**Java Data Types :**

Data types are divided into two groups:

* Primitive data types - includes byte, short, int, long, float, double, boolean and char
* Non-primitive data types - such as [String](https://www.w3schools.com/java/java_strings.asp), [Arrays](https://www.w3schools.com/java/java_arrays.asp) and [Classes](https://www.w3schools.com/java/java_classes.asp) (you will learn more about these in a later chapter)

## Primitive Data Types

A primitive data type specifies the size and type of variable values, and it has no additional methods.

There are eight primitive data types in Java:

|  |  |  |
| --- | --- | --- |
| **Primitive Data Type** | **Size** | **Details** |
| Byte | 1 byte | Stores positive and negative numbers ranging from -128 to 127. |
| Int | 4 bytes | Stores positive and negative numbers ranging from  -2,147,483,648 to 2,147,483,647. |
| Short | 2 bytes | Stores positive and negative numbers ranging from  -32,768 to 32,767. |
| Long | 8 bytes | Stores positive and negative numbers from  -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807. |
| Float | 4 bytes | Stores Decimal numbers. It can be used for storing numbers having  6 to 7 decimal digits |
| Double | 8 bytes | Stores Decimal numbers. It can be used for storing numbers having  15 decimal digits. |
| Boolean | 1 bit | Can Store Only true or false. |
| Char | 2 bytes | It can be used for storing only a single character,  letter or ASCII values |

## What is token in Java?

The Java compiler breaks the line of code into text (words) is called **Java tokens**. These are the smallest element of the [Java program](https://www.javatpoint.com/java-programs). The Java compiler identified these words as tokens. These tokens are separated by the delimiters. It is useful for compilers to detect errors. Remember that the delimiters are not part of the Java tokens.

Types of Tokens

Java token includes the following:

* Keywords
* Identifiers
* Literals
* Operators
* Separators
* Comments
* **Keywords:** These are the **pre-defined** reserved words of any programming language. Each [keyword](https://www.javatpoint.com/java-keywords) has a special meaning. It is always written in lower case. Java provides the following keywords:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 01. abstract | 02. boolean | 03. byte | 04. break | 05. class |
| 06. case | 07. catch | 08. char | 09. continue | 10. default |
| 11. do | 12. double | 13. else | 14. extends | 15. final |
| 16. finally | 17. float | 18. for | 19. if | 20. implements |
| 21. import | 22. instanceof | 23. int | 24. interface | 25. long |
| 26. native | 27. new | 28. package | 29. private | 30. protected |
| 31. public | 32. return | 33. short | 34. static | 35. super |
| 36. switch | 37. synchronized | 38. this | 39. thro | 40. throws |
| 41. transient | 42. try | 43. void | 44. volatile | 45. while |
| 46. assert | 47. const | 48. enum | 49. goto | 50. strictfp |

**Identifier:** Identifiers are used to name a variable, constant, function, class, and array. It usually defined by the user. It uses letters, underscores, or a dollar sign as the first character. The label is also known as a special kind of identifier that is used in the goto statement. Remember that the identifier name must be different from the reserved keywords. There are some rules to declare identifiers are:

* The first letter of an identifier must be a letter, underscore or a dollar sign. It cannot start with digits but may contain digits.
* The whitespace cannot be included in the identifier.
* Identifiers are case sensitive.

Some valid identifiers are:

1. PhoneNumber
2. PRICE
3. radius
4. a
5. a1
6. \_phonenumber
7. $circumference
8. jagged\_array
9. 12radius   //invalid

**Literals:** In programming literal is a notation that represents a fixed value (constant) in the source code. It can be categorized as an integer literal, string literal, Boolean literal, etc. It is defined by the programmer. Once it has been defined cannot be changed. Java provides five types of literals are as follows:

* Integer
* Floating Point
* Character
* String

|  |  |
| --- | --- |
| **Literal** | **Type** |
| 23 | Int |
| 9.86 | Double |
| false, true | Boolean |
| 'K', '7', '-' | Char |
| "javatpoint" | String |
| Null | any reference type |

* Boolean

**Separators:** The separators in Java is also known as **punctuators**. There are nine separators in Java, are as follows:

* **Square Brackets []:** It is used to define array elements. A pair of square brackets represents the single-dimensional array, two pairs of square brackets represent the two-dimensional array.
* **Parentheses ():** It is used to call the functions and parsing the parameters.
* **Curly Braces {}:** The curly braces denote the starting and ending of a code block.
* **Comma (,):** It is used to separate two values, statements, and parameters.
* **Assignment Operator (=):** It is used to assign a variable and constant.
* **Semicolon (;):** It is the symbol that can be found at end of the statements. It separates the two statements.
* **Period (.):** It separates the package name form the sub-packages and class. It also separates a variable or method from a reference variable.

**Operators:**

In programming, operators are the special symbol that tells the compiler to perform a special operation. Java provides different types of operators that can be classified according to the functionality they provide. There are eight types of [operators in Java](https://www.javatpoint.com/operators-in-java), are as follows:

1)Arithmetic operators

2)Relational operators

3)Logical Operators

4)Assignment operators

5)Conditional operator

6)Bitwise Operator

7)Special Operator (2)

**Arithmetic Operators :**

Arithmetic operators are used to perform arithmetic operations on variables and data.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| + (Addition) | Adds values on either side of the operator. | A + B will give 30 |
| - (Subtraction) | Subtracts right-hand operand from left-hand operand. | A - B will give -10 |
| \* (Multiplication) | Multiplies values on either side of the operator. | A \* B will give 200 |
| / (Division) | Divides left-hand operand by right-hand operand. | B / A will give 2 |
| % (Modulus) | Divides left-hand operand by right-hand  operand and returns remainder. | B % A will give 0 |
| ++ (Increment) | Increases the value of operand by 1. | B++ gives 21 |
| -- (Decrement) | Decreases the value of operand by 1. | B-- gives 19 |

public class Test {

public static void main(String args[]) {

int a = 10;

int b = 20;

int c = 25;

int d = 25;

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

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

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

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

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

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

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

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

// Check the difference in d++ and ++d

System.out.println("d++ = " + (d++) );

System.out.println("++d = " + (++d) );

}

}

**Relational Operators :**

Used to check relation between two operands.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| == (equal to) | Checks if the values of two operands are equal or not, if yes then condition becomes true. | (A == B) is not 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) is 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) is not 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) is 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 becomes true. | (A >= B) is not 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) is true. |

**Logical Operators :**

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

public class Test {

public static void main(String args[]) {

boolean a = true;

boolean b = false;

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

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

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

}

}

Output :

a && b = false

a || b = true

!(a && b) = true

**Assignment Operator :**

Used to assign values to variable

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| = | Simple assignment operator. Assigns values from right side operands 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 the 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 the 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 the 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 the 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 the 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 Test {

public static void main(String args[]) {

int a = 10;

int b = 20;

int c = 0;

c = a + b;

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

c += a ;

System.out.println("c += a = " + c );

c -= a ;

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

c \*= a ;

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

a = 10;

c = 15;

c /= a ;

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

a = 10;

c = 15;

c %= a ;

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

c <<= 2 ;

System.out.println("c <<= 2 = " + c );

c >>= 2 ;

System.out.println("c >>= 2 = " + c );

c >>= 2 ;

System.out.println("c >>= 2 = " + c );

c &= a ;

System.out.println("c &= a = " + c );

c ^= a ;

System.out.println("c ^= a = " + c );

c |= a ;

System.out.println("c |= a = " + c );

}

}

Output:

c = a + b = 30

c += a = 40

c -= a = 30

c \*= a = 300

c /= a = 1

c %= a = 5

c <<= 2 = 20

c >>= 2 = 5

c >>= 2 = 1

c &= a = 0

c ^= a = 10

c |= a = 10

**Conditional Operator :**

The conditional operator is also known 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

**public class Test {**

**public static void main(String args[]) {**

**int a, b;**

**a = 10;**

**b = (a == 1) ? 20: 30;**

**System.out.println("Value of b is: " + b);**

**b = (a == 10) ? 20: 30;**

**System.out.println(“Value of b is: " + b);**

**}**

**}**

**Output :**

**Value of b is: 30**

**Value of b is: 20**

**Bitwise Operators :**

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| & (bitwise and) | Binary AND Operator copies a bit to the result if it exists in both operands. | (A & B) will give 12 which is 0000 1100 |
| | (bitwise or) | Binary OR Operator copies a bit if it exists in either operand. | (A | B) will give 61 which is 0011 1101 |
| ^ (bitwise XOR) | Binary XOR Operator copies the bit if it is set in one operand but not both. | (A ^ B) will give 49 which is 0011 0001 |
| ~ (bitwise compliment) | Binary Ones Complement Operator is unary and has the effect of 'flipping' bits. | (~A ) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number. |
| << (left shift) | Binary Left Shift Operator. 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 |
| >> (right shift) | Binary Right Shift Operator. 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 |
| >>> (zero fill right shift) | Shift right zero fill operator. 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 |

public class Test {

public static void main(String args[]) {

int a = 60; /\* 60 = 0011 1100 \*/

int b = 13; /\* 13 = 0000 1101 \*/

int c = 0;

c = a & b; /\* 12 = 0000 1100 \*/

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

c = a | b; /\* 61 = 0011 1101 \*/

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

c = a ^ b; /\* 49 = 0011 0001 \*/

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

c = ~a; /\*-61 = 1100 0011 \*/

System.out.println("~a = " + c );

c = a << 2; /\* 240 = 1111 0000 \*/

System.out.println("a << 2 = " + c );

c = a >> 2; /\* 15 = 1111 \*/

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

c = a >>> 2; /\* 15 = 0000 1111 \*/

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

}

}

Output :

a & b = 12

a | b = 61

a ^ b = 49

~a = -61

a << 2 = 240

a >> 2 = 15

a >>> 2 = 15

**Special Operators :**

**1)InstanceOf :**

This operator is used only for object reference variables.

**public class Test {**

**public static void main(String args[]) {**

**String name = "James"; // following will return true since name is type of String**

**boolean result = name instanceof String;**

**System.out.println( result );**

**}**

**}**

**Output :true**

**2)Dot Operator :**

Dot operator is a syntactic element, i.e. it denotes the separation between class and package, method and class, variable and reference variable. It can also be called as separator operator.

Class person{

String name;

int age;

void display(){

System.out.println(“Name : ”+name);

System.out.println(“Age : ”+age);

}

public static void main(String[] args){

Person p = new Person();

p.name = “abc”;

p.age = “10”;

}

}

Output :

Name : abc

Age : 10

**Comments:** [Comments](https://www.javatpoint.com/java-comments) allow us to specify information about the program inside our Java code. Java compiler recognizes these comments as tokens but excludes it form further processing. The Java compiler treats comments as whitespaces. Java provides the following two types of comments:

* **Line Oriented:** It begins with a pair of forwarding slashes (**//**).
* **Block-Oriented:** It begins with /\* and continues until it founds **\*/**.