MOD -01 -> Java Fundamentals

Java : Getting Started

Relational operators

Operator	Description
==	Check if two operand are equal
!=	Check if two operand are not equal.
>	Check if operand on the left is greater than operand on the right
<	Check operand on the left is smaller than right operand
>=	check left operand is greater than or equal to right operand
<=	Check if operand on left is smaller than or equal to right operand

```
class RelationalDemo{
    public static void main(String[] args) {

        int numOne=7;
        int numTwo=20;

        System.out.println("numOne is " + numOne + " and numTwo is " + numTwo);

        System.out.println("numOne == numTwo -> "+(numOne == numTwo));  // false

        System.out.println("numOne != numTwo -> "+(numOne != numTwo));  // true

        System.out.println("numOne > numTwo -> "+(numOne > numTwo));  // false

        System.out.println("numOne < numTwo -> "+(numOne < numTwo));  // true

        System.out.println("numOne >= numTwo -> "+(numOne >= numTwo));  // false

        System.out.println("numOne <= numTwo -> "+(numOne <= numTwo));  // true
  }

}
```

```
H:\Luminar\corejava>javac RelationalDemo.java

H:\Luminar\corejava>java RelationalDemo
numOne is 7 and numTwo is 20
numOne == numTwo -> false
numOne != numTwo -> true
numOne > numTwo -> false
numOne < numTwo -> true
numOne >= numTwo -> false
numOne <= numTwo -> true

H:\Luminar\corejava>
```

Logical operators

Java supports following 3 logical operators. Suppose a=1 and b=0;

Operator	Description	Example
&&	Logical AND	(a && b) is false
||	Logical OR	(a || b) is true
!	Logical NOT	(!a) is false

```
class LogicalDemo {
	public static void main(String[] args) {
		int numOne=70;
		int numTwo=20;
		int numThree=15;
		System.out.println((numOne > numTwo) && (numOne > numThree));  // true
		//System.out.println((70 > 20) && (70 > 15));  // true

		System.out.println((numOne > numTwo) || (numThree > numTwo));  // true

		System.out.println((numOne < numTwo) || (numTwo > numThree));  // true

		System.out.println(!(numOne == numTwo));  // true
		System.out.println(!(numTwo > numThree));  // false
	}
}
```

```
H:\Luminar\corejava>javac LogicalDemo.java

H:\Luminar\corejava>java LogicalDemo
true
true
true
true
false

H:\Luminar\corejava>
```

Bitwise operators

Java defines several bitwise operators that can be applied to the integer types long, int, short, etc

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
>>	left shift
<<	right shift

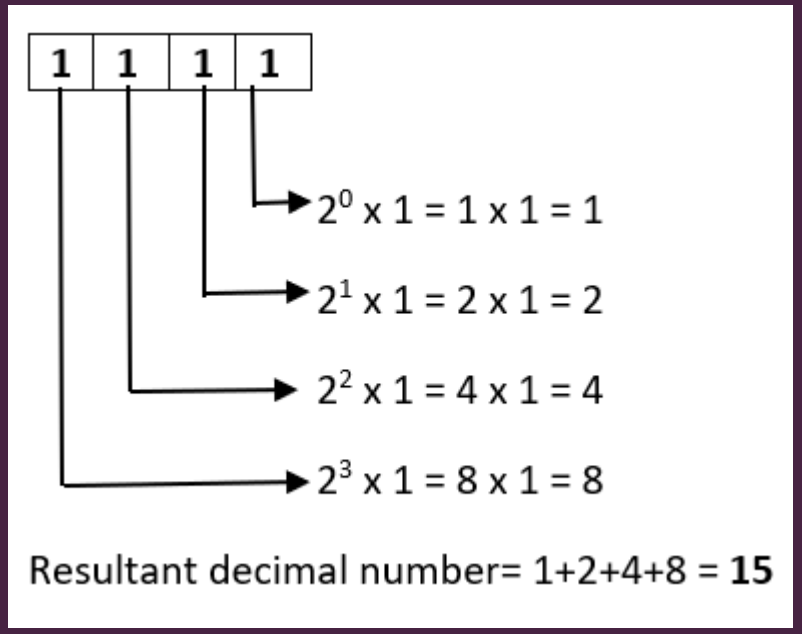
How to Convert from Binary to Decimal?

Binary numbers are numbers that are understandable by computer machines. It is in a combination of 0’s & 1’s.

Binary to Decimal Formula

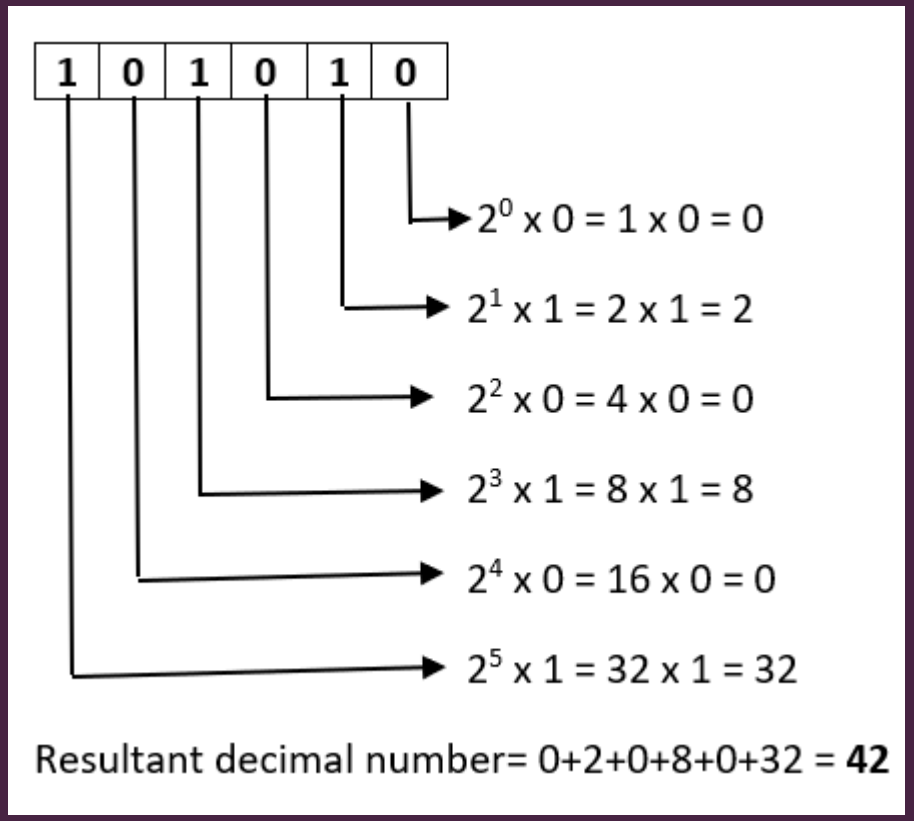
multiplication operation on each digit of a binary number from right to left with powers of 2 starting from 0 and add each result to get the decimal number of it.

binary number 1111. We need to convert this binary number to a decimal number.



Represent any binary number with this format (xxxx)2 and decimal in (xxxx)10 format.

Convert $(101010)_2$->$(?)_{10}$



1. Java Bitwise AND Operator

The bitwise AND & operator returns 1 if and only if both the operands are 1. Otherwise, it returns 0. The following table demonstrates the working of the bitwise AND operator.

Let a and b be two operands that can only take binary values i.e. 1 and 0

a	b	a & b
0	0	0
0	1	0
1	0	0
1	1	1

bitwise AND operation of two integers 12 and 25.

12 = 00001100 (In Binary)
25 = 00011001 (In Binary)

```
// Bitwise AND Operation of 12 and 25
   00001100
 & 00011001
 ____________
   00001000 = 8 (In Decimal)
```

2. Java Bitwise OR Operator

The bitwise OR | operator returns 1 if at least one of the operands is 1. Otherwise, it returns 0.

The following truth table demonstrates the working of the bitwise OR operator. Let a and b be two operands that can only take binary values i.e. 1 or 0.

a	b	a | b
0	0	0
0	1	1
1	0	1
1	1	1

bitwise OR operation of two integers 12 and 25.

12 = 00001100 (In Binary)
25 = 00011001 (In Binary)

Bitwise OR Operation of 12 and 25

```
   00001100
 | 00011001
 ____________
   00011101 = 29 (In Decimal)
```

3. Java Bitwise XOR Operator

The bitwise XOR ^ operator returns 1 if and only if one of the operands is 1. However, if both the operands are 0 or if both are 1, then the result is 0.

The following truth table demonstrates the working of the bitwise XOR operator. Let a and b be two operands that can only take binary values i.e. 1 or 0.

a	b	a ^ b
0	0	0
0	1	1
1	0	1
1	1	0

bitwise XOR operation of two integers 12 and 25.

12 = 00001100 (In Binary)
25 = 00011001 (In Binary)

```
// Bitwise XOR Operation of 12 and 25
  00001100
^ 00011001
_____________
  00010101 = 21 (In Decimal)
```

```
class Bitwise{
	public static void main(String[] args) {

		int numOne = 12;
		int numTwo = 25;

		// bitwise AND between 12 and 25

		System.out.println(numOne & numTwo);	// prints 8

		// bitwise OR between 12 and 25

		System.out.println(numOne | numTwo);	// prints 29

	// bitwise XOR between 12 and 25

		System.out.println(numOne ^ numTwo);	// prints 21
	}

}
```

```
H:\Luminar\corejava>javac Bitwise.java

H:\Luminar\corejava>java Bitwise
8
29
21

H:\Luminar\corejava>
```

4. Shift Operators

Right Shift (>>):
Bits move to the right, and new bits come in from the left.
(It’s like removing bits from the right side.)

Left Shift (<<):
Bits move to the left, and new bits come in from the right.
(It’s like adding zeroes on the right.)

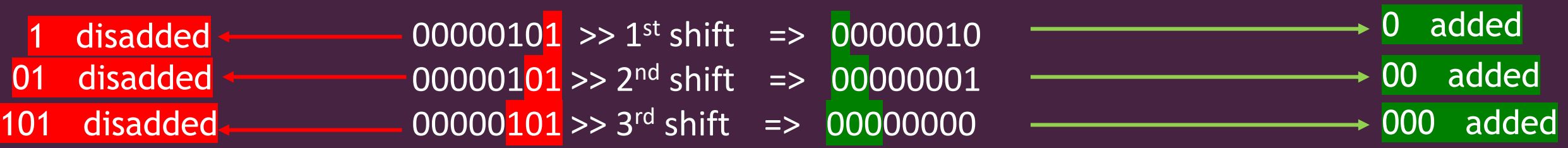
Example: Right Shift

right-shift 5 by 3 positions(5 >> 3)

Step-by-step:

5 in binary = 00000101(in 8-bit representation)

Shift 3 bits to the right:



Result=0

1. because the bits are shifted out of the right side (lower order end),
2. and zeros are filled in from the left

(for positive numbers).

General Rule:
Right shifting by **n** bits is equivalent to dividing by **2^n** (ignoring the decimal part).

2^(3)=> 2x2x2=8

5 >> 3 ≡ 5 / 8 = 0 (integer division)

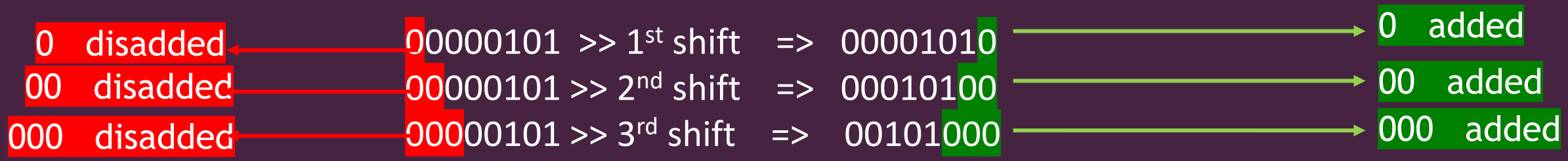
Example: Left Shift

left-shift 5 by 3 positions(5 << 3)

Step-by-step:

5 in binary = 00000101(in 8-bit representation)

Shift 3 bits to the left:



Result=40

1. because the bits are shifted out of the left side (higher-order) end,
2. and zeros are filled in from the right (lower-order end)

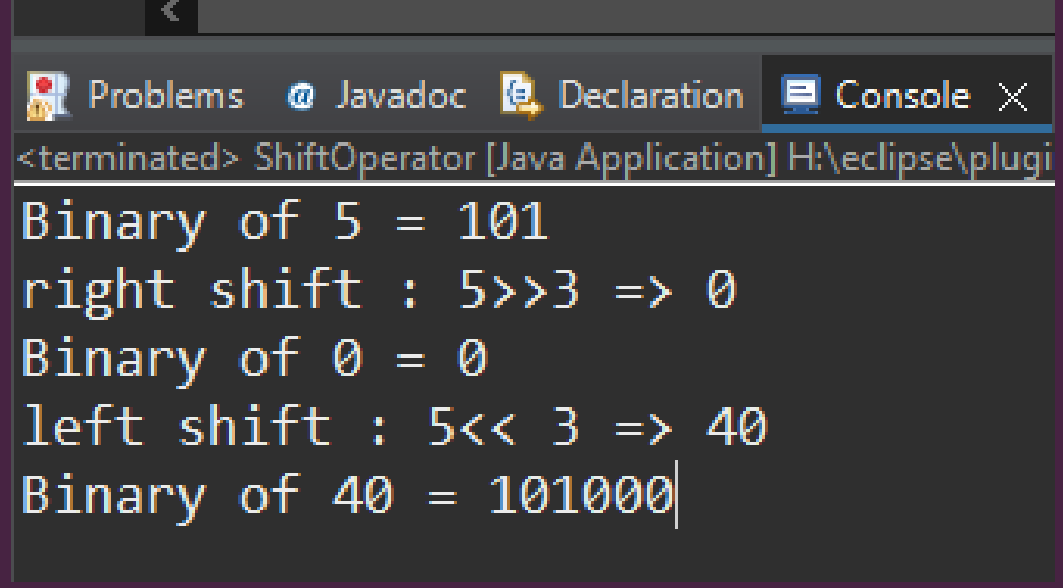
for positive numbers

General Rule:
Left shifting by **n** bits is equivalent to multiply by 2^n

2^(3)=> 2x2x2=8

5 << 3 ≡ 5 * 8 = 40

```java
package com;


public class ShiftOperator {


    public static void main(String[] args) {
            int result;
            int num=5;
            System.out.println("Binary of "+num+" = "+Integer.toBinaryString(num));

            result= num >> 3;
            System.out.println("right shift : "+num+">>3 => "+result); // 5/8
            System.out.println("Binary of "+result+" = "+Integer.toBinaryString(result));

            result= num << 3;
            System.out.println("left shift : "+num+"<< 3 => "+result); // 5*8
            System.out.println("Binary of "+result+" = "+Integer.toBinaryString(result));
    }


}
```



8-bit representation means:You display the binary number using 8 digits, by adding 0s on the left side (called leading zeros) if the binary number is shorter than 8 digits.

Eg. 5 in binary
Binary without padding: 101
8-bit representation: 00000101

Eg. 40 in binary
Binary without padding: 101000
8-bit representation: 00101000

Why do we do this?

To standardize the binary output.Useful in digital electronics and computer science where 8-bit, 16-bit, etc., are common.

Assignment operators

Operator	Description	Example
=	assigns values from right side operands to left side operand	a = b
+=	adds right operand to the left operand and assign the result to left	a+=b is same as a=a+b
-=	subtracts right operand from the left operand and assign the result to left operand	a-=b is same as a=a-b
*=	mutiply left operand with the right operand and assign the result to left operand	a*=b is same as a=a*b
/=	divides left operand with the right operand and assign the result to left operand	a/=b is same as a=a/b
%=	calculate modulus using two operands and assign the result to left operand	a%=b is same as a=a%b

```java
class AssignmentDemo{
	public static void main(String[] args) {

		int no = 4;
		int num;
		num=no; // assignment operator
		System.out.println("num using = operator -> " + num);

		System.out.println("num+=no ->"+(num+=no));
		System.out.println("num+=2 ->"+(num+=2));

		System.out.println("num-=1 ->"+(num-=1));
		System.out.println("num*=7 ->"+(num*=7));
		System.out.println("num/=3 ->"+(num/=3));
		System.out.println("num%=2 ->"+(num%=2));
	 }

}
```

```
H:\Luminar\corejava>javac AssignmentDemo.java

H:\Luminar\corejava>java AssignmentDemo
num using = operator -> 4
num+=no ->8
num+=2 ->10
num-=1 ->9
num*=7 ->63
num/=3 ->21
num%=2 ->1

H:\Luminar\corejava>
```

Misc operators

Conditional operator

It is also known as ternary operator and used to evaluate Boolean expression

epr1? expr2: expr3

If epr1 Condition is true? Then value expr2 : Otherwise value expr3

```
class ConditionOperatorDemo {
    public static void main(String args[]) {
        int num;
        int result;
        num = 10;

        result = (num == 1) ? 20: 30;
        System.out.println("Value of result is: " + result);
        result = (num == 10) ? 20: 30;
       System.out.println("Value of result is: " + result);
}
}
```

```
H:\Luminar\corejava>javac ConditionOperatorDemo.java

H:\Luminar\corejava>java ConditionOperatorDemo
Value of result is: 30
Value of result is: 20

H:\Luminar\corejava>
```

instanceOf operators

operator used to test if an object is of a given type. The result of the operation is either true or false. It's also known as type comparison operator because it compares the instance with type.

```java
package com;


class Employee{
}

public class InstanceOfDemo {
    public static void main(String[] args) {


            String str = "Luminar";
            Employee obj=new Employee();

            System.out.println("Is str an object(instance) of String? " + (str instanceof String));

            //System.out.println("Is str an object(instance) of Employee? " + (str instanceof
            Employee));//incompatible types

            //System.out.println("Is obj an object(instance) of String? " + (obj instanceof
            String));//incompatible types

            System.out.println("Is obj an object(instance) of Employee? " + (obj instanceof Employee));
    }
}
```

```
H:\Luminar\corejava>javac InstanceOfDemo.java

H:\Luminar\corejava>java InstanceOfDemo
Is str an object(instance) of String? true
Is obj an object(instance) of Employee? true

H:\Luminar\corejava>
```

Thank you ☺ Happy Learning ☺