SNS COLLEGE OF ENGINEERING

Kurumbapalayam (Po), Coimbatore – 641 107

An Autonomous Institution

Accredited by NAAC – UGC with 'A' Grade

Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

23CSB101

OBJECT ORIENTED PROGRAMMING

Operators

By

M.Kanchana

Assistant Professor/CSE

Operators

- Operators are symbols that perform operations on variables and values.

- Java supports different types of operators to handle arithmetic, logical, relational, and other operations.

Types of Operators in Java

1. Arithmetic Operators
2. Relational (Comparison) Operators
3. Logical Operators
4. Bitwise & Shift Operators
5. Assignment Operators
6. Unary Operators
7. Ternary Operator

Arithmetic Operators

- Java arithmetic operators are used to perform addition, subtraction, multiplication, and division.
- They act as basic mathematical operations

Operator	Description	Example (x = 10, y = 5)	Result
`+`	Addition	`x + y`	15
`-`	Subtraction	`x - y`	5
`*`	Multiplication	`x * y`	50
`/`	Division	`x / y`	2
`%`	Modulus (Remainder)	`x % y`	0

Arithmetic Operators

Java Follows **BODMAS** rules for evaluating **Arithmetic Expressions**

BODMAS Stands for:

1. Brackets ()
2. Orders (exponents: ^, √)
3. Division / and Multiplication * (from left to right)
4. Addition + and Subtraction - (from left to right)

Arithmetic Operators

```
class Arithmetic
{
public static void main(String args[])
{
System.out.println(10*10/5+3-1*4/2);
}}
```

10 * 10 = 100
100 / 5 = 20
(20 + 3 - 1 * 4 / 2)
1 * 4 = 4
(20 + 3 - 4 / 2)
4 / 2 = 2
(20 + 3 - 2)
20 + 3 = 23
23 - 2 = 21

Output

21

Relational (Comparison) Operators

- Java relational operators are used to compare values and return a boolean (true or false).

Operator	Description	Example (x = 10, y = 5)	Result
`==`	Equal to	`x == y`	false
`!=`	Not equal to	`x != y`	true
`>`	Greater than	`x > y`	true
`<`	Less than	`x < y`	false
`>=`	Greater than or equal to	`x >= y`	true
`<=`	Less than or equal to	`x <= y`	false

Relational (Comparison) Operators

```
class Relational
{
public static void main(String args[])
{
int x = 10, y = 5;
System.out.println("x > y : "+(x > y));
System.out.println("x < y : "+(x < y));
System.out.println("x >= y : "+(x >= y));
 System.out.println("x <= y : "+(x <= y));
System.out.println("x == y : "+(x == y));
System.out.println("x != y : "+(x != y));
}
}
```

Output:

```
x > y : true
x < y : false
x >= y : true
x <= y : false
x == y : false
x != y : true
```

Logical Operators

- Used for logical operations, mostly in decision-making (if conditions).

Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	x < 5 && x < 10
||	Logical or	Returns true if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result, returns false if the result is true	!(x < 5 && x < 10)

Logical Operators

```
class logical{
public static void main(String args[])
{
boolean x = true;
boolean y = false;
System.out.println("x && y : " + (x && y));
System.out.println("x || y: " + (x || y));
System.out.println("!x : " + (!x));
}
}
```

Output:

```
x && y : false
x || y: true
!x : false
```

Bitwise and Shift Operators

- Used for bit-level operations

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 Operators

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

Bitwise Operators

1. Write the binary representation (32 bit)of the number (if negative, ignore the sign for now).
2. Invert all bits (0 → 1, 1 → 0).
3. Add 1 to the inverted bits.
4. Apply the negative sign

Bitwise Operators

Right Shift (>>) : (5>>3)

The right shift (>>) operator in Java shifts the bits of a number to the right by a specified number of positions.

- Each shift divides the number by 2 (ignoring the remainder).
- The sign bit (leftmost bit) is preserved (for signed numbers).
- It is a form of integer division by powers of 2.

0000 0000 0000 0000 0000 0000 0000 0101
0000 0000 0000 0000 0000 0000 0000 0000 -----0

5÷2^3=5÷8=0.625≈0 (rounded down)

Bitwise Operators

Left Shift (<<) : (5<<3)

The left shift (<<) operator in Java shifts the bits of a number to the left by a specified number of positions.

- Each shift multiplies the number by 2.
- The empty rightmost bits (_) are filled with 0.
- No sign change occurs for positive numbers.

0000 0000 0000 0000 0000 0000 0000 0101
0000 0000 0000 0000 0000 0000 0010 1000 -------- 40

5×2^3=5×8=40

Bitwise Operators

```
class bitwise
{
public static void main(String args[])
{       int a = 5;
       int b = 3;
            System.out.println("a & b  : " + (a & b));
              System.out.println("a | b  : " + (a | b));
       System.out.println("a ^ b  : " + (a ^ b));
        System.out.println("~a     : " + (~a));
       System.out.println("a >> 1 : " + (a >> 1));
       System.out.println("a >>> 1: " + (a >>> 1));
       System.out.println("a << 1 : " + (a << 1));
```

```
a &= b;
 System.out.println("a &= b : " + a);
 a |= b;
  System.out.println("a |= b : " + a);
       a ^= b;
       System.out.println("a ^= b : " + a);
       a >>= 1;
       System.out.println("a >>= 1: " + a);
                  a >>>= 1;
   System.out.println("a >>>= 1: " + a);
                 a <<= 1;
       System.out.println("a <<= 1: " + a);
    }
}
```

Bitwise Operators

```
class bitwise
{
public static void main(String args[])
{       int a = 5;
        int b = 3;
              System.out.println("a & b  : " + (a & b));
               System.out.println("a | b  : " + (a | b));
        System.out.println("a ^ b  : " + (a ^ b));
         System.out.println("~a     : " + (~a));
        System.out.println("a >> 1 : " + (a >> 1));
        System.out.println("a >>> 1: " + (a >>> 1));/
        System.out.println("a << 1 : " + (a << 1));
a &= b;
 System.out.println("a &= b : " + a);
 a |= b;
  System.out.println("a |= b : " + a);
        a ^= b;
        System.out.println("a ^= b : " + a);
        a >>= 1;
        System.out.println("a >>= 1: " + a);
                    a >>>= 1;
   System.out.println("a >>>= 1: " + a);
                   a <<= 1;
        System.out.println("a <<= 1: " + a);
    }
}
```

Output:

```
a & b  : 1
a | b  : 7
a ^ b  : 6
~a     : -6
a >> 1 : 2
a >>> 1: 2
a << 1 : 10
a &= b : 1
a |= b : 3
a ^= b : 0
a >>= 1: 0
a >>>= 1: 0
a <<= 1: 0
```

Assignment Operators

- '=' Assignment operator is used to assign a value to any variable.
- It has right-to-left associativity, i.e. value given on the right-hand side of the operator is assigned to the variable on the left, and therefore right-hand side value must be declared before using it or should be a constant.

The general format of the assignment operator is:

variable = value;

a=10;

Assignment Operators

In many cases, the assignment operator can be combined with others to create shorthand compound statements. For example, a += 5 replaces a = a + 5. Common compound operators include:

+= , Add and assign.
-= , Subtract and assign.
*= , Multiply and assign.
/= , Divide and assign.
%= , Modulo and assign.

Unary Operators

The Java unary operators require only one operand. Unary operators are used to perform various operations

- incrementing/decrementing a value by one
- negating an expression(~)
- inverting the value of a Boolean(!)

Unary Operators

Example	Description
val = a++;	Store the value of "a" in "val" then increments.
val = a--;	Store the value of "a" in "val" then decrements.
val = ++a;	Increments "a" then store the new value of "a" in "val".
val = --a;	Decrements "a" then store the new value of "a" in "val".

Unary Operators

```
public class unaryop {
 public static void main(String[] args) {
  int r = 6;
  System.out.println("r=: " + r++);
  System.out.println("r=: " + r);
  int x = 6;
  System.out.println("x=: " + x--);
  System.out.println("x=: " + x);
  int y = 6;
  System.out.println("y=: " + ++y);
  int p = 6;
  System.out.println("p=: " + --p);
 }
}
```

Unary Operators

Output :

r=: 6
r=: 7
x=: 6
x=: 5
y=: 7
p=: 5

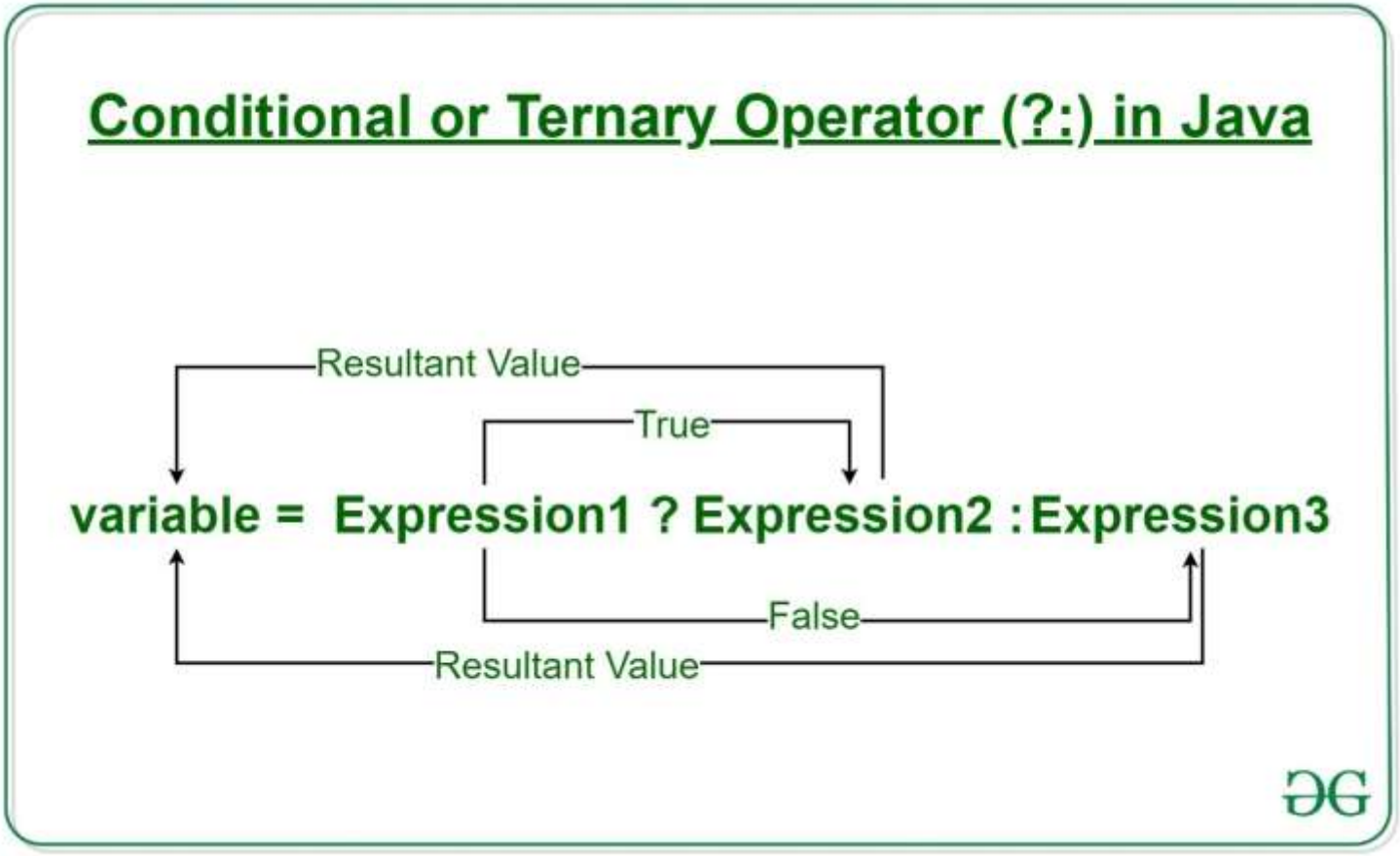
Ternary Operators

- The **Conditional operator** is the only ternary (operator takes three arguments) operator in Java.
- The operator evaluates the **first** argument and, if **true,** evaluates the **second argument**.
- If the first argument evaluates to **false**, then the **third argument** is evaluated. The conditional operator is the expression equivalent of the if-else statement.

variable = Expression1 ? Expression2: Expression3

Z =x > y ? x : y;

Ternary Operators



Ternary Operators

```
public static void main(String[] args) {
  int februaryDays = 29;
  String result;
   result = (februaryDays == 28) ? "Not a leap year" : "Leap year";
  System.out.println(result);
}
```

Output:

Leap year

Operator Precedence

- The order in which operators are applied is known as precedence. Operators with a higher precedence are applied before operators with a lower precedence.

- If two operators have the same precedence, they are applied in the order they appear in a statement. That is, from left to right. You can use parentheses to override the default precedence.

Operator Precedence

Category	Operator	Associativity
Postfix	() [] . (dot operator)	Left to right
Unary	++ - - ! ~	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	>> >>> <<	Left to right
Relational	> >= < <=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= |=	Right to left
Comma	,	Left to right

MCQ

```
public class Test {
    public static void main(String[] args) {
        int a = 10;
        int b = 5;
        System.out.println(a++ * --b);
    }
}
```

A) 50
B) 45
C) 40
D) 55

MCQ

```
public class Test {
    public static void main(String[] args) {
        int a = 10;
        int b = 5;
        System.out.println(a++ * --b);
}
}
```

A) 50
B) 45
C) 40 ✓
D) 55

MCQ

```
public class Test {
    public static void main(String[] args) {
        int a = 10;
        int b = 5;
System.out.println(--a * b++);
    }
}
```

A) 50
B) 45
C) 40
D) 55

MCQ

```
public class Test {
    public static void main(String[] args) {
        int a = 10;
        int b = 5;
System.out.println(--a * b++);
    }
}
```

A) 50
B) 45 ✓
C) 40
D) 55

MCQ

```
public class Test {
    public static void main(String[] args) {
        int x = 5;
        int y = 2;
        System.out.println(x / y * 1.0);
    }
}
```

A) 2
B) 2.5
C) 2.0
D) Compilation Error

MCQ

```
public class Test {
    public static void main(String[] args) {
        int x = 5;
        int y = 2;
        System.out.println(x / y * 1.0);
    }
}
```

A) 2
B) 2.5
C) 2.0 ✓
D) Compilation Error

MCQ

What will be the value of x after the following execution?

```
class Main
{
public static void main(String args[])
{
        int x = 10;
           x += x++ + ++x;
      System.out.println(x );

   }
}
```

A) 31
B) 32
C) 30
D) 33

MCQ

What will be the value of x after the following execution?

```
class Main
{
public static void main(String args[])
{          int x = 10;
            x += x++ + ++x;
       System.out.println(x );}}
```

A) 31
B) 32 ✓
C) 30
D) 33

Post-increment (x++): Uses the original value **before increasing**.
Pre-increment (++x**): Increases first**, then returns the new value.
Understanding precedence: x++ + ++x is evaluated before being added to x

MCQ

What will be the value of x after the following execution?

```
public class ShiftPuzzle {
 public static void main(String[] args) {
int x = 8;
System.out.println(x >> 1);
 System.out.println(x << 2);
}
 }
```

A) 4 and 16
B) 4 and 32
C) 8 and 16
D) 2 and 16

MCQ

What will be the value of x after the following execution?

```
public class ShiftPuzzle {
 public static void main(String[] args) {
int x = 8;
System.out.println(x >> 1);
 System.out.println(x << 2);
}
 }
```

A) 4 and 16
B) 4 and 32 ✓
C) 8 and 16
D) 2 and 16

Right Shift (>> n): **Divides by 2^n**
Left Shift (<< n): **Multiplies by 2^n**

THANK YOU