### ****Operators & Control Statements****

### ****Java Operators****

In Java, operators are special symbols or keywords used to perform operations on variables and values. Java provides a variety of operators, which can be classified into several categories based on their functionality.

Here’s a breakdown of the different types of operators in Java, along with examples:

### ****1. Arithmetic Operators****

These operators are used to perform basic arithmetic operations such as addition, subtraction, multiplication, division, and modulus.

| **Operator** | **Operation** | **Example** | **Result** |
| --- | --- | --- | --- |
| + | Addition | a + b | 5 + 3 = 8 |
| - | Subtraction | a - b | 5 - 3 = 2 |
| \* | Multiplication | a \* b | 5 \* 3 = 15 |
| / | Division | a / b | 5 / 3 = 1 |
| % | Modulus (Remainder) | a % b | 5 % 3 = 2 |

#### ****Example:****

public class ArithmeticOperatorsExample {

public static void main(String[] args) {

int a = 5, b = 3;

System.out.println("a + b = " + (a + b)); // Addition

System.out.println("a - b = " + (a - b)); // Subtraction

System.out.println("a \* b = " + (a \* b)); // Multiplication

System.out.println("a / b = " + (a / b)); // Division

System.out.println("a % b = " + (a % b)); // Modulus

}

}

### ****2. Relational (Comparison) Operators****

Relational operators are used to compare two values. They return a boolean value (true or false).

| **Operator** | **Operation** | **Example** | **Result** |
| --- | --- | --- | --- |
| == | Equal to | a == b | true or false |
| != | Not equal to | a != b | true or false |
| > | Greater than | a > b | true or false |
| < | Less than | a < b | true or false |
| >= | Greater than or equal | a >= b | true or false |
| <= | Less than or equal | a <= b | true or false |

#### ****Example:****

public class RelationalOperatorsExample {

public static void main(String[] args) {

int a = 5, 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 < b: " + (a < b));

System.out.println("a >= b: " + (a >= b));

System.out.println("a <= b: " + (a <= b));

}

}

### ****3. Logical Operators****

Logical operators are used to combine multiple boolean expressions or values.

| **Operator** | **Operation** | **Example** | **Result** |
| --- | --- | --- | --- |
| && | Logical AND | a && b | true or false |
| || | Logical OR | a || b | true or false |
| ! | Logical NOT | !a | true or false |

#### ****Example:****

public class LogicalOperatorsExample {

public static void main(String[] args) {

boolean a = true, b = false;

System.out.println("a && b: " + (a && b)); // AND operation

System.out.println("a || b: " + (a || b)); // OR operation

System.out.println("!a: " + (!a)); // NOT operation

}

}

### ****4. Assignment Operators****

Assignment operators are used to assign values to variables.

| **Operator** | **Operation** | **Example** | **Result** |
| --- | --- | --- | --- |
| = | Simple Assignment | a = b | a = b |
| += | Addition Assignment | a += b | a = a + b |
| -= | Subtraction Assignment | a -= b | a = a - b |
| \*= | Multiplication Assignment | a \*= b | a = a \* b |
| /= | Division Assignment | a /= b | a = a / b |
| %= | Modulus Assignment | a %= b | a = a % b |

#### ****Example:****

public class AssignmentOperatorsExample {

public static void main(String[] args) {

int a = 5, b = 3;

a += b; // a = a + b

System.out.println("a += b: " + a);

a -= b; // a = a - b

System.out.println("a -= b: " + a);

a \*= b; // a = a \* b

System.out.println("a \*= b: " + a);

a /= b; // a = a / b

System.out.println("a /= b: " + a);

a %= b; // a = a % b

System.out.println("a %= b: " + a);

}

}

### ****5. Unary Operators****

Unary operators perform operations on a single operand.

| **Operator** | **Operation** | **Example** | **Result** |
| --- | --- | --- | --- |
| + | Unary Plus | +a | a |
| - | Unary Minus | -a | -a |
| ++ | Increment (Prefix/Postfix) | a++ or ++a | Increments a by 1 |
| -- | Decrement (Prefix/Postfix) | a-- or --a | Decrements a by 1 |
| ! | Logical NOT | !a | Negates boolean value |

#### ****Example:****

public class UnaryOperatorsExample {

public static void main(String[] args) {

int a = 5;

System.out.println("++a: " + (++a));

System.out.println("a++: " + (a++));

System.out.println("--a: " + (--a));

System.out.println("a--: " + (a--));

System.out.println("!true: " + !true);

}

}

### ****6. Bitwise Operators****

Bitwise operators perform operations on the bits of integer data types.

| **Operator** | **Operation** | **Example** | **Result** |
| --- | --- | --- | --- |
| & | Bitwise AND | a & b | Bitwise AND of a and b |
| | | Bitwise OR | a | b | Bitwise OR of a and b |
| ^ | Bitwise XOR | a ^ b | Bitwise XOR of a and b |
| ~ | Bitwise Complement | ~a | Bitwise NOT of a |
| << | Left Shift | a << 2 | Shifts bits of a to the left |
| >> | Right Shift | a >> 2 | Shifts bits of a to the right |
| >>> | Unsigned Right Shift | a >>> 2 | Shifts bits of a to the right (with zero fill) |

#### ****Example:****

public class BitwiseOperatorsExample {

public static void main(String[] args) {

int a = 5, 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 << 2: " + (a << 2));

System.out.println("a >> 2: " + (a >> 2));

System.out.println("a >>> 2: " + (a >>> 2));

}

}

### ****7. Ternary (Conditional) Operator****

The ternary operator is a shorthand for if-else and is used to evaluate a boolean condition and return one of two values based on the result.

| **Operator** | **Operation** | **Example** | **Result** |
| --- | --- | --- | --- |
| ?: | Conditional (Ternary) | condition ? value1 : value2 | Returns value1 if condition is true, else returns value2 |

#### ****Example:****

public class TernaryOperatorExample {

public static void main(String[] args) {

int a = 5, b = 3;

int max = (a > b) ? a : b;

System.out.println("Max value: " + max);

}

}

### ****8. instanceof Operator****

The instanceof operator is used to test whether an object is an instance of a specific class or subclass.

#### ****Example:****

public class InstanceofExample {

public static void main(String[] args) {

String str = "Hello";

System.out.println(str instanceof String);

}

}

### ****Control Statements in Java****

Control statements in Java allow you to control the flow of execution in your program. These are divided into three categories:

1. **Conditional Statements** (used to make decisions)
2. **Branching Statements** (used to alter the flow of control)
3. **Looping Statements** (used to repeat a block of code)

Here’s a list of all the **Control Statements** in Java:

### ****1. Conditional Statements (Decision Making)****

Conditional statements are used to execute a block of code only if a certain condition is true.

| **Statement** | **Description** |
| --- | --- |
| if | Executes a block of code if the condition is true. |
| if-else | Executes one block of code if the condition is true, and another block if it is false. |
| else-if | Chains multiple conditions to check one by one. |
| switch | Executes one of many blocks of code based on the value of an expression. |

#### ****Examples:****

**if Statement:**

if (x > 10) {

System.out.println("x is greater than 10");

}

**if-else Statement:**

if (x > 10) {

System.out.println("x is greater than 10");

} else {

System.out.println("x is less than or equal to 10");

}

**else-if Statement:**

if (x > 10) {

System.out.println("x is greater than 10");

} else if (x == 10) {

System.out.println("x is equal to 10");

} else {

System.out.println("x is less than 10");

}

**switch Statement:**

switch (day) {

case 1:

System.out.println("Monday");

break;

case 2:

System.out.println("Tuesday");

break;

case 3:

System.out.println("Wednesday");

break;

default:

System.out.println("Invalid day");

}

### ****2. Branching Statements (Flow Control)****

Branching statements are used to jump to another part of the program, either inside or outside of loops or conditional blocks.

| **Statement** | **Description** |
| --- | --- |
| break | Exits the current loop or switch statement immediately. |
| continue | Skips the current iteration of a loop and continues with the next iteration. |
| return | Exits the current method and optionally returns a value to the caller. |

#### ****Examples:****

**break Statement:**

for (int i = 0; i < 10; i++) {

if (i == 5) {

break; // Exit the loop when i equals 5

}

System.out.println(i);

}

**continue Statement:**

for (int i = 0; i < 10; i++) {

if (i == 5) {

continue; // Skip the iteration when i equals 5

}

System.out.println(i);

}

**return Statement:**

public int sum(int a, int b) {

return a + b; // Exits the method and returns the sum

}

### ****3. Looping Statements (Repetition)****

Looping statements are used to repeat a block of code multiple times based on a condition.

| **Statement** | **Description** |
| --- | --- |
| for | Executes a block of code a specific number of times. |
| while | Executes a block of code as long as the condition is true. |
| do-while | Executes a block of code at least once, then repeats based on a condition. |

#### ****Examples:****

**for Loop:**

for (int i = 0; i < 10; i++) {

System.out.println(i); // Prints numbers 0 to 9

}

**while Loop:**

int i = 0;

while (i < 10) {

System.out.println(i); // Prints numbers 0 to 9

i++;

}

**do-while Loop:**

int i = 0;

do {

System.out.println(i); // Prints numbers 0 to 9

i++;

} while (i < 10);