Java Data Types,Expressions, Expression Evaluation,Priority and Associativity,Types of Operators and Examples

-K.L.Madhavi

Java Data Types

Data types specify the type of the data that is stored in a variable.There are divided into 2 type, they are:

1.Primitive data types:Includes boolean,char,byte,short,int,long,float,double

2.Non-Primitive data types:Includes classes,interfaces,arrays etc..

2.Non-Primitive data types: Unlike primitive data types, these are not predefined. These are user-defined data types created by programmers. These data types are used to store multiple values.

Types of Non-primitive data types:

There are five types of non-primitive data types in Java. They are as follows:

- Class
- Object
- String
- Array
- Interface

- **Class and objects:** A class in Java is a user defined data type i.e. it is create by the user.It is blue print of an object.It consists of no.of objects.

  Any entity that has state and behaviour is known as object.It is an instance of class.

- **String:**A string represents a sequence of characters for example "Java", "Hello world", etc. String is the class of Java . **Ex:**String str = "You're the best";

- **Array:**An array is a data type which can store multiple homogenous variables i.e., variables of same type in a sequence.They are stored in an indexed manner starting with index 0. **Ex:**int [ ] marks;

- **Interface:**An interface is similar to a class however the only difference is that its methods are abstract by default i.e. they do not have body.

Java Expressions

- A Java expression is a combination of variables, literals, operators, and method calls that evaluates to a single value. Expressions are used to perform calculations, comparisons, and assignments.
- **Types of Java Expressions :**

- Arithmetic expressions
- Relational expressions
- Logical expressions
- Bitwise Expressions
- Assignment Expressions
- Compound Expressions

1.Arithmetic expressions: Use arithmetic operators (+, -, *, /, %).

Example: int sub = a-b;

2.Relational expressions: Use relational operators (==, !=, <, >, <=, >=).

Example: boolean result = x > y;

3.Logical expressions: Use logical operators (&&, ||, !).

Example: boolean result = (a > b) && (b < c);

4.Bitwise Expressions: Use bitwise operators (&, |, ^, ~, <<, >>).

Example: int bitResult = a & b;

5.Assignment Expressions: Use assignment operators (=, +=, -=, *=, /=, etc.).

Example: a -= b; // equivalent to a = a - b;

6.Compound Expressions: Combine multiple operators and operands.

Example: int result = (a + b) * c - d / e;

Expression Evaluation

- Expression evaluation is the process of calculating the value of an expression. Java evaluates expressions by following the order of operations, which is determined by priority and associativity.

Steps :

1.Parentheses

2.Exponents

3.Multiplication and Division

4.Addition and Subtraction

Title	Text	Example
Parentheses	Evaluate expressions inside parentheses first	(x + 5) * 2
Exponents	Evaluate exponential expressions next	x ^ 2
Multiplication and Division	Evaluate multiplication and division from left to right	x * 5 / 2
Addition and Subtraction	Evaluate addition and subtraction from left to right	x + 5 - 2

Priority And Associativity

- **Priority :** Priority refers to the order in which operators are evaluated. Operators with higher priority are evaluated first.
- **Associativity :** Associativity determines the order in which operators with the same priority are evaluated.
- **Types of Associativity :**

  Left-to-right associativity

  Right-to-left associativity

- **Steps to Evaluate :** int x=5+3*2

1. Multiply 3 and 2: 3 * 2 = 6
2. Add 5 and 6: 5 + 6 = 11
3. Assign result to x: x = 11

Operators	Priority	Associativity
++ , --	1	N/A
+ , - , ! , ~	2	Right-to-left
/ , %	3	Left-to-right
+ , -	4	Left-to-right
==, !=	5	Left-to-right
&&	6	Left-to-right
||	7	Left-to-right
=, +=, -=, *=, /=	8	Right-to-left

Operators in Java

- **Types Of Operators :**

1.Arithmetic Operators : Perform mathematical operations.

Addition (+) : x + y

Subtraction (-) : x – y

Multiplication (*) : x * y

Division (/) : x / y →returns quotient of division operation.

Modulus (%) : x % y →returns remainder of division operation.

Ex:

```
int a = 10, b = 5;
int sum = a + b; // 15
int product = a * b; // 50
int remainder = a % b; // 0
```

2.Assignment Operators : Assign values to variables.

Simple (=) : x = y

Addition (+=) : x += y →equivalent to x=x+y

Subtraction (-=) : x -= y → equivalent to x=x-y

Multiplication (*=) : x *= y → equivalent to x=x*y

Division (/=) : x /= y →equivalent to x=x/y

Modulus (%=) : x % =y → equivalent to x=x%y

Ex:

```
int x = 10;
x += 5; // Equivalent to x = x + 5; now x is 15
x *= 2; // Equivalent to x = x * 2; now x is 30
```

3.Comparison Operators : Compare values

Equal to (==) : x == y

Not Equal to (!=) : x != y

Greater than (>) : x > y

Less than (<) : x < y

Less than or equal to (<=) : x<=y

Greater than or equal to (>=) : x>=y

Ex:

```
int x = 10, y = 20;
boolean isEqual = (x == y); // false
boolean isGreater = (x > y); // false
```

4.Logical Operators : Perform logical operations

AND (&&) : x && y

OR (||) : x || y

NOT (!x) : !x

Ex:

```
boolean a = true, b = false;
boolean result = a && b; // false
boolean orResult = a || b; // true
boolean notResult = !a; // false
```

5.Bitwise Operators : Perform bit level operation

Bitwise AND (&) : x & y

Bitwise OR(|) : x | y

Bitwise XOR (^) : x ^ y

Bitwise NOT(~) :~x

Left shift(<<) : x<<y

Right shift(>>) : x>>y

Ex:

```
int a = 5;  // 0101 in binary
int b = 3;  // 0011 in binary
int andResult = a & b; // 0001 (1 in decimal)
int orResult = a | b;  // 0111 (7 in decimal)
```

6.Increment/Decrement Operators : Increment/Decrement values

Pre-Increment (++) : ++x

Post-Increment (++) : x++

Pre-Decrement(--) : --x

Post-Decrement(--) : x—

Ex:

```
int x = 10;
x++; // Increment x; now x is 11
x--; // Decrement x; now x is 10
```

7.Conditional (Ternary) Operator : Evaluate conditions

Conditional (?:) : x > y ? x : y

Ex:

```
int a = 10, b = 20;
int max = (a > b) ? a : b; // max is 20
```