Data Types in Java

In programming languages, every value or data has an associated type known as data type. Java supports various data types. These data types are categorized into,

1. Primitive Data Types
2. Non-Primitive Data Types

Primitive Data Types Primitive data types are those that are predefined by the programming language (Java).

boolean: In general, anything that can take one of two possible values is considered a boolean. In Java, true and false are considered boolean values.

boolean canLearn = true;

byte: The byte data type is used to store integers without any fractional part whose values are in the range -128 to 127.

byte points = 100;

short : The short data type is used to store integers without any fractional part whose values are in the range -32,768 to 32,767.

short number = 28745;

int: The int data type is used to store integers without any fractional part whose values are in the range -2^{31} to $2^{31}-1$.

int distance = 2036372;

long: The long data type is used to store integers without any fractional part whose values are in the range -2^{63} to $2^{63}-1$. The long values should contain the suffix 'l' or 'L'.

long area = 2036372549999L;

float: The float data type is used to store any number with a decimal point. The float data type stores a value up to 7 point precision (ex: 12.1234567). The float values should contain the suffix 'f' or 'F'.

float height = 5.10f;

double: The double data type is used to store any number with a decimal point. The double data type stores a value up to 16 point precision. The double values may contain the suffix 'd' or 'D'.

double breadth = 9.2345D;

Non-Primitive Data Types These data types are used to store multiple values. Non-primitive data types are defined by the programmer. In Java programming, all non-primitive data types are simply called objects. Some commonly used non-primitive data types are,

- String
- Array
- Class

String: The String data type is simply a sequence of characters. In Java, double quotes (") are used to represent a string.

```
String name = "Rahul";
```

Array: In Java, an array is an object used to store similar data type values. In Java, the number of elements that an array can store is always fixed.

```
int[] arr = {60, 25, 20, 15, 30};
```

Conditional Statements

Conditional Statement: Conditional Statement allows us to execute a block of code only when a specific condition is true.

If...Else Statements: When an if...else conditional statement is used, the if block of code executes when the condition is true, otherwise the else block of code is executed.

```
int token = 20;
if (token == 20)
   System.out.println("Collect Water Bottle");
else
   System.out.println("Invalid Token");
```

// Output is:

```
Collect Water Bottle
```

Else if Statement: Java provides an else if statement to have multiple conditional statements between if and else. The else if statement is optional. We can add any number of else if statements after if conditional block.

```
int token = 20;
if (token == 10)
   System.out.println("Collect Chips");
else if(token == 20)
   System.out.println("Collect Soft Drink");
else
   System.out.println("Invalid Token");
```

// Output is:

```
Collect Soft Drink
```

Switch: A switch block can have multiple case or default labels. The switch statement allows us to execute a block of code among many cases.

```
switch (100 / 10) {
  case 10:
    System.out.println("Ten");
    break;
  case 20:
    System.out.println("Twenty");
    break;
  default:
    System.out.println("Other Number");
    break;
}
// Output is:
Ten
```

Nested Conditions: The conditional statements inside another conditional statement is called a nested conditional statement.

```
int token = 30;
char softDrink = 'S';
if (token == 30) {
  if (softDrink == 'S')
    System.out.println("Collect Sprite");
  else if (softDrink == 'F')
    System.out.println("Collect Fanta");
  else
    System.out.println("Invalid Option");
}
else
  System.out.println("Invalid Token");
// Output is:
 Collect Sprite
```

String - working with strings

Creating Strings:

- **Using String Literals**: A string literal is a value which is enclosed in double quotes (" ").

```
String str = "ABC";
```

- **Using new Keyword**: A string can be created by initializing the String class with the new keyword.

```
String str1 = new String(str);
```

String Concatenation: The concat() method appends one string to the end of another string. Using this method we can concatenate only two strings.

```
String concatenatedStr = "Hello ".concat("World");

System.out.println(concatenatedStr);  // Hello World

System.out.println("Hello " + "World");  // Hello World
```

Length of String: The string object has a length() method that returns the number of characters in a given string.

```
String name = "Rahul";

int strLength = name.length();

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

String Indexing: We can access an individual character in a string using their positions. These positions are also called indexes. The charAt() method is used to get the character at a specified index in a string.

```
String name = "Rahul";

char firstLetter = name.charAt(0);

System.out.println(firstLetter); // R
```

String Slicing: Obtaining a part of a string is called string slicing. The substring() method can be used to access a part of the string.

```
String message = "Welcome to Java";

String part = message.substring(0, 5);

System.out.println(part); // Welco
```

Slicing to end

```
String message = "Welcome to Java";

String part = message.substring(11);

System.out.println(part); // Java
```

String Repetition: The repeat() method returns a string whose value is the concatenation of the given string repeated N times. If the string is empty or the count is zero then the empty string is returned.

```
String str = "Rahul";

System.out.println(str.repeat(2)); // RahulRahul
```

Calculations in Java

Addition: Addition is denoted by + sign. It gives the sum of two numbers.

System.out.println(2 + 5); *// 7*

System.out.println(1 + 1.5); *// 2.5*

Subtraction: Subtraction is denoted by - sign. It gives the difference between the two numbers.

System.out.println(5 - 2); *// 3*

Multiplication: Multiplication is denoted by * sign. It gives the product of two numbers.

System.out.println(5 * 2); *// 10*

System.out.println(5 * 0.5); *// 2.5*

Division: Division is denoted by / sign. It returns the quotient as a result.

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

System.out.println(4 / 2); *// 2*

Modulo operation: Modulo is denoted by % sign. It returns the modulus (remainder) as a result.

System.out.println(5 % 2); *// 1*

System.out.println(4 % 2); *// 0*

Input and Output Basics

Take Input From User: The Scanner class in the package java.util is used to read user input.

```
import java.util.Scanner;

class Main {

  public static void main(String[] args) {

    Scanner input = new Scanner(System.in);

    String username = input.nextLine();

    System.out.println(username);

    input.close();

  }

}
```

Following are the methods, used to read different types of inputs from the user:

Method	**Description**
nextInt()	Reads an int value
nextLong()	Reads a long value

Method	**Description**
nextFloat()	Reads a float value
nextBoolean()	Reads a boolean value
next()	Reads a String value only until a space(" ") is encountered
nextLine()	Reads a String value till the end of line
nextDouble()	Reads a double value
nextShort()	Reads a short value
nextByte()	Reads a byte value

Printing the Output In Java, we have different methods available to print the output to the console.

The print() accepts a value as a parameter and prints text on the console. It prints the result in the same line.

System.out.print("Hello ");

System.out.print("Rahul");

// Output is:

Hello Rahul

The println() accepts a value as a parameter and prints text on the console. It prints the result in the new line.

System.out.println("Hello");

System.out.println("Rahul");

// Output is:

Hello

Rahul

Comments Single-line comments start with two forward slashes //

// This is a comment

Multi-line comments start with /* and end with */. In between these, we can write any number of statements.

/ This is a*

*Multi-line Comment */*

String Methods

Method	**Syntax**	**Usage**
trim()	str.trim();	removes all the leading and trailing spaces of the given string and returns a new string.
toLowerCase()	str.toLowerCase();	converts each character of the given string to lowercase and returns a new string.
toUpperCase()	str.toUpperCase();	converts each character of the given string to uppercase and returns a new string.
startsWith()	str.startsWith(value);	returns true if the given string starts with the specified value. Otherwise, false is returned.
endsWith()	str.endsWith(value);	returns true if the given string ends with the specified value. Otherwise, false is returned.
replace()	str.replace(old, latest);	replaces all the occurrences of the old character/substring with the latest character/substring and returns a new string.
replaceFirst()	str.replaceFirst(old, latest);	returns a new string after replacing the first occurrence of the old substring with the latest substring.
split()	str.split(separator);	used to split the string at the specified separator. It returns an array of substrings. https://www.geeksforgeeks.org/split-string-java-examples/
join()	String.join(delimiter, str1, str2, ...);	joins the given elements with the specified delimiter and returns a new string. https://www.w3schools.com/java/ref_string_join.asp
equals()	str1.equals(str2);	It returns true if the given strings are equal. Otherwise false is returned.
equalsIgnoreCase()	str1.equalsIgnoreCase(str2);	It works similar to equals(), but ignores the case difference between the strings.
compareTo()	str1.compareTo(str2);	used to compare two strings based on the Unicode values.

Method	**Syntax**	**Usage**
compareToIgnoreCase()	str1.compareToIgnoreCase(str2);	It works similar to the compareTo() but ignores the case difference between the strings.

String Formatting As Java is a statically typed language, it requires the data type of value to be specified that is going to be formatted. This is done with the help of format specifiers.

The format specifiers define the type of data that is going to be used. Below are the commonly used format specifiers:

Specifier	**Description**
%s, %S	Used for string values
%b, %B	Used for boolean values
%c, %C	Used for characters
%d	Used for integers i.e., int, long etc.
%f	Used for floating-point values like float and double
%e, %E	Used for a scientific notation for floating-point values

String.format() Method: In Java, string formatting can be done using the format() method of the String class. This method returns the formatted string.

```
String name = "James";

int age = 20;

String formattedStr;

formattedStr = String.format("%s is %d years old", name, age);

System.out.println(formattedStr);
```

// Output is:

```
James is 20 years old
```

printf() Method: The System.out.printf() method can be used to directly format and print the string.

```
String name = "James";

int age = 20;

System.out.printf("%s is %d years old", name, age);
```

// Output is:

```
James is 20 years old
```

Formatting Decimal Numbers: For floating-point numbers, we can specify the number of digits after the decimal point needs to be formatted.

```
float runRate = 10.2564f;

System.out.printf("Actual Run rate: %f\n", runRate);

System.out.printf("Run rate rounded to 2 decimals: %.2f", runRate);

// Output is:

Actual Run rate: 10.256400

Run rate rounded to 2 decimals: 10.26
```

DecimalFormat Class: The DecimalFormat class from the package java.text used to format decimal numbers. The format() method takes a double or long value and returns a String value. If float or int are data types are passed, they are implicitly converted to double and long respectively.

```
double doubleNum = 1.666;

DecimalFormat df = new DecimalFormat("#.##");

System.out.println(df.format(doubleNum)); // 1.67
```

Numbering Format Specifiers: We can provide an argument index for the format specifiers. It indicates which argument to be inserted at its position.

```
String name = "James";

int age = 20;

System.out.printf("%2$s is %1$d years old", age, name);

// Output is:

James is 20 years old
```

Character Methods

Method	**Syntax**	**Usage**
isLetter()	Character.isLetter(value);	checks whether the given character is an alphabet and returns a boolean value.
isDigit()	Character.isDigit(value);	checks whether the given character is a number/digit and returns a boolean value.
isWhiteSpace()	Character.isWhitespace(value);	checks whether the given character is a whitespace and returns a boolean value.
isUpperCase()	Character.isUpperCase(value);	checks whether the given character is an uppercase letter and returns a boolean value.

Method	Syntax	Usage
isLowerCase()	Character.isLowerCase(value);	checks whether the given character is a lowercase letter and returns a boolean value.
toUpperCase()	Character.toUpperCase(value);	converts the given lowercase letter to uppercase and returns the new character.
toLowerCase()	Character.toLowerCase(value);	converts the uppercase letter to lowercase and returns the new character.
toString()	Character.toString(value);	converts the character into the respective string.

Relational & Logical Operators

Relational Operators are used to compare values. It returns true or false as the result of a comparison.

Operator	Name	Example	Output
>	Is greater than	2 > 1	true
<	Is less than	5 < 10	true
==	Is equal to	3 == 4	false
<=	Is less than or equal to	2 <= 1	false
>=	Is greater than or equal to	2 >= 1	true
!=	Is not equal to	2 != 1	true

Logical operators are used to perform logical operations on boolean values. These will also produce a true or false as a result.

Name	Example	Output
&&	(5 < 10) && (1 < 2)	true
||	(5 < 10) || (2 < 2)	true
!	!(2 < 3)	false

Logical Operators Truth Table:

A	B	A && B
true	true	true

A	**B**	**A && B**
true	false	false
false	false	false
false	true	false
A	**B**	**A || B**
true	true	true
true	false	true
false	false	false
false	true	true
A	**!A**	
true	false	
false	true	

Ternary Operator

Ternary Operator: Ternary Operator is a conditional operator which works similar to if...else statements.

```
int num1 = 345

int num2 = 689

int largest = num1 > num2 ? num1 : num2 ;

System.out.println(largest); // 689
```

Nested Ternary Operator: A Ternary operator can be used inside another ternary operator. It is called the nested ternary operator.

```
int a = 345

int b = 689

int c = 986

int largest = (a >= b) ? ((a >= c) ? a : c) : ((b >= c) ? b : c);

System.out.println(largest); // 986
```

More Arithmetic Operators

Compound Assignment Operators: Compound assignment operators provide a shorter way to assign an expression to a variable. There are different compound assignment operators available in Java: +=, -=, *=, /=, %=, etc.

```
int a = 10;

int b = 11;

a -= 2;

b %= 3;

System.out.println(a); // 8

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

Unary Operators: The unary operators are those that operate upon a single operand and produce a new value. We have learned some of the unary operators like the logical NOT (!) operator. A few other unary operators are,

- Increment Operator
- Decrement Operator

Increment Operator: The Increment Operator (++) is an operator which is used to increment the value of a variable by 1, on which it is applied. The increment operator can be used in two ways:

Prefix (++x): If an Increment operator is used in front of an operand, then it is called a Prefix.

```
int a = 10;

++a;

int number = ++a;

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

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

// Output is:

a = 12

number = 12
```

Postfix (x++): If an Increment operator is used after an operand, then is called a Postfix.

```
int a = 10;

a++;

int number = a++;

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

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

// Output is:

a = 12
```

```
number = 11
```

Decrement Operator: The Decrement operator is an operator which is used to decrease the value of the variable by 1, on which it is applied. The decrement operator can also be used in two ways:

Prefix (--x): Prefix decrement is similar to prefix increment, except that the variable value is decremented by one instead of being incremented.

```
int a = 10;

--a;

int number = --a;

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

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

// Output is:

a = 8

number = 8
```

Postfix (x--): Postfix decrement is similar to postfix increment, except that the variable value is decremented by one instead of being incremented.

```
int a = 10;

a--;

int number = a--;

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

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

// Output is:

a = 8

number = 9
```

Escape Sequences: A character with a backslash \ just before a character is called an escape character or escape sequence. Most commonly used escape sequences include:

- \n → New Line
- \t → Tab Space
- \\ → Backslash
- \' → Single Quote
- \" → Double Quote