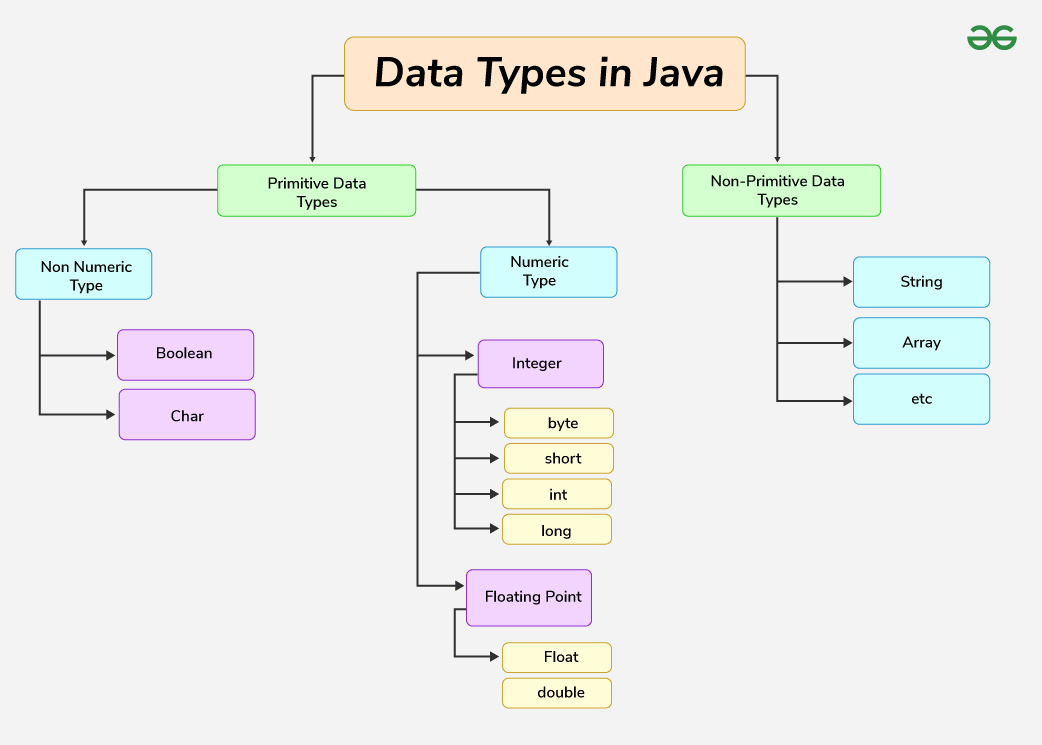
**Data Types in Java**

**Data types in Java** are of different sizes and values that can be stored in the variable that is made as per convenience and circumstances to cover up all test cases. Java has two categories in which data types are segregated

1. **Primitive Data Type:** such as boolean, char, int, short, byte, long, float, and double. The Boolean with uppercase B is a wrapper class for the primitive data type boolean in Java.
2. **Non-Primitive Data Type or Object Data type:** such as String, Array, etc.



Data Types in JAVA

**Primitive Data Types in Java**

Primitive data are only single values and have no special capabilities.  There are 8 primitive data types. They are depicted below in tabular format below as follows:

| **Type** | **Description** | **Default** | **Size** | **Example Literals** | **Range of values** |
| --- | --- | --- | --- | --- | --- |
| **boolean** | true or false | false | 8 bits | true, false | true, false |
| **byte** | twos-complement integer | 0 | 8 bits | (none) | -128 to 127 |
| **char** | Unicode character | \u0000 | 16 bits | ‘a’, ‘\u0041’, ‘\101’, ‘\\’, ‘\’, ‘\n’, ‘β’ | characters representation of ASCII values  0 to 255 |
| **short** | twos-complement integer | 0 | 16 bits | (none) | -32,768 to 32,767 |
| **int** | twos-complement intger | 0 | 32 bits | -2,-1,0,1,2 | -2,147,483,648  to  2,147,483,647 |
| **long** | twos-complement integer | 0 | 64 bits | -2L,-1L,0L,1L,2L | -9,223,372,036,854,775,808  to  9,223,372,036,854,775,807 |
| **float** | IEEE 754 floating point | 0.0 | 32 bits | 1.23e100f , -1.23e-100f , .3f ,3.14F | upto 7 decimal digits |
| **double** | IEEE 754 floating point | 0.0 | 64 bits | 1.23456e300d , -123456e-300d , 1e1d | upto 16 decimal digits |

Let us discuss and implement each one of the following data types that are as follows:

**1. boolean Data Type**

The boolean data type represents a logical value that can be either true or false. Conceptually, it represents a single bit of information, but the actual size used by the virtual machine is implementation-dependent and typically at least one byte (eight bits) in practice. Values of the boolean type are not implicitly or explicitly converted to any other type using casts. However, programmers can write conversion code if needed.

**Syntax:**

boolean booleanVar;

**Size:** Virtual machine dependent

**2. byte Data Type**

The byte data type is an 8-bit signed two’s complement integer. The byte data type is useful for saving memory in large arrays.

**Syntax:**

byte byteVar;

**Size:** 1 byte (8 bits)

**3. short Data Type**

The short data type is a 16-bit signed two’s complement integer. Similar to byte, use a short to save memory in large arrays, in situations where the memory savings actually matters.

**Syntax:**

short shortVar;

**Size:** 2 bytes (16 bits)

**4. int Data Type**

It is a 32-bit signed two’s complement integer.

**Syntax:**

int intVar;

**Size:** 4 bytes ( 32 bits )

**Remember:** In Java SE 8 and later, we can use the int data type to represent an unsigned 32-bit integer, which has a value in the range [0, 2 32 -1]. Use the Integer class to use the int data type as an unsigned integer.

**5. long Data Type**

The range of a long is quite large. The long data type is a 64-bit two’s complement integer and is useful for those occasions where an int type is not large enough to hold the desired value. The size of the Long Datatype is 8 bytes (64 bits).

**Syntax:**

long longVar;

**Remember:** In Java SE 8 and later, you can use the long data type to represent an unsigned 64-bit long, which has a minimum value of 0 and a maximum value of 2 64 -1. The Long class also contains methods like comparing Unsigned, divide Unsigned, etc to support arithmetic operations for unsigned long.

**6. float Data Type**

The float data type is a single-precision 32-bit IEEE 754 floating-point. Use a float (instead of double) if you need to save memory in large arrays of floating-point numbers. The size of the float data type is 4 bytes (32 bits).

**Syntax:**

float floatVar;

**7. double Data Type**

The double data type is a double-precision 64-bit IEEE 754 floating-point. For decimal values, this data type is generally the default choice. The size of the double data type is 8 bytes or 64 bits.

**Syntax:**

double doubleVar;

**Note:** Both float and double data types were designed especially for scientific calculations, where approximation errors are acceptable. If accuracy is the most prior concern then, it is recommended not to use these data types and use BigDecimal class instead.

It is recommended to go through [rounding off errors in java.](https://www.geeksforgeeks.org/rounding-off-errors-java/)

**8. char Data Type**

The char data type is a single 16-bit Unicode character with the size of 2 bytes (16 bits).

**Syntax:**

char charVar;

**Why is the Size of char 2 bytes in Java?**

So, other languages like C/C++ use only ASCII characters, and to represent all ASCII characters 8 bits is enough. But Java uses the **Unicode system not the ASCII code System** and to represent the Unicode system 8 bits is not enough to represent all characters so Java uses 2 bytes for characters. **Unicode** defines a fully international character set that can represent most of the world’s written languages. It is a unification of dozens of character sets, such as Latin, Greek, Cyrillic, Katakana, Arabic, and many more.

**Example:**

// Java Program to Demonstrate Char Primitive Data Type

// Class

class GFG {

// Main driver method

public static void main(String args[])

{

// Creating and initializing custom character

char a = 'G';

// Integer data type is generally

// used for numeric values

int i = 89;

// use byte and short

// if memory is a constraint

byte b = 4;

// this will give error as number is

// larger than byte range

// byte b1 = 7888888955;

short s = 56;

// this will give error as number is

// larger than short range

// short s1 = 87878787878;

// by default fraction value

// is double in java

double d = 4.355453532;

// for float use 'f' as suffix as standard

float f = 4.7333434f;

// need to hold big range of numbers then we need

// this data type

long l = 12121;

System.out.println("char: " + a);

System.out.println("integer: " + i);

System.out.println("byte: " + b);

System.out.println("short: " + s);

System.out.println("float: " + f);

System.out.println("double: " + d);

System.out.println("long: " + l);

}

}

**Output**

char: G

integer: 89

byte: 4

short: 56

float: 4.7333436

double: 4.355453532

long: 12121

**Non-Primitive (Reference) Data Types**

The **Non-Primitive (Reference) Data Types** will contain a memory address of variable values because the reference types won’t store the variable value directly in memory. They are strings, objects, arrays, etc.

**1. Strings**

[Strings](https://www.geeksforgeeks.org/strings-in-java/) are defined as an array of characters. The difference between a character array and a string in Java is, that the string is designed to hold a sequence of characters in a single variable whereas, a character array is a collection of separate char-type entities. Unlike C/C++, Java strings are not terminated with a null character.

**Syntax:** Declaring a string

<String\_Type> <string\_variable> = “<sequence\_of\_string>”;

**Example:**

// Declare String without using new operator   
String s = "GeeksforGeeks";   
// Declare String using new operator   
String s1 = new String("GeeksforGeeks");

**2. Class**

A [class](https://www.geeksforgeeks.org/classes-objects-java/) is a user-defined blueprint or prototype from which objects are created.  It represents the set of properties or methods that are common to all objects of one type. In general, class declarations can include these components, in order:

1. **Modifiers** : A class can be public or has default access. Refer to [access specifiers for classes or interfaces in Java](https://www.geeksforgeeks.org/access-specifiers-for-classes-or-interfaces-in-java/)
2. **Class name:** The name should begin with an initial letter (capitalized by convention).
3. **Superclass(if any):** The name of the class’s parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.
4. **Interfaces(if any):** A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.
5. **Body:** The class body is surrounded by braces, { }.

**3. Object**

An [Object](https://www.geeksforgeeks.org/classes-objects-java/) is a basic unit of Object-Oriented Programming and represents real-life entities.  A typical Java program creates many objects, which as you know, interact by invoking methods. An object consists of :

1. **State** : It is represented by the attributes of an object. It also reflects the properties of an object.
2. **Behavior** : It is represented by the methods of an object. It also reflects the response of an object to other objects.
3. **Identity** : It gives a unique name to an object and enables one object to interact with other objects.

**4. Interface**

Like a class, an [interface](https://www.geeksforgeeks.org/interfaces-in-java/) can have methods and variables, but the methods declared in an interface are by default abstract (only method signature, no body).

* Interfaces specify what a class must do and not how. It is the blueprint of the class.
* An Interface is about capabilities like a Player may be an interface and any class implementing Player must be able to (or must implement) move(). So it specifies a set of methods that the class has to implement.
* If a class implements an interface and does not provide method bodies for all functions specified in the interface, then the class must be declared abstract.
* A Java library example is [Comparator Interface](https://www.geeksforgeeks.org/comparator-interface-java/) . If a class implements this interface, then it can be used to sort a collection.

**5. Array**

An [Array](https://www.geeksforgeeks.org/arrays-in-java/) is a group of like-typed variables that are referred to by a common name. Arrays in Java work differently than they do in C/C++. The following are some important points about Java arrays.

* In Java, all arrays are dynamically allocated. (discussed below)
* Since arrays are objects in Java, we can find their length using member length. This is different from C/C++ where we find length using size.
* A Java array variable can also be declared like other variables with [] after the data type.
* The variables in the array are ordered and each has an index beginning with 0.
* Java array can also be used as a static field, a local variable, or a method parameter.
* The **size** of an array must be specified by an int value and not long or short.
* The direct superclass of an array type is Object.
* Every array type implements the interfaces [Cloneable](https://www.geeksforgeeks.org/marker-interface-java/) and [java.io.Serializable](https://www.geeksforgeeks.org/serialization-in-java/).

**Key Points to Remember**

* **Strong Typing**: Java enforces strict type checking at compile-time, reducing runtime errors.
* **Memory Efficiency**: Choosing the right data type based on the range and precision needed helps in efficient memory management.
* **Immutability of Strings**: Strings in Java cannot be changed once created, ensuring safety in multithreaded environments.
* **Array Length**: The length of arrays in Java is fixed once declared, and it can be accessed using the length attribute

**Conclusion**

Understanding Java’s data types is fundamental to efficient programming. Each data type has specific use cases and constraints, making it essential to choose the right type for the task at hand. This ensures optimal memory usage and program performance while leveraging Java’s strong typing system to catch errors early in the development process.

**Check Out:** [Quiz on Data Type in Java](https://www.geeksforgeeks.org/java-gq/data-types-2-gq/)

**Data Types in Java – FAQs**

**What are Data Types in Java?**

Data types are of different sizes and values that can be stored in the variable that is made as per convenience and circumstances to cover up all test cases.

**What are the 8 Data Types that use in Java?**

There are 8 main primitive data types in java as mentioned below:

* boolean
* byte
* char
* short
* int
* long
* float
* double

**Which is a Primitive Type in Java?**

Primitive data types are the types in java that can store a single value and do not provide any special capability.

**Why char uses 2 bytes in Java and what is \u0000?**

Char uses 2 bytes in java because it uses the Unicode system rather than the ASCII system. “\u000” is the lowest range of the Unicode system.

Last Updated : 20 Nov, 2024

Variables are the containers for storing the data values or you can also call it a memory location name for the data. Every variable has a:

* **Data Type** – The kind of data that it can hold. For example, int, string, float, char, etc.
* **Variable Name** – To identify the variable uniquely within the scope.
* **Value** – The data assigned to the variable.

**There are three types of variables in Java – Local, Instance, and Static.**

**Example:**

int age = 27; // integer variable having value 27  
  
String name = “ADYAR0" // string variable

**How to Declare Java Variables?**

We can declare variables in Java as pictorially depicted below:

From the image, it can be easily perceived that while declaring a variable, we need to take care of two things that are:

1. **datatype**: Type of data that can be stored in this variable.
2. **data\_name:** Name was given to the variable.

In this way, a name can only be given to a memory location. It can be assigned values in two ways:

* Variable Initialization
* Assigning value by taking input

**How to Initialize Java Variables?**

It can be perceived with the help of 3 components explained above:

**Example:**

// Declaring float variable  
float simpleInterest;   
  
// Declaring and initializing integer variable  
int time = 10, speed = 20;   
  
// Declaring and initializing character variable  
char var = 'h';

Variables are the basic units of storage in Java. For a deeper understanding of variable types, scope, and memory management, the [**Java Programming Course**](https://gfgcdn.com/tu/S5r/) provides a thorough exploration with hands-on coding tasks.

**Types of Java Variables**

Now let us discuss different types of variables  which are listed asfollows:

1. Local Variables
2. Instance Variables
3. Static Variables

Let us discuss the traits of every type of variable listed here in detail.

**1. Local Variables**

A variable defined within a block or method or constructor is called a local variable.

* The Local variable is created at the time of declaration and destroyed after exiting from the block or when the call returns from the function.
* The scope of these variables exists only within the block in which the variables are declared, i.e., we can access these variables only within that block.
* Initialization of the local variable is mandatory before using it in the defined scope.

**Example 1:**



1

// Java Program to show the use of local variables

2

import java.io.\*;

3

​

4

class GFG {

5

public static void main(String[] args)

6

{

7

// Declared a Local Variable

8

int var = 10;

9

​

10

// This variable is local to this main method only

11

System.out.println("Local Variable: " + var);

12

}

13

}

**Output**

Local Variable: 10

**Example 2:**



1

// Java Program to show the use of

2

// Local Variables

3

import java.io.\*;

4

public class GFG {

5

public static void main(String[] args)

6

{

7

// x is a local variable

8

int x = 10;

9

​

10

// message is also a local

11

// variable

12

String message = "Hello, world!";

13

​

14

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

15

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

16

​

17

if (x > 5) {

18

// result is a

19

// local variable

20

String result = "x is greater than 5";

21

System.out.println(result);

22

}

23

​

24

// Uncommenting the line below will result in a

25

// compile-time error System.out.println(result);

26

​

27

for (int i = 0; i < 3; i++) {

28

String loopMessage

29

= "Iteration "

30

+ i; // loopMessage is a local variable

31

System.out.println(loopMessage);

32

}

33

​

34

// Uncommenting the line below will result in a

35

// compile-time error

36

// System.out.println(loopMessage);

37

}

38

}

**Output**

x = 10

message = Hello, world!

x is greater than 5

Iteration 0

Iteration 1

Iteration 2

**2. Instance Variables**

Instance variables are non-static variables and are declared in a class outside of any method, constructor, or block.

* As instance variables are declared in a class, these variables are created when an object of the class is created and destroyed when the object is destroyed.
* Unlike local variables, we may use access specifiers for instance variables. If we do not specify any access specifier, then the default access specifier will be used.
* Initialization of an instance variable is not mandatory. Its default value is dependent on the data type of variable. For *String* it is *null,* for *float* itis *0.0f,* for *int* it is *0,* for Wrapper classes like *Integer* it is *null, etc.*
* Instance variables can be accessed only by creating objects.
* We initialize instance variables using [constructors](https://www.geeksforgeeks.org/constructors-in-java/) while creating an object. We can also use [instance blocks](https://www.geeksforgeeks.org/using-instance-blocks-in-java/) to initialize the instance variables.

**Example:**



1

// Java Program to show the use of

2

// Instance Variables

3

import java.io.\*;

4

​

5

class GFG {

6

​

7

// Declared Instance Variable

8

public String geek;

9

public int i;

10

public Integer I;

11

public GFG()

12

{

13

// Default Constructor

14

// initializing Instance Variable

15

this.geek = "Shubham Jain";

16

}

17

​

18

// Main Method

19

public static void main(String[] args)

20

{

21

// Object Creation

22

GFG name = new GFG();

23

​

24

// Displaying O/P

25

System.out.println("Geek name is: " + name.geek);

26

System.out.println("Default value for int is "+ name.i);

27

28

// toString() called internally

29

System.out.println("Default value for Integer is "+ name.I);

30

}

31

}

**Output**

Geek name is: Shubham Jain

Default value for int is 0

Default value for Integer is null

**3. Static Variables**

Static variables are also known as class variables.

* These variables are declared similarly to instance variables. The difference is that static variables are declared using the static keyword within a class outside of any method, constructor, or block.
* Unlike instance variables, we can only have one copy of a static variable per class, irrespective of how many objects we create.
* Static variables are created at the start of program execution and destroyed automatically when execution ends.
* Initialization of a static variable is not mandatory. Its default value is dependent on the data type of variable. For *String* it is *null*, for *float* it is *0.0f*, for *int* it is *0*, for *Wrapper classes* like *Integer* it is *null,* etc.
* If we access a static variable like an instance variable (through an object), the compiler will show a warning message, which won’t halt the program. The compiler will replace the object name with the class name automatically.
* If we access a static variable without the class name, the compiler will automatically append the class name. But for accessing the static variable of a different class, we must mention the class name as 2 different classes might have a static variable with the same name.
* Static variables cannot be declared locally inside an instance method.
* [Static blocks](https://www.geeksforgeeks.org/static-blocks-in-java/) can be used to initialize static variables.

// Java Program to show the use of

// Static variables

import java.io.\*;

class ADYARO {

// Declared static variable

public static String yaro = "Shubham Jain";

public static void main(String[] args)

{

// geek variable can be accessed without object

// creation Displaying O/P ADYARO.yaro --> using the

// static variable

System.out.println(" Name is : " + ADYARO.yaro);

// static int c = 0;

// above line, when uncommented,

// will throw an error as static variables cannot be

// declared locally.

}

}