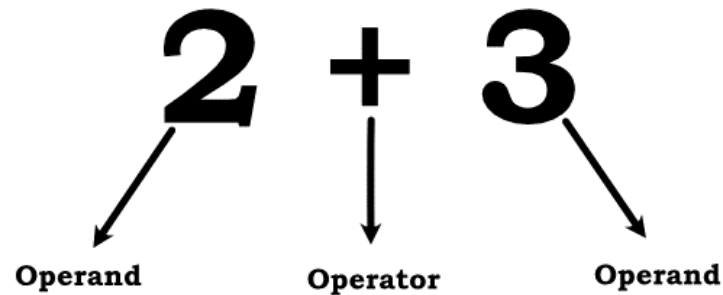


# Operations In Java

# Operator

- An operator is a symbol that operates on one or more arguments to produce a result.
- Java provides a rich set of operators to manipulate variables.

## Expression



# Operand

- An operands are the values on which the operators act upon
- An operand can be:
  - **A numeric variable** - integer, floating point or character
  - **Any primitive type variable** - numeric and Boolean
  - **Reference variable** to an object
  - **A literal** - numeric value, Boolean value, or string.
  - **An array element**, "a[2]"
  - **char primitive**, which in numeric operations is treated as an unsigned two-byte integer

# Type of Operators

1. Assignment Operators
2. Increment Decrement Operators
3. Arithmetic Operators
4. Bitwise Operators
5. Relational Operators
6. Logical Operators
7. Ternary Operators
8. Comma Operators
9. Instanceof Operators

# ASSIGNING VALUES EXAMPLE

## Assigning primitive value

```
int a, b;  
a = 2;    // 2 is assigned to variable a  
b = 5;    // 5 is assigned to variable b
```

## Assigning references

```
Home home1 = new Home(); // new object created  
Home home2 = home1;      // assigning the reference of home1 in home2
```

# INCREMENT AND DECREMENT OPERATORS

## ++ AND --

The increment and decrement operators add an integer

variable by one.

increment operator:

two successive plus signs, ++

decrement operator: --

# INCREMENT AND DECREMENT OPERATORS

++ AND --

## Common

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


```
a = a - 1;
```

## Shorthand

```
a++; or ++a;
```

```
a--; or --a;
```

```
public class Example
{
    public static void main(String[] args)
    {
        int j, p, q, r, s;
        j = 5;
        p = ++j;    //  j = j + 1;    p = j;
        System.out.println("p = " + p);
        q = j++;    //  q = j;        j = j + 1;
        System.out.println("q = " + q);
        System.out.println("j = " + j);
        r = --j;    //  j = j - 1;    r = j;
        System.out.println("r = " + r);
        s = j--;    //  s = j;        j = j - 1;
        System.out.println("s = " + s);
    }
}
```



```
> java example
p = 6
q = 6
j = 7
r = 6
s = 6
>
```



# ARITHMETIC OPERATORS

- The arithmetic operators are used to construct mathematical expressions as in algebra.
- Their operands are of numeric type.

# ARITHMETIC OPERATORS

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

```
public class Example {  
    public static void main(String[] args) {  
        int j, k, p, q, r, s, t;  
        j = 5;  
        k = 2;  
        p = j + k;  
        q = j - k;  
        r = j * k;  
        s = j / k;  
        t = j % k;  
        System.out.println("p = " + p);  
        System.out.println("q = " + q);  
        System.out.println("r = " + r);  
        System.out.println("s = " + s);  
        System.out.println("t = " + t);  
    }  
}
```



```
> java Example  
p = 7  
q = 3  
r = 10  
s = 2  
t = 1  
>
```

# BITWISE OPERATORS

- Java's bitwise operators operate on individual bits of integer (int and long) values.
- If an operand is shorter than an int, it is promoted to int before doing the operations.

Operator	Name	Description
<code>a &amp; b</code>	and	1 if both bits are 1.
<code>a   b</code>	or	1 if either bit is 1.
<code>a ^ b</code>	xor	1 if both bits are different.
<code>~a</code>	not	Inverts the bits.
<code>n &lt;&lt; p</code>	left shift	Shifts the bits of n left p positions. Zero bits are shifted into the low-order positions.
<code>n &gt;&gt; p</code>	right shift	Shifts the bits of n right p positions. If n is a 2's complement signed number, the sign bit is shifted into the high-order positions.
<code>n &gt;&gt;&gt; p</code>	right shift	Shifts the bits of n right p positions. Zeros are shifted into the high-order positions.

```
c = a ^ b; /* 49 = 0011 0001 */  
System.out.println("a ^ b = " + c);
```

```
c = ~a; /* -61 = 1100 0011 */  
System.out.println("~a = " + c);
```

```
c = a << 2; /* 240 = 1111 0000 */  
System.out.println("a << 2 = " + c);
```

```
c = a >> 2; /* 215 = 1111 */  
System.out.println("a >> 2 = " + c);
```

```
c = a >>> 2; /* 215 = 0000 1111 */  
System.out.println("a >>> 2 = " + c);
```

```
}}
```

**Output:**

**a & b = 12**

**a | b = 61**

**a ^ b = 49**

**~a = -61**

**a << 2 = 240**

**a >> 15 a >>> 15**

# RELATIONAL OPERATORS

- A relational operator compares two values and determines the relationship between them.
- For example, `!=` returns true if its two operands are unequal.
- Relational operators are used to test whether two values are equal, whether one value is greater than another, and so forth.

Operator	Description
==	Checks if the value of two operands are equal or not, if yes then condition becomes true.
!=	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.



Operator	Description
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.

```
public LessThanExample
{
    public static void main(String args[])
    {
        int a = 5; int b = 10;
        if(a < b)
        {
            System.out.println("a is less than b");
        }
    }
}
```

# LOGICAL OPERATORS

- These logical operators work only on Boolean operands. Their return values are always Boolean.

Operator	Description
&&	Called Logical AND operator. If both the operands are non zero then then condition becomes true.
	Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.

# TERNARY OPERATORS

- Java has a shorthand way by using `?:`: the ternary aka conditional operator for doing ifs that compute a value.
- Unlike the if statement, the conditional operator expression which can be used for

*// longhand with if:*

```
int answer;  
if ( a > b )  
    {  
        answer = 1;  
    }  
else  
    {  
        answer = -1;  
    }
```

*// can be written more tersely with the ternary operator as:* **int answer**  
**= a > b ? 1 : -1;**

# TERNARY OPERATORS

- Java has a shorthand way by using `?:`: the ternary aka conditional operator for doing ifs that compute a value.
- Unlike the if statement, the conditional operator expression which can be used for