Java Notes: Expressions, Data Types, and Operators

Aryan Singh
From, First Principles
`aryan115csphy@gmail.com`

1 Primitive Data Types in Java

Java has 8 primitive data types. A commonly used non-primitive (reference) type is also shown below:

Type	Size	Example
byte	8-bit	byte a = 10;
short	16-bit	short b = 200;
int	32-bit	int c = 5000;
long	64-bit	long d = 100000L;
float	32-bit	float e = 3.14f;
double	64-bit	double f = 3.14159;
char	16-bit	char g = 'A';
boolean	1-bit (logical)	boolean h = true;
String (non-primitive)	Reference type	String name = "Aryan";

2 What is an Expression?

An expression is a combination of variables, values, and operators that produces a single value.

Example:

```java
int result = 5 + 3;
```

3 What is a Boolean Expression?

A boolean expression is an expression that evaluates to either `true` or `false`.

Example:

```java
int a = 10, b = 5;
boolean check = a > b;
System.out.println(check); // true
```

4 Types of Operators

4.1 1. Arithmetic Operators

Used for mathematical calculations.

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus

Example:

```
int a = 10, b = 3;
System.out.println(a % b); // 1
```

4.2 2. Relational Operators

Return boolean values.

Operator	Meaning
==	Equal to
!=	Not equal to
¿	Greater than
¡	Less than
¿=	Greater or equal
¡=	Less or equal

4.3 3. Logical Operators

Used with boolean expressions.

- **&&** : Logical AND (short-circuit)
- **||** : Logical OR (short-circuit)
- **!** : Logical NOT

Example:

```
boolean x = true;
boolean y = false;
System.out.println(x && y); // false
```

Note: Short-Circuiting (with examples)

In Java, logical operators `&&` and `||` are short-circuit operators:

- For `A && B`: if `A` is `false`, Java does not evaluate `B`.
- For `A || B`: if `A` is `true`, Java does not evaluate `B`.

Example 1: Second part skipped with `&&`

```java
int a = 5;
if (a < 0 && ++a > 10) {
    System.out.println("Inside if");
}
System.out.println(a); // 5 (not 6)
```

Example 2: Second part evaluated with `&&`

```java
int a = 5;
if (a > 0 && ++a > 5) {
    System.out.println("Condition true");
}
System.out.println(a); // 6
```

Example 3: Second part skipped with `||`

```java
int b = 10;
if (b > 0 || ++b > 20) {
    System.out.println("True");
}
System.out.println(b); // 10 (not 11)
```

Example 4: Second part evaluated with `||`

```java
int b = -1;
if (b > 0 || ++b > 0) {
    System.out.println("Now true");
}
System.out.println(b); // 0
```

Example 5: Preventing division by zero

```java
int n = 0;
if (n != 0 && 10 / n > 1) {
    System.out.println("Safe");
}
// No ArithmeticException, because 10/n is not evaluated
```

Example 6: Method call skipped

```java
static boolean check() {
    System.out.println("check() called");
    return true;
}

boolean ready = true;
if (ready || check()) {
    System.out.println("Proceed");
}
// check() is not called
```

4.4 4. Assignment Operators

Operator	Example
=	a = 5
+=	a += 2
-=	a -= 2
*=	a *= 2
/=	a /= 2
%=	a %= 2

4.5 5. Unary Operators

Operate on one operand.

- + (unary plus)
- - (unary minus)
- ++ (increment)
- – (decrement)
- ! (logical NOT)

4.6 Prefix vs Postfix Increment

Let:

```
int a = 5;
```

Prefix (++a)

```
int b = ++a;
```

Flow:

$$a = 5 \to a = 6 \to b = 6$$

Postfix (a++)

```
int c = a++;
```

Flow:

$$a = 6 \to c = 6 \to a = 7$$

5 Operator Precedence (High to Low)

1. ++, –, !
2. *, /, %
3. +, -
4. <, >, <=, >=
5. ==, !=
6. &&
7. ||
8. =, +=, -=, etc.

Example:

$$5 + 3 \times 2 = 11$$

6 Common Mistakes

6.1 1. Using = instead of ==

Wrong:

```
if(a = 5)
```

Correct:

```
if(a == 5)
```

6.2 2. Integer Division

```
System.out.println(5/2); // 2
```

6.3 3. Forgetting Postfix Behavior

```
int x = 5;
int y = x++;
```

Result:

$$y = 5, \quad x = 6$$

7 Summary

- Expressions produce values.
- Operators manipulate data.
- Prefix changes first, postfix uses first.
- Precedence controls evaluation order.
- Parentheses override precedence.

Quick Revision Points

- Use relational operators (`>`, `<`, `==`, `!=`) to build boolean expressions.
- Use `&&` and `||` for conditions; both support short-circuiting.
- Prefer parentheses in complex conditions for readability and correctness.
- Be careful with side effects like `++a` and `a++` inside conditions.
- Always guard risky operations (such as division) with safe checks first.