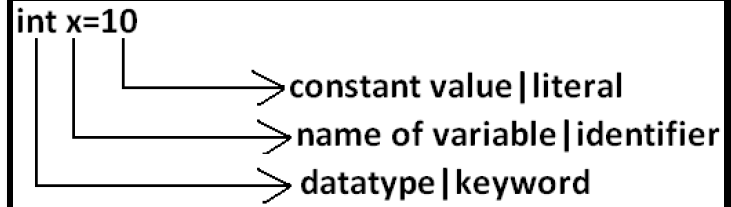
# Literals:

**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.**

**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;//Compilation Error: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 is within the range of byte, then compiler automatically treats as byte literal. Similarly short literal also.**

**Example:**

**byte b=127;(valid)**

**byte b=130;//Compilation Error:possible loss of precision(invalid)**

**short s=32767;(valid)**

**short s=32768;//Compilaton Error: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;//Compilation Error: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;//COMPILATION ERROR: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)**

**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\*102(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.**

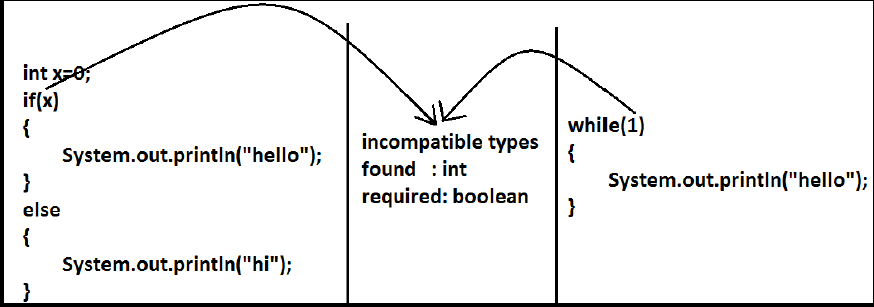
**Example:**

**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:**

**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)**

**Example:**

**char ch=97; (valid)**

**char ch=0xFace; (valid)**

**System.out.println(ch); //?**

**char ch=65536; //C.E: possible loss of precision(invalid)**

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

**Example:**

**char ch1='\u0061';**

**System.out.println(ch1); //a**

**char ch2=\u0062; //C.E:cannot find symbol**

**String literals:**

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

**Example: String s="Himanshu"; (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 1.6version we can specified literal value in the following ways:**

* **Decimal**
* **Octal**
* **Hexa decimal**

**But from 1.7version 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 1.7version 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**

**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**

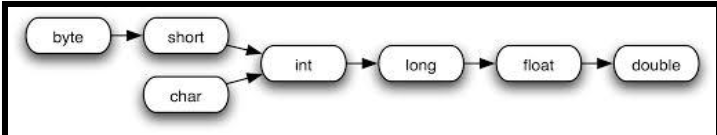
* **Implicit Type Casting**
* **Explicit Type Casting**

**implicit Type Casting :**

**int x='a';**

**System.out.println(x); //97**

* **The compiler is responsible to perform this type of 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:**

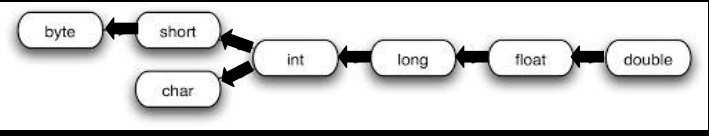
**1) Programmer is responsible for this type casting.**

**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.**

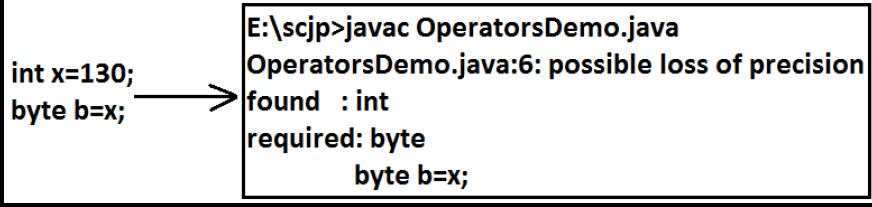
**4) There may be a chance of lose of information in this type casting.**

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



**Left**  **Right**  **Implicit Type Casting**

**Right**  **Left**  **Explicit Type Casting**



**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**

**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 after decimal point will be lost .**

**Example 4:**

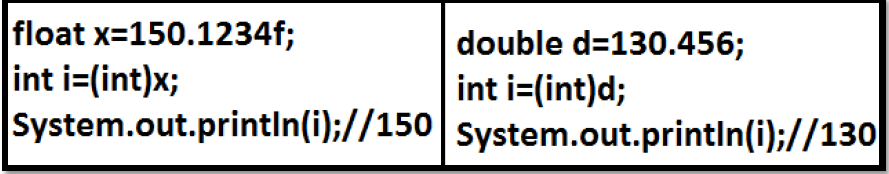
**double d=130.456 ;**

**int x=(int)d ;**

**System.out.println(x); //130**

**byte b=(byte)d ;**

**System.out.println(b); //-126**



**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?**

1. **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**