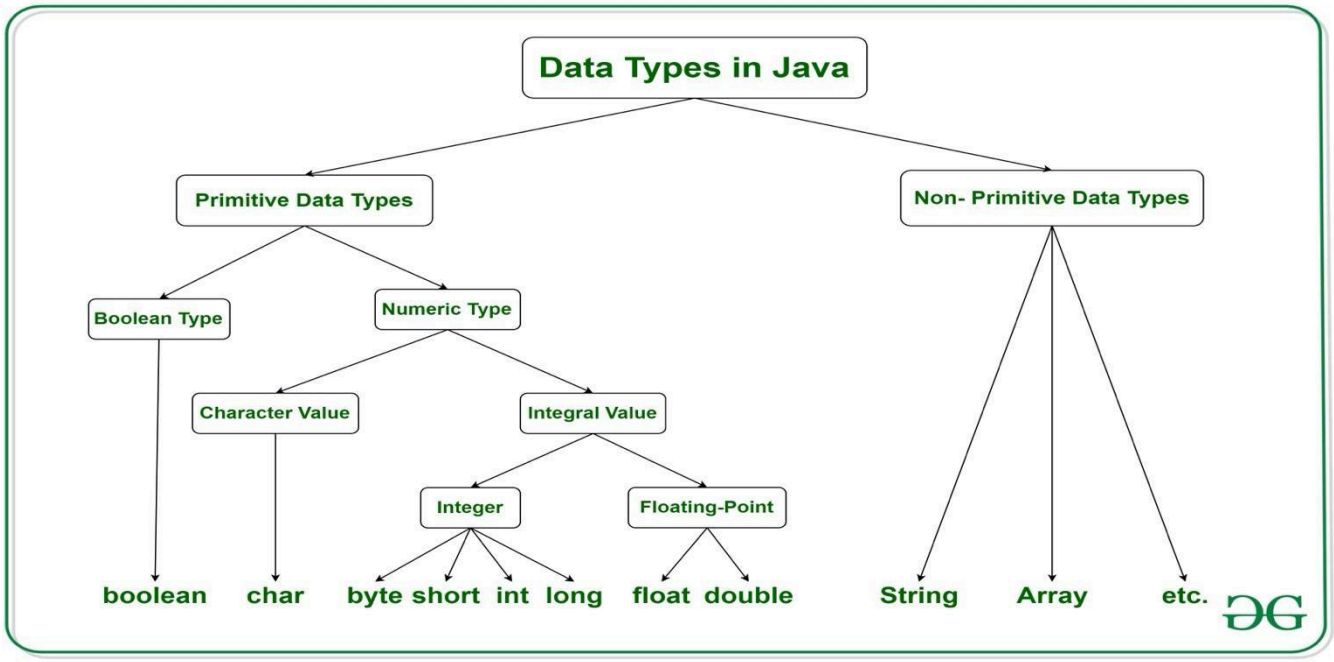
Data Types

Are the kinds of values that can be stored and manipulated within computer programs
Java is a strongly typed programming language



Type	Kind of Value	Default	Size	Range
boolean	true or false	false	1 bit	NA
char	Unicode character	\u0000	16 bits	\u0000 to \uFFFF
byte	Signed integer	0	8 bits	-128 to 127
short	Signed integer	0	16 bits	-32768 to 32767
int	Signed integer	0	32 bits	-2147483648 to 2147483647
long	Signed integer	0	64 bits	-9223372036854775808 to 9223372036854775807
float	floating point	0.0	32 bits	±1.4E-45 to ±3.4028235E+38
double	floating point	0.0	64 bits	±4.9E-324 to ±1.7976931348623157E+308

Primitive	Non-Primitive
Primitive types are used for simple, non-decomposable values such as an individual number or individual character.	*Non-primitive types* are used to store a group of values
Primitive types are predefined in Java	Non-primitive types are created by programmers (except *String*)
Primitive types *start with a lowercase letter*	Non-primitive types *start with an uppercase letter*

Variable – a location in main memory that holds only one value of a particular data type at a time, but that value can be changed o Must be declared with a name and a type before they can be used. double radius;

In Java, **the assignment** statement is used to change the value of a variable

variableName = Expression;

area = radius * radius * PI;

When an assignment statement is executed, the expression is first evaluated, and then the variable on the left-hand side of the equal sign is set equal to the value of the expression.

```
area = radius * radius * PI;
```

A **constant** is like a variable except that it holds the same value during its entire lifetime

In Java, the final modifier is used to declare a constant

final double PI = 3.14159;

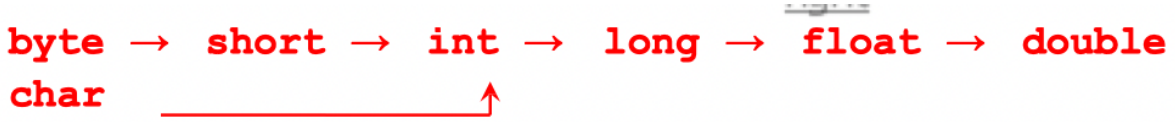
There are three kinds of variables in Java:

o Local variables o Instance variables o Class/Static variables

Local Variables

v Are declared inside the body of methods, constructors, or blocks v Are created when the method, constructor or block is entered and they will be destroyed once the method, constructor, or block ends **No ACCESS MODIFIER**

Changing a primitive value from one data type to another is known as data type conversion



Casting

Explicit casting is required when we perform a narrowing conversion o A value of broader (higher size) type like double can be converted to a value of a narrower (lower size) type like int with explicit casting.

int b = (int)d;

Literal – an item in Java which has one specific, fixed value that appears directly in the program. E.g., 4, 4.0, 'a', true, and "hello" are literals

Escape characters :

Backspace	\b
Horizontal tab	\t
NUL character	\0
Linefeed – new line	\n
Form Feed – new page	\f
Carriage return	\r
Double quote	\"
Single quote	\'
Backslash	\\

Precedence of Arithmetic Operators

Operator	Operation
*	Multiplication
/	Division
%	Remainder
+	Addition
–	Subtraction
=	Assignment

Java a lows you to combine assignment and addition operators using a shorthand operator, for example: i += 8; i=i+8;

The unary operators require only one operand; they perform various operations such as incrementing or decrementing a value by one, negating an expression, or inverting the value of a Boolean

Operator	Operation	Precedence (order of evaluation)
postfix (increment)	n++	Evaluated **first**. If there are several operators of this type, they're evaluated **from right to left**.
postfix (decrement)	n--	
Unary plus	+n	Evaluated **next**. If there are several operators of this type, they're evaluated **from right to left**.
unary minus	-n	
prefix (increment)	++n	
prefix (decrement)	--n	
logical compliment	!	

Precedence

1. Arithmetically, the expression in the parentheses is evaluated first. Parentheses can be nested, in which case the expression in the inner parentheses is executed first
2. When evaluating an expression without parentheses, the operators are applied according to the precedence and associativity rules
3. When two operators share an operand the operator with the higher precedence goes first, e.g., 1 + 2 * 3 is treated as 1 + (2 * 3)
4. When two operators have the same precedence, the expression is evaluated according to associativity rules, e.g., x = y = z = 17 is treated as x = (y = (z = 17))
5. Unary operators of equal precedence are grouped right-to-left
   +-+rate is evaluated as +(-(+rate))
6. Binary operators of equal precedence are grouped left-to-right
   base + rate - hours + days is evaluated as ((base + rate) - hours) + days
7. Assignment operators are grouped right-to-left
   n1 = n2 = n3; a = b += c = 5;is evaluated as is evaluated as n1 = (n2 = n3); a = (b += (c = 5));

Highest ↓ Lowest

Precedence	Associativity
`( )` parenthesis for explicit grouping	Left – Right
`++, --` (postfix as in `x++` and `x--`)	Right – Left
The unary operators: `-, +, ++, --` (prefix as in `++x` and `--x`), and `!`	Right – Left
Type cast `(type)`	Right – Left
The binary operators: `*, /, %`	Left – Right
The binary operators: `+, -`	Left – Right
The assignment operators: `= += -= *= /= %=`	Right – Left

When both operands are integer types, division results in an integer type. Any fractional part is discarded (not rounded)

The reminder % operator is usually used with operands of type int to recover the information lost after performing integer division.

The **char** type only represents one character 'A'.

sequence of characters, use the data type ca led **String**.

String str = "Welcom to Java";

You can also create String objects by using the new keyword and a constructor.

String s4 = new String();

Java strings can be concatenated (joined) using the + or += operators to create new **strings**

String str1 = "Java"; String str2 = "Hello " + str1; str2 += ", World";

When a String is combined with almost any other data type, it wil be implicitly converted to string before the concatenation

"The answer is " + 4 evaluates to "The answer is 4"

Use of the == operator only tests whether two String object

references refer to the same object (memory space). The == operator does not test whether the contents of the two Strings are equal.

Methods in String

Method	Description
charAt()	Returns the character at the specified index (position)
compareTo()	Compares two strings lexicographically
compareToIgnoreCase()	Compares two strings lexicographically, ignoring case differences
concat()	Appends a string to the end of another string
contains()	Checks whether a string contains a sequence of characters
equals()	Compares two strings. Returns true if the strings are equal, and false if not
equalsIgnoreCase()	Compares two strings, ignoring case considerations
indexOf()	Returns the position of the first found occurrence of specified characters in a string
lastIndexOf()	Returns the position of the last found occurrence of specified characters in a string
length()	Returns the length of a specified string
replace()	Searches a string for a specified value, and returns a new string where the specified values are replaced
substring()	Returns a new string which is the substring of a specified string
toLowerCase()	Converts a string to lower case letters
toUpperCase()	Converts a string to upper case letters

trim()	Removes whitespace from both ends of a string