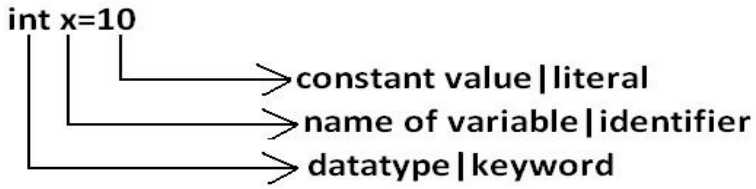
LITERALS

Agenda:

Literals ○ Integral Literals ○ Floating Point Literals ○ Boolean literals ○ Char literals ○ String literals

Literals:

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



Integral Literals:

For the integral data types (byte, short, int and long) we can specify literal value in the following ways.

1) Decimal literals: Allowed digits are 0 to 9.
Example: int x=10;

2) Octal literals: Allowed digits are 0 to 7. Literal value should be **prefixed with zero**.
Example: int x=**0**10;

3) Hexa Decimal literals:

- The allowed digits are 0 to 9, A to F.
- For the extra digits we can use both upper case and lower case characters.
- This is one of very few areas where java is not case sensitive. Literal value should be **prefixed with 0x(or)0X.**

Example: int x=**0x**10;

These are the only possible ways to specify integral literal.
Which of the following are valid declarations?

1. int x=0777; //(valid)
2. int x=0786; //C.E:integer number too large: 0786(invalid)
3. int x=**0x**FACE; (valid)
4. int x=**0x**beef; (valid)
5. int x=0xBeer; //C.E:';' expected(invalid) //:int x=0xBeer; ^// ^
6. int x=**0x**abb2cd;(valid)

Example:

```
int x=10;
int y=010;
int z=0x10;
System.out.println(x+"----"+y+"----"+z); //10----8----16
```

By default every integral literal is int type but we can specify explicitly as long type by **suffixing with small "l" (or) capital "L".**

Example:

```
int x=10;(valid)
long l=10L;(valid)
long l=10;(valid)
int x=10l;//C.E:possible loss of
precision(invalid)                found : long
required : int
```

There is no direct way to specify byte and short literals explicitly. But whenever we are assigning integral literal to the byte variables and its value within the range of byte compiler automatically treats as byte literal. Similarly short literal also.

Example:

```
byte b=127;(valid)
byte b=130;//C.E:possible loss of precision(invalid)
short s=32767;(valid) short s=32768;//C.E:possible loss
of precision(invalid)
```

Floating Point Literals:

Floating point literal is by default double type but we can specify explicitly as float type by **suffixing with f or F.**

Example:

```
float f=123.456;//C.E:possible loss of precision(invalid)
float f=123.456f;(valid) double d=123.456;(valid)
```

We can specify explicitly floating point literal as double type by suffixing with d or D.

Example:

```
double d=123.456D;
```

We can specify floating point literal only in decimal form and we can't specify in octal and hexadecimal forms.

Example:

```
double d=123.456;(valid)
double d=0123.456;(valid)  //it is treated as decimal
value  but not octal
double d=0x123.456;//C.E:malformed floating point
Literal(invalid)
```

Which of the following floating point declarations are valid?

1. float f=123.456; //C.E:possible loss of precision(invalid)
2. float f=123.456D; //C.E:possible loss of precision(invalid)
3. double d=0x123.456; //C.E:malformed floating point literal(invalid)
4. double d=0xFace; (valid)
5. double d=0xBeef; (valid)

We can assign integral literal directly to the floating point data types and that integral literal can be specified in decimal , octal and Hexa decimal form also.

Example:

```
double d=0xBeef;
System.out.println(d);//48879.0
```

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

Example:

```
int x=10.0;//C.E:possible loss of precision
```

We can specify floating point literal even in exponential form also(significant notation).

Example:

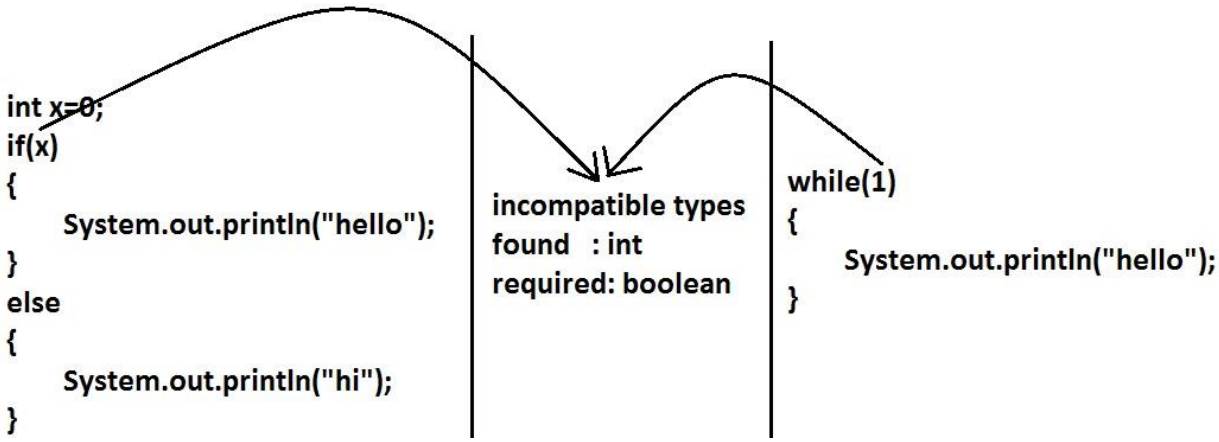
```
double d=10e2;//==>10*10^2(valid)
System.out.println(d);//1000.0
float f=10e2;//C.E:possible loss of precision(invalid)
float f=10e2F;(valid)
```

Boolean literals:

The only allowed values for the boolean type are true (or) false where case is important. i.e., lower case

Example:

1. boolean b=true;(valid)
2. boolean b=0;//C.E:incompatible types(invalid)
3. boolean b=True;//C.E:cannot find symbol(invalid)
4. boolean b="true";//C.E:incompatible types(invalid)



Char literals:

1) A char literal can be represented as single character within **single quotes.**

Example:

1. char ch='a';(valid)
2. char ch=a;//C.E:cannot find symbol(invalid)
3. char ch="a";//C.E:incompatible types(invalid)
4. char ch='ab';//C.E:unclosed character literal(invalid)

2) We can specify a char literal as integral literal which represents Unicode of that character. We can specify that integral literal either in decimal or octal or hexadecimal form but allowed values range is 0 to 65535.

Example:

1. char ch=97; (valid)
2. char ch=0xFace; (valid) System.out.println(ch); //?
3. char ch=65536; //C.E: possible loss of precision(invalid)

3) We can represent a char literal by Unicode representation which is nothing but **'\uxxxx'** (4 digit hexa-decimal number) .

Example:

1. char ch='\ubeef';
2. char ch1='\u0061';
   System.out.println(ch1); //a
3. char ch2=\u0062; //C.E:cannot find symbol
4. char ch3='\iface'; //C.E:illegal escape character 5. Every escape character in java acts as a char literal.

Example:

```
1) char ch='\n';   //(valid)
2) char ch='\l';   //C.E:illegal escape character(invalid)
```

Escape Character	Description
\n	New line
\t	Horizontal tab
\r	Carriage return
\f	Form feed
\b	Back space character
\'	Single quote
\"	Double quote
\\	Back space

Which of the following char declarations are valid?

1. char ch=a; //C.E:cannot find symbol(invalid)
2. char ch='ab'; //C.E:unclosed character literal(invalid)
3. char ch=65536; //C.E:possible loss of precision(invalid)
4. char ch=\uface; //C.E:illegal character: \64206(invalid)
5. char ch='/n'; //C.E:unclosed character literal(invalid)
6. none of the above. (valid) **String literals:**

Any sequence of characters with in double quotes is treated as String literal.

Example:
String s="Ashok"; (valid)