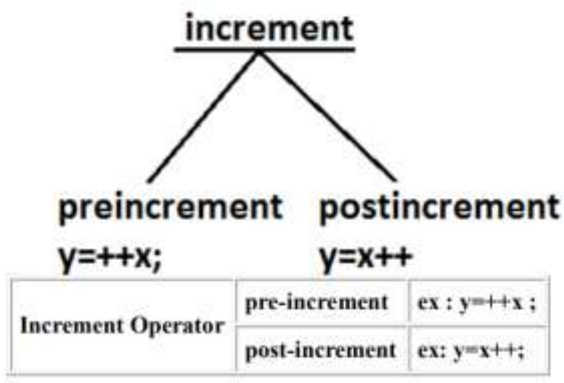
OPERATORS & ASSIGNMENTS

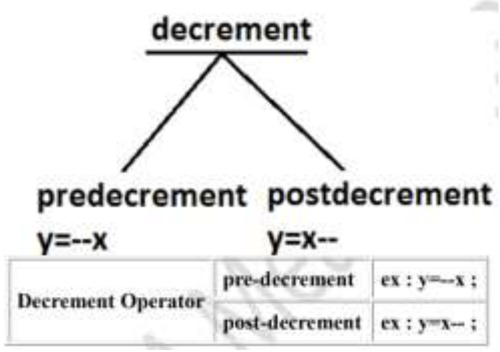
Agenda:

1. increment & decrement operators
2. arithmetic operators
3. string concatenation operators
4. Relational operators
5. Equality operators
6. instanceof operators
7. Bitwise operators
8. Short circuit operators
9. type cast operators
10. assignment operator
11. conditional operator
12. new operator
13. [ ] operator
14. Precedence of java operators
15. Evaluation order of java operands
16. new Vs newInstance( )
17. instanceof Vs isInstance( )
18. ClassNotFoundException Vs NoClassDefFoundError

Increment & Decrement operators :

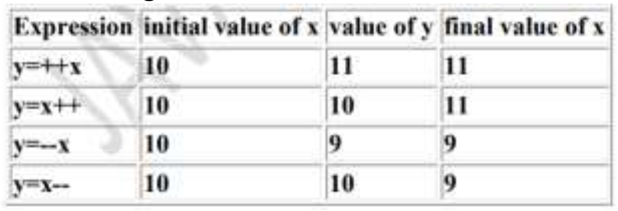


Increment Operator	pre-increment	ex : y=++x ;
	post-increment	ex: y=x++;



Decrement Operator	pre-decrement	ex : y=--x ;
	post-decrement	ex : y=x-- ;

The following table will demonstrate the use of increment and decrement operators.



Expression	initial value of x	value of y	final value of x
y=++x	10	11	11
y=x++	10	10	11
y=--x	10	9	9
y=x--	10	10	9

1. Increment & decrement operators we can apply only for variables but not for constant values.other wise we will get compile time error .

Ex 1 :

```
 int x = 4;
int y = ++x;
 System.out.pritnln(y); //output : 5
```

Ex 2 :

```
 int x = 4;
int y = ++4;
System.out.pritnln(y);
          C.E: unexpected type
          required: varialbe
          found : value
```

2: We can't perform nesting of increment or decrement operator, other wise we will get compile time error

```
int x= 4;
int y = ++(++x);
System.out.println(y);
            C.E: unexpected type
            required: varialbe
            found : value
```

3. For the final variables we can't apply increment or decrement operators ,other wise we will get compile time error

Ex:

```
final int x = 4;
x++; // x = x + 1
System.out.println(x);
            C.E : can't assign a value to final variable 'x' .
```

4. We can apply increment or decrement operators even for primitive data types except boolean .
Ex:

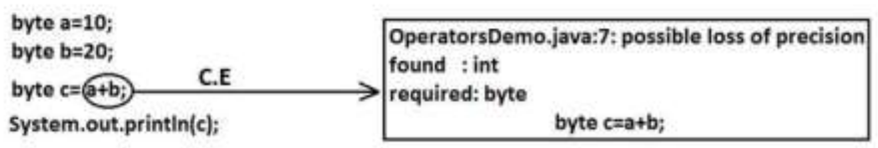
```
int x=10;
x++;
System.out.println(x); //output :11
```

```
char ch='a';
 ch++;
 System.out.println(ch); //b
```

```
double d=10.5;
 d++;
 System.out.println(d); //11.5
```

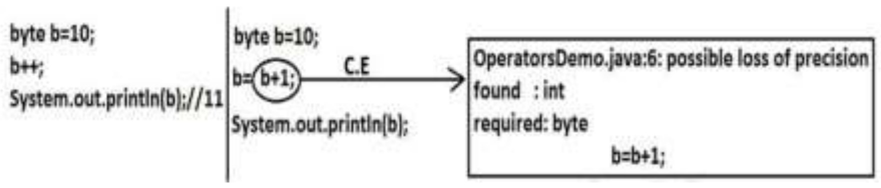
```
boolean b=true;
 b++;
 System.out.println(b);
             CE : operator ++ can't be applied to Boolean
```

Difference between b++ and b = b+1?
If we are applying any arithmetic operators b/w 2 operands 'a' & 'b' the result type is max(int , type of a , type of b)



```
byte a=10;
byte b=20;
byte c=a+b;
System.out.println(c);
```

```
OperatorsDemo.java:7: possible loss of precision
found   : int
required: byte
                  byte c=a+b;
```



```
byte b=10;
b++;
System.out.println(b);//11
```

```
byte b=10;
b=b+1;
System.out.println(b);
```

```
OperatorsDemo.java:6: possible loss of precision
found   : int
required: byte
               b=b+1;
```

Ex 1:

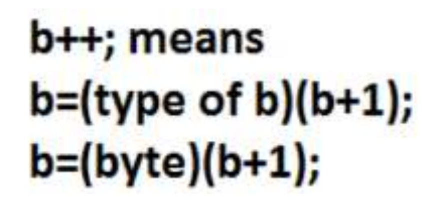
```
byte a=10;
byte b=20;
byte c=a+b; //byte c=byte(a+b); valid
System.out.println(c);
```

CE : possible loss of precession
found : int
required : byte

Ex 2:

```
byte b=20;
byte b=b+1; //byte b=(byte)b+1 ; valid
System.out.println(c);
        CE : possible loss of precession
        found : int
        required : byte
```

In the case of Increment & Decrement operators internal type casting will be performed automatically by the compiler



```
b++; means
b=(type of b)(b+1);
b=(byte)(b+1);
```

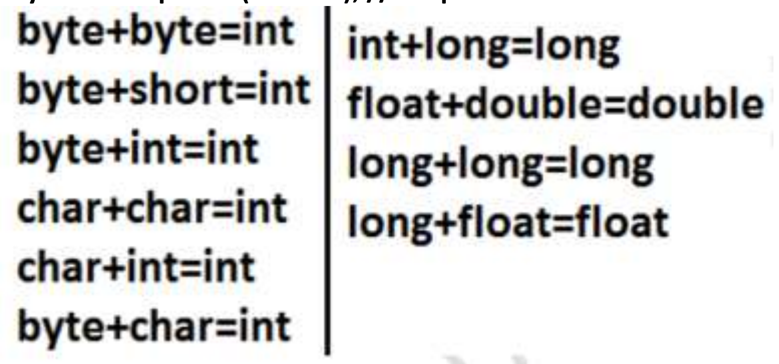
b++; => b=(type of b)b+1;

```
Ex:
byte b=10;
b++;
System.out.println(b); //output : 11
```

Arithmetic Operator :

1. If we apply any Arithmetic operation b/w 2 variables a & b ,
the result type is always max(int , type of a , type of b)
2. Example :

. byte + byte=int
. byte+short=int
. short+short=int
. short+long=long
. double+float=double
. int+double=double
. char+char=int
. char+int=int
. char+double=double
. System.out.println('a' + 'b'); // output : 195
. System.out.println('a' + 1); // output : 98
. System.out.println('a' + 1.2); // output : 98.2



byte+byte=int
byte+short=int
byte+int=int
char+char=int
char+int=int
byte+char=int

int+long=long
float+double=double
long+long=long
long+float=float

17. In integral arithmetic (byte , int , short , long) there is no way to represents infinity , if infinity is the result we will get the ArithmeticException / by zero

System.out.println(10/0); // output RE : ArithmeticException / by zero

But in floating point arithmetic(float , double) there is a way represents infinity.
System.out.println(10/0.0); // output : infinity

System.out.println(10/0); $\xrightarrow{\text{R.E}}$ $\boxed{\text{Exception in thread "main" java.lang.ArithmeticException: / by zero}}$

For the Float & Double classes contains the following constants :

1. POSITIVE_INFINITY
2. NEGATIVE_INFINITY

Hence , if infinity is the result we won't get any ArithmeticException in floating point arithmetics

Ex :

```
System.out.println(10/0.0); // output : infinity
System.out.println(-10/0.0); // output : - infinity
```

18. NaN(Not a Number) in integral arithmetic (byte , short , int , long) there is no way to represent undefine the results. Hence the result is undefined we will get ArithmericException in integral arithmetic

```
System.out.println(0/0); // output RE : ArithmeticException / by zero
```

But floating point arithmetic (float , double) there is a way to represents undefined the results .

For the Float , Double classes contains a constant NaN , Hence the result is undefined we won't get ArithmeticException in floating point arithmetics .

```
System.out.println(0.0/0.0); // output : NaN
System.out.println(-0.0/0.0); // output : NaN
```

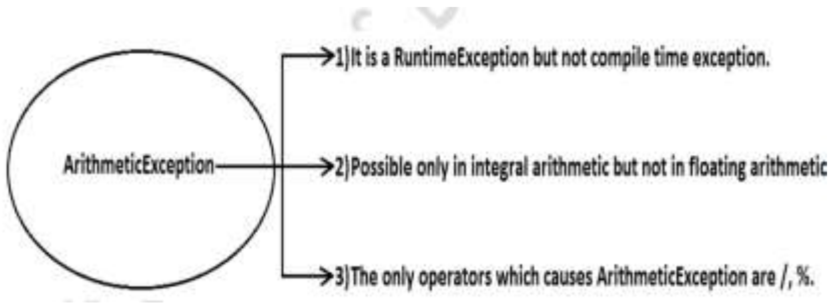
19. For any 'x' value including NaN , the following expressions returns false

System.out.println(0/0); $\xrightarrow{\text{R.E}}$ $\boxed{\text{Exception in thread "main" java.lang.ArithmeticException: / by zero}}$

```
20. // Ex : x=10;
21. System.out.println(10 < Float.NaN ); // false
22. System.out.println(10 <= Float.NaN ); // false
23. System.out.println(10 > Float.NaN ); // false
24. System.out.println(10 >= Float.NaN ); // false
25. System.out.println(10 == Float.NaN ); // false
26. System.out.println(Float.NaN == Float.NaN ); // false

28. System.out.println(10 != Float.NaN ); //true
29. System.out.println(Float.NaN != Float.NaN ); //true
```

30. ArithmeticException :

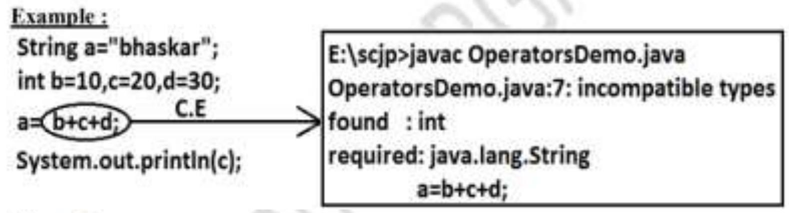
1. It is a RuntimeException but not compile time error
2. It occurs only in integral arithmetic but not in floating point arithmetic.
3. The only operations which cause ArithmeticException are : ' / ' and ' % '


ArithmeticException
1)It is a RuntimeException but not compile time exception.
2)Possible only in integral arithmetic but not in floating arithmetic
3)The only operators which causes ArithmeticException are /, %.

String Concatenation operator :

1. The only overloaded operator in java is ' + ' operator some times it access arithmetic addition operator & some times it access String concatenation operator.

2. If acts as one argument is String type , then '+' operator acts as concatenation and If both arguments are number type , then operator acts as arithmetic operator
3. Ex :
4. String a="ashok";
int b=10 , c=20 , d=30 ;
System.out.println(a+b+c+d); //output : ashok102030
System.out.println(b+c+d+a); //output : 60ashok
System.out.println(b+c+a+d); //output : 30ashok30
System.out.println(b+a+c+d); //output : 10ashok 2030

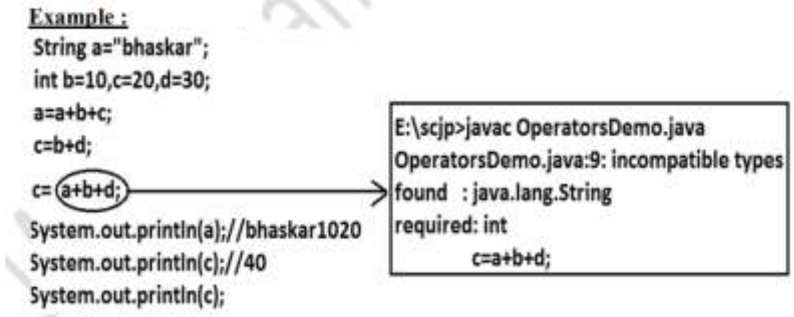


Example :

```
String a="bhaskar";
int b=10,c=20,d=30;
a=b+c+d;
System.out.println(c);
```

C.E

```
E:\scjp>javac OperatorsDemo.java
OperatorsDemo.java:7: incompatible types
found   : int
required: java.lang.String
          a=b+c+d;
```



Example :

```
String a="bhaskar";
int b=10,c=20,d=30;
a=a+b+c;
c=b+d;
c=a+b+d;
System.out.println(a);//bhaskar1020
System.out.println(c);//40
System.out.println(c);
```

```
E:\scjp>javac OperatorsDemo.java
OperatorsDemo.java:9: incompatible types
found   : java.lang.String
required: int
          c=a+b+d;
```

5. consider the following declaration
String a="ashok";
int b=10 , c=20 , d=30 ;
6. Example :
a=b+c+d ;

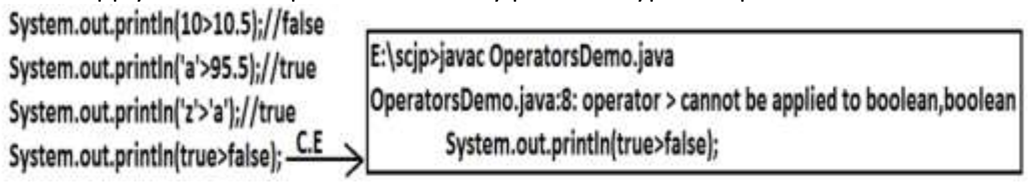
CE : incompatible type
found : int
required : java.lang.String

7. Example :
8.
a=a+b+c ; // valid
9. Example :
10.
b=a+c+d ;
11.

12.
CE : incompatible type
13.
found : java.lang.String
14.
required : int
15. Example :
16.
b=b+c+d ; // valid

Relational Operators(< , <= , > , >=)

We can apply relational operators for every primitive type except boolean .

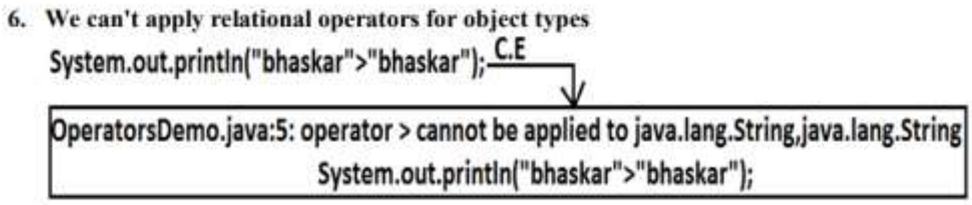


```
System.out.println(10>10.5);//false
System.out.println('a'>95.5);//true
System.out.println('z'>'a');//true
System.out.println(true>false);
```

C.E

```
E:\scjp>javac OperatorsDemo.java
OperatorsDemo.java:8: operator > cannot be applied to boolean,boolean
        System.out.println(true>false);
```

1. System.out.println(10 < 10.5); //true
2. System.out.println('a' > 100.5); //false
3. System.out.println('b' > 'a'); //true
4. System.out.println(true > false);
5. //CE : operator > can't be applied to boolean , Boolean



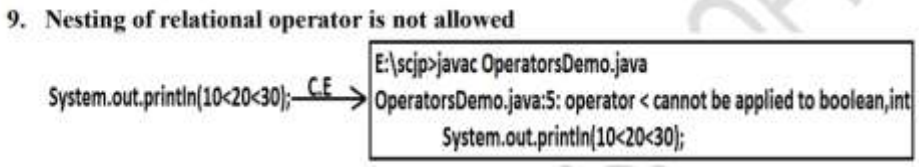
6. We can't apply relational operators for object types

```
System.out.println("bhaskar">"bhaskar");
```

C.E

```
OperatorsDemo.java:5: operator > cannot be applied to java.lang.String,java.lang.String
        System.out.println("bhaskar">"bhaskar");
```

7. System.out.println("ashok123" > "ashok");
**8. // CE: operator > can't be applied to java.lang.String ,
java.lang.String**



9. Nesting of relational operator is not allowed

```
System.out.println(10<20<30);
```

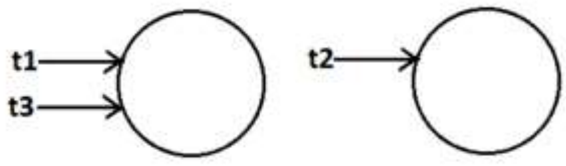
C.E

```
E:\scjp>javac OperatorsDemo.java
OperatorsDemo.java:5: operator < cannot be applied to boolean,int
        System.out.println(10<20<30);
```

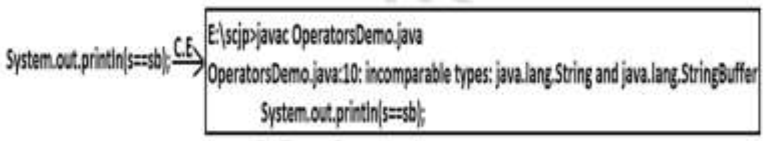
10. System.out.println(10 > 20 > 30); // System.out.println(true > 30);
11. //CE : operator > can't be applied to boolean , int

Equality Operators : (== , !=)

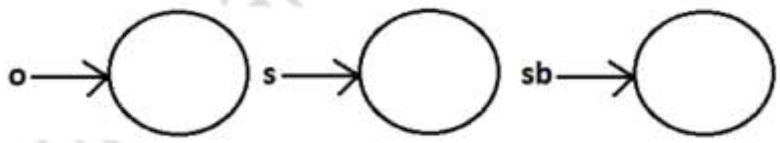
**1. We can apply equality operators for every primitive type including boolean type
also**
2. System.out.println(10 == 20) ; //false
3. System.out.println('a' == 'b'); //false
4. System.out.println('a' == 97.0) //true
5. System.out.println(false == false) //true
6. We can apply equality operators for object types also .
**For object references r1 and r2 , r1 == r2 returns true if and only if both r1 and r2 pointing to the same object. i.e., == operator meant for reference-comparision
Or address-comparision.**
7. Thread t1=new Thread( ) ;
8. Thread t2=new Thread( );
9. Thread t3=t1 ;
10. System.out.println(t1==t2); //false
11. System.out.println(t1==t3); //true



12. To use the equality operators between object type compulsory these should be some relation between argument types(child to parent , parent to child) , Otherwise we will get Compiletime error incompatible types
13. Thread t=new Thread( ) ;
14. Object o=new Object( );
15. String s=new String("durga");
16. System.out.println(t ==o); //false
17. System.out.println(o==s); //false
18. System.out.println(s==t);
19. CE : incompatible types : java.lang.String and java.lang.Thread

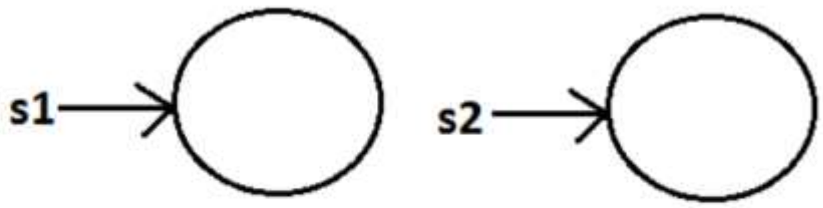


```
E:\scjp>javac OperatorsDemo.java
OperatorsDemo.java:10: incomparable types: java.lang.String and java.lang.StringBuffer
        System.out.println(s==sb);
```



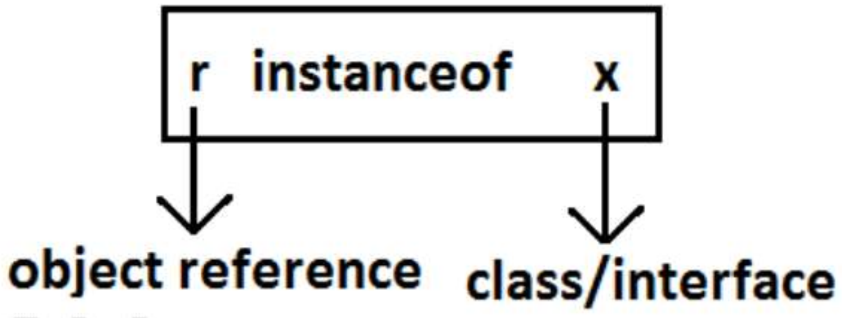
20. For any object reference of on r==null is always false , but null==null is always true .
21. String s= new String("ashok");
22. System.out.println(s==null); //output : false
23. String s=null ;
24. System.out.println(r==null); //true
25. System.out.println(null==null); //true

26. What is the difference between == operator and .equals( ) method ?
In general we can use .equals( ) for content comparision where as == operator for reference comparision
27.
28. String s1=new String("ashok");
29. String s2=new String("ashok");
30. System.out.println(s1==s2); //false
31. System.out.println(s1.equals(s2)); //true



instanceof operator :

1. We can use the instanceof operator to check whether the given an object is perticular type or not



```
Object o=l.get(0);           // l is an array name
if(o  instanceof Student) {
  Student s=(Student)o ;
       //perform  student  specific  operation
  }
  elseif(o instanceof Customer) {
    Customer c=(Customer)o;
      //perform  Customer specific  operations
        }
```

O instanceof X here O is object reference , X is ClassName/Interface name

```
            Thread  t = new  Thread( );
            System.out.println(t  instanceof Thread);   //true
            System.out.println(t instanceof Object);     //true
```

Bitwise Operators : (& , | , ^)

1. & (AND) : If both arguments are true then only result is true.
2. | (OR) : if at least one argument is true. Then the result is true.
3. ^ (X-OR) : if both are different arguments. Then the result is true.

Example:

System.out.println(true&false);//false
System.out.println(true|false);//true
System.out.println(true^false);//true

We can apply bitwise operators even for integral types also.

Example:

System.out.println(4&5);//4 using binary digits
System.out.println(4|5);//5 4-->100
System.out.println(4^5);//1 5-->101

Example :

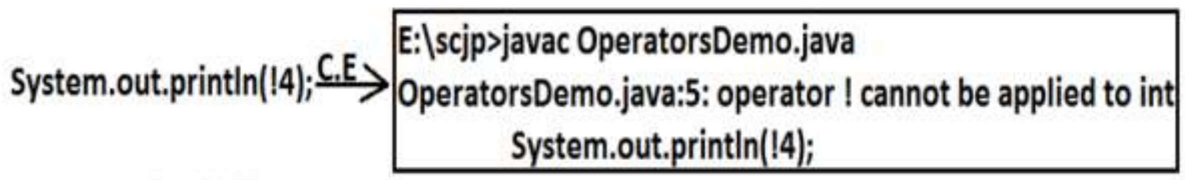
```
System.out.println(4&5);//4    100     100     100
System.out.println(4|5);//5    101     101     101
                              -----   -----   -----
System.out.println(4^5);//1    100     101     001
                              -----   -----   -----
```

Bitwise complement (~) (tilde symbol) operator:

1. We can apply this operator only for integral types but not for boolean types.
2. Example :
3. System.out.println(~true); // CE :opetator ~ can not be applied to boolean
4. System.out.println(~4); //-5

Boolean complement (!) operator:

This operator is applicable only for boolean types but not for integral types.



```
System.out.println(!4);
```

```
E:\scjp>javac OperatorsDemo.java
OperatorsDemo.java:5: operator ! cannot be applied to int
        System.out.println(!4);
```

Example :
Example:
System.out.println(!true);//false
System.out.println(!false);//true
System.out.println(!4);//CE : operator ! can not be applied to int
Summary:
&
| Applicable for both boolean and integral types.
^
~ --------Applicable for integral types only but not for boolean types.
! --------Applicable for boolean types only but not for integral types.

Type Cast Operator :

There are 2 types of type-casting

1. implicit

2. explicit

implicit type casting :

int x='a';

System.out.println(x); //97

1. The compiler is responsible to perform this type casting.

2. When ever we are assigning lower datatype value to higher datatype variable

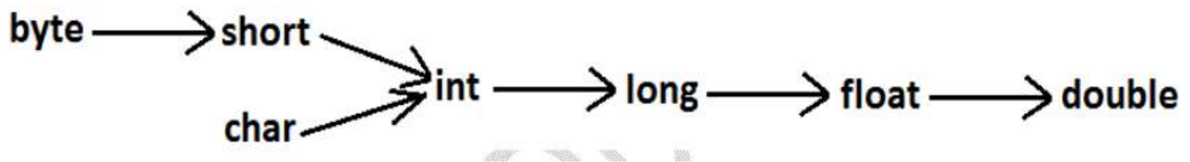
then implicit type cast will be performed .

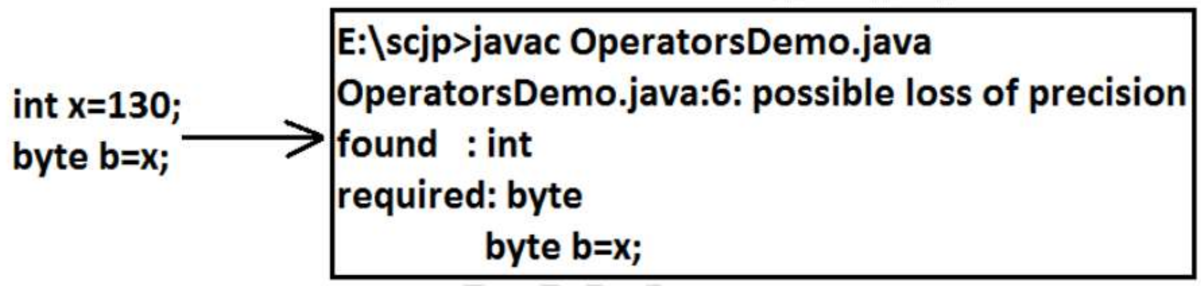
3. It is also known as Widening or Upcasting.

4. There is no lose of information in this type casting.

5. The following are various possible implicit type casting.

Diagram:





```
int x=130;
byte b=x;
```

```
E:\scjp>javac OperatorsDemo.java
OperatorsDemo.java:6: possible loss of precision
found   : int
required: byte
          byte b=x;
```

18. When ever we are assigning higher datatype value to lower datatype value variable by explicit type-casting ,the most significant bits will be lost i.e., we have considered least significant bits.

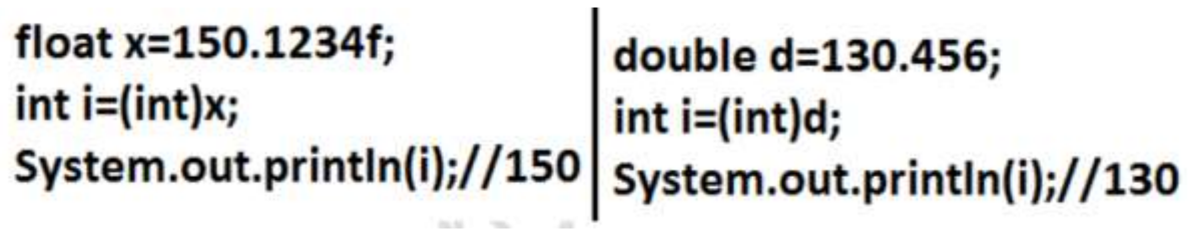
Example 3 :

```
int x=150;
short s=(short)x;
byte b=(byte)x;
System.out.println(s); //150
System.out.println(b); //-106
```

When ever we are assigning floating point value to the integral types by explicit type casting , the digits of after decimal point will be lost .

Example 4:

```
double d=130.456 ;

int x=(int)d ;
System.out.println(x); //130

byte b=(byte)d ;
System.out.println(b); //-206
```



```
float x=150.1234f;
int i=(int)x;
System.out.println(i);//150
```

```
double d=130.456;
int i=(int)d;
System.out.println(i);//130
```

Assignment Operator :

There are 3 types of assignment operators

1. Simple assignment:

Example: int x=10;

2. Chained assignment:

3. Example:

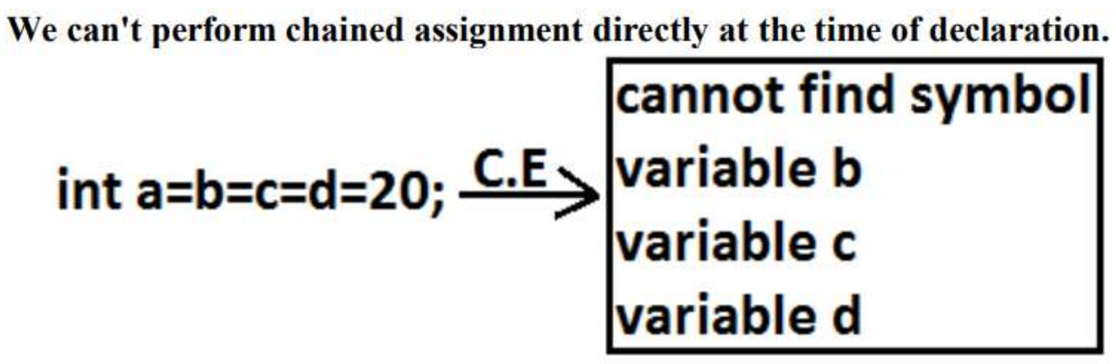
4. int a,b,c,d;

5. a=b=c=d=20;

6. System.out.println(a+"---"+b+"---"+c+"---"+d);//20---20---20---20

int b , c , d ;

. int a=b=c=d=20 ; //valid



We can't perform chained assignment directly at the time of declaration.

$$\texttt{int a=b=c=d=20;} \xrightarrow{\text{C.E}} \boxed{\begin{array}{l}\text{cannot find symbol}\\ \text{variable b}\\ \text{variable c}\\ \text{variable d}\end{array}}$$

Compound assignment:

1. Sometimes we can mixed assignment operator with some other operator

to form compound assignment operator.

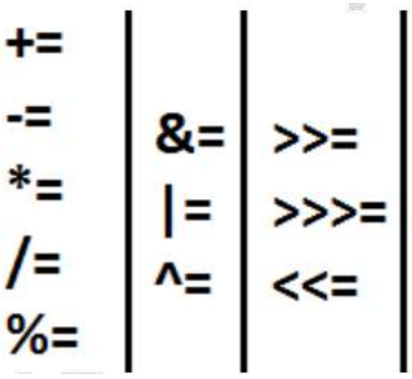
2. Ex:

3. int a=10 ;

4. a +=20 ;

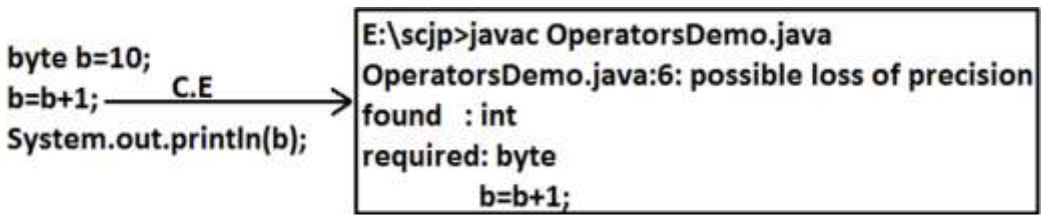
5. System.out.println(a); //30

6. The following is the list of all possible compound assignment operators in java.



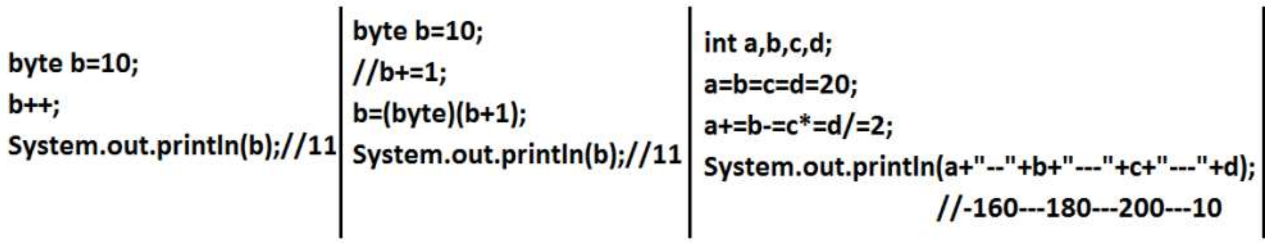
+=	&=	>>=
-=	|=	>>>=
*=	^=	<<=
/=		
%=		

In the case of compound assignment operator internal type casting will be performed automatically by the compiler (similar to increment and decrement operators.)



```
byte b=10;
b=b+1; ——C.E——>
System.out.println(b);
```

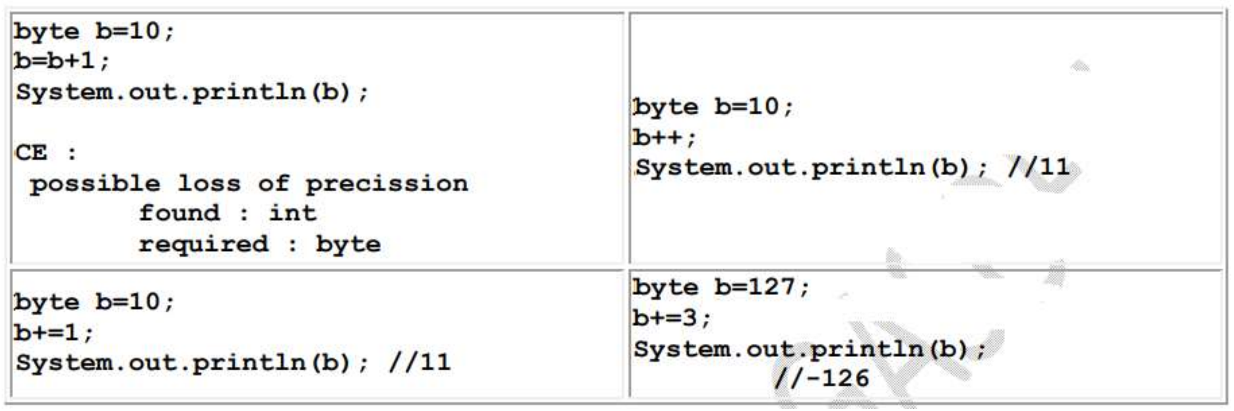
```
E:\scjp>javac OperatorsDemo.java
OperatorsDemo.java:6: possible loss of precision
found   : int
required: byte
          b=b+1;
```



```
byte b=10;
b++;
System.out.println(b);//11
```

```
byte b=10;
//b+=1;
b=(byte)(b+1);
System.out.println(b);//11
```

```
int a,b,c,d;
a=b=c=d=20;
a+=b-=c*=d/=2;
System.out.println(a+"--"+b+"---"+c+"---"+d);
                    //-160---180---200---10
```



```
byte b=10;
b=b+1;
System.out.println(b);

CE :
 possible loss of precission
        found : int
        required : byte
```

```
byte b=10;
b++;
System.out.println(b); //11
```

```
byte b=10;
b+=1;
System.out.println(b); //11
```

```
byte b=127;
b+=3;
System.out.println(b);
          //-126
```

int a , b , c , d ;

a=b=c=d=20 ;

a += b-= c *= d /= 2 ;

System.out.println(a+"---"+b+"---"+c+"---"+d);// -160...-180---200---10

Conditional Operator (? :)

The only possible ternary operator in java is conditional operator

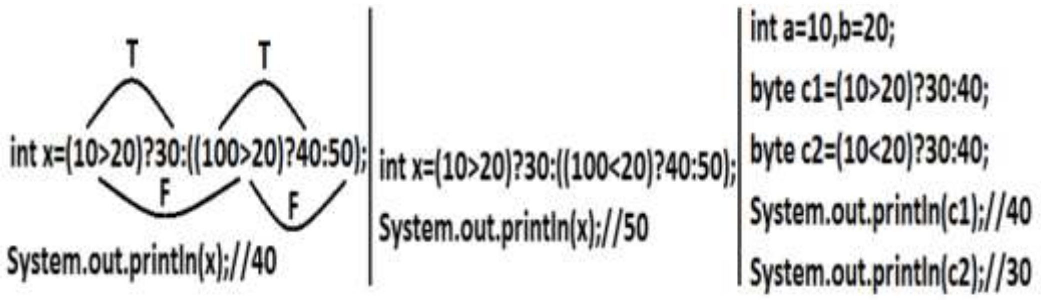
Ex 1 :

```
int x=(10>20)?30:40;

System.out.println(x); //40
```

Ex 2 :

```
int x=(10>20)?30:((40>50)?60:70);

System.out.println(x); //70
```

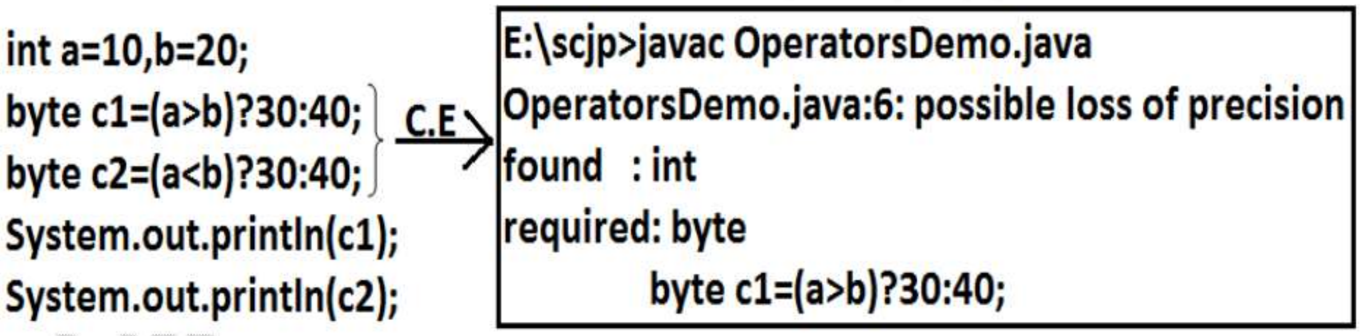
Nesting of conditional operator is possible



```
int x=(10>20)?30:((100>20)?40:50);
System.out.println(x);//40
```

```
int x=(10>20)?30:((100<20)?40:50);
System.out.println(x);//50
```

```
int a=10,b=20;
byte c1=(10>20)?30:40;
byte c2=(10<20)?30:40;
System.out.println(c1);//40
System.out.println(c2);//30
```



```
int a=10,b=20;
byte c1=(a>b)?30:40;
byte c2=(a<b)?30:40;
System.out.println(c1);
System.out.println(c2);
```

```
E:\scjp>javac OperatorsDemo.java
OperatorsDemo.java:6: possible loss of precision
found   : int
required: byte
            byte c1=(a>b)?30:40;
```

new operator :

1. We can use "new" operator to create an object.
2. There is no "delete" operator in java because destruction of useless objects is the responsibility of garbage collector.

[ ] operator:

We can use this operator to declare under construct/create arrays.