Integral Literal :

* If any numbeic literal (constant) does not contain **decimal OR Fraction** then it is called Integral Literal.
  Example : 12, 98, 34, 78, 90

* In integral literal we have total 4 data types :
  a) byte (8 bits)
  b) short (16 bits)
  c) int (32 bits)
  d) long (64 bits)

* An Integral literal we can represent in 4 different forms :
  a) Decimal Literal (Base is 10, 0 to 9)
  b) Octal Literal (Base is 8, 0 to 7)
  c) Hexadecimal Literal (Base is 16, 0 to 9 & A - F)
  d) Binary Literal (Base is 2, 0 and 1) [It is available from JDK 1.7V]

Decimal Literal :

By default our numeric literals are decimal literal. Here base is 10 so, It accepts 10 digits i.e. from 0-9.

Example :
int a = 123;
int b = 456;
int c = 223;
int d = 234;

Octal Literal :

If any integer literal starts with 0 (Zero) then it will become octal literal. Here base is 8 so it will accept 8 digits i.e 0 to 7.

Example :
int a = 0118; //Invalid because it contains digit 8 which is out of range
int b = 0777; //Valid
int c = 0123; //Valid

Hexadecimal Literal :

If any integer literal starts with 0X or 0x (Zero capital X Or 0 small x) then it is hexadecimal literal. Here base is 16 so it will accept 16 digits i.e 0 to 9 and A to F OR (a to f)

Example :

int a = 0X12; //Valid
int b = 0xFAC; //Valid
int c = 0XBack; //valid
int e = 0Xagc; //Invalid because character 'g' out of range

Binary Literal :

If an Integral literal starts with 0B (Zero capital B) or 0b (Zero small b) then it will become Binary literal. Binary literal is available from JDK 1.7v.
Here base is 2 so it will accept 2 digits i.e 0 and 1.

Example :

int a = 0B101; //valid
int y = 0b111; //Valid
int c = 0B112; //Invalid [2 is out of range]

* As we know the default literal is **decimal literal** so as a programmer we can represent our integral literal in decimal, octal, hexadecimal and Binary **but is order to get the output JVM will internally convert into decimal format only.**

Conversion :

$5 \times 8^1 + 2 \times 8^0 = 42$

public class OctalLiteral
{
public static void main(String[] args)
{
int a = 015;
System.out.println(a);
}
}

015 -> Here 0 describes that It is an octal literal.

$1 \times 8^1 + 5 \times 8^0$

$1 \times 8 + 5$

$8 + 5 = 13$

//Hexadecimal Example :

public class HexadecimalDemo
{
public static void main(String[] args)
{
int a = 0Xb40;
System.out.println(a);
}
}

0Xb40 -> Here 0X represents It is a hexadecimal number.

A = 10
B = 11
C = 12
D = 13
E = 14
F = 15

$a \times 16^2 + 0 \times 16^1 + 4 \times 16^0$

$10 \times 256 + 10 \times 16 + 10 \times 1$

$\Rightarrow 2560 + 160 + 13$
$\Rightarrow 2781$

//Example of Binary

public class BinaryLiteral
{
public static void main(String[] args)
{
int a = 0B101;
System.out.println(a);
}
}

How we are representing integral literal in memory :

byte a = 5; -> 00000101 [8 bits representation]

short b = 5; -> 0000000000000101 [16 bits representation]

int c = 5; -> 00000000000000000000000000000101 [32 bits representation]

long d = 5; -> 0000000000000000000000000000000000000000000000000000000000000101
[64 bits representation]

By default every integral literal is of type int only. byte and short are below than int so we can assign integral literal(which is by default int type) to byte/short but the values must be within the range. [for Byte -128 to 127 and for short -32768 to 32767]

Actually whenever we are assigning integral literal to byte and short data type then compiler internally converts into corresponding type.

byte b = (byte) 12; //Compiler is converting int to byte]
short s = (short) 12; //Compiler is converting int to short]

In order to represent long value explicitly we should use either L OR l [Capital L OR Small l] as a suffix to integral literal.

According to IT industry, we should use L because l looks like digit 1.

Different Case :

Case 1 :
byte b = 12; //[Compiler will convert int to byte]

Case 2:
short s = 25; //[Compiler will convert int to byte]

Case 3 :
byte b = 128; //[Compilation error 128 is out of the range of byte]

Case 4 :
short s = 32768; //[Compilation error 32768 is out of the range of short]

Case 5 :
int i = 12; //int to int [32 bits to 32 bits]

Case 5 :
long y = 90; // int to long [32 bit to 64 bits]

Case 6 :
long mob = 9812345678; //error [Integer number too large, It is out of the range of int]

Case 7 :
long mob = 9812345678L; //long to long

Case 8 :
public static void main(String[] args)
{
accept(5);
}

public static void accept(byte b)
{
}