What is **Widening** OR Automatic type casting in Java ?

* Assigning smaller data type value to bigger data type value is called Widening OR Automatic type casting.
* It is called automatic type casting because compiler will automatically convert smaller type to bigger type.

Widening.java

```
void main()
{
byte b = 12;
short s = b; //Automatic Type Casting OR WIDENING
IO.println(b);
}
```

What is narrowing OR explicit type casting in Java ?

* If we try to assign a bigger data type value to smaller data type then first of all we will get compilation error as shown below :

Explicit.java

```
void main()
{
short s = 8;
byte b = s; //error
}
```

Cast.java

```
void main()
{
short s = 12;
byte b = (byte)s;
IO.println(b);
}
```

While working with byte OR short when the **values** are out of the range then how to calculate the output :

byte range : -128 to 127 [Total Digits are 256]
short range : -32768 to 32767 [Total digits are 65536]

Working with byte :

Case 1 : If the value is within the range -128 to 127 then same output will be generated

Case 2 : If the value is **> 127 then subtract 256 from the given number till the value is not coming within the range**

Case 3 : If the value is < -128 then **add 256** in the given number **till the value is not coming within the range**

130 - 256 = -126

```
void main()
{
int c = 130;
byte b = (byte) c;
IO.println(b);
}
```

150 - 256 = -106

```
void main()
{
int c = 150;
byte b = (byte) c;
IO.println(b);
}
```

-112

```
void main()
{
int c = -112;
byte b = (byte) c;
IO.println(b);
}
```

121

```
void main()
{
int c = 121;
byte b = (byte) c;
IO.println(b);
}
```

608 - 256 = 344
344 - 256 = 88

```
void main()
{
int c = 608;
byte b = (byte) c;
IO.println(b);
}
```

Short type calculation :

Case 1 : If the value is within the range -32768 to 32767 then same output will be generated

Case 2 : If the value is > 32767 then subtract 65536 from the given number till the value is not coming within the range

Case 3 : If the value is < -32768 then add 65536 in the given number till the value is not coming within the range

```
void main()
{
int c = 32800;
short s = (short) c; //-32736
IO.println(s);
}
```

Cast.java

```
void main()
{
    byte a = (byte) 38;
    IO.println("a value = "+a);

    long l = 28L;
    IO.println("l value = "+l);

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

** Is Java pure object oriented language or not ?

* No, Java is not a pure object oriented language because it accepts primitive data type like byte, short, int, long, float, double, char and boolean.

* Except these **8** primitive data types, everything in Java is an object.

* Primitive data types **cannot move in the network, only objects are moving in the network.**

* Java has provided a concept called **Wrapper classes through which we can convert primitive data type into corresponding wrapper object.**

* From JDK 1.5V, Java has provided the following two concepts

a) Autoboxing [Converting the primitive data type into Wrapper object]
b) Unboxing [Converting the Wrapper object into Primitive type]

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

Wrapper.java

```
void main();
    Integer a = 24;
    Integer b = 36;
    Integer z = a + b;
    IO.println("The sum is "+z);

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

    Double d = 98.98;
    IO.println(d);

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

How to find out the size, minimum value & maximum value of Integeral literal data type :

The Wrapper classes(Byte) in 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) BYTE b) MIN_VALUE c) MAX_VALUE

public static final int BYTE;

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 Integeral 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);

}
```

Prevoding _ (underscore) in integeral 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_9876L;
    IO.println("Mobile Number is "+mobile);
}

void main()
{
    final int a = 127;
    byte b = a;
    IO.println(b);
}
```

How to convert **decimal number** to Octal, Hexadecimal and Binary :

Integer class has provided the following **static** methods to convert decimal to octal, hexadecimal and binary :

1) public static String toBinaryString(int decimal) : Will convert the decimal into binary in String format.

2) public static String toOctalString(int decimal) : Will convert the decimal into octal in String format.

3) public static String toHexadecimalString(int decimal) : Will convert the decimal into hexadecimal in String format.

Note : All the above static methods are available in Integer class.

```
void main()
{
    IO.println(Integer.toOctalString(121));//171
    IO.println(Integer.toHexadecimalString(2785));//ae0
    IO.println(Integer.toBinaryString(121));//1111
}
```

var keyword in Java :

* var keyword was introduced from JDK 10V

* var keyword we can only use as a **local variable.**

* While using var keyword, initialization is compulsory in the **same line.**

* **By using var keyword we can initialize any type of data.**

```
void main()
{
    var x = 90;
    IO.println(x);

    var y = true;
    IO.println(y);
}
```