# Operators

### 1. Arithmetic Operators

Used to perform basic mathematical operations.

| Operator | Description | Example |
|---|---|---|
| + | Addition | a + b |
| - | Subtraction | a - b |
| * | Multiplication | a * b |
| / | Division | a / b |
| % | Modulus (remainder) | a % b |

### 2. Unary Operators

Operate on a single operand.

| Operator | Description | Example |
|---|---|---|
| + | Unary plus (positive value) | +a |
| - | Unary minus (negation) | -a |
| ++ | Increment | ++a or a++ |
| -- | Decrement | --a or a-- |
| ! | Logical NOT | !true → false |

### 3. Relational (Comparison) Operators

Used to compare two values.

| Operator | Description | Example |
|---|---|---|
| == | Equal to | a == b |
| != | Not equal to | a != b |

| Operator | Description | Example |
|---|---|---|
| > | Greater than | a > b |
| < | Less than | a < b |
| >= | Greater than or equal to | a >= b |
| <= | Less than or equal to | a <= b |

**4. Logical Operators**

Used to perform logical operations.

| Operator | Description | Example |
|---|---|---|
| && | Logical AND | a > 5 && b < 10 |
| ! | Logical NOT | !(a > 5) |

**5. Bitwise Operators**

Operate on individual bits of integers.

| Operator | Description | Example |
|---|---|---|
| & | Bitwise AND | a & b |
| ` | ` | Bitwise OR |
| ^ | Bitwise XOR | a ^ b |
| ~ | Bitwise Complement | ~a |
| << | Left shift | a << 2 |
| >> | Right shift | a >> 2 |
| >>> | Unsigned right shift | a >>> 2 |

### 6. Assignment Operators

Used to assign values to variables.

| Operator | Description | Example |
|---|---|---|
| = | Simple assignment | a = 10 |
| += | Add and assign | a += 5 |
| -= | Subtract and assign | a -= 5 |
| *= | Multiply and assign | a *= 2 |
| /= | Divide and assign | a /= 2 |
| %= | Modulus and assign | a %= 2 |

### 7. Ternary Operator

A shorthand for if-else.

| Operator | Description | Example |
|---|---|---|
| ? : | Ternary (conditional) operator | a = (x > y) ? x : y; |

### 8. instanceof Operator

Checks if an object is an instance of a specific class or subclass.

| Operator | Description | Example |
|---|---|---|
| instanceof | Object type comparison | obj instanceof String |

---

### 9. Type Cast Operator

Used to convert data from one type to another.

| Syntax | Description | Example |
|---|---|---|
| (datatype) | Type conversion (casting) | (int) 10.5 |

---

**Operator Precedence (Important Notes)**

- **Highest to Lowest Precedence** (partial list):
- () -> . ++ -- (postfix)
- ! ~ ++ -- + - (unary)
- * / %
- + -
- << >> >>>
- < <= > >= instanceof
- == !=
- &
- ^
- |
- &&
- ||
- ? :
- = += -= etc.
- **Associativity**:
    - Most operators: **Left to Right**
    - Assignment (=, +=, etc.) and Ternary (?:): **Right to Left**

---

If you'd like I can also provide:

- A **Java code example** for each operator
- A **cheat sheet PDF**
- Or **interactive MCQs** for student testing

Let me know how you'd like to proceed.

Sure! Here's a **brief explanation of Bitwise Operators in Java**, with **examples and expected outputs**:

---

🔧 **Bitwise Operators in Java**

Bitwise operators work on the **binary representation** of integers. They compare or manipulate bits at the **bit level**.

📘 **Integer Representation:**

Every integer in Java is stored in **binary** using **2's complement format** (32 bits for int).

Example:

int a = 5;  // Binary: 00000000 00000000 00000000 00000101

int b = 3;  // Binary: 00000000 00000000 00000000 00000011

---

✅ **1. & (Bitwise AND)**

**Compares each bit of two numbers. If both bits are 1, result is 1. Else 0.**

int a = 5;  // 0101

int b = 3;  // 0011

System.out.println(a & b); // 0001 → 1

- **Output:** 1

---

✅ **2. | (Bitwise OR)**

**Compares each bit. If either bit is 1, result is 1.**

int a = 5;  // 0101

int b = 3;  // 0011

System.out.println(a | b); // 0111 → 7

- **Output:** 7

---

### ✅ 3. ^ (Bitwise XOR)

**Returns 1 if the bits are different.**

```
int a = 5;  // 0101
int b = 3;  // 0011
System.out.println(a ^ b); // 0110 → 6
```

- **Output:** 6

---

### ✅ 4. ~ (Bitwise NOT)

**Flips each bit (inverts 0 to 1 and 1 to 0).**

```
int a = 5;     // 00000000 00000000 00000000 00000101
System.out.println(~a);    // 11111111 11111111 11111111 11111010 → -6
```

- **Output:** -6

💡 In 2's complement, ~a = -(a + 1)

---

### ✅ 5. << (Left Shift)

**Shifts bits to the left. Adds 0 from the right.**

```
int a = 5;  // 0101
System.out.println(a << 1); // 1010 → 10
```

- **Output:** 10

---

### ✅ 6. >> (Right Shift)

**Shifts bits to the right. Preserves the sign bit.**

```
int a = 5;  // 0101
System.out.println(a >> 1); // 0010 → 2
```

- **Output:** 2

### 🧪 Summary Table

| Operator | Description | Example | Output |
|---|---|---|---|
| & | Bitwise AND | 5 & 3 | 1 |
| ` | ` | Bitwise OR | `5 |
| ^ | Bitwise XOR | 5 ^ 3 | 6 |
| ~ | Bitwise NOT | ~5 | -6 |
| << | Left shift | 5 << 1 | 10 |
| >> | Right shift | 5 >> 1 | 2 |

### 🔹 Ternary Operator in Java

### ✅ Syntax:

condition ? expression_if_true : expression_if_false;

- It's a **shorthand** for if-else.
- It evaluates the condition:
    - If **true** → returns expression_if_true
    - If **false** → returns expression_if_false

---

### ✅ Example:

```
public class TernaryExample {
    public static void main(String[] args) {
        int a = 10, b = 20;

        // Find the maximum of a and b
        int max = (a > b) ? a : b;

        System.out.println("Maximum is: " + max);
```

```
    }
}
```

---

🔸 **Explanation:**

- Condition: a > b → 10 > 20 → false
- Since condition is false, max = b → max = 20

---

🖨️ **Output:**

```
Maximum is: 20
```

---

✅ **Another Example: Check Even or Odd**

```
public class EvenOddTernary {
    public static void main(String[] args) {
        int number = 7;
        String result = (number % 2 == 0) ? "Even" : "Odd";
        System.out.println(number + " is " + result);
    }
}
```

🔸 number % 2 == 0 → 7 % 2 == 0 → false
🔸 So, result = "Odd"

🖨️ **Output:**

```
7 is Odd
```

---

✅ **Where Ternary Operator is Useful:**

- Compact replacement for simple if-else
- Good for **assigning values** based on conditions in a single line
- Can be **nested**, but should be used carefully for readability