Data Types in Java

Char (16 bit)	byte - (8 bit)	float (32 bit)	boolean
	short - (16 bit)	double (64 bit)	
	int - (32 bit)		
	long - (64 bit)		

Primitive data types in java

- In java, all data types are signed and in 2's complement form.
- In 1's complement - there r 2 representation of 0
  but in 2's " - " is 1 " " 0.

- **How to find range of data type in java:**

Step-1 (±) _ _ _ _ _ _ _ // Suppose there r n bits

1 bit reserved for sign — remaining (n-1) bits

↓

Step-2 Using (n-1) bits the range is 0 to $(2^{n-1} - 1)$

↓

Step-3 Now include -ve numbers.

$-(2^{n-1}) \ldots\ldots -2, -1, +0, +1, +2 \ldots\ldots (2^{n-1} - 1)$

There is no -ve 0 → this -1 is beoz of +0

so no -1 is here ↓

Step-4 $-(2^{n-1})$ to $+(2^{n-1} - 1)$

eg find range of long - 64 bit

$\pm$ _ _ _ _ _ . . . _

$\underbrace{\qquad\qquad\qquad\qquad}_{63 \text{ bits}}$

$\downarrow$

0 to $2^{63} - 1$

$\downarrow$

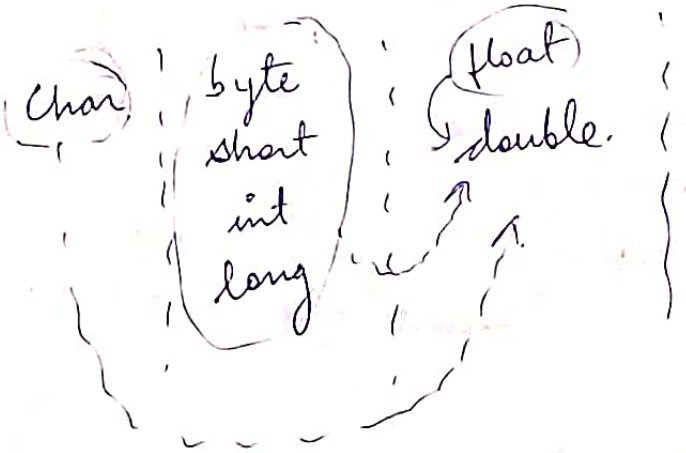
$$\boxed{-(2^{63}) \text{ to } +(2^{63} - 1)}$$

Data Type Conversion in java

There are 2 types of conversion:

1. Widening type conversion
   Automatic type conversion
   Implicit type casting.

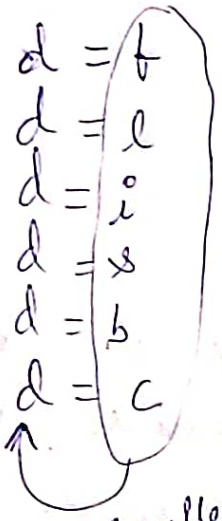
- When a smaller data type is stored in larger data type, then the smaller data type is automatically converted to larger data type



eg: Let's take var of each data type

```
char c = 'a';
byte b = 10;
short s = 20;
int i = 30;
long l = 50;
float f = 3.5f;
double d = 3.5;
```

- Double var d has the largest size, so any var of other data type can be stored in a var of double type.



- Smaller data type can be stored to a larger data type
- Smaller data type is automatically converted to larger data type.
- This conversion is automatic
  It's called auto type conversion, implicit type conversion.

② Narrowing Type Conversion / Manual type conversion / explicit type Conversion

- When a larger data type is stored in a smaller data type, then the larger data type needs to be ~~converted~~ "manually" converted to smaller data type.

- eg char, byte, short < smaller than int
  long, float, double > larger than int

- Now, to store ~~float~~ char, ~~long~~ byte, ~~double~~ short to int, they ~~need to be~~ can be directly copied without converting.
- ~~To~~ because they are smaller than int.

- To, store float, long, double to int, they need to manually converted to int becz they are larger than int.

- They are converted by casting manually.
  Syntax for casting is: (type)

```
i = c  ✓
i = b  ✓
i = s  ✓
```

smaller than int so auto converted to int

```
i = l  ×
i = f  ×
i = d  ×
```

larger than int Can't be converted to int auto

~~i = (int) long~~
~~i = (int) float~~
~~i = (int) do~~

i = (int) l ✓
i = (int) f ✓
i = (int) d ✓

explicitly type casted to int bcoz they r larger than int.

Truncation

- When a larger data type is stored in a smaller data type by using explicit type casting, then the value of larger data type is truncated / reduced.

eg `byte b = (byte) 373;`

b can store values b/w (-128 to 127)

→ 373 is larger than byte
→ So it is explicitly converted to byte.

- New value in b will be = 65
it lies b/w -128 to 127

• Steps to calculate new value

1) Calculate ; Old value %. 2^n ;

2) If . oldvalue %. 2^n is b/w range (-(0) to +(2^{n-1} - 1), then

; newvalue = oldvalue %. 2^n ;

3) otherwise

; newvalue = oldvalue %. 2^n - 2^n ;

Eg1 For byte, $2^n = 256$ as $n = 8$ bit

Range = -127 to +128.

Q1) byte b = (byte) 373;

Step-1 : oldvalue %. 256

= 373 %. 256

= 65

Step2 : 65 is b/w 0 to 127

Step3 :- Newvalue of b = 373 %. 256

= 65.

Q2) byte b = (byte) 874;

Step-1 : oldvalue %. 256

= 874 %. 256 = 106.

Step-2 :- 106 is b/w 0 to 127

Step3 :- Newvalue of b = 874 %. 256 = 106.

Q3) byte b = (byte)436;

Step1 :- 436 % 256 = 180

Step2 :- 180 is larger than 127

Step3 :- ∴ New value = oldvalue % 256 − 256

= 436 % 256 − 256

= 180 − 256

= −76

Q4) byte b = (byte)1024;

Step1 :- 1024 % 256 = 0.

Step2 :- It is b/w 0 to 127

Step :- New value = 0
of b

Q5) byte b = (byte)755.324

Step1 :- 755 % 256 = 243

Step2 :- Its larger than 127.

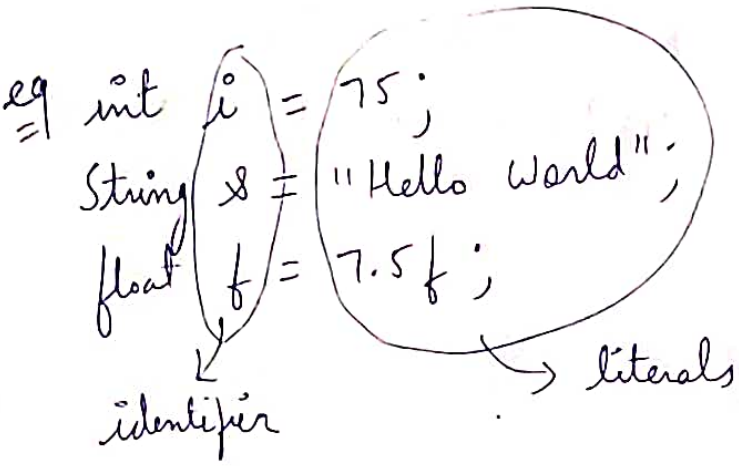
Step3 :- Newvalue = 755 % 256 − 256

= 243 − 256

= −13.

Literals in java

- Identifiers is the name of var, fun, class etc
- Literals r the values or constants



Data type of literals

1) By default numeric literals r 'int'.

eg Sopln (75);
↳ type is 'int'.

int i = 75;
↳ type is int

2) By default, decimal values r of type 'double'

eg Sopln (7.5);
↳ type is double.
not float.

How to change data type of a literal

- Data type can be changed by specifying a suffix.
  - d for double
  - f for float
  - l for long

eg `sopln(75);` → int

`sopln(75l);` → now it is long.

`sopln(7.5);` → double (by default)

`sopln(7.5f);` → float

Note: ** ~~Lithium~~ Decimal values can't be stored to a float var in java

eg `float f = 7.5;` ✗ error → type is double

float

// Larger data type can't be stored to a smaller data type.

Eg float f = 7.5f; // Valid
↳ float
↳ float

//Now both r float.
//∴ To store any no. w/ decimal pt. to a float var, put a suffix f; o/wise there is error.

Q) Which is valid?

1) int i = 10;
2) int i = 10l;
3) ~~int~~ long l = 10;
4) long l = 10l;

Answer:-

Ans.) 1) i = 10 ✓
↳ int ↳ int
//both r int – so valid.

2) i = 10l; ✗
↳ long
↳ int
//invalid – long can't be stored to int.

3) l = 10; ✓
↳ long ↳ int
// int can be stored in long.

4) l = 10l; ✓
↳ long
↳ long
//both r long.

Q) Valid / Invalid ? ***

1) float f = 7.5 ;
2) float f = 7.5f ;
3) float f = 7.5d ;
4) double d = 7.5 ;
5) double d = 7.5f ;
6) double d = 7.5d ;

Answer:

1) f = 7.5 → double. ✗
   ↳ float

// double can't be stored to float as double is larger than float.

2) f = 7.5f ; ✓
   ↳ float → float

3) f = 7.5d ; ✗ // double can't be stored in float.
   ↳ float → double

4) d = 7.5 ; ✓
   ↳ double → double

5) d = 7.5f ; ✓ // float can be stored in double
   ↳ double → float

6) d = 7.5d ; ✓
   ↳ double → double