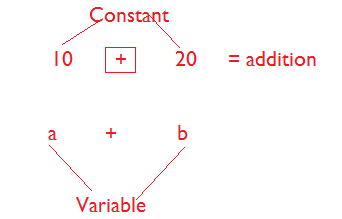
**Operator**

* Operator is a symbol that acts upon the operands to carry out certain type computations.
* A variable or constant otherwise known as operand.

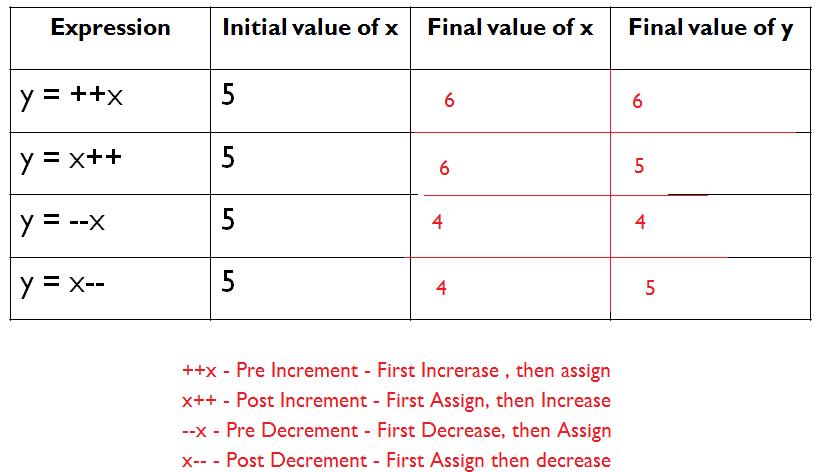


**Types of Operator**

* Unary Operator
  + Operator associated with one operand is known as unary operator.
* Binary Operator
  + Operator associated with two operands is known as Binary Operator.
* Ternary Operator
  + Operator associated with three operands is known as Ternary Operator.

**Increment Operator / Decrement Operator**

* Increment Operator
  + It is used to add 1 to the exiting value of the variable.
* Decrement Operator
  + It is used to subtract 1 from the value of the variable.



**Case1:**

* We can apply increment and decrement operator only for variables but not for constant values.

class Test

{

public static void main(String args[])

{

int x = 4;

int y = ++10;

System.out.println(y);

}

}

**Case2:**

* Nesting of increment and decrement operators is not allowed.

class Test

{

public static void main(String args[])

{

int x = 4;

int y = ++(++x);

System.out.println(y);

}

}

**Case3:**

* We can't apply increment and decrement operators for the final variable.

class Test

{

public static void main(String args[])

{

final int x = 2;

x++;

System.out.println(x);

}

}

**Case3:**

* We can apply increment and decrement operators for every primitive data type except boolean.

class Test

{

public static void main(String args[])

{

byte a = 10;

short b = 10;

int c = 10;

long d =10;

float e = 10.2f;

double f = 10.22;

char g = 'a';

boolean h = true;

System.out.println(++a);

System.out.println(++b);

System.out.println(++c);

System.out.println(++d);

System.out.println(++e);

System.out.println(++f);

System.out.println(++g);

//System.out.println(++h);

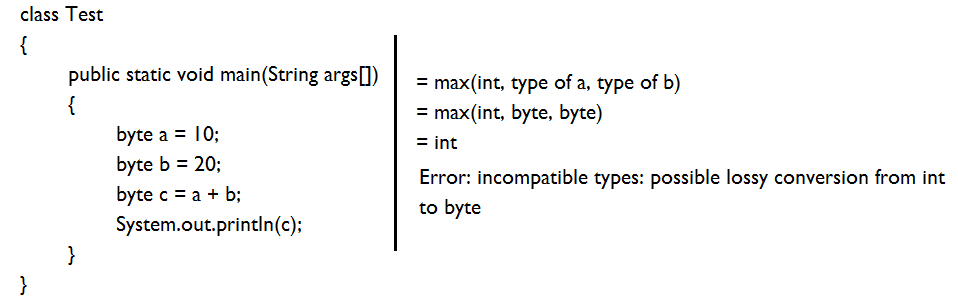
}

}

**Note**

* Whenever we are performing any arithmetic operation between two variables a and b the result type is always:

max(int, type of a, type of b)



**Solution**

class Test

{

public static void main(String args[])

{

byte a = 10;

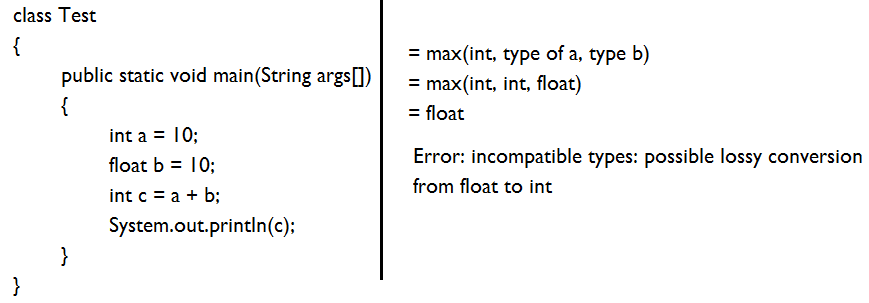
byte b = 20;

int c = a + b;

System.out.println(c);

}

}



**Solution**

class Test

{

public static void main(String args[])

{

int a = 10;

float b = 10;

float c = a + b;

System.out.println(c);

}

}

**Difference between b++ and b = b+1**

**Example#1**

class Test

{

public static void main(String args[])

{

byte b = 10;

b++;

System.out.println(b);

}

}

**Example#2**

class Test

{

public static void main(String args[])

{

byte b = 10;

b = b + 1;

System.out.println(b);

}

}

Error

**Example#3**

class Test

{

public static void main(String args[])

{

byte b = 10;

b = (byte) (b+1);

System.out.println(b);

}

}

**Note**

b++ is equal to b = b + 1 - wrong

b++ is equal to b = byte(b+1) – Correct

**Relational Operators**

* >
* <
* >=
* <=

**Note**

* We can apply relational operators for every primitive data type except Boolean.

class Test

{

public static void main(String args[])

{

System.out.println(10>20);

System.out.println(10<20);

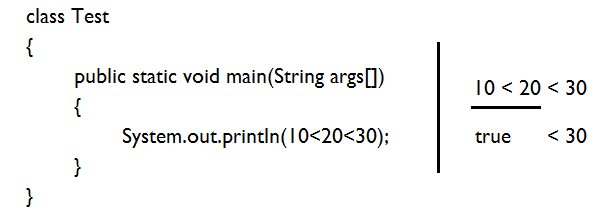
System.out.println(10>=10);

System.out.println(10<=20);

}

}

**Example#2**



**Equality Operator**

class Test

{

public static void main(String args[])

{

System.out.println(10 == 20);

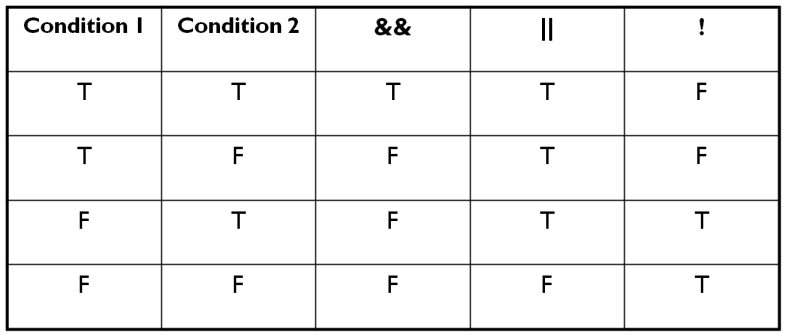
System.out.println(10 != 20);

}

}

**Logical Operator**

* These operators are used to combine more than one relational expression.



&& - both are true then true otherwise false

|| - If both are false then false otherwise true

**Example**

class Test

{

public static void main(String args[])

{

System.out.println(10<20 && 20>30);

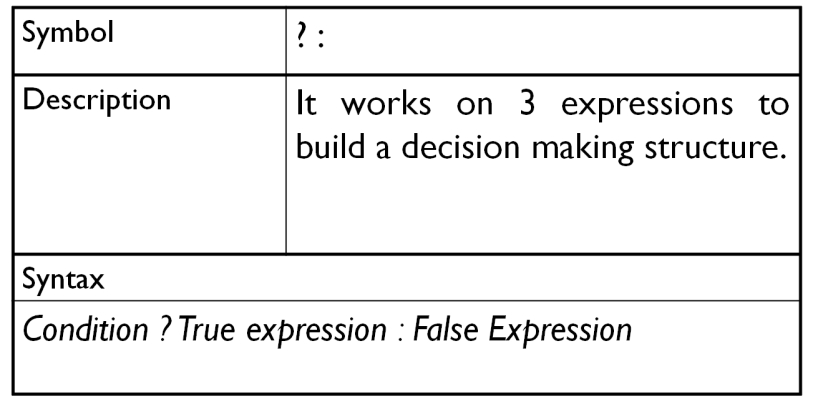
System.out.println(10<20 || 20>30);

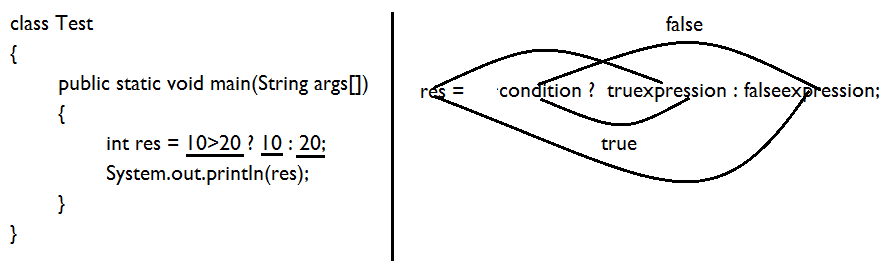
System.out.println(!(10<20));

}

}

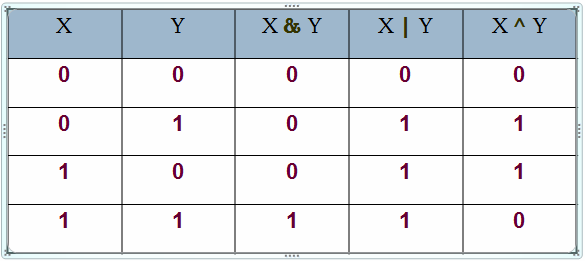
**Conditional Operator**

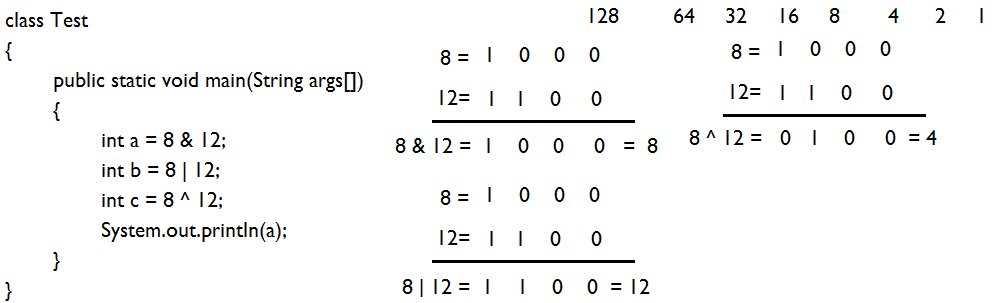




**Bitwise Operator**

* Manipulating data in bits level is known as bitwise operation.
* Operator used to do so is known as bitwise operator.
* Types
  + Bitwise AND &
  + Bitwise OR |
  + Bitwise XOR ^
  + Bitwise Left Shift <<
  + Bitwise Right Shift >>





**Bitwise Left Shift**

