JAVA Data Types

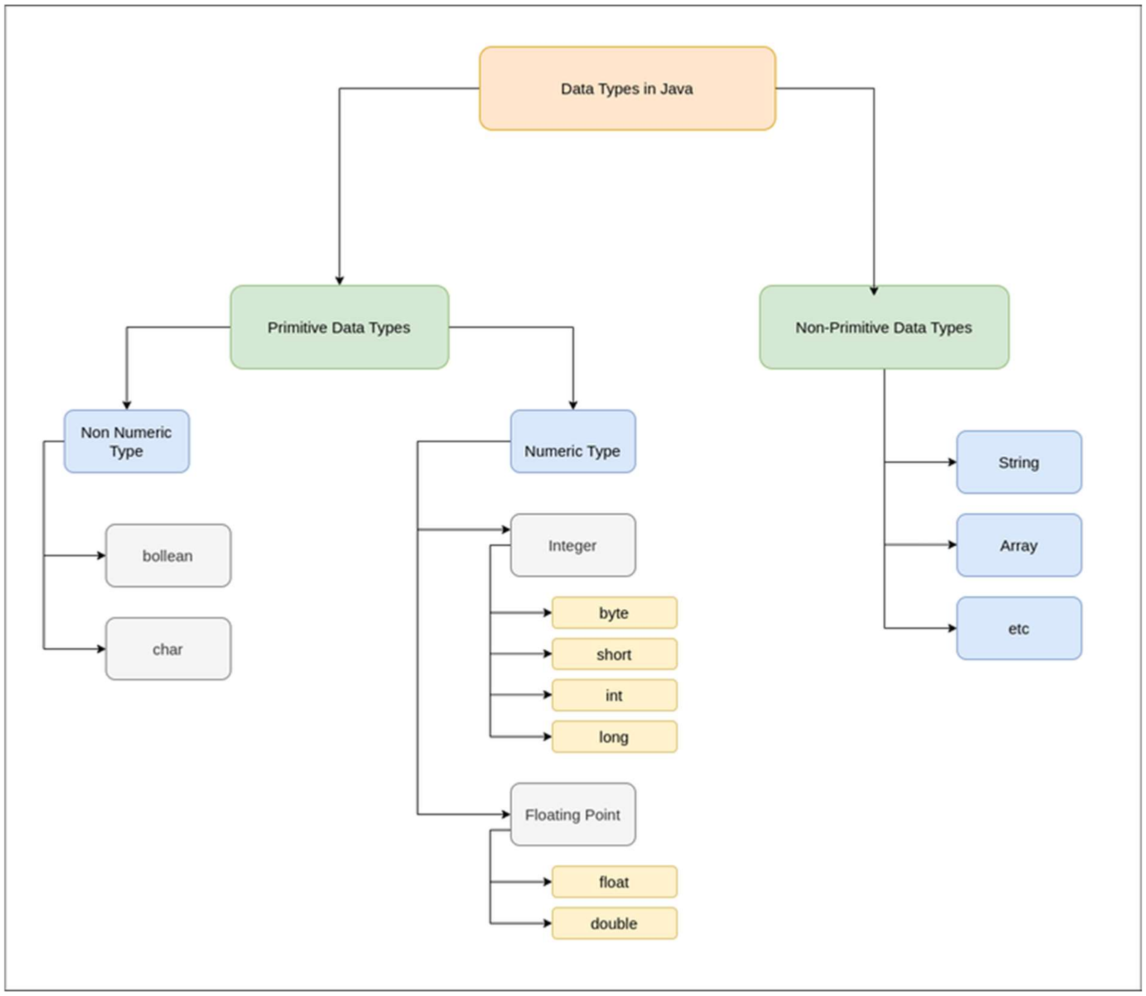
Java is statically typed and also a strongly typed language because, in Java, each type of data (such as integer, character, hexadecimal, packed decimal, and so forth) is predefined as part of the programming language and all constants or variables defined for a given program must be described with one of the Java data types.

What is DATA type?

Datatype specify the different sizes and values that can be stored in the variable. Java has two categories in which data types are segregated.

1. **Primitive Data Types:**
   - **These are the fundamental building blocks in Java. They include:**
     - **byte: An 8-bit integer that stores whole numbers from -128 to 127.**
     - **short: A 16-bit integer for whole numbers ranging from -32,768 to 32,767.**
     - **int: A 32-bit integer capable of storing whole numbers from -2,147,483,648 to 2,147,483,647.**
     - **long: An 64-bit integer that accommodates whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.**
     - **float: A 32-bit floating point type for fractional numbers (approximately 6 to 7 decimal digits).**
     - **double: A 64-bit floating point type for more precise fractional numbers (approximately 15 decimal digits).**
     - **char: A 16-bit Unicode character representing a single letter or symbol.**
     - **boolean: A 1-bit type that stores true or false values.**
2. **Non-Primitive Data Types:**
   - **These are more complex and include:**
     - **String: Represents sequences of characters (text).**
     - **Arrays: Collections of elements of the same type.**
     - **Classes: User-defined data types that encapsulate data and methods.**

- Interfaces: Blueprint for classes to implement.



What is type casting in java?

➔ Converting one datatype to another datatype is called type casting.

Two type of type casting in java

1. Implicit type casting
2. Explicit type casting

Implicit type casting:

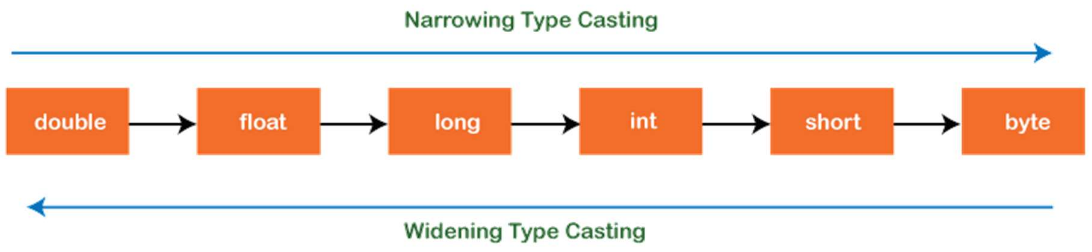
it is automatically performed by the compiler

Explicit type casting:

By default, the compiler doesn't allow the explicit type casting.

For example: double x=10.5;

int y=(int)x;



Type Casting in Java

variable

variable is the name of the memory location.

In other word we can say it is used define name which is given by user. Variable can store on type of value.

Ex:- int a=10

Here a is the variable.

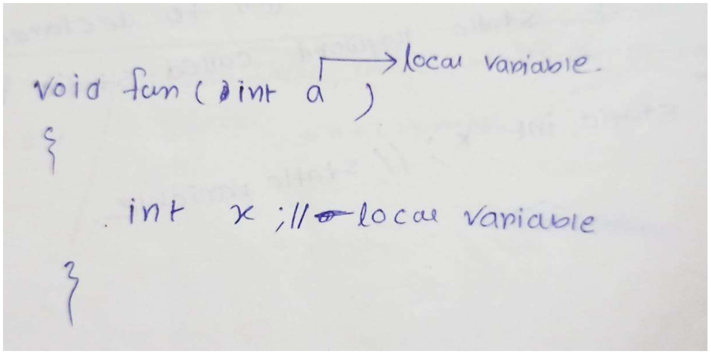
Type:

Three types of variables are in java there are

Local variable:

A variable which is declared inside the body of the method or method parameter call local variable.

Syntax:



```
                 ┌──→local variable.
void fun ( int a )
{
     int  x ;// local variable
}
```

Instance variable:

A variable which is declared inside the class but outside of all the methods call instance variable.

- 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 0.
- Instance variables can be accessed only by creating objects.

Static variable:

A variable written to declared with the help of static keyword call static variable.

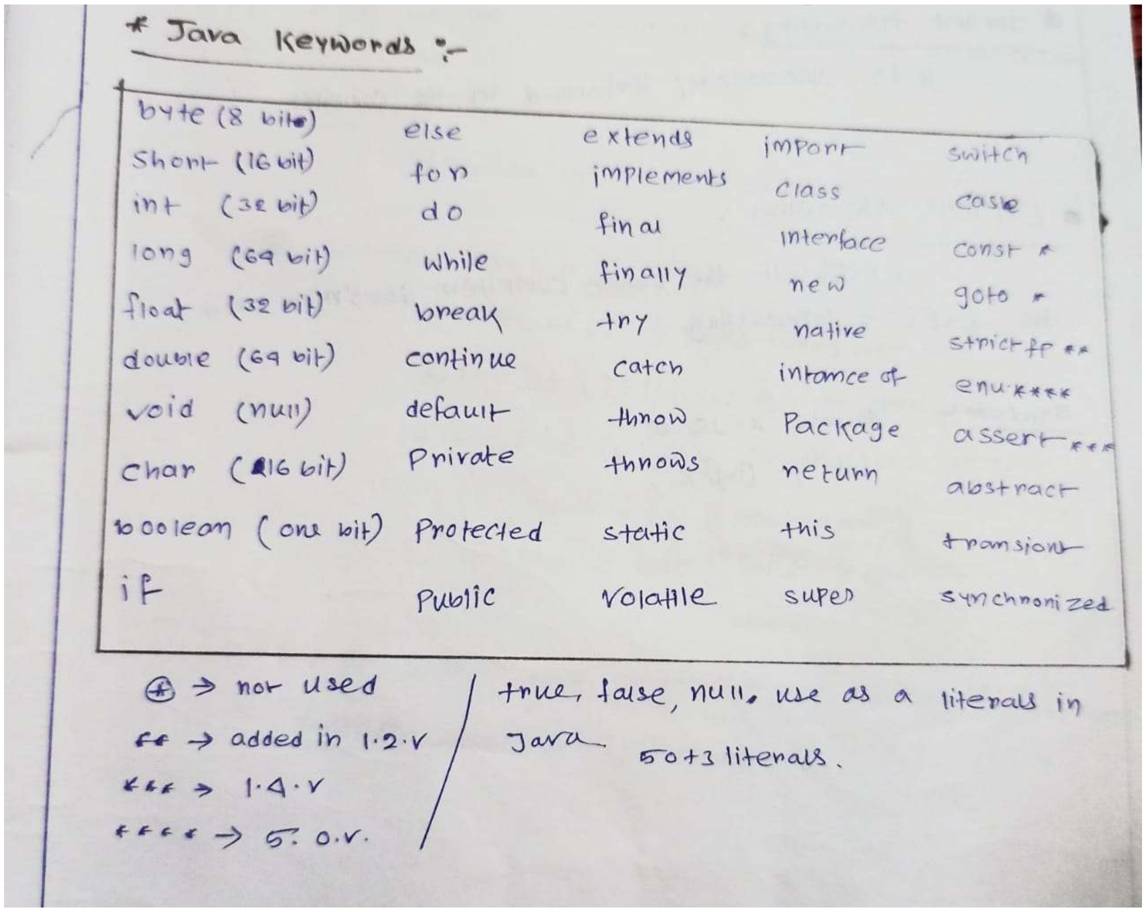
Syntax: static int x;

Keyword

Keyword are the reserved word whose meaning is already define in the java compiler.

Note:

- We can’t use keyword for our personal use.
- Keyword are the case-sensitive.



* Java Keywords :-

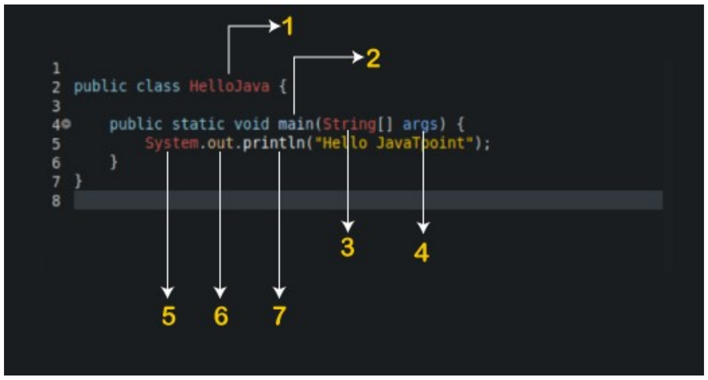
byte (8 bits)	else	extends	import	switch
Short (16 bit)	for	implements	class	case
int (32 bit)	do	final	interface	const *
long (64 bit)	while	finally	new	goto *
float (32 bit)	break	try	native	strictfp **
double (64 bit)	continue	catch	instance of	enu ****
void (null)	default	throw	Package	assert ***
char (16 bit)	Private	throws	return	abstract
booleam (one bit)	Protected	static	this	transient
if	Public	volatile	super	synchronized

* → not used
** → added in 1.2.v
*** → 1.4.v
**** → 5.0.v.

true, false, null, use as a literals in Java
50+3 literals.

Identifier

In java, an identifier is the name of the variable, method, class, package or interface that is used for the purpose of identification.
--- In java an identifier can be a class name, method name, variable name or label.



```
public class HelloJava {

    public static void main(String[] args) {
        System.out.println("Hello JavaTpoint");
    }
}
```

Here

HelloJava → class name

main →method name

String [] args →array

System.out.println()→ object/variable/method

Rules For Defining Java Identifiers

- The only allowed characters for identifiers are all alphanumeric characters**([A-Z],[a-z],[0-9]), '$'(dollar sign) and '_' (underscore).**For example "geek@" is not a valid Java identifier as it contains a '@' a special character.
- Identifiers should **not** start with digits(**[0-9]**). For example "123geeks" is not a valid Java identifier.
- Java identifiers are **case-sensitive**.
- There is no limit on the length of the identifier but it is advisable to use an optimum length of 4 – 15 letters only.

- **Reserved Words** can’t be used as an identifier. For example, “int while = 20;” is an invalid statement as a while is a reserved word. There are **53** reserved words in Java.

Token

Token is the smallest element of a program that is identify by a compiler.

- Every java statement and expression are created using token.

List of Token:

1. Keyword
2. Identifier
3. Operator
4. Separator
5. Literals

JAVA operators

Operator Type	Category	Precedence	Associativity
Unary	**postfix**	**a++, a--**	**Right to left**
	prefix	**++a, --a, +a, -a, ~, !**	**Right to left**
Arithmetic	**Multiplication**	***, /, %**	**Left to Right**
	Addition	**+, -**	**Left to Right**
Shift	**Shift**	**<<, >>, >>>**	**Left to Right**
Relational	**Comparison**	**<, >, <=, >=, instanceOf**	**Left to Right**
	equality	**==, !=**	**Left to Right**
Bitwise	**Bitwise AND**	**&**	**Left to Right**
	Bitwise exclusive OR	**^**	**Left to Right**
	Bitwise inclusive OR	**|**	**Left to Right**
Logical	**Logical AND**	**&&**	**Left to Right**
	Logical OR	**||**	**Left to Right**
Ternary	**Ternary**	**? :**	**Right to Left**
Assignment	**assignment**	**=, +=, -=, *=, /=, %=, &=, ^=, |=, <<=, >>=, >>>=**	**Right to Left**