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 and 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 convert into corresponding type.
Example :-
byte B = (byte) 12; //compiler is converting int to byte
short s = (short) 32; //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.

Test4.java

```
/* By default every integral literal is of type int only*/

void main()
{
        byte b = 12B; //error
        IO.println(b);

        short s = 32768L //error
        IO.println(s);
}
```

Test5.java

```
//Assigning smaller data type value to bigger data type
void main()
{
    byte a = 12;
    short b = a;  //Automatic OR WIDENING type casting
    IO.println(b);

    IO.println("  ,,,,");

    short c = 190;
    int d = c;    //Automatic OR WIDENING type casting
    IO.println(d);

}
```

Test6.java

```
//Converting bigger type to smaller type

void main()
{
    short a = 127;
    byte B = (byte) a;//Explicit OR Narrowing type casting
    IO.println(B);  //127

    short z1 = 128;
    byte b1 = (byte) z1;  //Explicit OR Narrowing type casting
    IO.println(b1);  //-128
}
```

Test7.java

```
void main()
{
        byte x = (byte) 18L;
        IO.println("x value = "+x);

        long l = 29L;
        IO.println("l value = "+l);

        int y = (int) 1L;
        IO.println("y value = "+y);
}
```

Is java pure object oriented language or not ?

NO, Java is not a pure object oriented language. Actually any language, if except primitive data type like byte, short, int, long, float, double, char and boolean then that language is not a pure object oriented language so Java is also not a pure object oriented language.

Except these 8 primitive data types, everything in **java is Object that means** due to these 8 primitive data types, Java is not a pure oblac oriented langue. A pure object oriented language like **Ruby and smalltalk are accepting everything in the form of object.**

The basic problem with primitive data types is, they cannot move in the network. Only Objects are moving in the network.

Java software people has provided a concept called "Wrapper classes" through which the primitive data type can be converted into Wrapper object.

Primary Data type		Wrapper Object
byte	:	Byte
short	:	Short
int	:	Integer
long	:	Long
float	:	Float
double	:	Double
char	:	Character
boolean	:	Boolean

Test8.java

```
//Wrapper classes
void main()
{
        Integer a = 26;
        Integer a = 26;
        int b = a + 5;
        IO.println("The sum is "+b);

        Boolean D = true;
        IO.println(D);

        Double d = 90.90;
        IO.println(d);

        Character c = 'K';
        IO.println(c);
}
```

How to find out the minimum, maximum value as well as size of different data types :

The Wrapper classes like Byte, Short, Integer and Long has provided predefined static and final fields to represent minimum value, maximum value as well as size of the respective data type.

Example :

If we want to get the minimum value, maximum value as well as size of byte data type then Byte class (Wrapper class) has provided the following final and static variables :

a) SIZE b) MIN_VALUE c) MAX_VALUE

public static final int SIZE;

public static final int MIN_VALUE;

public static final int MAX_VALUE;

Byte.MIN_VALUE : -128

Byte.MAX_VALUE : 127

Byte.SIZE : 8 (bits format)

```
//Program to find out the range and size of Integral Data type

void main()
{
        IO.println("\n Byte range:");
        IO.println(" min: " + BYTE.MIN_VALUE);
        IO.println(" max: " + Byte.MAX_VALUE);
        IO.println(" size: " + Byte.SIZE);

        IO.println("\n Short range:");
        IO.println(" min: " + Short.MIN_VALUE);
        IO.println(" max: " + Short.MAX_VALUE);
        IO.println(" size: "+Short.SIZE);

        IO.println("\n Integer range:");
        IO.println(" min: " + Integer.MIN_VALUE);
        IO.println(" max: " + Integer.MAX_VALUE);
        IO.println(" size: "+Integer.SIZE);

        IO.println("\n Long range:");
        IO.println(" min: " + Long.MIN_VALUE);
        IO.println(" max: " + Long.MAX_VALUE);
        IO.println(" size: "+Long.SIZE);
}
```

Providing _ (underscore) in integral Literal :-

In Order to enhance the readability of large numeric literals, Java software people has provided _ from JDK 1.7V. While writing the big numbers to separate the numbers we can use _

We can't start or end an integral literal with _, we will get compilation error.

//We can provide _ in integral literal [JDK 1.7]

```
void main()
{
        long mobile = 98_1234_5678L;
        IO.println("Mobile Number is "+mobile);
}
```