~ Ankita Bansal

LITERALS

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

e.g `int x = 10`

- `int` — data type
- `x` — variable / identifier
- `10` — constant value / literal

Integral literals

For integral datatypes (byte, short, int, long) we can specify literal values in the following base.

① **Decimal literals** (base-10)
Allowed values are 0-9
e.g int x = 10

② **Octal literals** (base-8)
Allowed values are 0 to 7.
Literal values should be prefix with 0.
e.g int x = 010

③ **Hexa-decimal form** (base-16)
Allowed digits ⇒ 0-9, a-f
For hexa digits (a-f), we can use both lowercase & uppercase letters. This is one of very few areas where Java is not case-sensitive.
The literal values should be prefix with 0x/0X.
e.g int x = 0X10.

These are only possible ways to specify literal values for integral data-types.

Ques! Which of the following declarations are valid?
(i) int x = 10; ✓
(ii) int x = 0786 X
(iii) int x = 0777 ✓
(iv) int x = 0XFace ✓
(v) int x = 0XBeef ✓
(vi) int x = 0xBeer X

Ques → What is the output?

```
class Test {
    public static void main(String[] args) {
        int x = 10;
        int y = 010;
        int z = 0x10;
        System.out.println(x+" "+y+" "+z);
    }
}
```

It will get printed only in decimal form so octal and hex values will be converted in decimal first.

$(10)_8 = (?)_{10}$

$0 \times 8^0 + 1 \times 8^1 = 8$

$(10)_{16} = (?)_{10}$

$0 \times 16^0 + 1 \times 16^1 = 16$

Output :- 10 8 16

* By default, every integral literal is of int type. But we can specify explicitly as long by adding suffix l/L.

  10l 010L 0x10l

```
int x = 10 ✓
long l = 10L ✓
int x = 10L X → CE :- PLP found: long required: int
long l = 10 ✓
```

* There is no direct way to specify byte and short literals explicitly but we can declare it indirectly. Whenever we are assigning integral literals to the byte value variable, until the value within the range of byte, compiler treat it automatically as byte literal, same for short literal.

```
byte b = 127 ✓
byte b = 128; X → CE :- PLP
```

Floating-point literals

By default, every floating-point literals are of double type hence we can't directly assign to float variables.
But we can assign floating type literals by suffixing with f/F.

float f = 123.456; X → CE:- PLP
float f = 123.456F; ✓ found: double
double d = 123.456; ✓ required int.

* We can specify floating type literal as double type by suffix with d/D. Ofcourse this convention is not required.
double d = 123.456D ✓
float f = 123.456 d X CE:- PLP
found: double required : float

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

e.g. double d = 123.456 ✓
double d = 0123.456 ✓ → it is decimal literal, not octal
double d = 0x123.456 X CE: malformed floating point literal

* We can assign integral literal directly to floating point variables. And the integral literal can either be specified in decimal / octal / hexa forms.

double d = 0786 X
double d = 0xface ✓
double d = 0786.0 ✓
double d = 0xface.0 X
double d = 10; ✓
double d = 0777; ✓

* We can't assign floating point literals to integer types.

double d = 10 ✓

int x = 10.0 ✗ CE :- PLP

found: float required: int

* We can specify floating point literal either in exponential form.

double d = 1.2e3; ✓

float f = 1.2e3; ✗

float f = 1.2e3F; ✓

1.2e3 = 1.2×10^3 = 1.2×1000 = 1200.0

Bool boolean literals

Only allowed values for boolean variables are true/false.

If we provide any other, get Compile Time error.

boolean b = true; ✓

boolean b' = 0; ✗

boolean b = True; ✗

boolean b = "true"; ✗

char literals

We can specify char literals as single letter b/w single quotes

e.g. char ch = 'a' ✓

char ch = a ✗ CE:- Can't find symbol.

char ch = "a" ✗ CE:- Incompatable type

char ch = 'ab' ✗ CE:- Unclosed char literal

We can specify char literals as integrals literals which

Represents its corresponding UNICode value. The integral literal can either be specified in decimal / octal / hexa-decimal form but allowed range is 0 - 65535.

(i) char ch = 97; ✓
    SOUT(ch); ⇒ a

(ii) char ch = 0xface ✓
(iii) char ch = 0777 ✓
(iv) char ch = 65535 ✓
(v) char ch = 65536 ✗ CE:- PLP

⇒ We can specify char literal even in unicode representation
'\uXXXX' → 4 digit hexa-decimal number.

a = 97 — 97 / 16 — 6 - quotient, 1 - remainder.
a = '\u0061'

⇒ Every escape character is a valid char literal.
char ch = '\n' ✓
char ch = '\t' ✓
char ch = '\m' ✗. CE:- illegal escape character

2)

Escape character	Description
\n	new line
\t	Horizontal tab
\r	Carriage return (first cell of next line)
\b	Back space ←
\f	form feed
\'	single quote
\"	double quote
\\	Backslash

Q. Which of the following are valid?

c = 65536 ✗ - Out of range
c = 0XBeer ✗ because of r
c = \uface ✗ because of quotes
c = '\ubeef' ✓
c = '\m' ✗ illegal backslash escape character
c = '\iface' ✗ Because of i

String literals

Any sequence of characters b/w double quotes is string literal.
String s = "Ankita"

⇒ 1.7 version enhancements wrt literals

1. Binary literals.
For integral data-type until 1.6 version, we can specify literal value only in octal/hexa/decimal form.
But from 1.7 version, we can specify in binary also but allowed values are 0 and 1, & it should be suffix with b/B.

e.g. int x = 0B1111
Sout(x) ⇒ 15

2. Usage of underscore symbol in numerics literals.
We can use '_' b/w digits of numeric literals.

e.g. double d = 123456.789
d = 1_23_456.7_8_9
d = 123_456.7_8_9
Advantage ⇒ improved readability

At the time of compilation, these '_' will be removed automatically Hence after compilation both will be the same.

double d = 123456.789

⇒ User can use more '_' also b/w the digits, anyway compiler is going to remove

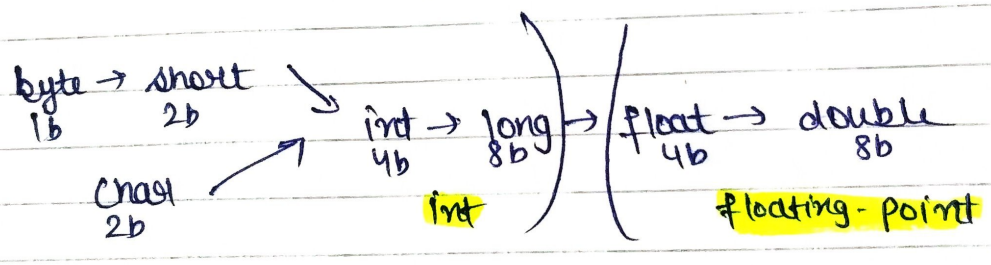
double d = 1___23_4_56_7.8_9 ✓

⇒ We can use '_' only b/w the digits. If we are using anywhere else we'll get compile time error.

e.g. d = _1-23_456.789 X

d = 1_23_456_.789 X

d = 1_23_456.789_ X



⇒ 8byte long value we can assign to 4 byte float value because both are following diff. memory allocation internally.

float f = 10l ✓