1) Identifier :-

→ A name in Java program is called identifier, it can be class name or variable name or method name or label name.

Eg:-
```
class Test                          → class name
{
    p.s.v. main(String [] args)     → method name
    {
        int x=10;
    }
}
```
✓ → is identifiers.

↳ variable name

* Rules to define identifiers :-

1) The only allowed characters in Java identifier are :

✓ (a to z, A to Z, 0 to 9, _, $)

→ If we are using any other character we will get Compiletime Error.

Eg:-
- ✓ all_member
- ✗ all#
- ✓ _$_$
- ✗ 098$_10

2) identifier can't starts with digit. Eg:- ✗ 123total
✓ total123.

3). Java identifiers are CaseSensitive.

```
Class Test
{
 int Number =10;
 int NUMBER =20;
 int NumBer = 30;
}
```

} We can differenciate w.r.t Case.

4) There is no Length Limit for Java identifiers. but it's not recommended to take morethan 15 length (>15).

5) Reserved words Can't be used as identifiers.

6) All predefined Java class names & interface names we can use as identifiers. ~~but~~ Eventhough it is legal, but it is not recommended.

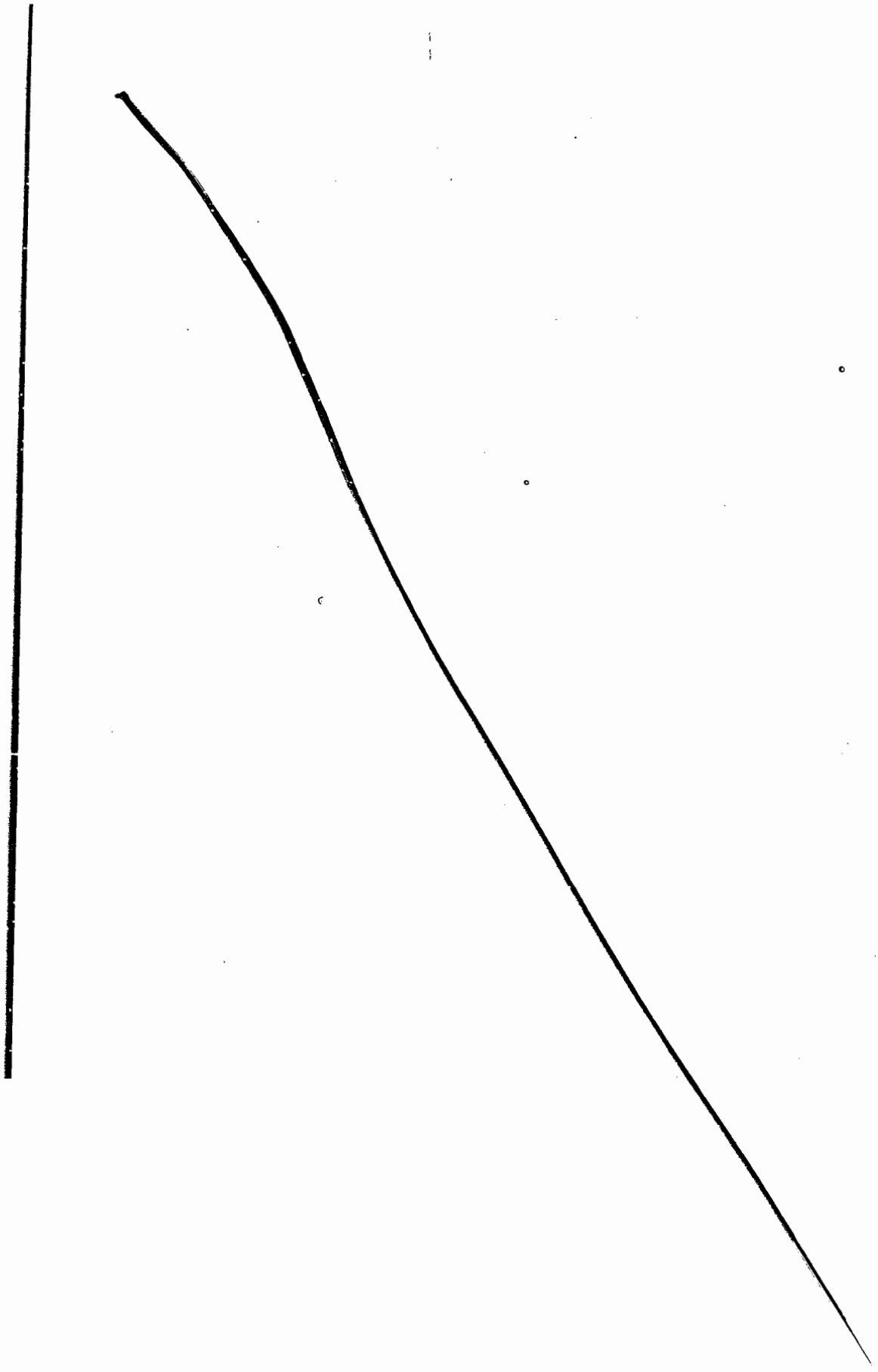
Eg:-

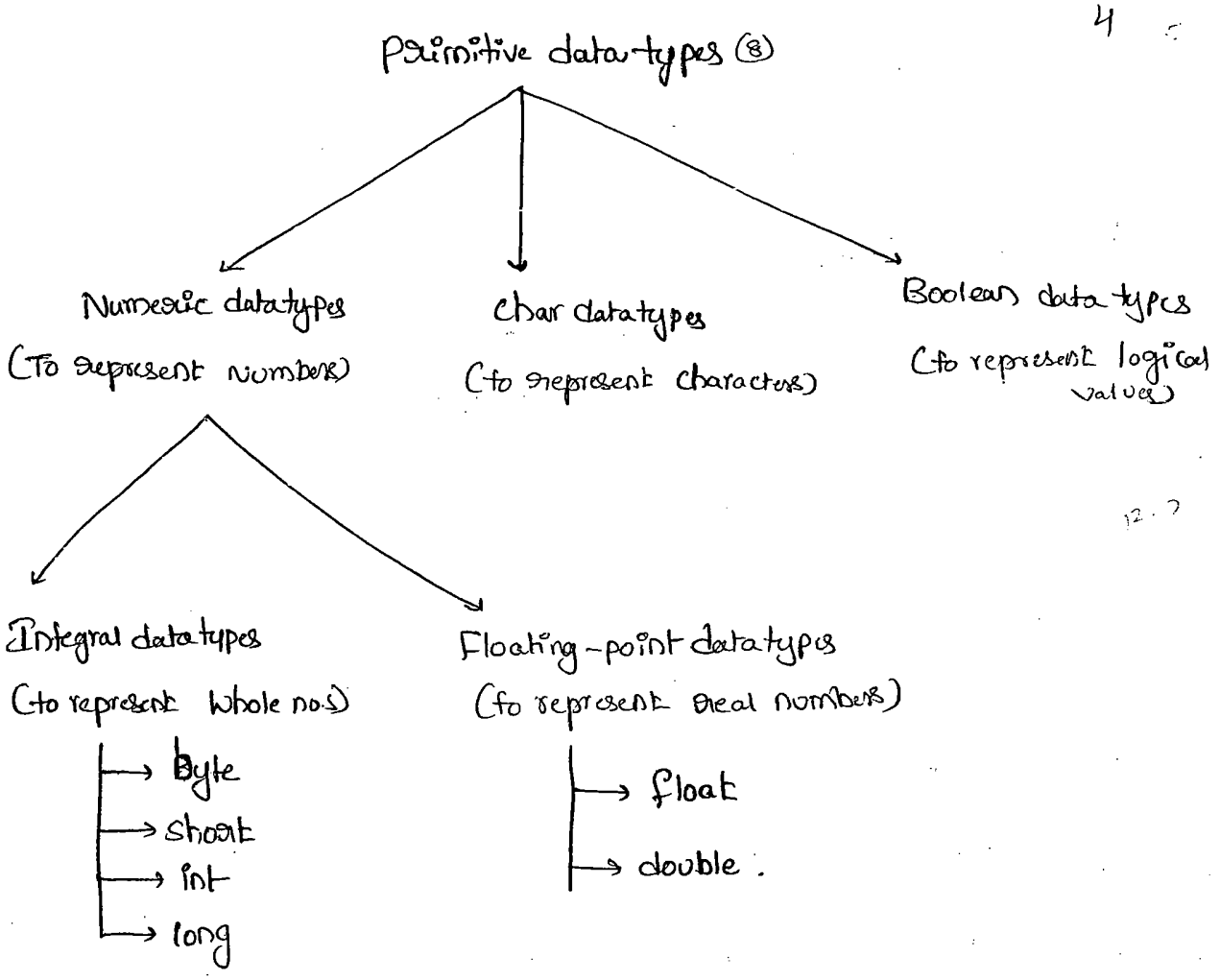
```
Class Test
{
int String = 10;
S.o.pln(String); 10
}
```

```
Class Test
{
int Runnable = 20;
S.o.pln(Runnable); 20
}
```

Q) Which ~~are~~ the following are valid Java identifiers?

✓ ① Java2Share
X ④ 4shared
X ③ all@hands
✓ ④ total_noof_Students
✓ ⑤ _$_
X ⑥ total#
X ⑦ int
✓ ⑧ Integer





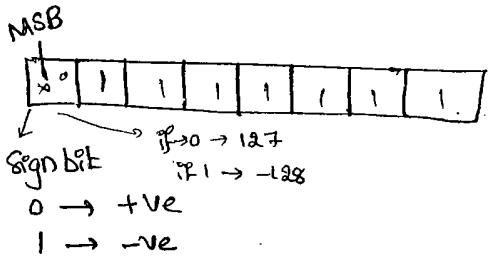
① Byte :-

Size = 8-bits (or) 1 Byte

Max_value = 127

Min_value = -128

Range = -128 to +127



→ The MSB Significant Bit is called "Sign bit". 0 means +ve value, 1 means -ve value

→ +ve numbers represented directly in the memory where as -ve numbers represented in 2's Complement form.

Ex:-

✓ byte b = 100;

✓ byte b = 127;

✗ byte b = 130; C.E:- possible loss of precision
found : int
Required : byte

✗ byte b = 123.456; C.E:- PLP
found : double
Required : byte

✗ byte b = true; C.E:- ~~PLP~~ inCompatible types
found : boolean
Required : byte

✗ byte b = "durga"; C.E:- inCompatible types
found : ~~String~~ .lang.String
Required : byte.

→ byte datatype is best suitable if we want to handle data in terms of streams either from the file or from the Network.

② Short :-

Size : 2-bytes (16-bits)

Range : -2^{15} to $2^{15}-1$

[-32768 to 32767]

Eg:- ✓ Short s = 32767

✓ Short s = -32768

✗ Short s = 32768 C.E:- PLP
found : int
Required : short

✗ Short S = 123.456 C.E:- PLP

found : double

Required : Short

✗ Short S = true C.E:- Incompatible types

found : boolean

Required : Short.

→ Most frequently used datatype in Java is "Short"

→ Short datatype is best suitable if we are using 16-bit processors like 8086 but these processors are Completly outdated & hence Corresponding Short datatype is also outdated.

(3) int :-

→ The most Commonly used datatype is "int"

Size : 4-bytes

Range : -2^{31} to $2^{31}-1$

[-2147483648 to 2147483647)

Note :-

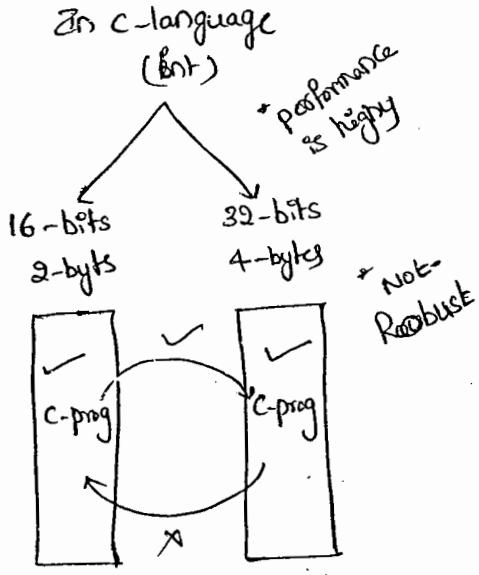
→ In C language the size of int is varied from platform to platform for 16-bit processors it is 2-bytes but for 32-bit processors it is 4-byte

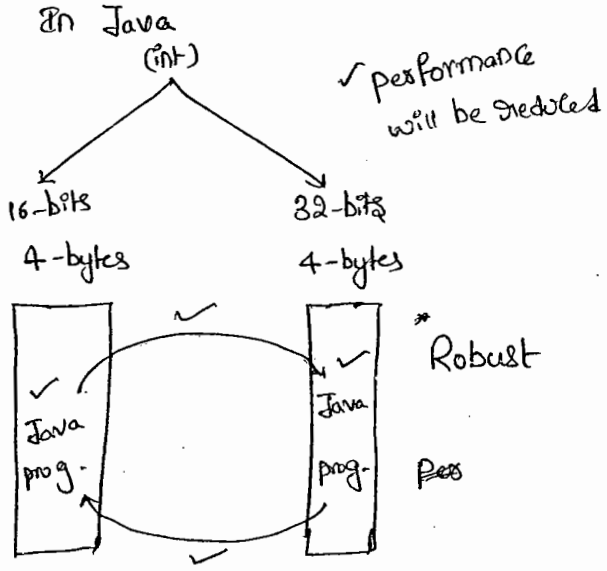
* The main advantage of this approach is read & write operation we can perform very effiently and performance will be improved. But the main

* disadvantage of this approach is the chance of failing C program is very very high if we are changing platform bcoz C language is not Considered as Robust.

→ But in Java the size of int is always 4-bytes irrespective of any platform. * The main advantage of this approach is the chance of failing Java program is very very less, if we are changing underlaying platform & Hence Java is considered as Robust lang.

* But the main disadvantage in this approach is read & write operations will become costly & performance will be reduced.





13/08/11

4) long :-

→ When ever int is not enough to hold big values then we should go for long data type.

Ex(1):- To represent the amount of distance travelled by light in 1000 days int is not enough Compulsary we should go for long type

Ex:- long l = 1,23,000 x 60 x 60 x 24 x 1000 miles;

Ex(2):-

To Count the no. of characters present in a big file. int may not enough Compulsary we should go for long data type.

Size = 8 bytes

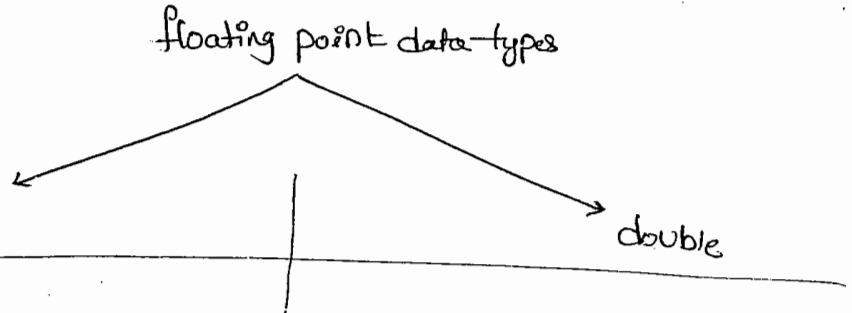
Range = -2^{63} to $2^{63}-1$

Note:-

→ All the above data types (byte, short, int, long) ment for representing whole values.

→ If we want to represent real numbers Compulsary we should go for floating point data types.

Floating Point data types :-



float	double
1) Size : 4-bytes	1) Size : 8-bytes
2) Range : -3.4e38 to 3.4e38	2) Range : -1.7e308 to 1.7e308
3) If we want 5 to 6 decimal places of accuracy then we should go for float	3) If we want 14 to 15 decimal places of accuracy then we should go for double.
4) float follows Single precision	4) double follows double precision

Boolean data type :-

Size : Not Applicable (Virtual machine dependent)

Range : Not Applicable [But allowed values are true/false]

Q) Which of the following boolean declarations are valid

✗ 1) boolean b = 0; C.E:- incompatible types
found : int
required : boolean

✓ 2) boolean b = true;

✗ 3) boolean b = True; C.E:- Can't find Symbol
Symbol : Variable True
Location : class Test

✗ 4) boolean b = "false" C.E:- incompatible types
found : java.lang.String
required : boolean

✓ 5) boolean True = true
boolean b = True
S.o.pln(b); true

Ex :-

```
int x=0;
if(x)
{
S.o.pln("Hello");
}
else
{
S.o.pln("Hi");
}
```

in Java ✗

C++ ✓ C.E:- incompatible types
found : int
required : boolean

```
while(1)
{
S.o.pln("Hello");
}
```

in Java ✗

in C++

→ The only allowed values for the boolean datatypes are "true" or "false" where Case is important.

Char data-type :-

→ In languages like C & C++ we can use only ASCII characters and to represent all ASCII characters 8-bits are enough. hence char size is 1-byte.

→ But in java we can use unicode characters which covers world wide all alphabetes sets. The no of unicode characters is >256 & hence 1-byte is not enough to represent all characters Compulsary we should go for 2-bytes.

Size : 2-bytes

Range : 0 to 65535

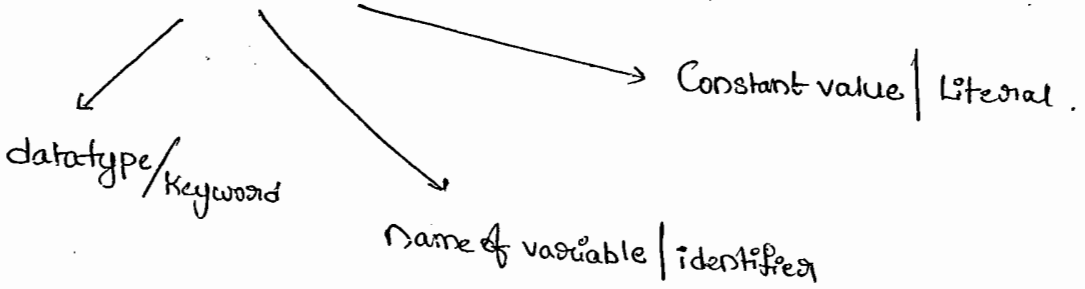
Summary of primitive data types :-

datatype	Size	Range	Corresponding Wrapper classes	default value
byte	1-byte	-2^7 to 2^7-1 [-128 to 127]	Byte	0
Short	2-bytes	-2^{15} to $2^{15}-1$ [-32768 to 32767]	Short	0
int	4-bytes	-2^{31} to $2^{31}-1$ [-2147483648 to 2147483647]	Integer	0
long	8-bytes	-2^{63} to $2^{63}-1$	Long	0
float	4-bytes	-3.4e38 to 3.4e38	Float	0.0
double	8-bytes	-1.7e308 to 1.7e308	Double	0.0
Char	2-bytes	0 to 65535	Character	0 [represents blank space]
boolean	NA	NA [true/false are allowed]	Boolean	false (true in C++)

Literals :-

↳ A Constant value which can be assign to the variable is called "Literal"

Ex:- `int x = 10;`



Integral Literals :-

→ For the Integral datatypes (byte, short, int, long) the following are various ways to specify Literal value

1) decimal literals :-

allowed digits are 0 to 9

Ex:- `int x = 10;`

2) Octal literals :-

→ allowed digits are 0 to 7

→ literal value should be prefixed with "0" [zero]

Ex:- `int x = 010;`

3) Hexadecimal literals :-

→ allowed digits are 0 to 9, a to f (or) A to F

→ For the extra digits we can use both upper case & lower case. This is one of very few places where Java is not case sensitive

→ Literal value should be prefixed with 0x (or) 0X

Ex:- int x = 0x10
(or)
int x = 0X10

→ these are the only possible ways to specify integral literal.

Ex:-
```
class Test
{
  p.s.v.m(String[] args)
  {
    int x = 10;
    int y = 010;
    int z = 0X10;
    S.o.pln(x+"----"+y+"------"+z);
  }
}
```
10 8 16

$(10)_8 = (8)_{10}$

$0\times8 + 1\times8^1 = 8$

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

$0\times16^0 + 1\times16^1 = 16$

Refer all

Q) Which of the following declarations are valid.

✓ ① int x = 10;

✓ ② int x = 066;

X ③ int x = 0786; C.E:- integer number too large

✓ ④ int x = 0XFACE; 64206

X ⑤ int x = 0XBEER$ C.E:- (after Bee) ; Expected

✓ ⑥ int x = 0XBea; 3050

→ Bydefault Every integral literal is of int type but we can Specify explicitly as long type by suffixing with l or L.

Ex:-

✓ 1) int i = 10;

✗ 2) int i = 10l; C.E:- PLP
found : long
required : int

✓ 3) long l = 10l;

✓ 4) long l = 10;

→ There is no way to Specify integral literal is of byte & short types explicitly.

→ If we are assigning integral literal to the byte variable & that integral literal is with in the range of byte then it treats as byte literal automatically. Similarly short literal also.

Ex:- byte b = 10; ✓

byte b = 130; ✗ C.E:- PLP
found : int
Required : byte

floating point Literals:-

→ Every floating point literal is bydefault double type & hence we can't assign directly to float variable

→ But we can Specify explicitly floating point literal is the float type by suffixing with 'f' or 'F'.

Ex:- ✗ float f = 123.456; ∵ P.L.P
found : double
required : float

✓ float f = 123.456f;

✓ double d = 123.456;

→ We Can Specify floating point literal Explicitly as double type by Suffixing with d or D.

Eg: ✓ double d = 123.4567D;

✗ float f = 123.4567d; C.E:- PLP

found : double
required : float

→ We Can Specify floating point literal only in decimal form & we Can't Specify in Octal & Hexa decimal form.

Ex:-

✓ 1) double d = 123.456;

✓ 2) double d = 0123.456; o/p:- 123.456

✗ 3) double d = 0x123.456; C.E:- malformed floating point literal

Q) Which of the following floating point declarations are valid?

✗ 1) float f = 123.456;

✓ 2) double d = 0123.456;

✗ 3) double d = 0x123.456;

✓ 4) double d = 0xface; // 64206.0

✓ 5) float f = 0xBea; // 418.0

✓ 6) float f = 0642;

Because these 3 are not floating point so, that values are taking int type.

→ We Can assign integral literal directly to the floating point datatypes & that integral Literal Can be Specified either in decimal form or Octal form or hexa decimal form.

→ But we can't assign floating point literals directly to the integral types.

double

Eg:- ✗ int i = 123.456; PLP

found : double

required : int

✓ double d = 1.2e3;

S.o.println(d); 1200.0

→ we can specify floating point literal even in scientific form also [exponential form]

Ex:- ✓ 1) double d = 1.2e3;

S.o.println(d); 1200.0

✗ 2) float f = 1.2e3; C.E:- PLP

found : double

required : float

✓ 3) float f = 1.2e3f;

o/p:- 1200.0

Boolean Literals:-

→ The only possible values for the Boolean data types are true/false

Q) Which of the following Boolean declarations are valid:

✗ ① boolean b = 0; C.E:- Incompatible types

found : int

required : boolean

✗ ② boolean b = True; C.E:- Can't find Symbol

Symbol : variable True

✓ ③ boolen b = true;

✗ ④ boolen b = "true"; C.E:- Incompatible types

found : java.lang.String required : boolean

1) Ex:- int x=0;

```
if(x)
{
S.o.pln("Hello");
}
else
{
S.o.pln("Hi");
}
```

```
while(1)
{
S.o.pln("Hello");
}
```

C.E:- Incompatible types
found : int
required : boolean

Ex②:-

✗	✓	✓	✓

int x=10;
if(x==20)
{
S.o.pln("Hello");
}
else
{
S.o.pln("Hi");
}
	int x=10;
	if(x==20)
	{
	S.o.pln("Hello");
	}
	else
	{
	S.o.pln("Hi");
	}
		boolean b=true;
		if(b=false)
		{
		S.o.pln("Hello");
		}
		else
		{
		S.o.pln("Hi");
		}
			boolean b=true;
			if(b==true)
			{
			S.o.pln("Hello");
			}
			else
			{
			S.o.pln("Hi");
			}

C.E:- IT f : int R : boolean	O/P: Hi	O/P:- Hi	O/P:- Hello.

Char Literals :-

1) → A char literal can be represented as single character with in single codes

Ex:- ✓ `char ch = 'a';`

✗ `char ch = a;` C.E:- Can't find Symbol
Symbol : variable a
location : class xxxx

✗ `Chaa ch='ab';`
→ C.E:- unclosed character literal
C.E:- unclosed " "
C.E : not a statement

ప్రశ్న:

2) → A char Literal can be represented as integral Literal which represents unicode of that character.

→ we can specify integral literal either in decimal form or octal form or hexa decimal form. But allowed range 0 to 65535.

Ex:- ✓ 1) `char ch = 97;`
`S.o.p(ch);` a

✓ 2) `char ch = 65535;`
`S.o.pln(ch);`

✗ 3) `char ch = 65536;` C.E:- PLP
found : int
required : char

✓ 4) `char ch = 0XFACE;`

✓ 5) `char ch = 0642;`

3) → A char literal can be represented in unicode representation which is nothing but `\u xxxx` 4-digit hexa decimal no.

Eg:- ✓ 1) `char ch = '\u0061';`

`S.o.p(ch);` a

✗ 2) `char ch = '\uabcd';` → semicolon missing

✓ 3) `char ch = '\uface';`

✗ 4) `char ch = '\i beaf';`

4. → Every escape character is a char literal

Eg:- ✓ 1) `char ch = '\n';`

✓ 2) `char ch = '\t';`

✗ 3) `char ch = '\l';`

escape character	meaning
\n	new line
\t	horizontal tab
\r	Carriage Return
\b	Back space
\f	form feed
\'	Single Quads
\"	Double Quads
\\	Back slash

Q) Which of the following are valid char declarations.

✓ 1) char ch = 0xbeaf;

✗ 2) char ch = \ubeaf; because ' '

✗ 3) char ch = -10;

✗ 4) char ch = '\x';

✓ 5) char ch = 'a';

String Literals :-

→ Any Sequence of characters with in " " (double codes) is called String literal.

Ex:- String s = "java";

→ The following promotions will be performed automatically by the Compiler