Operators
(Chapter 3 of Schilit)

Object Oriented Programming BS (CS/SE) II

Compiled By

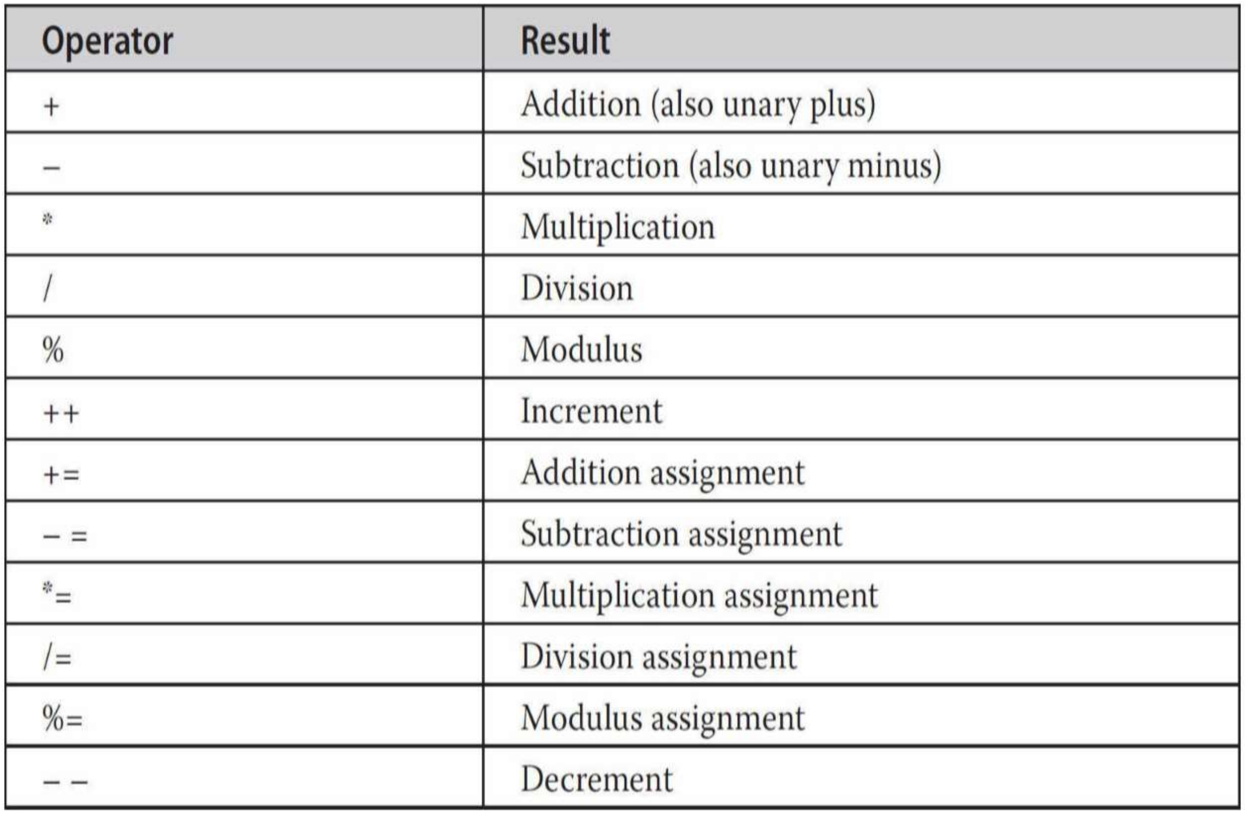
Abdul Haseeb Shaikh

Operators

- Arithmetic
- Bitwise
- Relational
- Logical

sk1

Arithmetic operators



Operator	Result
+	Addition (also unary plus)
–	Subtraction (also unary minus)
*	Multiplication
/	Division
%	Modulus
++	Increment
+=	Addition assignment
– =	Subtraction assignment
*=	Multiplication assignment
/=	Division assignment
%=	Modulus assignment
– –	Decrement

Operands to these operators must be numeric

Slide 3

sk1 sher khalil, 25/03/2021

Example (arithmetic with int and double)

Unary Operator

- Unary Minus
- NOT(!)
- Increment(++) (pre & post)
- Decrement(--) (pre & post)

Modulus Operator(%)

- Floating
- Integer
- What happens when left side is smaller than right side?

Take a floating point number as input, find its remainder when divided with 5

Compound Assignment Operators

var = <var> op <expression> **Equal to** var op= <expression>;

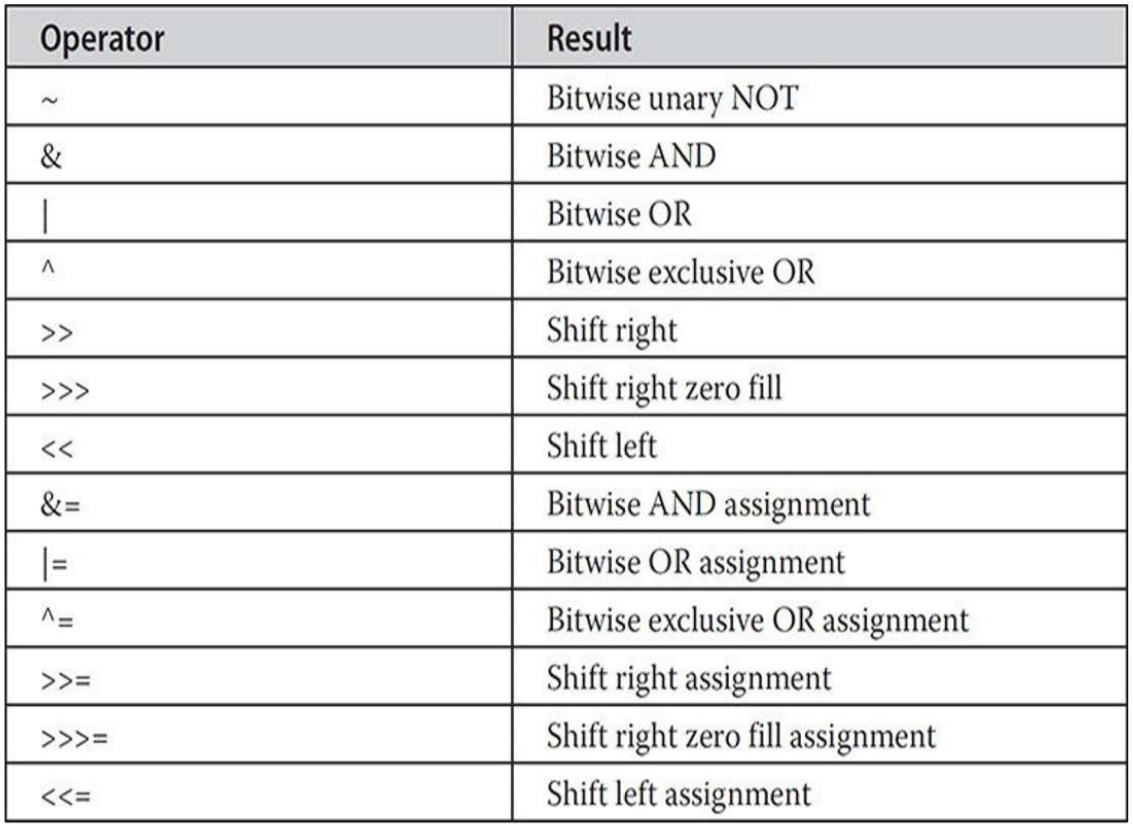
Example (compound operator)

How integers are stored in memory by Java and representation of sign

- In java integers are signed:
    - Store negative as well as positive values

- To store negative numbers, use the concept of Two’s complement:
    - Invert all the bits and add 1 to the result from LSB
    - Example 8 is represented in binary as 00001000
    - Invert all bits= 11110111
    - +1
    - 10000000

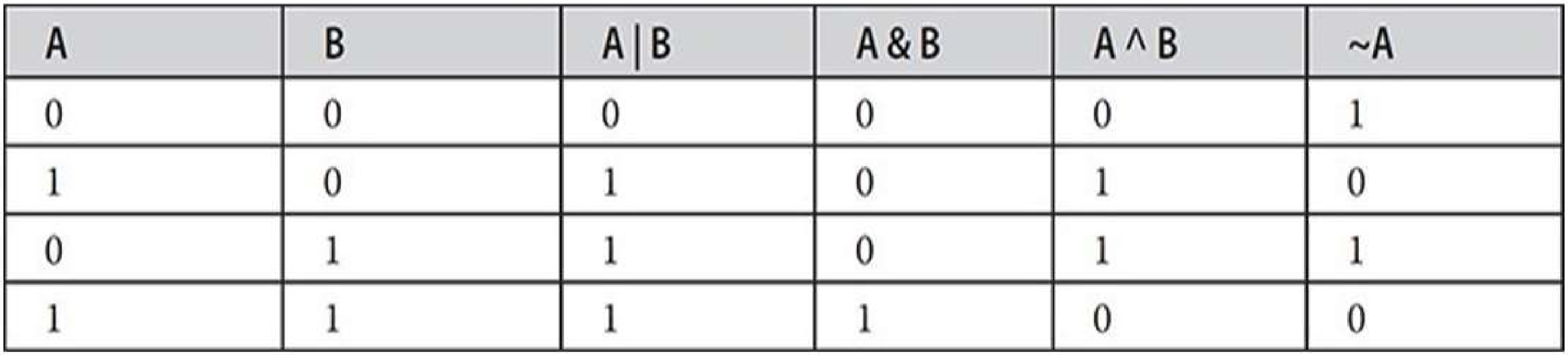
Bitwise Operators



Operator	Result
~	Bitwise unary NOT
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
>>	Shift right
>>>	Shift right zero fill
<<	Shift left
&=	Bitwise AND assignment
|=	Bitwise OR assignment
^=	Bitwise exclusive OR assignment
>>=	Shift right assignment
>>>=	Shift right zero fill assignment
<<=	Shift left assignment

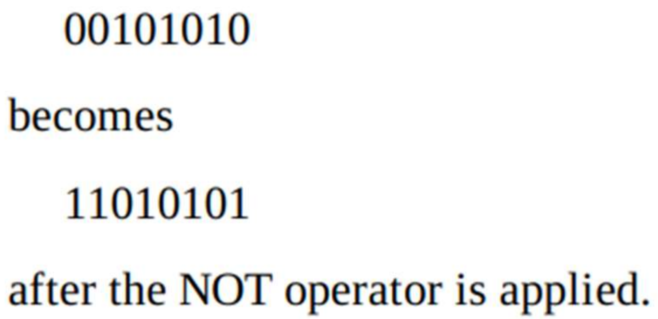
Bitwise Logical Operators

&, |, ^, and ~



A	B	A|B	A&B	A ^ B	~A
0	0	0	0	0	1
1	0	1	0	1	0
0	1	1	0	1	1
1	1	1	1	0	0

Bitwise NOT(Complement) ~



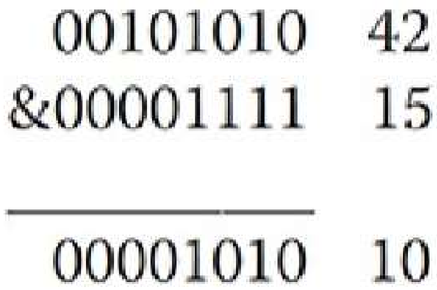
    00101010

becomes

    11010101

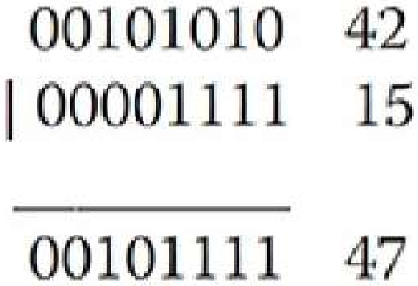
after the NOT operator is applied.

Bitwise AND &



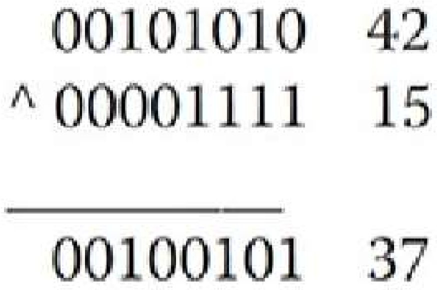
```
 00101010   42
&00001111   15
_________
 00001010   10
```

Bitwise OR |



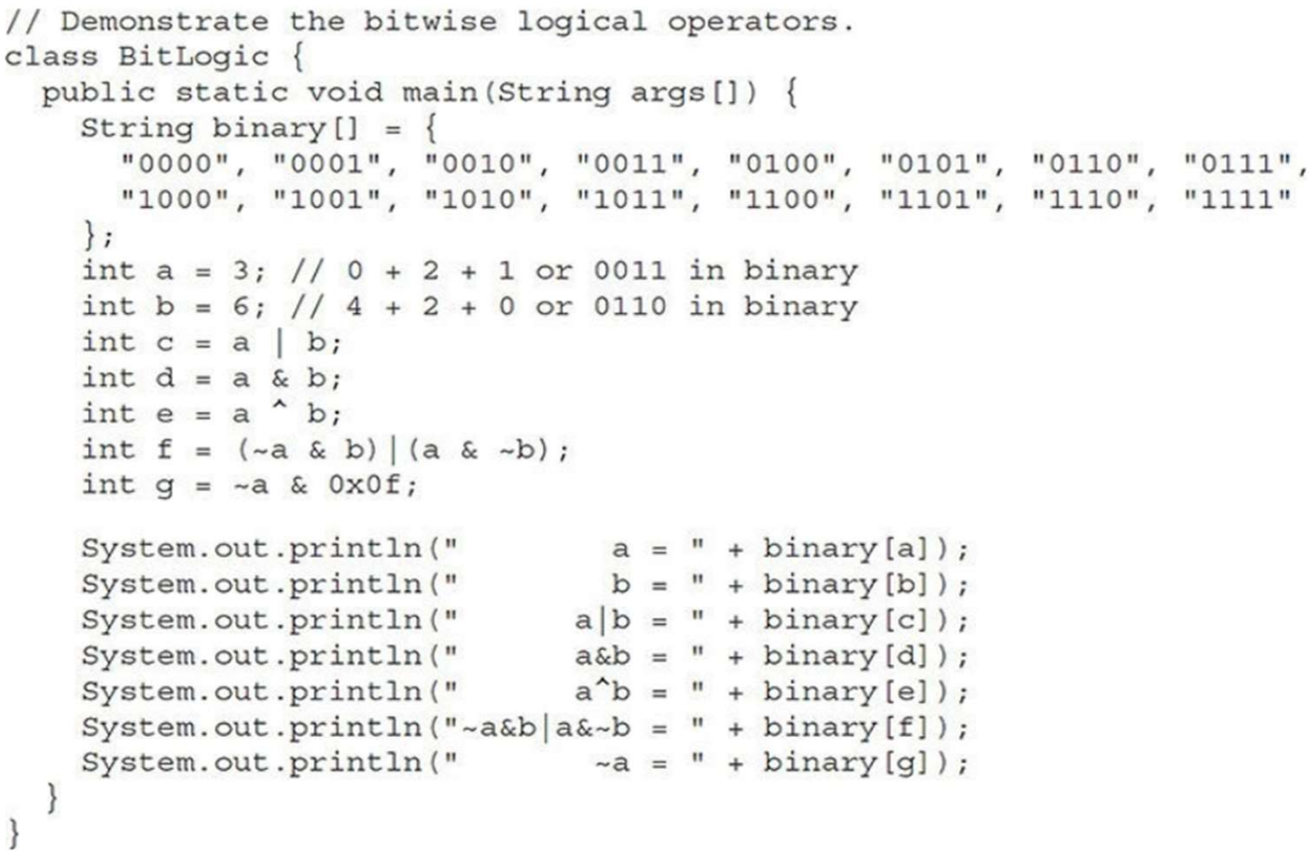
```
 00101010   42
|00001111   15
 ________
 00101111   47
```

Bitwise XOR ^



```
  00101010   42
^ 00001111   15
__________
  00100101   37
```

Use of Bitwise Logical Operators



```
// Demonstrate the bitwise logical operators.
class BitLogic {
  public static void main(String args[]) {
    String binary[] = {
      "0000", "0001", "0010", "0011", "0100", "0101", "0110", "0111",
      "1000", "1001", "1010", "1011", "1100", "1101", "1110", "1111"
    };
    int a = 3; // 0 + 2 + 1 or 0011 in binary
    int b = 6; // 4 + 2 + 0 or 0110 in binary
    int c = a | b;
    int d = a & b;
    int e = a ^ b;
    int f = (~a & b)|(a & ~b);
    int g = ~a & 0x0f;

    System.out.println("        a = " + binary[a]);
    System.out.println("        b = " + binary[b]);
    System.out.println("      a|b = " + binary[c]);
    System.out.println("      a&b = " + binary[d]);
    System.out.println("      a^b = " + binary[e]);
    System.out.println("~a&b|a&~b = " + binary[f]);
    System.out.println("       ~a = " + binary[g]);
  }
}
```
]

a=0011
b=0110

a|b=0111

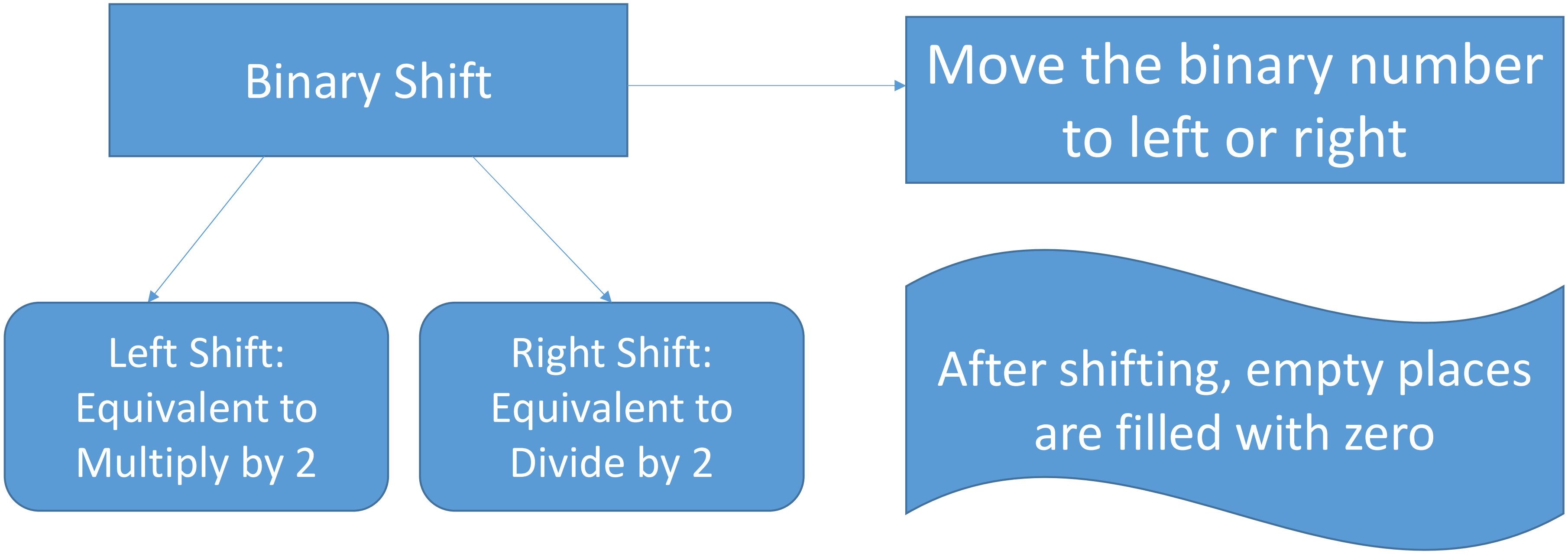
a=0011
b=0110

a&b=0010

a=0011
b=0110
a^b=0101

a=0011
b=0110
a&b=0010

LOGICAL BINARY SHIFTS



Binary Shift

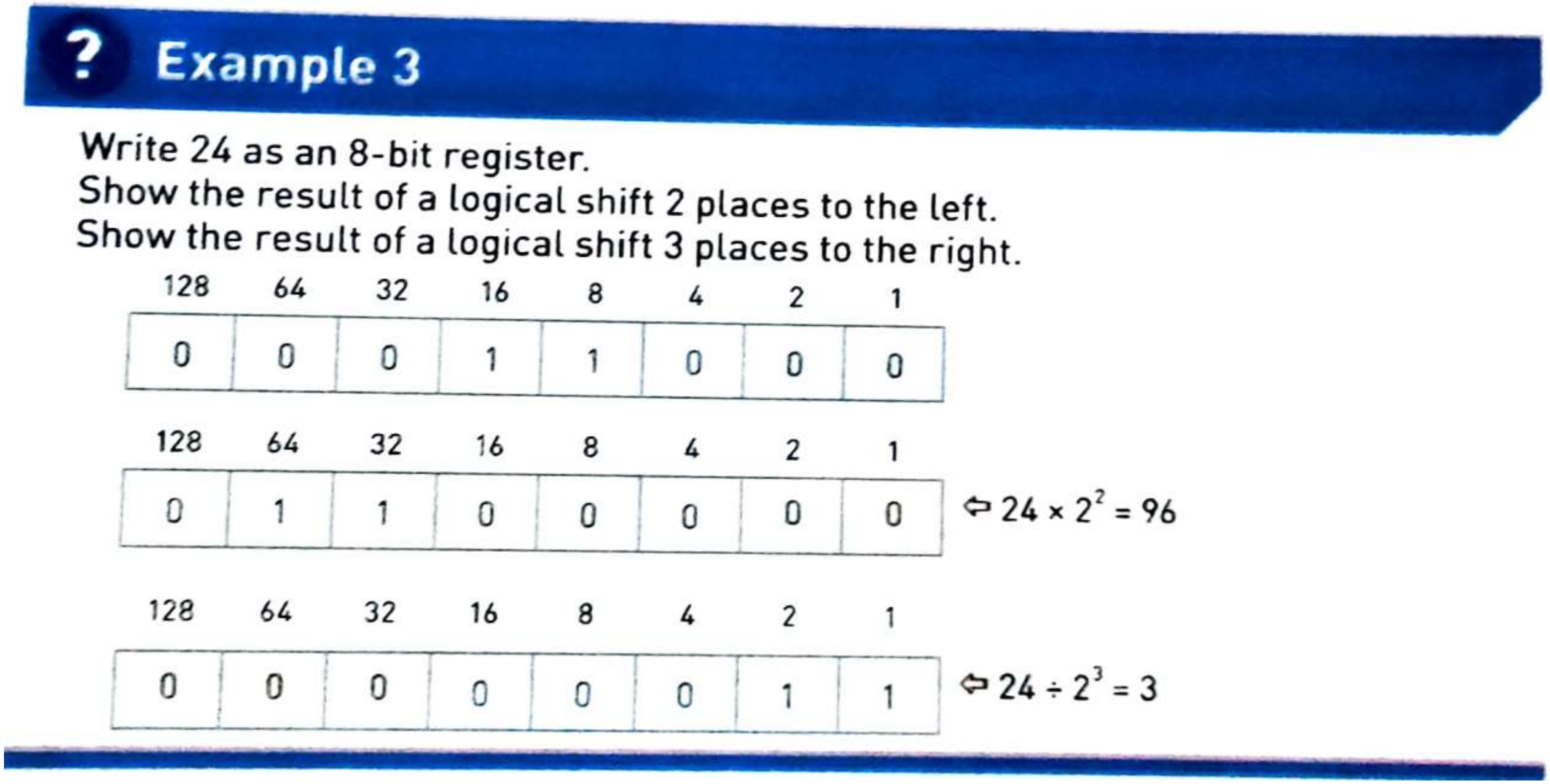
Move the binary number to left or right

Left Shift: Equivalent to Multiply by 2

Right Shift: Equivalent to Divide by 2

After shifting, empty places are filled with zero

Left Shift and Right Shift



Example 3

Write 24 as an 8-bit register.
Show the result of a logical shift 2 places to the left.
Show the result of a logical shift 3 places to the right.

128	64	32	16	8	4	2	1
0	0	0	1	1	0	0	0

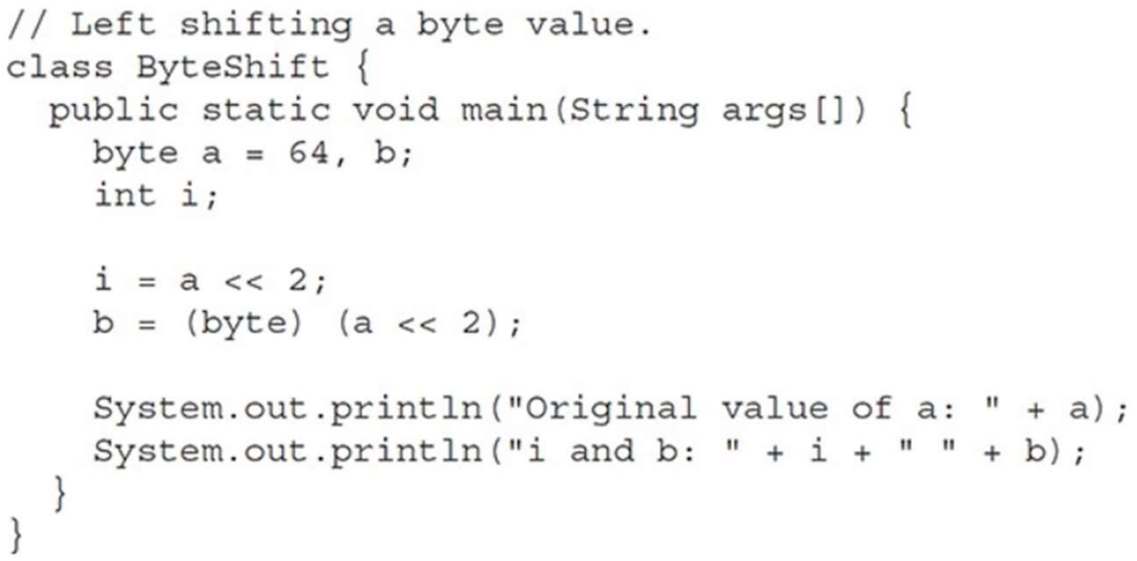
128	64	32	16	8	4	2	1
0	1	1	0	0	0	0	0

⇦ $24 \times 2^2 = 96$

128	64	32	16	8	4	2	1
0	0	0	0	0	0	1	1

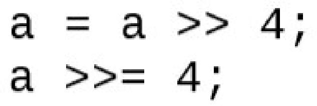
⇦ $24 \div 2^3 = 3$

Left Shift and Right Shift Demo

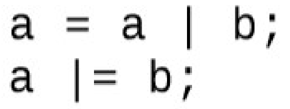


```
// Left shifting a byte value.
class ByteShift {
  public static void main(String args[]) {
    byte a = 64, b;
    int i;

    i = a << 2;
    b = (byte) (a << 2);

    System.out.println("Original value of a: " + a);
    System.out.println("i and b: " + i + " " + b);
  }
}
```

Bitwise Operator Compound Operator

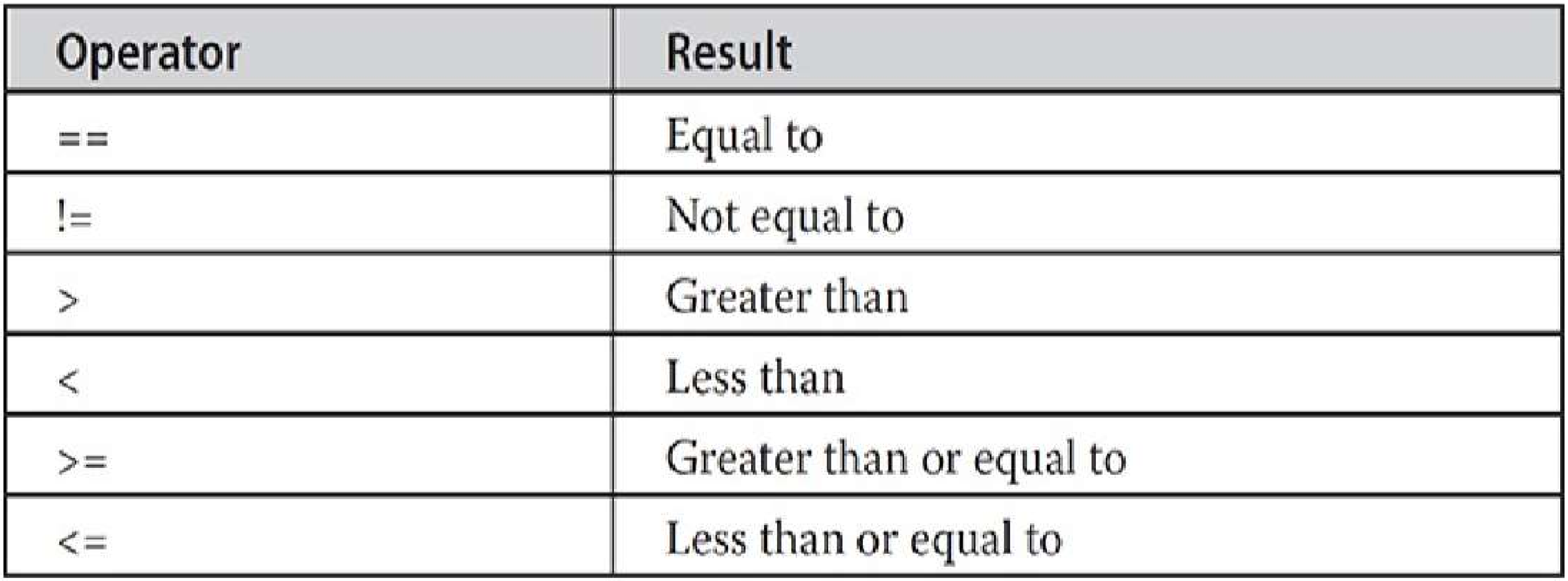


```
a = a >> 4;
a >>= 4;
```



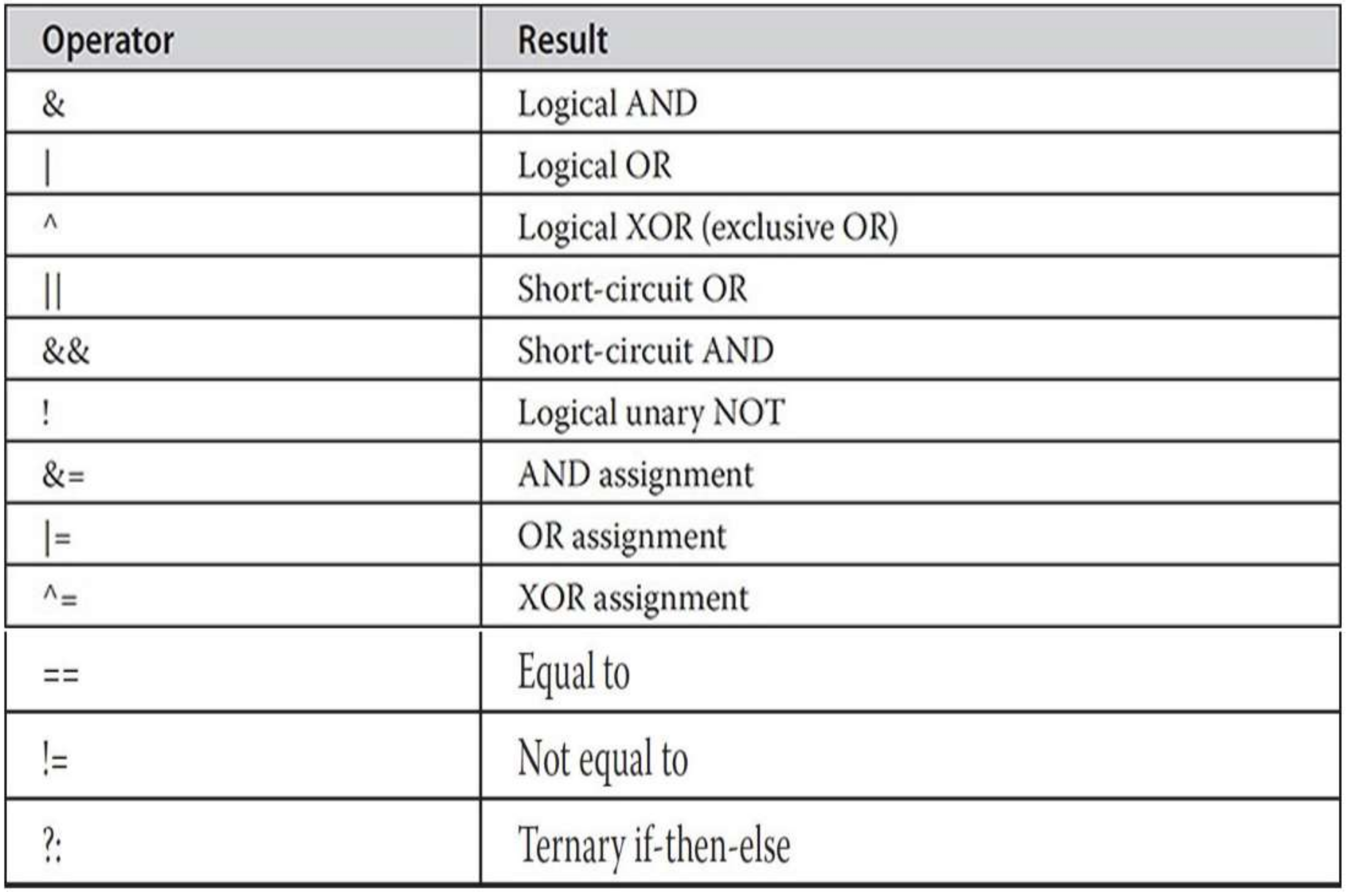
```
a = a | b;
a |= b;
```

Relational Operators (Boolean Outcome)



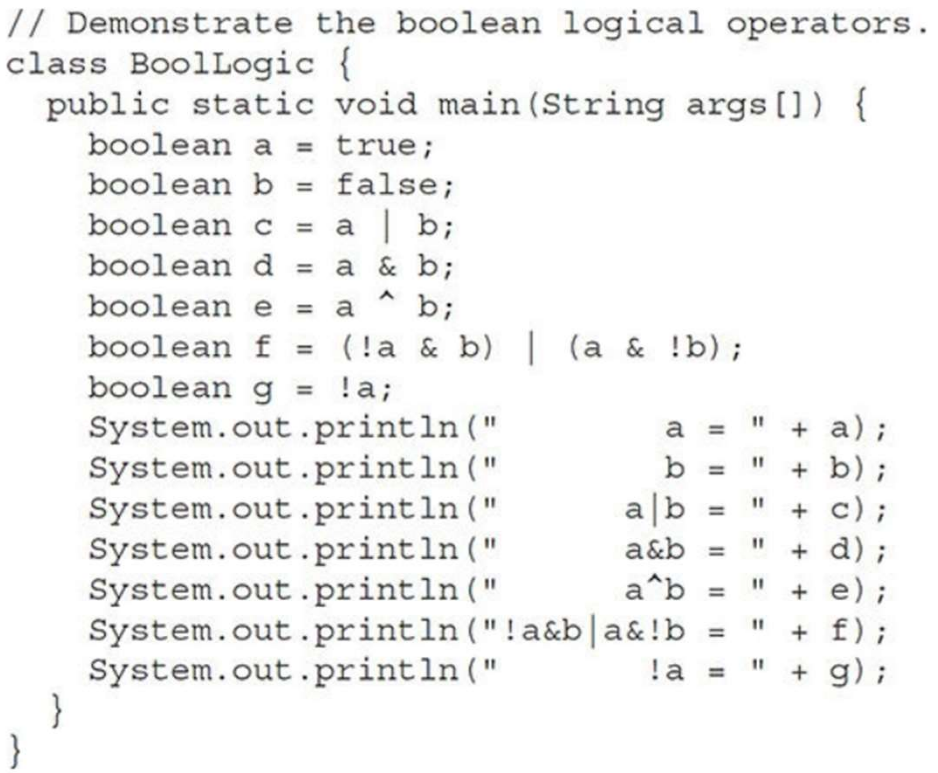
Operator	Result
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Boolean Logical Operators

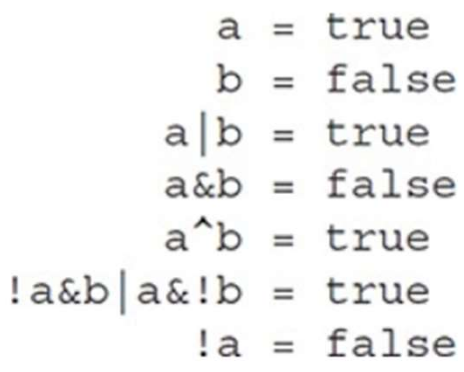


Operator	Result
&	Logical AND
|	Logical OR
^	Logical XOR (exclusive OR)
||	Short-circuit OR
&&	Short-circuit AND
!	Logical unary NOT
&=	AND assignment
|=	OR assignment
^=	XOR assignment
==	Equal to
!=	Not equal to
?:	Ternary if-then-else

Boolean Logical Operators

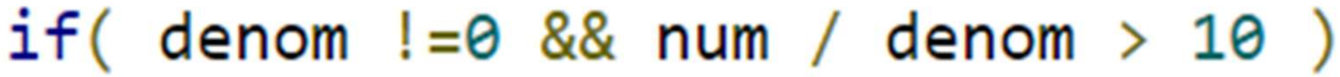


```
// Demonstrate the boolean logical operators.
class BoolLogic {
  public static void main(String args[]) {
    boolean a = true;
    boolean b = false;
    boolean c = a | b;
    boolean d = a & b;
    boolean e = a ^ b;
    boolean f = (!a & b) | (a & !b);
    boolean g = !a;
    System.out.println("        a = " + a);
    System.out.println("        b = " + b);
    System.out.println("      a|b = " + c);
    System.out.println("      a&b = " + d);
    System.out.println("      a^b = " + e);
    System.out.println("!a&b|a&!b = " + f);
    System.out.println("       !a = " + g);
  }
}
```



```
        a = true
        b = false
      a|b = true
      a&b = false
      a^b = true
!a&b|a&!b = true
       !a = false
```

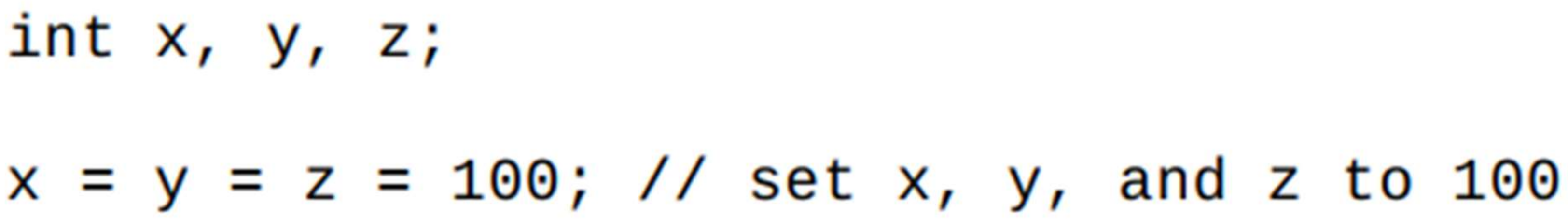
Short Circuit Logical Operator



```
if( denom !=0 && num / denom > 10 )
```

Assignment Operator

- = Operator

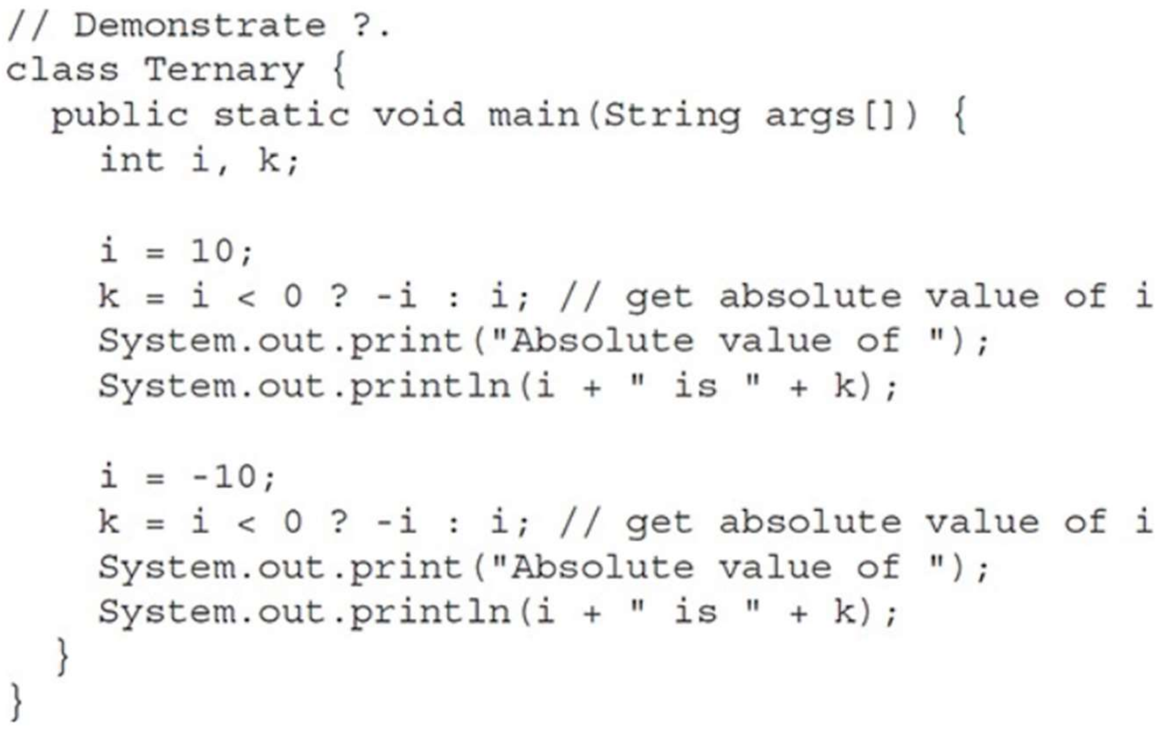


```
int x, y, z;

x = y = z = 100; // set x, y, and z to 100
```

The ? Operator

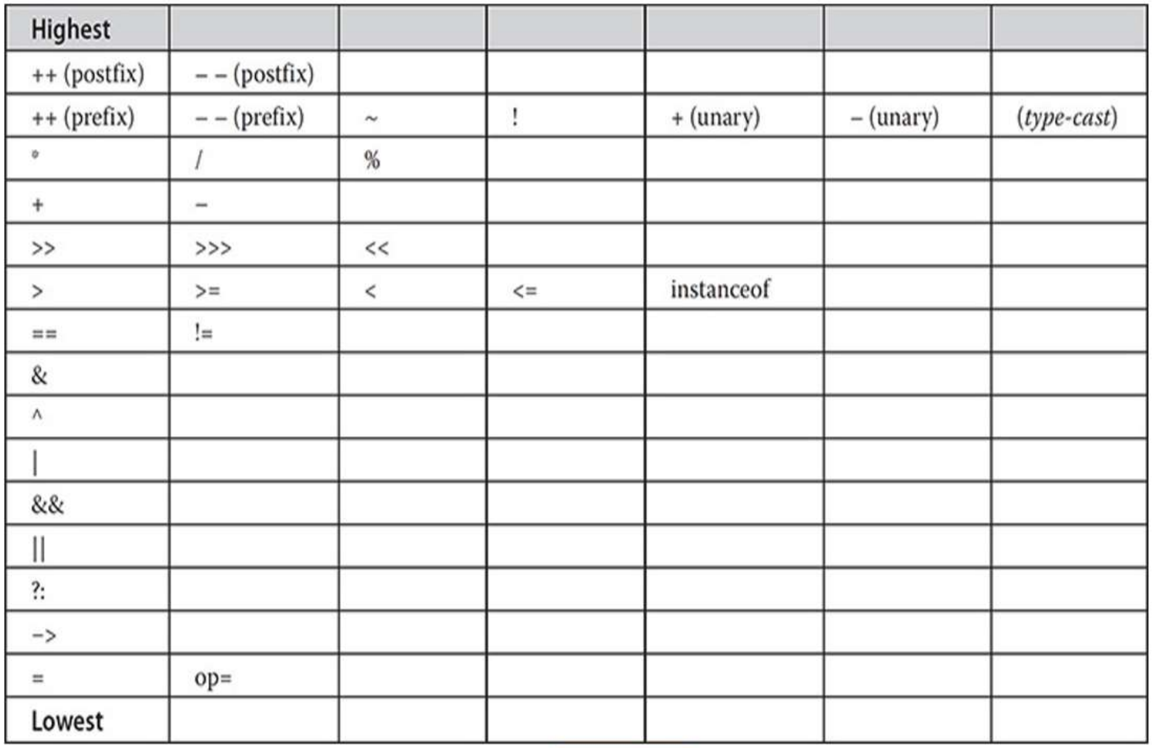
- Also known as ternary(three way) operator
- expression1?expression2:expression3

The ? Operator



```
// Demonstrate ?.
class Ternary {
  public static void main(String args[]) {
    int i, k;

    i = 10;
    k = i < 0 ? -i : i; // get absolute value of i
    System.out.print("Absolute value of ");
    System.out.println(i + " is " + k);

    i = -10;
    k = i < 0 ? -i : i; // get absolute value of i
    System.out.print("Absolute value of ");
    System.out.println(i + " is " + k);
  }
}
```

Operator Precedence



Highest						
++ (postfix)	– – (postfix)					
++ (prefix)	– – (prefix)	~	!	+ (unary)	– (unary)	(*type-cast*)
*	/	%				
+	–					
>>	>>>	<<				
>	>=	<	<=	instanceof		
==	!=					
&						
^						
|						
&&						
||						
?:						
–>						
=	op=					
Lowest						

Use of parentheses

```
a >> b + 3        →        a >> (b + 3)


            (a >> b) + 3
```