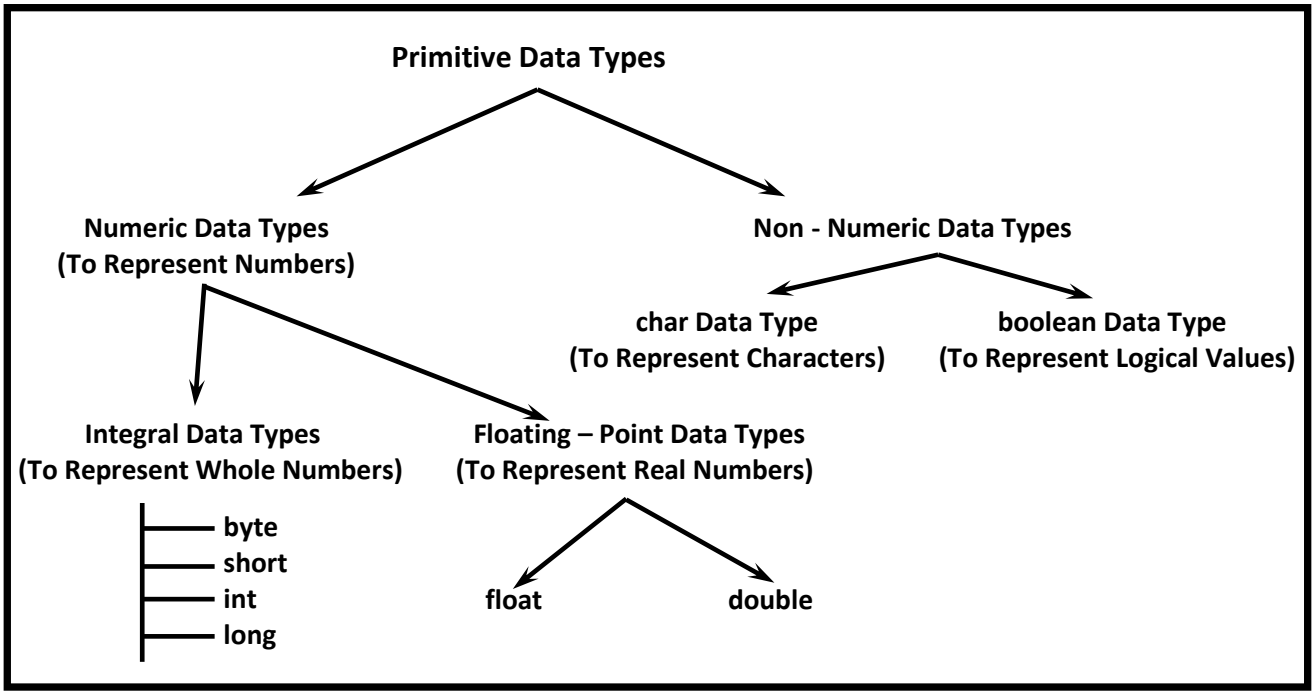
Data Types

Every variable has a type, every expression has a type and all types are strictly define more over every assignment should be checked by the compiler by the type compatibility hence java language is considered as strongly typed programming language.

Java is pure object oriented programming or not?

Java is not considered as pure object oriented programming language because several oops features (like multiple inheritance, operator overloading) are not supported by java. Moreover we are depending on primitive data types which are non objects.



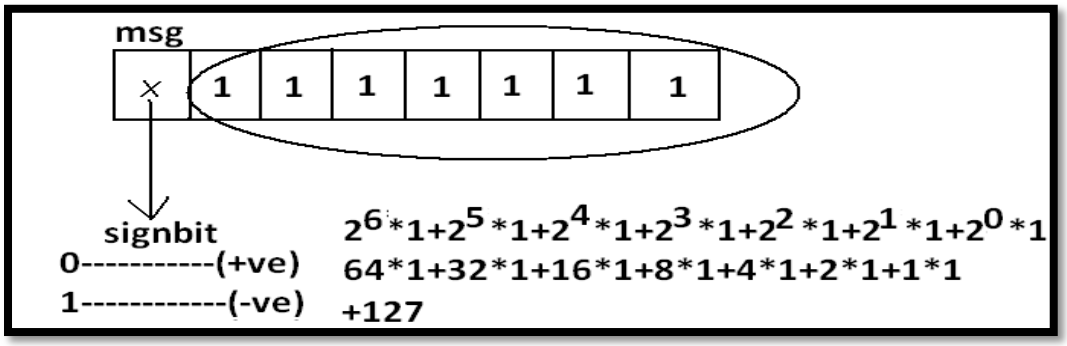
Except Boolean and char all remaining data types are considered as signed data types because we can represent both "+ve" and"-ve" numbers.

DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 040 – 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | www.durgasoft.com

Integral Data Types:

byte Data Type:

Size: 1byte (8bits)
Maxvalue: +127
Minvalue:-128
Range:-128to 127[-27 to 27-1]



The most significant bit acts as sign bit. "0" means "+ve" number and "1" means "–ve" number. "+ve" numbers will be represented directly in the memory whereas "–ve" numbers will be represented in 2's complement form.

Example:

```
byte b=10;
byte b2=130;//C.E:possible loss of precision
              found : int
              required : byte
byte b=10.5;//C.E:possible loss of precision
byte b=true;//C.E:incompatible types
byte b="ashok";//C.E:incompatible types
                found : java.lang.String
                required : byte
```

Note:
byte data type is best suitable if we are handling data in terms of streams either from the file or from the network.

DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 040 – 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | www.durgasoft.com

short Data Type:

The most rarely used data type in java is short.

Size: 2 bytes
Range: s-32768 to 32767(-215 to 215-1)

Example:

```
short s=130;
short s=32768;//C.E:possible loss of precision
short s=true;//C.E:incompatible types
```

Note:
short data type is best suitable for 16 bit processors like 8086 but these processors are completely outdated and hence the corresponding short data type is also out data type.

int Data Type:

This is most commonly used data type in java.
Size: 4 bytes
Range:-2147483648 to 2147483647 (-231 to 231-1)

Example:

```
int i=130;
int i=10.5;//C.E:possible loss of precision
int i=true;//C.E:incompatible types
```

long Data Type:

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

Eg 1:

To hold the distance travelled by light in 1000 days , int may not enough, compulsory we should go for long data type.

```
long l = 186000*60*24*1000 miles
```

Eg 2:

To hold the number of characters present in a big file, int may not enough, compulsory we should go for long data type. Hence the return type of length() method is long.

```
long l=f.length();//f is a file
```
Size: 8 bytes

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

Note: All the above data types (byte, short, int and long) can be used to represent whole numbers. If we want to represent real numbers then we should go for floating point data types.

Floating Point Data types:

Float	double
If we want to 5 to 6 decimal places of accuracy then we should go for float.	If we want to 14 to 15 decimal places of accuracy then we should go for double.
Size:4 bytes.	Size:8 bytes.
Range:-3.4e38 to 3.4e38.	-1.7e308 to1.7e308.
float follows single precision.	double follows double precision.

boolean data type:

Size: Not applicable (virtual machine dependent)
Range: Not applicable but allowed values are true or false.

Q. Which of the following boolean declarations are valid?

```
boolean b=true;
boolean b=True;//C.E:cannot find symbol
boolean b="True";//C.E:incompatible types
boolean b=0;//C.E:incompatible types
```

Example 2:

```
int x=0;
if(x)
{
    System.out.println("hello");
}
else
{
    System.out.println("hi");
}
```

```
incompatible types
found   : int
required: boolean
```

```
while(1)
{
    System.out.println("hello");
}
```

char data type:

Old languages like C & C++ are ASCII code based and the number of ASCII characters are < 256. To represent these 256 characters, 8 – bits are enough and hence char size in old languages 1 byte.

But, in java we are allowed to use worldwide any alphabet character and java is Unicode based. The number of unicode characters are > 256 and <= 65536. To represent all these characters one byte is not enough compulsory we should go for 2 bytes.
Size: 2 bytes
Range: 0 to 65535

Example:

```
char ch1=97;
char ch2=65536;//C.E:possible loss of precision
```

Summary of java primitive data type:

data type	Size	Range	Corresponding Wrapper class	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
boolean	Not applicable	Not applicable(but allowed values true|false)	Boolean	false
char	2 bytes	0 to 65535	Character	0(represents blank space)

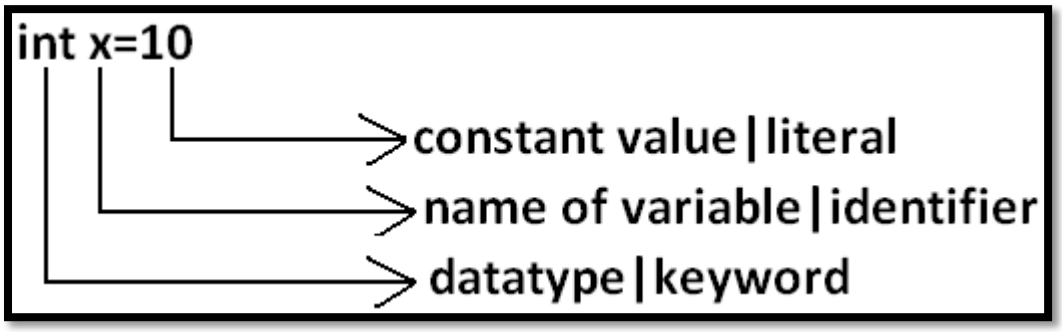
Note:

The default value for the object references is "null".

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(base -10):

Allowed digits are 0 to 9.
Example: int x=10;

2) Octal literals(base-8):

Allowed digits are 0 to 7. Literal value should be prefixed with zero.
Example: int x=010;

3) Hexa Decimal literals(base-16):

The allowed digits are 0 to 9, A to Z.
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 ox(or)oX.
Example: int x=0x10;

These are the only possible ways to specify integral literal.

DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 040 – 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | www.durgasoft.com

Which of the following are valid declarations?

int x=0777; //(valid)
int x=0786; //C.E:integer number too large: 0786(invalid)
int x=0xFACE; (valid)
int x=0xbeef; (valid)
int x=0xBeer; //C.E:';' expected(invalid) //:int x=0xBeer; ^// ^
int x=0xabb2cd;(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 of 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?
float f=123.456; //C.E:possible loss of precision(invalid)
float f=123.456D; //C.E:possible loss of precision(invalid)
double d=0x123.456; //C.E:malformed floating point literal(invalid)
double d=0xFace; (valid)
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

float f = 100f:
System.out.println(f); //100.0
```

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

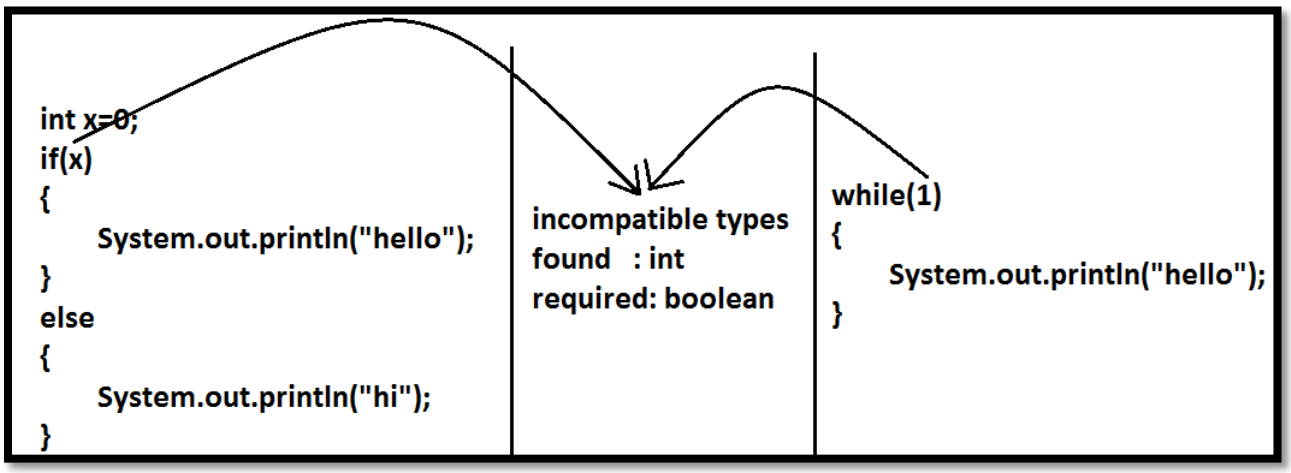
float f=10e2;//C.E:possible loss of precision(invalid)
float f=10e2F;(valid)

Boolean literals:

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

Example:

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



Char literals:

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

Example:

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

DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 040 – 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | www.durgasoft.com

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:

```
char ch=97; (valid)
char ch=0xFace; (valid)
System.out.println(ch); //?
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:

```
char ch='\ubeef';
char ch1='\u0061';
System.out.println(ch1); //a
char ch2=\u0062; //C.E:cannot find symbol
char ch3='\iface'; //C.E:illegal escape character
```

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 slash

Which of the following char declarations are valid?

char ch=a; //C.E:cannot find symbol(invalid)
char ch='ab'; //C.E:unclosed character literal(invalid)
char ch=65536; //C.E:possible loss of precision(invalid)
char ch=\uface; //C.E:illegal character: \64206(invalid)
char ch='/n'; //C.E:unclosed character literal(invalid)
none of the above. (valid)

String literals:

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

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

1.7 Version enhancements with respect to Literals:

The following 2 are enhancements

- Binary Literals
- Usage of '_' in Numeric Literals

Binary Literals:

For the integral data types until 1.6v we can specified literal value in the following ways
Decimal
Octal
Hexa decimal
But from 1.7v onwards we can specify literal value in binary form also.
The allowed digits are 0 to 1.
Literal value should be prefixed with Ob or OB .

```
int x = 0b111;
System.out.println(x);  // 7
```

Usage of _ symbol in numeric literals:

From 1.7v onwards we can use underscore(_) symbol in numeric literals.

```
double d = 123456.789;  //valid

double d = 1_23_456.7_8_9;  //valid
double d = 123_456.7_8_9;   //valid
```


DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 040 – 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | www.durgasoft.com

The main advantage of this approach is readability of the code will be improved At the time of compilation ' _ ' symbols will be removed automatically , hence after compilation the above lines will become double d = 123456.789
We can use more than one underscore symbol also between the digits.

Ex : double d = 1_23_ _456.789;
We should use underscore symbol only between the digits
double d=_1_23_456.7_8_9; //invalid
double d=1_23_456.7_8_9_; //invalid
double d=1_23_456_.7_8_9; //invalid

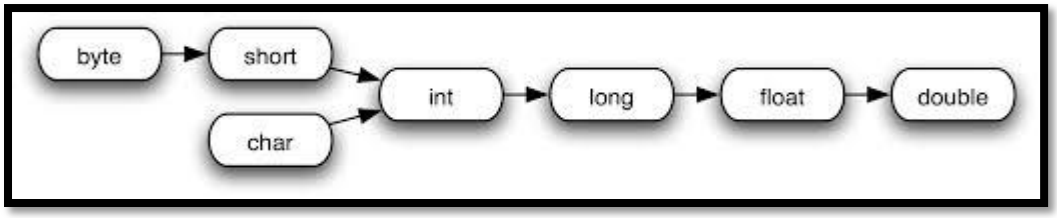
Primitive Type Casting

There are 2 types of type-casting
Implicit Type Casting
Explicit Type Casting

implicit Type Casting :

```
int x='a';
System.out.println(x);  //97
```

The compiler is responsible to perform this type casting.
Whenever we are assigning lower datatype value to higher datatype variable then implicit type cast will be performed .
It is also known as Widening or Upcasting.
There is no lose of information in this type casting.
The following are various possible implicit type casting.



Example 1:

```
int x='a';
System.out.println(x);//97
```

Note: Compiler converts char to int type automatically by implicit type casting.

Example 2:

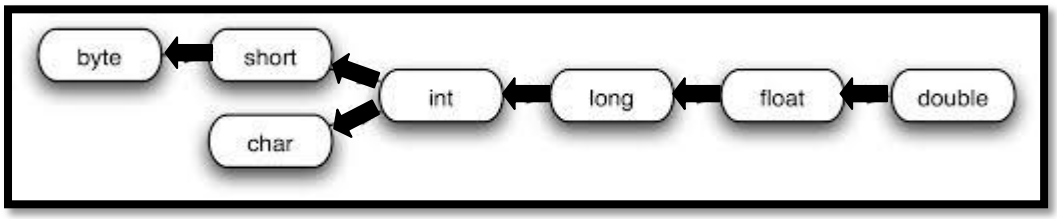
```
double d=10;
System.out.println(d);//10.0
```

Note: Compiler converts int to double type automatically by implicit type casting.

Explicit type casting:

1) Programmer is responsible for this type casting.
2) Whenever we are assigning bigger data type value to the smaller data type variable then explicit type casting is required.
3) Also known as Narrowing or down casting.
4) There may be a chance of lose of information in this type casting.

The following are various possible conversions where explicit type casting is required.



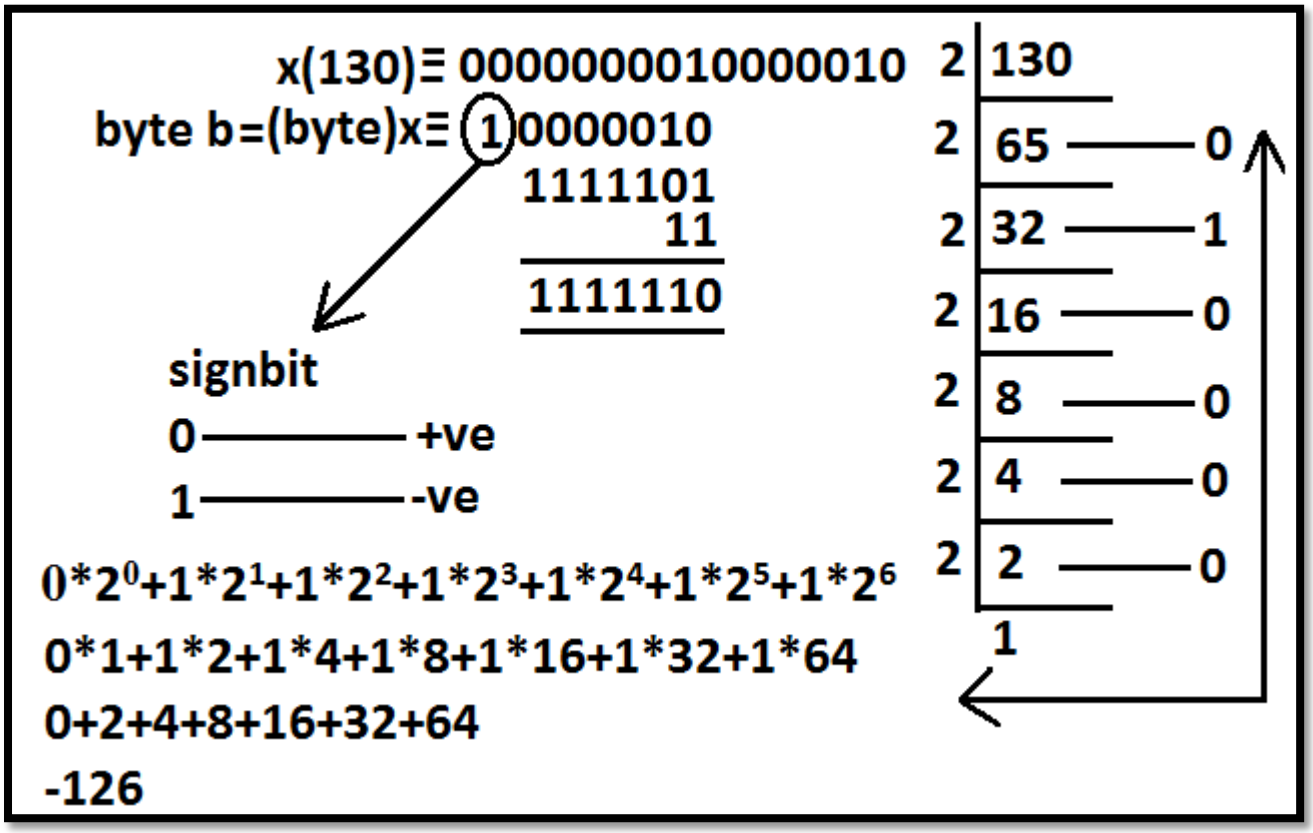
Left → Right → Implicit Type Casting
Right → Left → Explicit Type Casting

```
int x=130;
byte b=x;
```

```
E:\scjp>javac OperatorsDemo.java
OperatorsDemo.java:6: possible loss of precision
found   : int
required: byte
                byte b=x;
```

Example:

```
int x=130;
byte b=(byte)x;
System.out.println(b);  //-126
```



Example 2:

```
int x=130;
byte b=x;
System.out.println(b);  //CE : possible loss of precision
```

Whenever we are assigning higher datatype value to lower datatype value variable by explicit type casting ,the most significant bits will be lost i.e., we have considered least significant bits .

Example 3 :

```
int x=150;
short s=(short)x;
byte b=(byte)x;
System.out.println(s); //150
System.out.println(b);  //-106
```

Whenever we are assigning floating point value to the integral types by explicit type casting, the digits of after decimal point will be lost .

Example 4:

```
double d=130.456 ;

int x=(int)d ;
System.out.println(x);  //130

byte b=(byte)d ;
System.out.println(b);  //-206
```

```
float x=150.1234f;
int i=(int)x;
System.out.println(i);//150
```

```
double d=130.456;
int i=(int)d;
System.out.println(i);//130
```

Q1. Which of the following conversions will be performed automatically in Java?

A) int to byte
B) byte to int
C) float to double
D) double to float
E) None of the above

Answer: B, C

Q2. In which of the following cases explicit Type casting is required ?

A) int to byte
B) byte to int
C) float to double
D) double to float
E) None of the above

Answer: A, D

Q3. Consider the code

```
int i =100;
float f = 100.100f;
double d = 123;
```

Which of the following assignments won't compile?
A) i=f;

B) f=i;
C) d=f;
D) f=d;
E) d=i;
F) i=d;

Answer: A,D,F

Q4. In which of the following cases we will get Compile time error?

A) float f =100F;
B) float f =(float)1_11.00;
C) float f =100;
D) double d = 203.22;
  float f = d;
E) int i =100;
  float f=(float)i;

Answer: D

Q5. Consider the code

```
1)  public class Test
2)  {
3)     public static void main(String[] args)
4)     {
5)        int a=10;
6)        float b=10.25f;
7)        double c=100;
8)        a = b;
9)        b = a;
10)       c = b;
11)       c = a;
12)    }
13) }
```

Which change will enable the code fragment to compile successfully?

A) Replace a = b; with a=(int)b;
B) Replace b = a; with b=(float)a;
C) Replace c = b; with c=(double)b;
D) Replace c = a; with c=(double)a;

Answer: A