Operators
========
 1. increment and decrement operator
 2. arithmetic operator
 3. String concatination operators
 4. Relational operator
 5. Equality operator
 6. instance operator
 7. bitwise operator
 8. shortciruit operator
 9. typecast operator
 10. conditional operator
 11. assignement operator

increment
 a. pre-increment(y=++x)[increment and use]
 b. post-increment(y=x++)[use and increment]

decrement
 a. pre-decrement(y=--x)[decrement and use]
 b. post-decrement(y=x--)[use and decrement]

```
int x = 10;
y = ++x;(x = 11 , y =11)
y = x++;(x = 11 , y =10)
y = --x;(x = 9 , y =9)
y = x--;(x = 9 , y =10)
```

```
eg::
int x= 4;
int y=++x;
print x;//5
print y;//5
```

```
eg::
int x= 4;
int y=++4;//CE: increment can be applied only on variables not on direct literals.
print x;
print y;
```

```
eg::
int x= 4;
int y=++(++x);//CE: increment can be applied only on variables not on direct
literals.
print x;
print y;
```

```
eg::
final int x = 4;
++x;//CE
print x;
```

Note:: We can use final access modifiers on
      a. variable :: If we mark variable as final, the it would treated as "CompiletimeConstant"
                  CompileTimeConstant -> value will be known to compiler and these values should not be changed
                  through out the program.

b. class :: If we mark class as final, the those classes would not participate in "inheritance".
c. method :: If we mark method as final, the those methods cannot be "Overriden".

eg::
```
boolean b = true;
 b++;//CE: ++ operator won't work w.r.t boolean types.
print b;
```

What is the difference b/w b++ and b=b+1?

```
byte b = 5;
b++; //b = (byte)(b + 1);
print b;//6
```

```
byte b = 5;
b = b+1;//CE
print b;
```

Q>
```
class Test
{
	public static void main(String[] args)
	{
		int a=100;
		System.out.println(-a++);
	}
}
```

A. -101
B. 99
c. Compilation error
d. -100
e. -99

Q>
```
class Test
{
	public static void main(String[] args)
	{
		int a = 20;
		int var= --a * a++ + a-- - --a;
		System.out.println("a = " + a);
		System.out.println("var = " + var);
	}
}
```
A.
```
 a = 18
 var=363
```

B.
```
    a = 363
    var=363
```

C. Compilation Error

D.

```
  a = 25
var= 363
```

answer A

Q>

```
class Test
{
      public static void main(String[] args)
      {
            int i = 5;
            if (i++ < 6)
            {
                  System.out.println(i++);
            }
      }
}
```

A. 5
B. 6
C. Program executes succesfully but nothing is printed on to console
D. 7

Answer: B

String concatination
==========

=> On String '+' operator is used for concatination
=> if one operand is String and other operand is other type like int,float,double then it perform concatination.
=> if both operands are of number type then only '+' operator performs "Addition".

```
String a = "sachin";
int b= 10,c=20,d=30;
System.out.println(a+b+c+d);//sachin102030
System.out.println(b+c+d+a);//60sachin
System.out.println(b+c+a+d);//30sachin30
System.out.println(b+a+c+d);//10sachin2030
```

```
String a = "sachin";
int b= 10,c=20,d=30;
 a=b+c+d;//CE
 print a;
```

```
String a = "sachin";
int b= 10,c=20,d=30;
 b=a+c+d;//CE
 print b;
```

```
String a = "sachin";
int b= 10,c=20,d=30;
 b=b+c+d;
 print b;//60
```

RelationalOperator
==========

  <,<=
  >,>=

Output of relational operator is boolean type.
eg::

```
 System.out.println("sachin"< "kohli");//CE
 System.out.println(true<true); //CE
 System.out.println(10<10.5);//true
Note: Nesting of relational operator is not possible.
 System.out.println(10<20<30);//CE

Equality operator
=================
  a. ==(It is also called as comparison operator)
  b. !=

System.out.println(false == false);//true
System.out.println('a' == 97);//true
System.out.println(10 == 20);//false

Q> == operator on reference type would always compare the reference.
      It is used to check whethere both the reference are ponting to same object or
not
      If both are ponting to same object,then it would return true otherwise it
would return false.

Thread t1 =new Thread();
Thread t2 =new Thread();
Thread t3 = t1;
 System.out.println(t1 == t2);//false
 System.out.println(t1 == t3);//true

Note: To compare the reference of the object, there should be a relationship b/w 2
objects,if relationship doesnot exists
      then it would result in "CompiletimeError".

Thread t = new Thread();
  Object  o = new Object();
 String s = new String("sachin");
 System.out.println(t == o);//false
 System.out.println(o == s);//false
 System.out.println(s == t);//CE
 System.out.println(o == t);//false

                        Object
                          |
String,StringBuilder,StringBuffer,Number,Thread

String name = new String("sachin");
System.out.println(name == null);//false
System.out.println(null == null);//true

bitwise operator
===============
1. &(if both arguments are true,then result is true)
2. |(if atleast one argument is true,then result is true)
3. ^(if both are different arguments,then result is true otherwise it is false)


System.out.println(true&false);//false
System.out.println(true|false);//true
System.out.println(true^false);//true
System.out.println(4&5);//4
System.out.println(4|5);//5
```

```
System.out.println(4^5);//1

4 ===> 100
5 ===> 101

4&5 ==>100
4|5 ==>101
4^5 ==>001


bitwise compliment operator(~)
==============================
System.out.println(~true);//CE
System.out.println(~4);//-5

4 =>  0 0100
2's compliment of 4 is 1's compliment is 1.

~4 ==>1 1011[inverting bits,since number is negative,store it in 2's compliment
manner]
   ==>1 0100
           1
      =======
      1 0101

Boolean compliment operator(!)
==============================
System.out.println(!true);//false
System.out.println(!4);//Ce


Tomo topics :: operators and control statement and oops
```