LAB # 2

JAVA OPERATORS

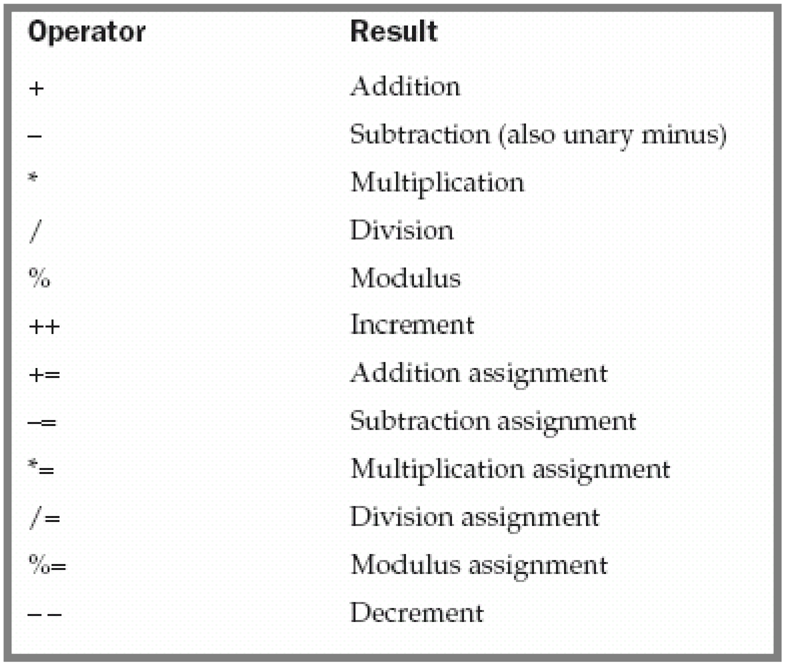
OBJECTIVE

To Study Arithmetic, Bitwise Logical, Bitwise Operator Assignments, Left and Right Shift, relational and Boolean Logical Operators.

THEORY

Arithmetic Operators

Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra. The following table lists the arithmetic operators:



Operator	Result
+	Addition
–	Subtraction (also unary minus)
*	Multiplication
/	Division
%	Modulus
++	Increment
+=	Addition assignment
–=	Subtraction assignment
*=	Multiplication assignment
/=	Division assignment
%=	Modulus assignment
– –	Decrement

Arithmetic Assignment Operators

Java provides special operators that can be used to combine an arithmetic operation with an assignment.

```
// Demonstrate several assignment operators.
public class OpEquals {
public static void main(String args[]) {
int a = 1;
int b = 2;
int c = 3;
a += 5;
b *= 4;
c += a * b;
c %= 6;
System.out.println("a = " + a);
System.out.println("b = " + b);
System.out.println("c = " + c);
}
}
```

Output

```
a = 6
b = 8
c = 3
```

Increment and Decrement

- The ++ and the – – are Java’s increment and decrement operators.
- ++ a and a ++ are pre and post increment and -- a and a-- are pre and post decrement

The following program demonstrates the increment operator.

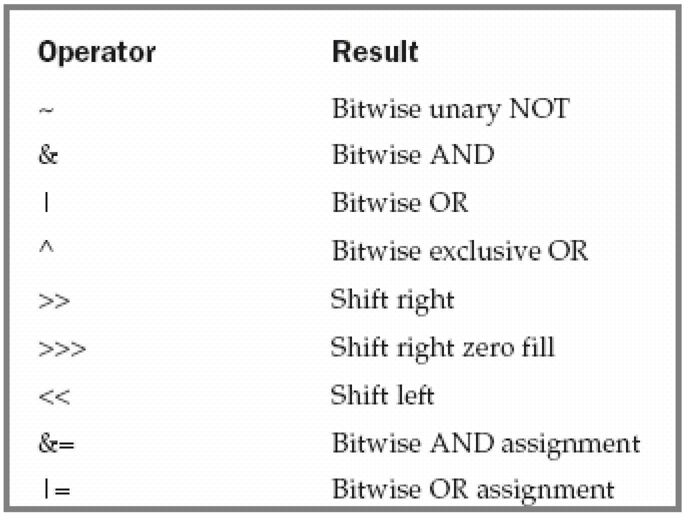
```
// Demonstrate ++.
public class IncDec {
public static void main(String args[]) {
int a = 1;
int b = 2;
int c;
int d;
c = ++b;
d = a++;
c++;
System.out.println("a = " + a);
System.out.println("b = " + b);
System.out.println("c = " + c);
System.out.println("d = " + d);
}
}
```

Output

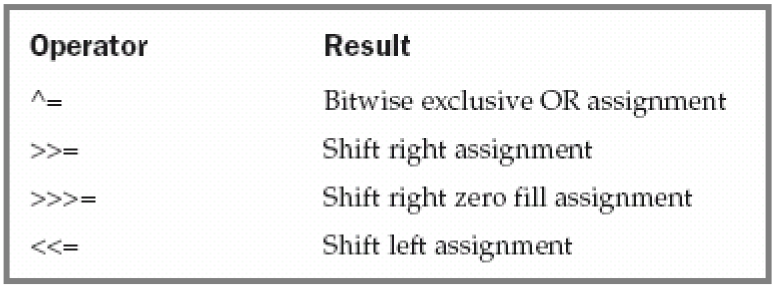
```
a = 2
b = 3
c = 4
```

The Bitwise Operators

Java defines several *bitwise operators* which can be applied to the integer types, **long**, **int**, **short**, **char**, and **byte**. These operators act upon the individual bits of their operands. They are summarized in the following table:



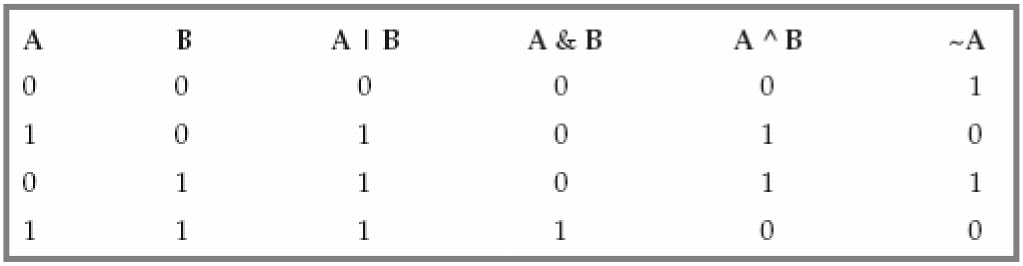
Operator	Result
~	Bitwise unary NOT
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
>>	Shift right
>>>	Shift right zero fill
<<	Shift left
&=	Bitwise AND assignment
|=	Bitwise OR assignment



Operator	Result
^=	Bitwise exclusive OR assignment
>>=	Shift right assignment
>>>=	Shift right zero fill assignment
<<=	Shift left assignment

The Bitwise Logical Operators

The bitwise logical operators are **&**, |, ^, and **~**. The following table shows the outcome of each operation. In the discussion that follows, keep in mind that the bitwise operators are applied to each individual bit within each operand.



A	B	A | B	A & B	A ^ B	~A
0	0	0	0	0	1
1	0	1	0	1	0
0	1	1	0	1	1
1	1	1	1	0	0

The following program demonstrates the bitwise logical operators:

```
// Demonstrate the bitwise logical operators.
public class Bitwise{
public static void main (String[] args){
int a=10;
int b=28;
System.out.println(a&b);
System.out.println(a|b);
System.out.println(a^b);
}
}
```

The Left Shift

```
// Left shifting a byte value.
public class ByteShift {
public static void main(String args[]) {
byte a = 64, b;
int i;
i = a << 2;
b = (byte) (a << 2);
System.out.println("Original value of a: " + a);
System.out.println("i and b: " + i + " " + b);
}
}
```

Output

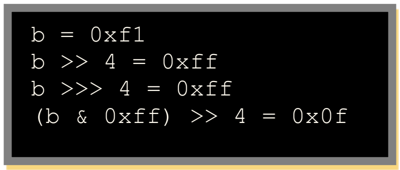
```
Original value of a: 64
i and b: 256   0
```

The Unsigned Right Shift

As you have just seen, the >> operator automatically fills the high-order bit with its previous contents each time a shift occurs. This preserves the sign of the value.

```
// Unsigned shifting a byte value.
public class ByteUShift {
   public static void main(String args[]) {
char hex[] = {'0', '1', '2', '3', '4', '5', '6', '7',
                      '8', '9', 'a', 'b', 'c', 'd', 'e', 'f'};
byte b = (byte) 0xf1;
byte c = (byte) (b >> 4);
byte d = (byte) (b >>> 4);
byte e = (byte) ((b & 0xff) >> 4);
System.out.println(" b = 0x"+ hex[(b >> 4) & 0x0f] + hex[b & 0x0f]);
System.out.println(" b >> 4 = 0x"+ hex[(c >> 4) & 0x0f] + hex[c & 0x0f]);
System.out.println(" b >>> 4 = 0x"+ hex[(d >> 4) & 0x0f] + hex[d & 0x0f]);
System.out.println("(b & 0xff) >> 4 = 0x"+ hex[(e >> 4) & 0x0f] + hex[e & 0x0f]);
}
}
```

Output

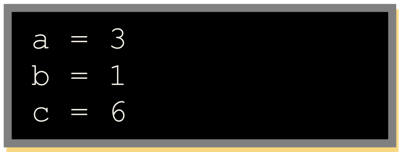


```
b = 0xf1
b >> 4 = 0xff
b >>> 4 = 0xff
(b & 0xff) >> 4 = 0x0f
```

Bitwise Operator Assignments

```
public class OpBitEquals {
public static void main(String args[]) {
int a = 1;
int b = 2;
int c = 3;
a |= 4;
b >>= 1;
c <<= 1;
a ^= c;
System.out.println("a = " + a);
System.out.println("b = " + b);
System.out.println("c = " + c);
}
}
```

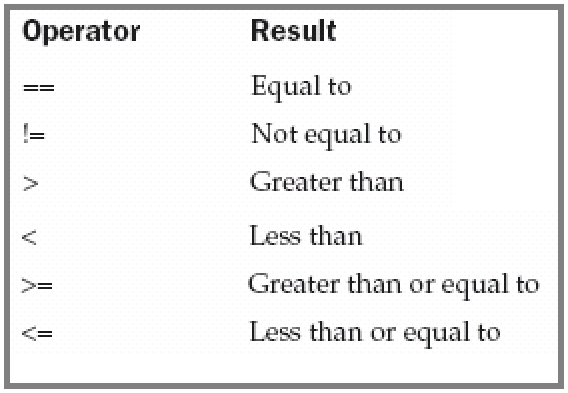
Output



```
a = 3
b = 1
c = 6
```

Relational Operators

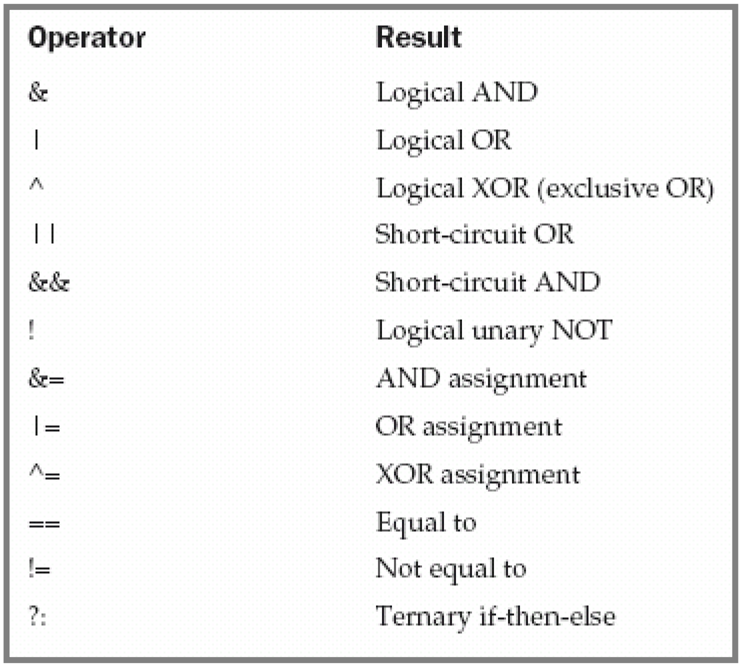
The *relational operators* determine the relationship that one operand has to the other. Specifically, they determine equality and ordering. The relational operators are shown here:



Operator	Result
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Boolean Logical Operators

The Boolean logical operators shown here operate only on **boolean** operands. All of the binary logical operators combine two **boolean** values to form a resultant **boolean** value.



Operator	Result
&	Logical AND
|	Logical OR
^	Logical XOR (exclusive OR)
||	Short-circuit OR
&&	Short-circuit AND
!	Logical unary NOT
&=	AND assignment
|=	OR assignment
^=	XOR assignment
==	Equal to
!=	Not equal to
?:	Ternary if-then-else

LAB TASK

1. Write a program to implement calculator that can perform all function define above.

2. Write a program that inputs a decimal integer and displays its value in hexadecimal.