ArithmeticOperators
+++++++++++++++++++
 => +,-,*,/,%

/ and % -> possiblity of getting ArithmeticException when operated on integral arithmetic type,but not on floating type.

/ => extracts the quotient and performs the operation.
% => extracts the remainder and performs the operation.

Nan(Not a Number) is a integral arithmetic(byte,short,int,long) there is no way to represent the undefined result, so it would throw
an Exception called "ArithmeticException".
But floating point arithmetic(double,float) there is a way to represent the undefined result, so the result would be "Nan".

eg#1.
```
System.out.println(0.0/0.0); //double/double -> double
System.out.println(-0.0/0.0);//double/double -> double
System.out.println(0/0);     //int/int -> int
```

Output
```
Nan
Nan
ArithmeticException : /by zero
```

Note:
 for any value of 'x' including NaN, the result will be false.

```
//<, <=, >, >= , ==, !=
System.out.println(10<Float.NaN);//false
System.out.println(10<=Float.NaN);//false
System.out.println(10>Float.NaN);//false
System.out.println(10>=Float.NaN);//false
System.out.println(10==Float.NaN);//false
System.out.println(Float.NaN == Float.NaN);//false

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

++
'+' operator applied on Strings in Java
++

'+' operator in java is refered as "Overloaded-Operator".
'+' operator will perform addition if both the operands are of numeric type(byte,short,int,long,float,double)
'+' operator will perform concatination, if one of the operand is of "String" type.

eg#1.
```
String name="sachin";
int a= 10,b=20,c=30;

//int + String => concatination(joining String)
System.out.println(a+b+c+name);//60sachin

System.out.println(name+a+b+c);//sachin102030
System.out.println(a+b+name+c);//30sachin30
System.out.println(a+name+b+c);//10sachin2030


eg#2.
String name="sachin";
int a= 10,b=20,c=30;

name = a+b+c;//CE:incompatible type
System.out.println(name);

eg#3.
String name="sachin";
int a= 10,b=20,c=30;

name=name+a+b;
c=a+b;

c=a+b+name; //CE: incompatible
System.out.println(a);
System.out.println(b);
System.out.println(c);

+++++++++++++++++++++++++++
RelationalOperator in Java
++++++++++++++++++++++++++
 <,<=,>,>=

=> We can apply relational operators only on primtive types except "boolean types".
=> we cannot apply relational operators only reference types(on objects)
=> Nesting of relational operator is not possible in java.

eg#1.
System.out.println(10>10.5);//false
System.out.println('a'>10.5);//true
System.out.println('z'>'a');//true
System.out.println(true>false);//CE
System.out.println("sachin">"kohli");//CE
System.out.println(10<20<30);//CE
System.out.println(10>20>30);//CE

++++++++++++++++
EqualityOperator
++++++++++++++++
 ==,!=
=> We can apply equality operators on primitive types including boolean types

eg#1.
System.out.println(10  == 20);//false
System.out.println('a' == 'b');//false
System.out.println('a' == 97.0);//true
System.out.println(false == false);//true


=> we can apply equality operators on reference type also.

eg#2.
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 use == operator on reference type, we need to check whether there exists a relationship b/w 2 operands.
=> If relationship exists, it should be parent-child relationship, otherwise it would result in "CompileTimeError".

eg#3.
```
Thread t = new Thread();
Object o = new Object();
String s = new String("sachin");
StringBuffer sb =new StringBuffer("dhoni");

System.out.println(t==o);//false
System.out.println(o==s);//false
System.out.println(s==t);//CE
System.out.println(s==sb);//CE
```

++++++++++++++++
Bitwise Operator
++++++++++++++++
1. & => If both the arguments are true, then only result in true.
2. | => If atleast one argument is true, then result is true.
3. ^ => If both arguments are of different type, then result in true.
4. ~ => It is negation operator.

&,|,^   => These operators can be applied on boolean and even on integral types.
~(bitwsise complement)   => This operator can be applied on integral types, but not on boolean types.
!(boolean complement)   => This operator can be applied only on boolean types, but not on integral types.

eg#1.
```
System.out.println(4&5);//4
System.out.println(4|5);//5
System.out.println(4^5);//1

System.out.println(~4);
System.out.println(~true);//CE

System.out.println(!true);
System.out.println(!4);//CE
```

Note:
=> Negative no will be stored inside the computer using 2's complements.
=> For a negative number MSB will be 1, where as for a positive number MSB will be 0.

++++++++++++++++++++++
ShortCircuit Operator
++++++++++++++++++++++
  &&, ||
These operators are exactly same as "& and |",but with small differences.

```
&,|
 => Both the arguments should be evaluated always
 => Performance is relatively slow.
 => It is applicable for both integral and boolean types.

&&,||
 => Second argument evaluation is optional.
 => Performance is relatively high
 => It is applicable only for boolean types, not for integral types.

Note:
 x  && y
  => y will be evaluated only if x is true,otherwise y won't be evaluated.
  => x is true, then y will be evaluated.

 x || y
  => y will be evaluated only if x is false, otherwise y won't be evaulated.
  => x is true, then y won't be evaulated.

eg#1.
int x = 10, y= 15;
if (++x < 10  || ++y>15)
      x++;
else
      y++;

System.out.println("x = "+x);//12
System.out.println("y = "+y);//16

Type casting operator
++++++++++++++++++++
 1. implicit/narrowing => compiler will automatically do(no loss of data)
 2. explicit/widening  => programmer should do(loss of data might happen)


=> if we work with floating point data and if we try to assign it to integral type
using explicit typecasting then the data after
   decimal value will be lost.

eg#1.
float x = 150.1234f;
int i =(int) x;
System.out.println(i);//150

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

=> While working with integral types, storing higher value in lower type using
explicit typecasting might lead to data loss.
eg#1.
int x = 150;
short s  =(short) x;
System.out.println(s);//150

//minRange + (result-maxRange-1)
/*
      =-128 + (150-127-1)
      =-128 + 150-128
      = -128+22
      = -106
*/
byte b = (byte)x;
System.out.println(b);//-106

eg#2.
double d= 130.456;
int x= (int)d;
System.out.println(x);//130

//minRange + (result-maxRange-1)
/*
      =-128 + (130-127-1)
      =-128 + 130-128
      = -128+2
      = -126
*/
byte b = (byte)d;
System.out.println(b);//-126

+++++++++++++++++++
Assignment Operator
+++++++++++++++++++
There are 3 types of assignment operator
      a.Simple assignment
            eg: int a= 10;
                int b= 20;
      b.Chained assignment
      c.Compound assignment


Chained assignment
eg#1
   int a,b,c,d;
   a= b= c= d= 10; //valid

eg#2.
  int a=b=c=d=10; //invalid
  System.out.println(a+" " + b + " "+c+" "+d);

eg#3.
 int b,c,d;
 int a=b=c=d=10;//valid
 System.out.println(a+" " + b+" " + c +" " +d);

eg#4.
 int a,b,c,d=10; // only d value is initialized
 System.out.println(d);//10

Compound assignment
++++++++++++++++++++
  += ,-=, /=, *=, %=
  &=,|=,^=

Note: In case of compound assignment operator, internally type casting will be
performed automatically by the JVM similar to
      increment or decrement operator.
```

```
eg#1.
 int a= 10;
  a+=20;
 System.out.println(a);

eg#2.
 byte b=10;
   b = b+1;//incompatible types: required : int,found : byte
  System.out.println(b);

eg#3.
 byte b = 10;
   b++; // b= (byte)(b+1);
 System.out.println(b);//11

eg#4
 byte b=10;
   b+=1;//b = (byte)(b+1);
  System.out.println(b);//11

eg#5.
 int a,b,c,d;
 a=b=c=d=20;
 a+=b-=c*=d/=2;
/*
      d= d/2; 20/2   = 10
      c= c*d; 20*10  = 200
      b= b-c; 20-200 =-180
      a= a+b; 20-180 =-160
*/
    System.out.println(a+" " +b +" " +c +" " +d);


++++++++++++++++++++
Conditional Operator
++++++++++++++++++++
 It is also called as "Ternary operator".
syntax:
    condition ? true : false

eg#1.
int x= 10==10 ? 100 : 500;
System.out.println(x);//100

eg#2.
int x= (10>20) ? 30 : (40>50) ? 60 : 70;
System.out.println(x);//70

eg#3.
int x= (10>20) ? 30 : (100>20) ? 40 : 50;
System.out.println(x);//40

eg#4.
byte c= (10 > 20) ? 30: 40; // byte c = 40;
byte d= (10 < 20) ? 30: 40; // byte d = 30;
System.out.println(c + " " + d); // 40  30

final int a= 10,b= 20;
byte c1= (a > b) ? 30: 40;// byte c = 40;
byte d1= (a < b) ? 30: 40;// byte d = 30;
System.out.println(c1+ " " + d1); // 40 30

int a1= 10,b1= 20;
byte c2= (a1 > b1) ? 30: 40;//CE
byte d2= (a1 < b1) ? 30: 40;//CE
System.out.println(c2 + " " + d2);
```

Snippets
++++++++
```
public class Test {
 public static void main(String [] args) {
    int x = 5;
    boolean b1 = true;
    boolean b2 = false;

   if ((x == 4) && !b2 )
    System.out.print("1 ");
   System.out.print("2 ");
   if ((b2 = true) && b1 )
     System.out.print("3 ");
 }
 }
```
What is the result?
A. 2
B. 3
C. 1 2
D. 2 3[Answer]
E. 1 2 3
F. Compilation fails.
G. An exception is thrown at runtime.

Q>
What will be the result of compiling and executing Test class?
```
public class Test {
    public static void main(String[] args) {
        System.out.println("Hello" + 1 + 2 + 3 + 4);
    }
}
```
A. Hello10
B. Hello19
C. Hello1234[Answer]
D. Hello10

3.
What will be the result of compiling and executing Test class?
```
public class Test {
    public static void main(String[] args) {
        System.out.println(1 + 2 + 3 + 4 + "Hello");
    }
}
```
A. 1234Hello
B. 10Hello[Answer]
C. 64Hello
D. 10 Hello

4.What will be the result of compiling and executing Test class?
```
public class Test {
    public static void main(String[] args) {
        System.out.println("Output is: " + 10 != 5);
    }
}
```
A. Output is : true
B. Output is : false
C. Compilation error
D. Output is : 10 !=5

Answer: C

5.What will be the result of compiling and executing Test class?
```
public class Test {
    public static void main(String[] args) {
        System.out.println("Output is: " + (10 != 5));
    }
}
```
A. Output is : true
B. Output is : false
C. Compilation error
D. Output is : 10 !=5

Answer: A

6.What will be the result of compiling and executing Bonus class?
```
public class Bonus {
    public static void main(String[] args) {
        int $ = 80000;
        String msg = ($ >= 50000) ? "Good bonus" : "Average bonus";
        System.out.println(msg);
    }
}
```
A. Good bonus[Answer]
B. Average bonus
C. compilation error
D. None of the above

7. What will be the result of compiling and executing Test class?
```
public 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 =25
  var=363
B. a= 363
  var=363
C. a=18
  var=363
D. compilation error

```
a = 19,20,19,18
int var = --a * a++ + a-- - --a;
        = 19  * 19  + 20  - 18
        = 361 + 2
        = 363
Answer : C
```

8.
What will be the result of compiling and executing Test class?

```
public class Test {
    public static void main(String[] args) {
        int a = 7;
        boolean res = a++ == 7 && ++a == 9 || a++ == 9; //line-n1
        System.out.println("a = " + a);
        System.out.println("res = " + res);
    }
}
```

```
A. a=10
   res=true
B. a=9
   res=true
C. a=10
   res=false
D. compilation error
```

```
a = 7,8,9
boolean res = (true && true)|| a++ == 9
          = true  || a++==9
          = true
```

Answer: B