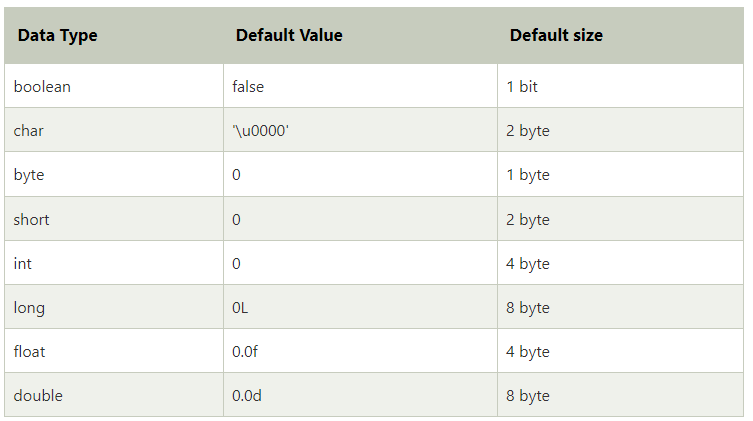
Day-2

Features Of java

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [Simpile](https://www.javatpoint.com/features-of-java#Simple) | [Object-Oriented](https://www.javatpoint.com/features-of-java#Object-Oriented) | [Portable](https://www.javatpoint.com/features-of-java#Portable) | [Platform independent](https://www.javatpoint.com/features-of-java#Platform-independent) | [Secured](https://www.javatpoint.com/features-of-java#Secured) | [Robust](https://www.javatpoint.com/features-of-java#Robust) |
| [Architecture neutral](https://www.javatpoint.com/features-of-java#Architecture-neutral) | [Interpreted](https://www.javatpoint.com/features-of-java#Interpreted) | [High Performance](https://www.javatpoint.com/features-of-java#High-Performance) | [Multithreaded](https://www.javatpoint.com/features-of-java#Multithreaded) | [Distributed](https://www.javatpoint.com/features-of-java#Distributed) | [Dynamic](https://www.javatpoint.com/features-of-java#Dynamic) |

**Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.





* Boolean Data Type

The Boolean data type is used to store only two possible values: true and false. This data type is used for simple flags that track true/false conditions.

The Boolean data type specifies one bit of information, but its "size" can't be defined precisely.

**Example:**

Boolean one = **false**

* Byte Data Type

The byte data type is an example of primitive data type. It isan 8-bit signed two's complement integer. Its value-range lies between -128 to 127 (inclusive). Its minimum value is -128 and maximum value is 127. Its default value is 0.

The byte data type is used to save memory in large arrays where the memory savings is most required. It saves space because a byte is 4 times smaller than an integer. It can also be used in place of "int" data type.

**Example:**

**byte** a = 10, **byte** b = -20

## Short Data Type

The short data type is a 16-bit signed two's complement integer. Its value-range lies between -32,768 to 32,767 (inclusive). Its minimum value is -32,768 and maximum value is 32,767. Its default value is 0.

The short data type can also be used to save memory just like byte data type. A short data type is 2 times smaller than an integer.

**Example:**

**short** s = 10000, **short** r = -5000

* Int Data Type

The int data type is a 32-bit signed two's complement integer. Its value-range lies between - 2,147,483,648 (-2^31) to 2,147,483,647 (2^31 -1) (inclusive). Its minimum value is - 2,147,483,648and maximum value is 2,147,483,647. Its default value is 0.

The int data type is generally used as a default data type for integral values unless if there is no problem about memory.

**Example:**

**int** a = 100000, **int** b = -200000

* Long Data Type

The long data type is a 64-bit two's complement integer. Its value-range lies between -9,223,372,036,854,775,808(-2^63) to 9,223,372,036,854,775,807(2^63 -1)(inclusive). Its minimum value is - 9,223,372,036,854,775,808and maximum value is 9,223,372,036,854,775,807. Its default value is 0. The long data type is used when you need a range of values more than those provided by int.

**Example:**

**long** a = 100000L, **long** b = -200000L

* Float Data Type

The float data type is a single-precision 32-bit IEEE 754 floating point.Its value range is unlimited. It is recommended to use a float (instead of double) if you need to save memory in large arrays of floating point numbers. The float data type should never be used for precise values, such as currency. Its default value is 0.0F.

**Example:**

**float** f1 = 234.5f

* Double Data Type

The double data type is a double-precision 64-bit IEEE 754 floating point. Its value range is unlimited. The double data type is generally used for decimal values just like float. The double data type also should never be used for precise values, such as currency. Its default value is 0.0d.

**Example:**

**double** d1 = 12.3

* Char Data Type

The char data type is a single 16-bit Unicode character. Its value-range lies between '\u0000' (or 0) to '\uffff' (or 65,535 inclusive).The char data type is used to store characters.

**Example:**

**char** letterA = 'A'

**Non-primitive data types:**

The non-primitive data types include [Classes](https://www.javatpoint.com/object-and-class-in-java), [Interfaces](https://www.javatpoint.com/interface-in-java), and [Arrays](https://www.javatpoint.com/array-in-java).

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.

For example, consider an [array](https://www.javatpoint.com/array-in-java) that stores a group of values. Class is also a primitive type that stores different methods and variables. Therefore, these are also called as **advanced data types** in Java.

Whenever a non-primitive data type is defined, it refers a memory location where the data is stored in heap memory i.e., it refers to the memory location where an object is placed. Therefore, a non-primitive data type variable is also called **referenced data type** or **simply object reference variable**.

An object reference variable lives on the stack memory and the object to which it points always lives on the heap memory. The stack holds a pointer to the object on the heap.

In Java programming, all non-primitive data types are simply called objects that are created by instantiating a class.

### Key points:

1. The default value of any reference variable is null.
2. Whenever we are passing a non-primitive data type to a method, we are passing the address of that object where the data is stored.

## Types of Non-primitive data types

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

1. Class
2. Object
3. String
4. Array
5. Interface

### 1. Class and objects:

A [class](https://www.javatpoint.com/object-and-class-in-java) in Java is a user defined data type i.e. it is created by the user. It acts a template to the data which consists of member variables and methods.

An [object](https://www.javatpoint.com/object-and-class-in-java) is the variable of the class, which can access the elements of class i.e. methods and variables.

**Example:**

In the following example, we are creating a class containing the variables and methods ( **add() and sub()** ). Here, we are accessing the methods using the object of the Class **obj**.

**ClassExample.java**

1. **public** **class** ClassExample {
3. // defining the variables of class
4. **int** a = 20;
5. **int** b = 10;
6. **int** c;
8. // defining the methods of class
9. **public** **void** add () {
10. **int** c = a + b;
11. System.out.println("Addition of numbers is: " + c);
12. }
14. **public** **void** sub () {
15. **int** c = a - b;
16. System.out.println("Subtraction of numbers is: " + c);
17. }
19. // main method
20. **public** **static** **void** main (String[] args) {
21. // creating the object of class
22. ClassExample obj = **new** ClassExample();
24. // calling the methods
25. obj.add();
26. obj.sub();
27. }
28. }

**Output:**

Addition of numbers is: 30

Subtraction of numbers is: 10

### 2. Interface:

An [interface](https://www.javatpoint.com/interface-in-java) is similar to a class however the only difference is that its methods are abstract by default i.e. they do not have body. An interface has only the final variables and method declarations. It is also called a fully abstract class.

**Example:**

In the following example, we are creating the interface CalcInterface with two abstract methods ( **multiply() and divide()** ). Here, the class InterfaceExample implements the interface and further defines the methods of that interface. Then, the object of class is used to access those methods.

**InterfaceExample.java**

1. **interface** CalcInterface {
2. **void** multiply();
3. **void** divide();
4. }
5. **public** **class** InterfaceExample **implements** CalcInterface {
7. // defining the variables of class
8. **int** a = 10;
9. **int** b = 20;
10. **int** c;
12. // implementing the interface methods
13. **public** **void** multiply() {
14. **int** c = a \* b;
15. System.out.println("Multiplication of numbers is: " + c);
16. }
17. **public** **void** divide() {
18. **int** c = a / b;
19. System.out.println("Division of numbers is: " + c);
20. }
21. // main method
22. **public** **static** **void** main (String[] args) **throws** IOException {
23. InterfaceExample obj = **new** InterfaceExample();
24. // calling the methods
25. obj.multiply();
26. obj.divide();
27. }
28. }

### 3. String:

A string represents a sequence of characters for example "Javatpoint", "Hello world", etc. String is the class of Java.

One of the ways to create a string and store a value in it is shown below:

1. String str = "You're the best";

Here, String type variable **str** has the value "You're the best". Click here to understand more about [String in Java](https://www.javatpoint.com/java-string-split).

**Example:**

In the following example, we are creating a string with a value. Here, we are using one of the String class methods, [**substring()**](https://www.javatpoint.com/substring) which prints the specified indexed part of the string.

**StringExample.java**

1. **public** **class** StringExample {
2. **public** **static** **void** main(String[] args) {
4. // creating a string and initializing it
5. String str = "Hello! This is example of String type";
7. // applying substring() on above string
8. String subStr = str.substring(0,14);
10. // printing the string
11. System.out.println(subStr);
12. }
13. }

**Output:**

Hello! This is

### 4. Array:

An [array](https://www.javatpoint.com/array-in-java) 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. The variables can be either primitive or non-primitive data types.

Following example shows how to declare array of primitive data type **int**:

1. **int** [ ] marks;

Following example shows how to declare array of non-primitive data type:

1. Student [ ] students;

where, **Student** is the class name and [ ] creates an array of object **students**.

**Example:**

In the following example, we are creating two basic array, in which one is initialized and the other is declared (input is read from the user). Further, we are printing those array using the for loop.

**ArrayExample.java**

1. // importing required packages
2. **import** java.io. \* ;
3. **import** java.util. \* ;
5. **public** **class** ArrayExample {
6. **public** **static** **void** main(String[] args) **throws** IOException {
7. **int** i;
8. Scanner sc = **new** Scanner(System. in );
9. // declaring and initializing an array
10. **int** arr[] = {1, 2, 3, 6, 9};
11. // defining another array arr1
12. **int** arr1[] = **new** **int**[5];
13. // reading values from the user
14. System.out.println("Enter the numbers (size = 5) :");
15. **for** (i = 0; i < 5; i++) {
16. arr1[i] = sc.nextInt();
17. }
18. System.out.println("Previous array with initialized size is: ");
19. **for** (i = 0; i < 5; i++) {
20. System.out.print(arr[i] + " ");
21. }
22. System.out.println("\nThe new array we have entered is:");
23. **for** (i = 0; i < 5; i++) {
24. System.out.print(arr1[i] + " ");
25. }
26. }
27. }

**Output:**

Enter the numbers (size = 5) :

56

43

22

1

7

Previous array with initialized size is:

1 2 3 6 9

The new array we have entered is:

56 43 22 1 7

## Difference between Primitive and Non-primitive Data types in Java

1. In Java, the primitive data types are system defined however we have to create and define the non-primitive data types.
2. In primitive data type, variables can store only one value at a time. However in non-primitive data types, either multiple values of the same type or different type or both can be stored.
3. All the data for primitive type variables are stored on the stack whereas, for reference types, the stack holds a pointer to the object on the heap.
4. A primitive type starts with a lowercase letter, while non-primitive types start with an uppercase letter.
5. The size of a primitive type depends on the data type, while non-primitive types have all the same size.

//Adding Two Numbers

import java.util.Scanner;

class ArithmaticOp{

public static void main(String[] args){ //program to demonstrate use of arithmatic operations

int a,b,su; //declaretion of veriables

Scanner sc.=new Scanner(Sysyem.in); //initialize a Scanner

System.out.println("Pleare Two Number");

a=sc.nextInt(); //store the value throw user is put

b=sc.nextInt(); //store the value throw user is put

su=a+b;

System.out.println("Your Result Is:-"+su);

}

}