1.3. DATA TYPES

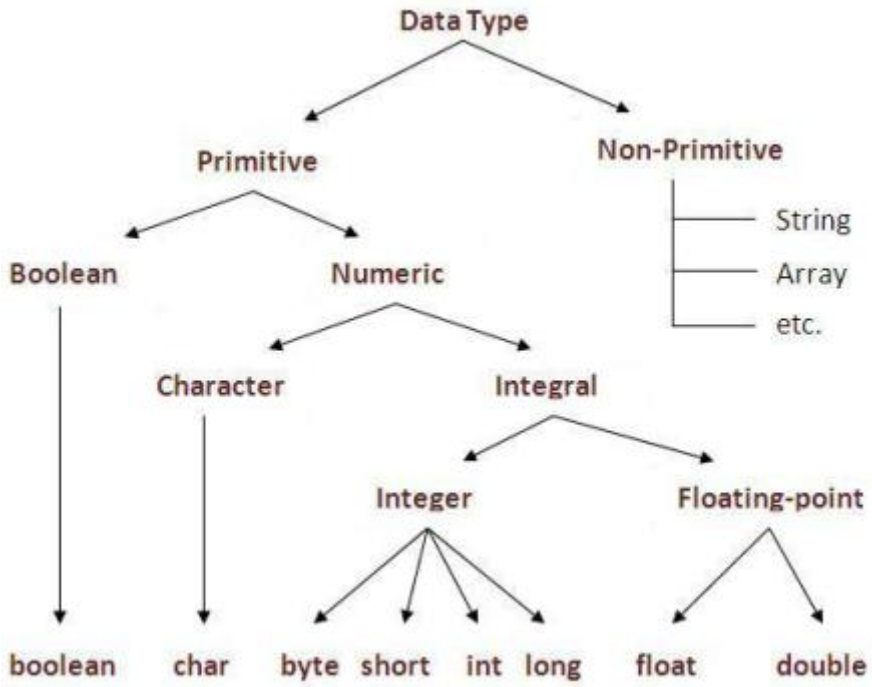
There are two data types available in Java.

Primitive Data Types

- There are eight primitive data types supported by Java.
- Primitive data types are predefined by the language and named by a keyword.
- Let us now look into the eight primitive data types in detail.

byte

- Byte data type is an 8-bit signed two's complement integer.
- Minimum value is -128 (-2^7).
- Maximum value is 127 (inclusive)(2^7-1).
- Default value is 0.
- Byte data type is used to save space in large arrays, mainly in place of integers, since a byte is four times smaller than an integer.
- Example: byte a = 100, byte b = -50.



short

- Short data type is a 16-bit signed two's complement integer.
- Minimum value is -32,768 (-2^{15}).
- Maximum value is 32,767 (inclusive) (2^{15} -1).
- Short data type can also be used to save memory as byte data type. A short is 2 times smaller than an integer.
- Default value is 0.
- Example: short s = 10000, short r = -20000.

int

- Int data type is a 32-bit signed two's complement integer.
- Minimum value is - 2,147,483,648 (-2^{31}).
- Maximum value is 2,147,483,647(inclusive) (2^{31} -1).
- Integer is generally used as the default data type for integral values unless there is a concern about memory.
- The default value is 0.
- Example: int a = 100000, int b = -200000.

long

- Long data type is a 64-bit signed two's complement integer.
- Minimum value is -9,223,372,036,854,775,808(-2^{63}).
- Maximum value is 9,223,372,036,854,775,807 (inclusive)(2^{63} -1).
- This type is used when a wider range than int is needed.
- Default value is 0L.
- Example: long a = 100000L, long b = -200000L.

float

- Float data type is a single-precision 32-bit IEEE 754 floating point.
- Float is mainly used to save memory in large arrays of floating point numbers.
- Default value is 0.0f.
- Float data type is never used for precise values such as currency.
- Example: float f1 = 234.5f.

double

- double data type is a double-precision 64-bit IEEE 754 floating point.
- This data type is generally used as the default data type for decimal values, generally the default choice.
- Double data type should never be used for precise values such as currency.
- Default value is 0.0d.
- Example: double d1 = 123.4.

boolean

- boolean data type represents one bit of information.
- There are only two possible values: true and false.
- This data type is used for simple flags that track true/false conditions.
- Default value is false.
- Example: boolean one = true.

char

- char data type is a single 16-bit Unicode character.

- Minimum value is '\u0000' (or 0).
- Maximum value is '\uffff' (or 65,535 inclusive).
- Char data type is used to store any character.
- Example: char letterA = 'A'.

Non-primitive Data Types

- In java, non-primitive data types are the reference data types or user-created data types.
- All non-primitive data types are implemented using object concepts. Every variable of the non-primitive data type is an object.
- The non-primitive data types may use additional methods to perform certain operations.
- The default value of non- primitive data type variable is null.
- In java, examples of non-primitive data types are String, Array, List, Queue, Stack, Class, Interface, etc.

Variables

- A variable is the holder that can hold the value while the java program is executed.
- A variable is assigned with a datatype. It is name of *reserved area allocated in memory*.
- In other words, it is a *name of memory location*. There are three types of variables in java: local, instance and static.

Declaring a Variable:

Before using any variable, it must be declared.

datatype variable_neme = value;

To declare more than one variable of the specified type, use a comma-separated list.

Example

```
int a, b, c;                    // Declaration of variables a, b, and c.
int a = 20, b = 30;  // initialization
byte B = 22;                    // Declaratrion initializes a byte type variable B.
```

Types of Variable:

There are three types of variables in java:

- ✓ local variable
- ✓ instance variable
- ✓ static variable

Local Variable

- Local variables are declared inside the methods, constructors, or blocks.
- Local variables are created when the method, constructor or block is entered
- Local variable will be destroyed once it exits the method, constructor, or block.
- Local variables are visible only within the declared method, constructor, or block.
- Local variables are implemented at stack level internally.

- There is no default value for local variables, so local variables should be declared and an initial value should be assigned before the first use.
- Access specifiers cannot be used for local variables.Instance Variable
- Instance variables are declared in a class, but outside a method, constructor or any block.
- Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.
- Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object's state that must be present throughout the class.
- Instance variables can be declared in class level before or after use.
- Access modifiers can be given for instance variables.
- The instance variables are visible for all methods, constructors and block in the class. It is recommended to make these variables as private. However, visibility for subclasses can be given for these variables with the use of access modifiers.
- Instance variables have default values.
    - numbers, the default value is 0,
    - Booleans it is false,
    - Object references it is null.
- Values can be assigned during the declaration or within the constructor.
- Instance variables cannot be declared as static.
- Instance variables can be accessed directly by calling the variable name inside the class. However, within static methods (when instance variables are given accessibility), they should be called using the fully qualified name. ObjectReference.VariableName.

Static variable

- Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block.
- Static variables are rarely used other than being declared as constants. Constants are variables that are declared as public/private, final, and static. Constant variables never change from their initial value.
- Static variables are stored in the static memory. It is rare to use static variables other than declared final and used as either public or private constants.
- Static variables are created when the program starts and destroyed when the program stops.
- Visibility is same as instance variables. However, most static variables are declared public since they must be available for users of the class.
- Default values are same as instance variables.
    - numbers, the default value is 0;

- Booleans, it is false;
- Object references, it is null.

1.4 OPERATORS:

Operator in java is a symbol that is used to perform operations. Java provides a rich set of operators to manipulate variables.For example: +, -, *, / etc.

All the Java operators can be divided into the following groups −

- ✓ Arithmetic Operators :
- ✓ Relational Operators
- ✓ Bitwise Operators
- ✓ Logical Operators
- ✓ Assignment Operators: =
- ✓ Ternary operator: ? :
- ✓ Unary operator

The Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations in the same way as they are used in algebra.

Example:

int A=10,B=20;

Operator	Description	Example	Output
+ (Addition)	Adds values A & B.	A + B	30
- (Subtraction)	Subtracts B from A	A - B	-10
* (Multiplication)	Multiplies values A & B	A * B	200
/ (Division)	Divides B by A	B / A	2
% (Modulus)	Returns remainder.	B % A	0

```
// Java program to illustrate arithmetic operators
public class Aoperators
{
public static void main(String[] args)
{
int a = 20, b = 10, c = 0, d = 20, e = 40, f = 30;
String x = "Thank", y = "You";
System.out.println("a + b = "+(a + b));
System.out.println("a - b = "+(a - b));
                System.out.println("x + y = "+x + y);
System.out.println("a * b = "+(a * b)); System.out.println("a
```

/ b = "+(a / b));

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

}

}

The Relational Operators

The following relational operators are supported by Java language.

Example: int A=10,B=20;

Operator	Description	Example	Output
== (equal to)	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B)	true
!= (not equal to)	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B)	true
> (greater than)	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B)	true
< (less than)	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B)	true
>= (greater than or equal to)	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition be- comes true.	(A >= B)	true
<= (less than or equal to)	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B)	true
instance of Operator	checks whether the object is of a partic- ular type (class type or interface type) (Object reference variable) instanceof (class/interface type)	boolean result = name instanceof String;	True

// Java program to illustrate relational operators

public class operators

{

public static void main(String[] args)

{

int a = 20, b = 10;

boolean condition = true;

```
//various conditional operators
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));
System.out.println("condition==true :" + (condition == true));
}
}
```

Bitwise Operators

- Java supports several bitwise operators, that can be applied to the integer types, long, int, short, char, and byte.
- Bitwise operator works on bits and performs bit-by-bit operation.

Example:

int a = 60,b = 13;

binary format of a & b will be as follows −

a = 0011 1100 b = 0000 1101

Bitwise operators follow the truth table:

a	b	a&b	a|b	a^b	~a
0	0	0	0	1	1
0	1	0	1	0	1
1	0	0	1	0	0
1	1	1	1	1	0

Operator	Description	Example	Output
& (bit- wise and)	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B)	12 (in binary form: 0 0 0 0 1100)
bitwise or)	Binary OR Operator copies a bit if it exists in either operand.	(A | B)	61 (in binary form: 0011 1101)
^ (bitwise XOR)	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B)	49 (in binary form: 0011 0001)

~ (bitwise compli-ment)	Binary Ones complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61 which is 1100 0011	-61 (in binary form: 1100 0011)
<< (left shift)	The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000	240 (in binary form: 1111 0000)
>> (right shift)	The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 1111	15 (in binary form: 1111)
>>> (zero fill right shift)	The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.	A >>>2 will give 15 which is 0000 1111	15 (in binary form: 0000 1111)

```
// Java program to illustrate bitwise operators
public class operators
{
public static void main(String[] args)
{
int a = 10; int b = 20;
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);
}
}
```

Logical Operators

The following are the logical operators supported by java.

Example:

A=true; B=false;

Operator	Description	Example	Output
&& (logical and)	If both the operands are non-zero, then the condition becomes true.	(A && B)	false

|| (logical or)	If any of the two operands are non- zero, then the condition becomes true.	(A || B)	true
! (logical not)	Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B)	true

Assignment Operators

The following are the assignment operators supported by Java.

Operator	Description	Example
= (Simple assignment operator)	Assigns values from right side oper- ands to left side operand.	C = A + B will assign value of A + B into C
+= (Add AND assignment operator)	It adds right operand to the left operand and assigns the result to left operand.	C += A is equivalent to C = C + A
-= (Subtract AND assignment operator)	It subtracts right operand from the left operand and assigns the result to left operand.	C -= A is equivalent to C = C – A
*= (Multiply AND assignment operator)	It multiplies right operand with the left operand and assigns the result to left operand.	C *= A is equivalent to C = C * A
/= (Divide AND assignment operator)	It divides left operand with the right operand and assigns the result to left operand.	C /= A is equivalent to C = C / A
%= (Modulus AND assignment operator)	It takes modulus using two operands and assigns the result to left operand.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2

>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator.	C |= 2 is same as C= C | 2

```
public class AssignmentOperators
{
        public static void main(String[] args)
        {
int a = 20, b = 10, c, d, e = 10, f = 4, g = 9; c = b;
System.out.println("Value of c = " + c);
a += 1;
b -= 1;
e *= 2;
f /= 2;
System.out.println("a, b, e, f = " +a + "," + b + "," + e + "," + f);
        }
}
```

Ternary Operator

Conditional Operator (? :)

- Since the conditional operator has three operands, it is referred as the **ternary operator**.
- This operator consists of three operands and is used to evaluate Boolean expressions. The goal of the operator is to decide, which value should be assigned to the variable.
- The operator is written as

variable x = (expression) ? value if true : value if false

Example:

```
public class example
{
        public static void main(String args[])
```

```
{
        int a, b;
        a = 10;
        b = (a == 0) ? 20: 30;
        System.out.println( "b : " +  b );
}}
```

Unary Operators

Unary operators use only one operand. They are used to increment, decrement or negate a value.

Operator	Description
- Unary minus	negating the values
+ Unary plus	converting a negative value to positive
++ :Increment operator	incrementing the value by 1
— : Decrement operator	decrementing the value by 1
! : Logical not operator	inverting a boolean value

```
public class operators
{
        public static void main(String[] args)
        {
int a = 20, b = 10, c = 0, d = 20, e = 40, f = 30;
boolean condition = true;
c = ++a;
System.out.println("Value of c (++a) = " + c);
c = b++;
System.out.println("Value of c (b++) = " + c);
c = --d;
System.out.println("Value of c (--d) = " + c);
c= --e;
System.out.println("Value of c (--e) = " + c);
System.out.println("Value of !condition =" + !condition);
}       }
```

Precedence of Java Operators

- Operator precedence determines the grouping of operands in an expression. This affects how an expression is evaluated.

➢ Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator.

x = 10 + 5 * 2;

5 * 2 is evaluated first. Because operator * has higher precedence than +.

Category	Operator	Associativity
Postfix	>() [] . (dot operator)	Left to right
Unary	>++ - - ! ~	Right to left
Multiplicative	>* /	Left to right
Additive	>+ -	Left to right
Shift	>>> >>> <<	Left to right
Relational	>> >= < <=	Left to right
Equality	>== !=	Left to right
Bitwise AND	>&	Left to right
Bitwise XOR	>^	Left to right
Bitwise OR	>|	Left to right
Logical AND	>&&	Left to right
Logical OR	>||	Left to right
Conditional	?:	Right to left
Assignment	>= += -= *= /= %= >>= <<= &= ^= |=	Right to left