &= ^= |= <<= >>= >>>=	Left Assoc. (Low)	

Operator Precedence

- a = x + y - z ; // Left Association
- b = -x ; // Right Association
- c = -x++ ;
- d = -++x ;
- e = -x + z ;

Operator instanceof

- Operator instance of digunakan untuk mengecek class suatu obyek.
- Pengecekan dilakukan pada saat runtime.

```
import java.awt.*;
class CompareTest {
  public static void main(String [] args) {
        Button b = new Button(“Exit”);
        boolean compare1 = b instanceof Button;
        boolean compare2 = b instanceof Componenet;
        System.out.println(“Is b a Button?” + compare1)
        System.out.println(“Is b a Component?” + compare2)
  }
}
```

Operator instanceof

- Hasil:

```
Is b a Button? true
Is b a Component? true
```

- Argumen sebelah kiri adalah object reference expression.
- Argumen sebelah kanan adalah class, interface, atau array