Literals

Any constant value which can be assigned to the variable is called literal.

Eg → `int x = 10;`

- `int` → Data type (keyword)
- `x` → Name of variable (identifier)
- `10` → constant value (literal)

Integral literals

For integral data types (byte, short, int, long) we can specify literal values in the following ways -

① Decimal form (Base 10)
Allowed digits are 0 to 9

eg → `int x = 10;`

② Octal form (Base 8)
Allowed values are 0 to 7

Literal value should be prefix with zero.

eg → `int x = 010;`

Date ___/___/_____ Page No.: _____

③ Hexa-decimal form (Base 16)
Allowed digits are 0 to 9, a to f

For extra digits (a to g) we can use both lower & upper case (character) this is one of very few areas where JAVA is not case-sensitive.

literal value should be prefix with 0X.

eg → int x = 0X10;

these are only possible ways for literals for integral value (data types).

Ex → ✓ int x = 10;

✗ int x = 0786; C.E → integer number too large.

✓ int x = 0777;

✓ int x = 0XFace;

✓ int x = 0XBeef;

✗ int x = 0XBeer;

Ex →

```
class Test
{
public static void main (String [] args)
 {
    int x = 10;
    int y = 010;
    int z = 0X10;
$.pln Super (x + "..." + y + "..." + z)
 }
}
```

output : 10 ... 8 ... 16.

(*) By default, every integral literal is of int type.
But we can specify explicitly ~~with~~ suffixed with (l or L).

✓ int x = 10;

✓ long l = 10L;

X int x = 10L; C.E → Possible loss of precision
found : long
required : int

✓ long l = 10;

Date ___/___/_____ Page No.: _____

There is no direct way byte & short literal explicitly, but indirectly we can specify, whenever we are assigning integral literal to the byte value & if the value within the range of byte then compiler treats it as automaticlly as byte literal similarly short literal also.

Eg → ✓ byte b = 10;
✓ byte b = 127;
× byte b = 128; → C.E → PLP

✓ short s = 32767;
× short s = 32768; → CE → PLP

Floating point literals

By default every floating point literal is of double type & hence we can't assign directly to the float variable, but we can specify floating point literal as float type by suffixed with f or F.

eg → float f = 123.456; → C.E → possible loss of precision found: double required: float

float f = 123.456F;

~~float~~ double d = 123.456;

Date ___/___/_____ Page No.: _____

We can specify explicitly floating point literal as double type by suffixed with d or D.
Ofcourse this convention is not required.

eg → double d = 123.456 D;
float f = 123.456 d;
↓
C.E → Possible loss of Precision
found : double
required : float

We can specify floating point literals only in decimal form & we can't specify in octal & hexa-decimal forms.

eg → ✓ double d = 123.456;
✓ double d = 0123.456;
✗ double d = 0X123.456;
↓
C.E → mal formed floating point literal

We can assign integral literal directly to floating point variables & that integral literal can be specified either in decimal, octal or hexa-decimal forms.

eg → ✗ double d = 0786;
C.E → integer too large.
✓ double d = 0XFace;

Date ___/___/_____ Page No.: _____

✓ double d = 0786.0;
✗ double d = 0xFace.0;
✓ double d = 10;
✓ double d = 0771;

We can't assign floating point literals to integral type.

eg → ✓ double d = 10;
✗ int x = 10.0;
↳ C.E → P L P
found: double
required: int.

We can specify floating point literal even in exponential form.

eg → ✓ double d = 1.2e3;
S.o.p(n ~~super~~ (d);
output → 1200.0.

✗ float f = 1.2e3;
C.E → P L P found: double
✓ float f = 1.2e3F;

Boolean literals

The only allowed value are T/F.

eg → ✓ boolean b = true;
✗ boolean b = 0;
✗ boolean b = True;
✗ boolean b = "True";

Date 2/12/24 Page No.:

char literal

1st way: We can specify char literal as single character within single quotes.

eg → ✓ char ch = 'a';

(X) char ch = a; → C.F → cannot find symbol → symbol → var a, location → class Test

(X) char ch = "a";
↓
C.E → incompatible types found = Java lang String required : char

(X) char ch = 'ab';
↓
C.E : unclosed char literal
CE 2 : unclosed char literal
CE 2 : not a statement

2nd Way: We can specify char literal as integral literal which represents unicode value of the character and that integral literal can be specify either in decimal, octal & Hexa decimal forms.
But allowed range 0 to 65535.

eg → ✓ char ch = 97
✓ char ch = 0XFace;
✓ char ch = 65535;
✓ char ch = 0777;

[9→97;];[1970→?] ; [197→+]
↳ why ?

Date __/__/__ Page No.: _____

✗ char ch = 65536;
CE: Possible loss of precision
found: int
required: char

3rd way We can represent char literal as in unicode representation which is nothing but '\uxxxx'
4-digit Hexa-decimal number

eg → char ch = '\u0061';
S.p/n (ch);
output → a.

Query escape character is a valid char literal.

eg → ✓ char ch = '\n';
✓ char ch = '\t';
✗ char ch = '\m';
CE: illegal escape char

Escape char	Description
\n	new line
\t	Horizontal tab
\r	carriage return
\b	Back space
\f	Form feed
\'	single quote
\"	Double quote
\\	Back slash

"No real change in history has ever been acheived by discussions." —Subhash chandra bose

Date ___/___/_____ Page No.: _____

carriage return → moving to 1st cell of second line.

Ques :- which of the following are valid ?

✗ char ch = 65536;

✗ char ch = 0xBeer;

✗ char ch = \uface;

✓ char ch = '\uBeef';

✗ char ch = '\m';

✗ char ch = '\iface';

String literal

Any sequence of characters within double quotes is treated as string literal.

eg → String s = "nonu";

* 1.7 version enhancements wrt literals

Date ___/___/_____ Page No.: _____

① Binary literals

for integral datatypes until 1.6 version we can specify literal values in the following ways -

1) decimal
2) octal
3) Hexa-decimal

→ But from 1.7 version onwards, we can specify literal values in Binary form also.
→ Allowed digits are 0 & 1.

→ Literal value should be prefixed with 0B or 0b.

eg → int x = 0B1111;
S.pln ~~super~~ (x)
output : 15.

② usage of underscore symbol in numeric literals from 1.7 version onwards we can use underscore symbols between digits of numeric literal.

eg → double d = 123456.789
⇓
double d = 1_23_456.7_8_9;
US system → double d = 123_456.7_8_9;

"No real change in history has ever been acheived by discussions." –Subhash chandra bose

Date ___/___/_____ Page No.: _____

→ The main advantage of this approach is to improve the readibility of code.

→ At the time of compilation these underscore symbols will be removed automatically.
Hence, after compilation these line will become
⇒ double d = 123456.789;

→ We can use more than one underscore also between the digits.

eg → ✓ double d = 1--23__45--6.7-8-9;

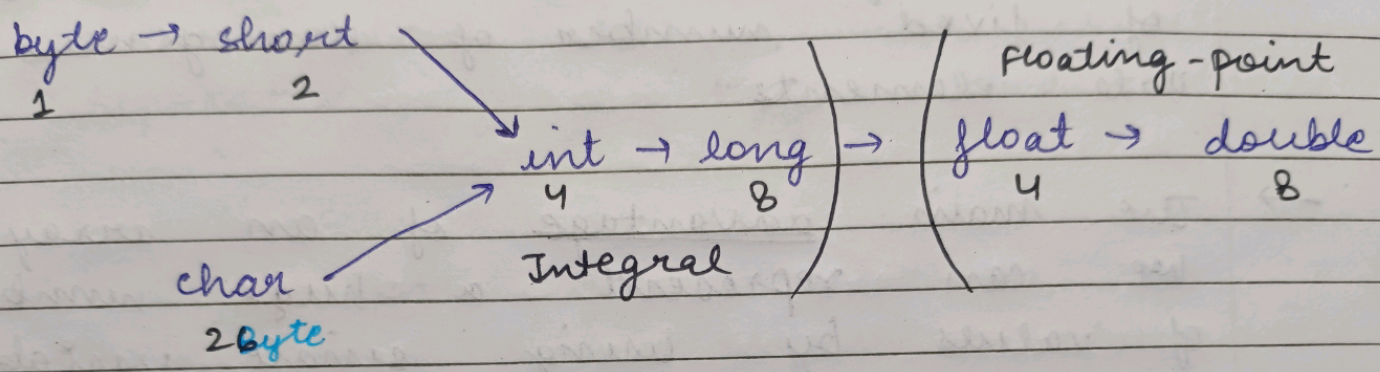
✓ double d = 1---23---45--6.7-8-9;

→ We can use underscore symbols only blw the digits if we are using anywhere else, we will get compile time error.

eg → all are invalid
× double d = 1--23--456.7-8_9;
× double d = _1--23_456.789;
× double d = 1-23--456.7-8-9_;

Date ___/___/___

→ from lower to higher



NOTE: 8 byte long value we can assign to 4 byte float variable b coz both are following different memory representation.

eg → float f = 10l;
S.pln ~~sapln~~ (f); = 10.0;