(7) Data types and Incrementation, Decrementation operator :-

02/11/22

↳ Nitin Sir

→ Numeric Datatype :-

(a) Whole number :-
1. byte (1byte)
2. short (2 bytes)
3. int (4 bytes) → Commonly used datatype.
4. long (8 bytes)

(b) Real number :-
1. float
2. double.

① float :-

Size of float = 32 bits (4 bytes)

Max Value of float = 3.4028235E38

Min Value of float = 1.4E-45

→ S.O.P("Size" + float.SIZE);

S.O.P("Max" + float.MIN_VALUE);

S.O.P("Min" + float.MAX_VALUE);

Eg:- float a = 10.5; → C.E

↳ Incompatible types.

↳ Possible lossy Conversion from double to float.

→ Compiler treats as double by default.

→ to treat it as float we have use "f" (or 'f')

Note:- by default if you specify any real number (decimal number) compiler will treat it as 'double', to specify to the compiler to treat it as float, we need to suffix it with 'f' (or 'f').

Eg:- ① float a = 10.5; → Compiler Error
↳ Possibly loss of Precision.
float b = 10.5f; ✓
float c = 25.5F; ✓

② **double Data type:-**

Size of double = 64
Max Value of = 4.9e-324
Min value = 1.79769313486231570e308

Eg:- double d = 23.567;
→ Compiler treats real number by default as double
→ double d = Max Value of double;

Note:- Datatypes are actually represented to the Compiler and JVM using reserve words.

→ reserve words are normally in "lower case".
→ To map primitive data as object in java from JDK 1.5 Concept of "wrapper class" was introduced.

Eg:-

Primitive	Class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

③ **Character ~~Data type~~ :-**

③ Character Data types [Char] ← → Hyder Sir

Why data type?

→ to store data.
→ to optimize data
→ to store real world data
→ these data types will follow specific format behind the scenes. that format will help to store the data in the form of 0's and 1's in the memory.

→ int → follows Base 2 format.

Char → Primitive data type.

→ to store characters like A, B, #, C etc.

$2^1 = 2$

A	0
B	1

$2^2 = 4$

A	0 0
B	0 1
C	1 0
D	1 1

$2^3 = 8$

000	A
001	B
010	C
100	D
101	E
110	F
011	G
111	H

→ As the no. of characters increasing then number of bits also increased.

$2^1 = 2 \Rightarrow$ 2 char → 1 bit
$2^2 = 4 \Rightarrow$ 4 char → 2 bits
$2^3 = 8 \Rightarrow$ 8 char → 3 bits.

→ Americans discovered 128 characters.
Eg: A B ... z - @ .. a, b, ... 3 # ...

→ for all these 128 characters they have given binary representation, Decimal, Hexadecimal representation

→ $2^7 = 128$
Memory → 7 bits + 1 bit
↳ add to stadardization of Memory.

#ASCII

→ 1 byte ⇒ 8 bit

Character → Binary (00000000) → Decimal → Hexadecimal

They given this representation for Characters

ASCII

#Unicode

→ 1111 → they told to find all characters of diffent languages

2^{16} → 65536 → character / Symbol

→ They give

→ UTF-16

$2^{16} \Rightarrow 65536$ (16 bits ⇒ 2 bytes)

→ UTF ⊢ Unicode Transformation format

→ 2 bytes of Memory allocated for a character

→ Java follows UTF. 2 bytes of memory allocated character.

ASCII ⊢ they given Binary, Decimal and Hexadecimal for 128 characters (2^7)

unicode ⊢ they given Binary, decimal and Hexadecimal for 65536 (2^{16}) Characters.

→ for unicode and ASCII 0 to 128 characters are same. (decimal representation)

→ Java follows UTF. (unicode)
↳ 2 bytes of Memory allocated Character.

Syntax ⊢ `Char a = 'A'`

→ Character in java is within Single quote.!!

Char Syntax ⊢

`Char a = 'A'`

→ Character in java is within Single quote
' '

`char a = "A";` (not valid)

`Char a = 'AB';` (invalid)

→ it should be Single character and Single quote.

Eg: `char a='A';`
`char b = 'i';`

→ for char the class is Character.

→ In other programming languages String and array also treated as data types.

→ In java array and string treated as object.

→ Java is impure object oriented Language.
↳ because we primitive datatypes

→ 100% Pure object oriented when we use wrapper classes.

→ Alternative for primitive data type is wrapper class where everything is treated as object.

④ Truncation / Rounding zero :-

①
```
int a = 25;
int b = 2;
int c = a/b;
S.O.P(c);
```
→ 25/2 ⇒ 12.5 → 12
→ o/p ⇒ 12

②
```
int a = 25;
int b = 2;
float c = a/b;
S.O.P(c);
```
→ 25/2 ⇒ 12.5
→ o/p = 12.0

⇒ Truncation is happening

→ Int / Int = Int.

→ The Result is Integer when we perform operation between Integers irrespective of where it is stored.

→ 0.5 → Truncated (or) Rounding to zero. getting

Eg: ③
```
double a=25
double b=2;
double c=a/b;
S.O.P(c);
```
→ 25/2 = 12.5
→ o/p = 12.5

Check :- a=0, b=2, o/p = 0.0
a=0, b=0, o/p = NaN
↳ not a number.

Formats followed by Datatypes :-

→ byte - 1 bytes
Short - 2 bytes
int - 4 bytes
long - 8 bytes
→ follow base 2 format

float - 4 bytes
double - 8 bytes
→ follows IEEE Single and double Precision format

Char - 2 bytes → follows UNICODE

Eg: 2|45
2|22-1
2|11-0
2|5-1
2|2-1
1-0

45 stored in binary.
1 0 1 1 0 1
32 8 4 1

→ IEEE is More Efficient
→ whatever long takes 8 bytes to store something float takes 4 bytes for same Purpose.

⇒ Why Itt Single and double precision format is Efficient?

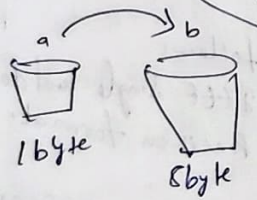
(A) what Ever longs takes 8 byte to store Something float takes 4 byte for same purpose

Eg (1)

```
byte a=45;
double b;
b=a;
so.p(b); → 45.0
```

→ augment operator. Whatever there in Right side given left side.

→ nature of double.



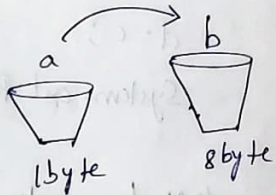
→ [redacted]

(5) Type Casting / Numeric Promotion 04/11/22

→ Changing type of data from one type to another type is called type casting / Numeric Promotion.

Eg

```
byte a=45;
double b;
b=a;
s.o.p(b); → 45.0
```



↳ nature of double

→ Converting data from one type to another type. ([redacted]) without any efforts (or) automatically is called implicit type Casting

⇒

→ implicit type Casting
- ↳ internally
- ↳ automatically
- ↳ behind the Scene [redacted]

[redacted] → Implicit type conversion possible

→ byte - short - int → long → float → double

Eg ②:- double c = 45.5;
byte d;
d = c;
System.out.Println(d); → Compilation Error.

⑥ Explicit typeCasting / Narrowing type Casting :-

→ Converting a higher data type into a lower one is called narrowing type casting. it is also known as Explicit Conversion (or narrowing.

Eg:- ① double a = 45.5;
byte b;
b = (byte) a;
System.out.Println(b); → 45

→ [redacted] -
→ Value after decimal - precision

→ o/p: 45.5 → loss of precision might occur.
x

Eg ②:- byte ab = 10;
byte ac = 20;
byte result = ab * ac;
S.o.p (result) → Error
↳ Cannot Convert from into to byte.

Eg ③:- [redacted]

byte ab = 10
byte ac = 20
int result = ab * ac;
S.o.p(result). → o/p: 200

⑥ operator :-

operators are used to perform operations on Variables and Values.

Eg ① int a = 10+5; → 15

(i) Incrementation :- Increasing Existing Value by 1.

Eg ① :- a++ → Incrementation

```
int a = 5;
a = a+1;  → 5+1 = 6
S.o.p(a);
```

Alter :-

```
int a = 5
a++;
System.out.println(a); → a=6
```

(ii) Decrementation :- decreasing Existing Value by 1.

Eg ① :-

```
int a = 5;
a = a-1; (or a--;)
S.o.p(a);
```

(i.a) Pre and Post Incrementation

a++ → Post incrementation

++a → Pre incrementation.

Eg ① :-

```
int a = 5;
S.o.p(a); → o/p: 5
a++;
S.o.p(a); → o/p: 6
```

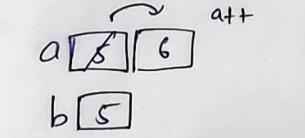
```
int a = 5;
S.o.p(a); → 5
++a;
S.o.p(a); → 6
```

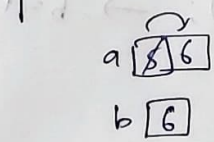
o/p:

Eg ② :-

```
int a = 5;
int b;
b = a++;
S.o.p(a); → 6
S.o.p(b); → 5
```

```
int a = 5
int b;
b = ++a
S.o.p(a) → 6
S.o.p(b) → 6
```





→ Post incrementation a is stored in b and incremented after that

Eg ③ :-

⑥ Eg

int a=5 ; a [5 6 7 8]

int b ; b [19]

b = a++ + a++ + ++a;
5 + 6 + 8
Post Post Pree

(i) S.o.p (a);

S.o.p (b);

Eg ③

int a=5; a [5 6 7 8]

int b; b [19]

Post Post Pree

b = a++ + a++ + ++a

5 + 6 + 8 = 19

S.o.P(a); ⟶ a=8

S.o.P(b); ⟶ b=19

b=a++	b=++a
↳ store it Incre it	Increment it Store it ↓

Eg ④ int a=5 a [ ]

int b; b [ ]

b= ++a + a++ + ++a + a--;

S.o.P(a)

S.o.P(b)

Eg ④

int a=5; a [5 6 7 8 7]

int b; b [28]

b = (++a) + (a++) + (++a) + (a--); → 28

6 6 8 + 8 = 28

S.o.P (a); ⟶ 7

S.o.P(b); ⟶ 28

Eg ⑤ int a=5, b;

b=(a++) + (--a) - (a--) - (a++);

5 + 5 5 - 4

S.o.P (a); ⟶ 4

S.o.P (b); ⟶ 1

a [5 4 4 4]

b [1]