ជំពូកទី៣

សញ្ញាណនព្វន្តលេខ

១. **<u>សញ្ញាណនព្វន្តលេខគណិត</u>**(*Arithmetic operators*):

សញ្ញាណនព្វន្តលេខគណិត ត្រូវបានប្រើនៅក្នុងកន្សោមលេខគណិតវិទ្យា ដែលដូចទៅនឹងការប្រើនៅក្នុងពិជគណិតដែរ។ តារាងខាងក្រោមនេះរាយសញ្ញាណនព្វន្តលេខគណិត:

<u>សញ្ញាណនព្វន្តលេខ</u>	**<u>មុខងារ</u>**
+	បូក (Addition)
-	ដក (Subtraction or unary minus)
*	គុណ(Multiplication)
/	ចែក (Division)
%	ចែករកសំណល់ (Modulus)
++	កំនើនមួយឯកតា (Increment)
+=	កំនត់តំលៃនៃផលបូក (Addition assignment)
-=	កំនត់តំលៃនៃផលដក (Subtraction assignment)
*=	កំនត់តំលៃនៃផលគុណ(Multiplication assignment)
/=	កំនត់តំលៃនៃផលចែក (Division assignment)
%=	កំនត់តំលៃនៃផលចែកសំណល់ (Modulus assignment)
--	តំហយមួយឯកតា (Decrement)

តួលេខលៃនសញ្ញាណនព្វន្តត្រូវតែជាតួលេខ។ យើងមិនអាចប្រើវាទៅលើប្រភេទ Boolean ប៉ុន្តែយើងអាចប្រើវាជាមួយ char បានដោយសារ char នៅក្នុងភាសា Java ជាផ្នែកមួយនៃ int។

១.១ **<u>សញ្ញាណនព្វន្តលេខគណិតសំខាន់ៗ</u>** (The basic arithmetic operators)

ការធ្វើប្រមាណវិធីលេខ បូក ដក គុណនិង ចែក សុទ្ធតែមានលក្ខណៈជាលេខ។ សញ្ញាណនព្វន្តលេខដកមានលក្ខណៈជា unary (ប្រើតែមួយលេខ) ដែលអោយតំលៃអវិជ្ជមានដល់តួលេខរបស់វា។ គួរចងចាំថាកាលណាលេខចែក ប្រើជាមួយចំនួនគត់ នោះលទ្ធផលរបស់វាគ្មានផ្នែកទសភាគទេ។

កម្មវិធីខាងក្រោមនេះ បង្ហាញពីការប្រើសញ្ញាណលេខគណិត។ វាបញ្ជាក់ពីភាពខុសគ្នារវាងការចែកចំនួនទសភាគ និង ការចែកចំនួនគត់ផងដែរ។

```
//demonstrate the basic of arithmetic operators
class BasicMath{
        public static void main(String Args[]){
                //arithmetic using integers
                System.out.println("Integer Arithmetic");
                int a = 1 + 1;
                int b = a * 3;
                int c = b / 4;
                int d = c - a;
                int e = -d;
                System.out.println("a = " + a);
                System.out.println("b = " + b);
                System.out.println("c = " + c);
                System.out.println("d = " + d);
                System.out.println("e = " + e);

                //arithmetic using double
                double da = 1 + 1;
                double db = da * 3;
                double dc = db / 4;
                double dd = dc - a;
                double de = -dd;
                System.out.println("da = " + da);
                System.out.println("db = " + db);
                System.out.println("dc = " + dc);
                System.out.println("dd = " + dd);
                System.out.println("de = " + de);
        }
}
```

១.២ **សញ្ញាណនព្វន្តលេខចែករកសំណល់**(The modulus operator)

សញ្ញាណនព្វន្តលេខចែករកសំនល់ (%) អោយសំណល់នៃការធ្វើប្រមាណវិធីចែក។ វាអាចប្រើជាមួយចំនួនទសភាគ ក៍ដូចជាចំនួនគត់ផងដែរ។ ត្រង់នេះវាខុសពី C និង C^{++} ដែល % អាចប្រើបានចំពោះតែចំនួនគត់ប៉ុណ្ណោះ។ ចូរពិនិត្យមើលកម្មវិធីខាងក្រោម:

```
//Demonstrate of Modulus operator
class Modulus{
        public static void main(String Args[]){
                int x=42;
                double y=43.3;
                System.out.println("x mod 10= " + x % 10);
                System.out.println("y mod 10= " + y % 10);
        }
}
```

១.៣ **សញ្ញាណនព្វន្តលេខកំនត់តំលៃ**(Arithmetic assignment operators)

Java ផ្តល់នូវសញ្ញាណនព្វន្តលេខពិសេស ដែលអាចប្រើរួមបញ្ចូលសញ្ញាណនព្វន្តលេខគណិតជាមួយការកំនត់តំលៃ។ វាមានសញ្ញាកំនត់តំលៃ សំរាប់គ្រប់សញ្ញាណនព្វន្តលេខគណិតប្រភេទ binary (សញ្ញាណនព្វន្តលេខគណិតមួយអមដោយពីរតួលេខ) ទាំងអស់។ ដូច្នេះឃ្លាណាមួយមានទំរង់:

var = var op expression;

អាចសរសេរជា:

var op = expression;

ឧទាហរណ៍ a = a +4; យើងងអាចសរសេរទៅជា a + = 4;

នេះជាឧទាហរណ៍មួយទៀត a = a % 2; អាចសរសេរជា a % = 2; កម្មវិធីខាងក្រោមនេះ បង្ហាញពីការប្រើប្រាស់សញ្ញាណនព្វន្តលេខកំនត់តំលៃ *op*= ជាច្រើន:

```
//Demonstate several assignment operators.
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);
        }
}
```

១.៤ <u>សញ្ញាណនព្វន្តលេខកំនើន និង តំហាយមួយឯកតា</u>(Increment and decrement)

សញ្ញា ++ និង – គឺជាសញ្ញាណនព្វន្តលេខ កំនើន និង តំហាយមួយឯកតានៅក្នុង Java។ សញ្ញាណនព្វន្តលេខកំនើនមួយឯកតា បង្កើនតំលៃអោយតួលេខរបស់វា គឺ មួយ។ សញ្ញាណនព្វន្តលេខតំហាយមួយឯកតា បន្ថយតំលៃនៃតួលេខរបស់វា គឺ មួយ។ ចូរពិនិត្យមើលឧទាហរណ៍ខាងក្រោម:

```
//Demonstrate ++
class IncDec{
        public 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);
        }
}
```

២. សញ្ញាណនព្វន្តលេខគោលពីរ(*The bitwise operators*):

Java មានសញ្ញាណនព្វន្តលេខគោលពីរច្រើន ដែលអាចប្រើជាមួយនិងប្រភេទចំនួនគត់: long, int, short, និង byte។ សញ្ញាណនព្វន្តលេខទាំងនេះ ធ្វើសកម្មភាពលើ bits ដោយឡែកនៃតួលេខរបស់វា។ សញ្ញាណទាំងនេះត្រូវបានសង្ខេបក្នុងតារាងខាងក្រោម:

សញ្ញាណនព្វន្តលេខ	មុខងារ
~	Bitwise unary NOT
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
>>	Shift right
>>>	Shift right zero fill
<<	Shift left
&=	Bitwise AND assignment
!=	Bitwise OR assignment
^=	Bitwise exclusive OR assignment
>>=	Shift right assignment
>>>=	Shift right zero fill assignment
<<=	Shift left assignment

សញ្ញាណនព្វន្តលេខគោលពីរ ប្រើ bits ក្នុងចំនួនគត់។ ប្រភេទចំនួនគត់ទាំងអស់ (លើកលែងតែ char) គឺជាចំនួនគត់មានសញ្ញា។ នេះមានន័យថា វាអាចតាងអោយតំលៃអវិជ្ជមានផង និង តំលៃវិជ្ជមានផង។ Java ប្រើការបំលែង code ជា tow's complement ដែលមានន័យថា ចំនួនអវិជ្ជមាន បង្ហាញឡើងដោយការបញ្ច្រាស់គ្រប់ bits ទាំងអស់ក្នុងតំលៃមួយ (ប្តូរ 1 ទៅ 0 ហើយច្រាស់មកវិញ)។ រួចហើយបូក 1 ទៅអោយលទ្ធផល។ ឧទាហរណ៏-42 តាងដោយការបញ្ច្រាស់ bits ទាំងអស់របស់ 42 ឬ 00101010 ទទួលបាន 11010101 រួយហើយបូកនិង 1 ដែលអោយលទ្ធផលជា 11010110 ឬ -42។ ដើម្បីប្តូរ code ជាចំនួនអវិជ្ជមាននោះ ដំបូងត្រូវបញ្ច្រាស់រាល់ bits ទាំងអស់ រួចហើយបូកនិង 1។ -42 ឬ 11010110 បានបញ្ច្រាស់អោយទៅជា 00101001 ឬ 41 រួចបូក 1 បានជា 42។

២.១ សញ្ញាណនព្វន្តលេខគោលពីរបែបតក្ក(The bitwise logical operator)

សញ្ញាណនព្វន្តលេខគោលពីរបែបតក្កមាន &, |, ^, និង ~។

២.១.១ សញ្ញាណនព្វន្តលេខគោលពីរឈ្មោះចំមិន (The bitwise NOT)

សញ្ញាណនព្វន្តលេខគោលពីរឈ្មោះចំមិន "~" ជាសញ្ញាណនព្វន្តលេខបែប unary អាចហៅម្យ៉ាងទៀតថា bitwise complement ប្រើសំរាប់បញ្ច្រាស់គ្រប់ bits ទាំងអស់នៃតួលេខរបស់វា។ ឧទាហរណ៏ លេខ 42 ដែលមានលំនាំ bit ដូចខាងក្រោម:

00101010 អោយទៅជា 11010101 បន្ទាប់ពីសញ្ញាណនព្វន្តខាងលើធ្វើប្រមាណវិធីរួច។

២.១.២ សញ្ញាណនព្វន្តលេខគោលពីរឈ្មោះប៉ន់និង (The bitwise AND)

សញ្ញាណនព្វន្តលេខគោលពីរឈ្មោះប៉ន់និង “&” អោយផល 1 bit បើសិនជាតួលេខទាំងពីរ ជាលេខ 1 ហើយក្រៅពីករណីនេះ អោយផល 0។ ឧទាហរណ៍:

```
  00101010      42
& 00001111      15
----------
  00001010      10
```

២.១.៣ សញ្ញាណនព្វន្តលេខគោលពីរឈ្មោះប៉ប (The bitwise OR)

សញ្ញាណនព្វន្តលេខគោលពីរឈ្មោះប៉ប “|” រួមបញ្ចូល bit នៅក្នុងតួលេខ។ បើសិនជា bit ណាមួយ ក្នុងតួលេខជាលេខ 1 នោះលទ្ធផលគឺ 1។ ចូរសង្កេត:

```
  00101010      42
| 00001111      15
----------
  00101111      47
```

២.១.៤ សញ្ញាណនព្វន្តលេខគោលពីរឈ្មោះប៉ **XOR** (The bitwise XOR)

សញ្ញាណនព្វន្តលេខគោលពីរឈ្មោះប៉ XOR “^” ផ្សំបញ្ចូលលេខបើសិនជាតួលេខណាមួយជាលេខ 1 នោះពេលនោះលទ្ធផលគឺ 1។ ផ្ទុយទៅវិញលទ្ធផលគឺ 0។

```
  00101010      42
^ 00001111      15
----------
  00100101      37
```

២.១.៥ ការប្រើសញ្ញាណនព្វន្តលេខគោលពីរបែបតក្ក (Using the bitwise logical operators)

កម្មវិធីខាងក្រោមនេះ បង្ហាញពីការប្រើប្រាស់ សញ្ញាណនព្វន្តលេខគោលពីរបែបតក្ក:

```
//Demonstrate the bitwise logical operators
class BitLogic{
        public static void main(String Args[]){
                String binary[]={
                        "0000","0001","0010","0011","0100","0101",
                        "0110","0111","1000","1001","1010","1011",
                        "1100","1101","1110","1111"
                };
                int a = 3;
                int b = 6;
                int c = a;
                int d = a & b;
                int e = a ^ b;
                int f = (~a & b)|(a & ~b);
                int g = ~a & 0x0f;

                System.out.println("             a= " + binary[a]);
                System.out.println("             b= " + binary[b]);
```

```
                System.out.println("          a|b= " + binary[c]);
                System.out.println("          a & b= " + binary[d]);
                System.out.println("          a ^ b= " + binary[e]);
                System.out.println("          ~a & b)|(a & ~b) = " + binary[f]);
                System.out.println("          ~a & 0x0f = " + binary[g]);
        }
}
```

២.១.៦ ការប្រើសញ្ញាណនព្វន្តលេខគោលពីរអំពីលឆ្វេង (The left shift operator)

សញ្ញាណនព្វន្តលេខគោលពីរអំពីលឆ្វេង “<<” រំកិលគ្រប់ bits ទាំងអស់នៅក្នុងតំលៃមួយ ទៅកាន់ខាងឆ្វេង តាមចំនួនខ្ទង់ដែលបានកំនត់។ វាមានទំរង់ទូទៅ:

value << num;

ក្នុងនេះ num កំនត់ចំនួនខ្ទង់តាមទិសដៅរំកិលទៅឆ្វេងនៃតំលៃក្នុងមួយ value។ បានន័យថា << ផ្លាស់ទីគ្រប់ bits ទាំងអស់ក្នុងតំលៃដែលបានកំនត់ទៅខាងឆ្វេង តាមចំនួនខ្ទង់ bits បញ្ជាក់ដោយ num។ ចំពោះ bits លំដាប់ខ្ពស់នៅខាងឆ្វេងបង្អស់ ត្រូវរំកិលហួស (នឹងត្រូវបាត់បង់) ហើយលេខសូន្យ ត្រូវបំពេញនៅខាងស្តាំ។ នេះមានន័យថា កាលណារំកិលមកខាងឆ្វេងបានអនុវត្តទៅលើលេខ int មួយ, bits ត្រូវបាត់បង់នៅពេលវារំកិលហួសខ្ទង់ bit ត្រង់ 31។ កាលណាតួលេខជា long ពេលនោះ bit ត្រូវបាត់បង់នៅត្រង់ bit ខ្ទង់ 63។

ការបង្កើនប្រភេទទិន្នន័យដោយស្វ័យប្រវត្តរបស់ Java ធ្វើអោយគេទទួលលទ្ធផលដែលគិតមិនដល់នៅពេលដែលយើងរំកិលតំលៃប្រភេទ byte និង short។ តំលៃអវិជ្ជមាន byte ឬ short មួយនឹងរក្សាសញ្ញា នៅពេលវាបង្កើនទៅជា int។ ហេតុនេះ bits លំដាប់ខ្ពស់នឹងត្រូវបំពេញដោយលេខ 1។ តាមរយៈហេតុផលនេះ ដើម្បីធ្វើការរំកិលមកឆ្វេងលើតំលៃប្រភេទ byte ឬ short នោះយើងត្រូវបោះបង់ចោលនូវ byte លំដាប់ខ្ពស់នៃលទ្ធផល int។ កម្មវិធីខាងក្រោមនេះបង្ហាញពីទស្សនៈនេះ:

```
//Left shifting a byte value
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);
        }
}
```

លទ្ធផលនៃកម្មវិធីនេះ បង្ហាញអោយឃើញដូចខាងក្រោម:

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

ដោយសារ a ត្រូវតំលើងទៅជា int ចំពោះគោលបំណងនៃការរង្វាយតំលៃ។ ការរំកិលមកឆ្វេងនូវតំលៃ 64 (0100 0000) ពីរដងអោយលទ្ធផល i ដែលមានតំលៃ 256 (1 0000 0000)។ ទោះជាយ៉ាងណាក៏ដោយ តំលៃក្នុង b ផ្ទុក 0 ព្រោះបន្ទាប់ពីរំកិល, byte លំដាប់ទាបគឺ 0។ វាមានតែមួយ bit ដែលត្រូវរំកិលចេញក្រៅ។

ដោយសារតែការរំកិលមកឆ្វេងម្តងៗ ធ្វើអោយមានឥទ្ធិពលបង្កើនតំលៃដើមទ្វេរនោះ អ្នកសរសេរកម្មវិធី ប្រើជាញឹកញាប់នូវលក្ខណៈនេះ ដូចទៅនឹងការគុណនឹង 2 ដោយមានប្រសិទ្ធិភាព ប៉ុន្តែយើងត្រូវយកចិត្តទុកដាក់អោយម៉ត់ចត់។ បើសិនជាយើងរំកិល 1 bit ទៅក្នុងទីតាំងលំដាប់ខ្ពស់ (ត្រង់ bit ទី 31 ឬ bit ទី 64), តំលៃនឹងប្តូរជាអវិជ្ជមាន។

២.១.៧ ការប្រើសញ្ញាលទ្ធន្តលេខគោលពីរំកិលស្តាំ (The right shift operator)

សញ្ញាណនព្វន្តលេខគោលពីរំកិលស្តាំ ">>" រំកិលគ្រប់ bits ទាំងអស់នៅក្នុងតំលៃមួយ ទៅកាន់ខាងស្តាំ តាមចំនួនដងដែលបានកំនត់។ វាមានទំរង់ទូទៅ:

value >> num;

ដែល num កំនត់ចំនួនខ្ទង់ដែលត្រូវរំកិលទៅស្តាំ ក្នុង value។ បានន័យថា >> ផ្លាស់ទីគ្រប់ bits ទាំងអស់ក្នុងតំលៃដែលបានកំនត់ទៅខាងស្តាំ តាមចំនួនខ្ទង់ bit បញ្ជាក់ដោយ num។

code ខាងក្រោមនេះ រំកិលតំលៃ 32 មកខាងស្តាំពីរខ្ទង់ អោយលទ្ធផលទៅកាន់ a កំនត់ដោយលេខ 8:

```
int a = 32;
a = a >> 2; // a now contains 8
```

ចូរសង្កេតមើលប្រមាណវិធីដូចគ្នាក្នុង binary វាបង្ហាញអោយឃើញកាន់តែច្បាស់ពីការកើតឡើងបែបនេះ។

```
0010 0011        35
>> 2
0000 1000        8
```

កាលណាយើងធ្វើការរំកិលមកស្តាំ bit នៅខាងឆ្វេងបង្អស់ត្រូវបំពេញដាក់មកស្តាំ តាមបន្ទុកតំលៃមុនរបស់ bit នៅខាងឆ្វេងបង្អស់។ នេះហៅថា *sign extension* ហើយប្រើសំរាប់រក្សាសញ្ញាអវិជ្ជមាន កាលណាយើងរំកិលមកស្តាំ។ ឧទាហរណ៍-8 >> 1 គឺ -4 ដែលមានទំរង់ binary ដូចខាងក្រោម:

```
1111 1000        -8
>>1
1111 1100        -4
```

វាមានចំនុចគួរអោយកត់សំគាល់ថា បើយើងរំកិល -1 មកស្តាំ លទ្ធផលនៅតែ -1 ជានិច្ច ដោយសារតែ *sign extension* នាំមកនូវសញ្ញាច្រើនឡើងនៅក្នុង bit លំដាប់ខ្ពស់។

២.១.៨ ការប្រើសញ្ញាលទ្ធន្តលេខគោលពីរំកិលស្តាំមិនគិតសញ្ញា (The unsigned right shift operator)

ពេលណាយើងធ្វើការរំកិល ដែលមិនតាងអោយតំលៃលេខនោះ យើងមិនត្រូវការ *sign-extension* នោះឡើយ។ ស្ថានភាពនេះប្រើជាទូទៅ នៅពេលយើងធ្វើការជាមួយតំលៃ pixel ឬ graphic។ នៅក្នុងករណីនេះ យើងធ្វើការរំកិលដោយបំពេញលេខ 0 ទៅក្នុង bit លំដាប់ខ្ពស់ ដោយមិនគិតពីតំលៃចាប់ផ្តើមរបស់វាជាអ្វីនោះឡើយ។ លក្ខណៈនេះហៅថា unsigned shift។ ដូច្នេះយើងនឹងប្រើសញ្ញានព្វន្តលេខគោលពីរំកិលស្តាំមិនគិតសញ្ញា

(>>>) ដែលជានិច្ចជាកាលបំពេញលេខ 0 ទៅក្នុង bits លំដាប់ខ្ពស់។

Code ខាងក្រោមនេះបង្ហាញពីការប្រើ >>>។ ក្នុងនោះ a ត្រូវបានកំនត់តំលៃ -1 ដែលកំនត់គ្រប់ 32 bits ទៅជា 1 ក្នុងទំរង់ binary។ ពេលនោះតំលៃនេះត្រូវរំកិលមកស្តាំចំនួន 24 bits រួចហើយបំពេញលេខ 0 ដោយមិនគិតសញ្ញា sign-extension។ ដូច្នេះកំនត់ a ទៅជា 255។

```
int a = -1;
a = a >>> 24;
```

ខាងក្រោមនេះជាប្រមាណវិធីដូចគ្នាមួយទៀត ដែលសរសេរជាលេខទំរង់ប្រព័ន្ធគោលពីរ ដើម្បីបង្ហាញបន្ថែមនូវអ្វី ដែលកើតឡើង:

```
1111 1111  1111 1111  1111 1111  1111 1111          -1 ទំរង់ជាលេខគោលពីរ និង ប្រភេទ int
>>> 24
0000 0000  0000 0000  0000 0000  1111 1111          255 ទំរង់ជាលេខគោលពីរ និង ប្រភេទ int
```

សញ្ញាណនព្វន្តលេខ >>> មិនសូវមានប្រយោជន៏ដូចយើងចង់បាននោះទេ ដោយសារវាប្រើសំរាប់ តែតំលៃ 32 និង 64 bits។ ចូរចងចាំថា តំលៃតូចជាងត្រូវតំលើងទៅជា int ដោយស្វ័យប្រវត្តក្នុងកន្សោមលេខ។ នេះមានន័យថា sign-extension កើតឡើង ហើយការរំកិល នឹងចាប់ផ្តើមអនុវត្តនៅលើតំលៃ 32 bits ជាជាងលើតំលៃ bit ឬ 16 bits។ នោះគេអាចរំពឹងទុកថា ការរំកិលមកស្តាំដោយមិនគិតសញ្ញានៅលើតំលៃ byte ដើម្បីបំពេញលេខ 0 ចាប់ផ្តើមនៅត្រង់ bit 7។ ប៉ុន្តែនេះមិនមែនជាករណីទេ តាមពិតដោយសារតំលៃ 32 bits ដែលត្រូវធ្វើការរំកិល។

២.១.៩ ការកំនត់តំលៃ សញ្ញាណនព្វន្តលេខគោលពីរ (Bitwise operator assignment)

សញ្ញាណនព្វន្តលេខគោលពីរទាំងអស់ មានទំរង់អក្សរកាត់ ស្រដៀងគ្នាទៅនឹងសញ្ញាណនព្វន្តលេខគណិតដែរ ហើយផ្សំឡើងដោយសញ្ញាណនព្វន្តគោលពីរ ជាមួយនឹងសញ្ញាកំនត់តំលៃ។ ឧទាហរណ៏ឃ្លាពីរខាងក្រោមនេះ រំកិលតំលៃនៅក្នុង a មកស្តាំចំនួន 4 bits ហើយសមមូលនឹង:

```
a = a >> 4;
a >> = 4;
```

កម្មវិធីខាងក្រោមនេះ បង្កើតអញ្ញាត្តិចំនួនគត់ពីរបី រួចហើយប្រើទំរង់អក្សរកាត់នៃការកំនត់តំលៃសញ្ញាណនព្វន្តលេខគោលពីរ ដើម្បីអនុវត្តទៅលើអញ្ញាត្តិទាំងនោះ:

```
class OpBitEquals{
        public static void main(String Args[]){
                int a = 1, b = 2, 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);
        }
}
```

៣. <u>សញ្ញាណនព្វន្តលេខធៀប</u>(*Relational operators*):

សញ្ញាណនព្វន្តលេខធៀប កំនត់ទំនាក់ទំនងគ្នា រវាងតួលេខមួយ និង តួលេខមួយទៀត។ ជាពិសេស វាកំនត់ភាពស្មើគ្នា និង លំដាប់។ សញ្ញាណនព្វន្តលេខធៀប បានបង្ហាញដូចខាងក្រោមនេះ:

<u>សញ្ញាណនព្វន្តលេខ</u>	<u>មុខងារ</u>
==	ស្មើគ្នានឹង
!=	មិនស្មើគ្នានឹង
>	ធំជាង
<	តូចជាង
>=	ធំជាង ឬ ស្មើគ្នានឹង
<=	តូចជា ឬ ស្មើគ្នានឹង

លទ្ធផលនៃប្រមាណវិធីនេះ ជាតំលៃ boolean។ សញ្ញាណនព្វន្តលេខធៀប ប្រើជាញឹកញាប់ក្នុងកន្សោម ផ្ទៀងផ្ទាត់លក្ខ័ណ្ឌ ដូចជា ឃ្លួ if និង ឃ្លាដំណើរវង្វិល (loop) ផ្សេងទៀត។

ដូចបាននិយាយរួចមកហើយថា លទ្ធផលបានមកពីសញ្ញានព្វន្តលេខធៀប គឺជាតំលៃ boolean។ ឧទាហរណ៍ code ខាងក្រោមនេះគឺពិតជាត្រឹមត្រូវ:

```
int a = 4;
int b = 1;
boolean c = a < b;
```

ក្នុងករណីនេះ លទ្ធផលនៃ a < b (គឺ false) ត្រូវផ្ទុកនៅក្នុង c។ ចំពោះអ្នកដែលមានចំនេះជឿង C/C^{++} ចូរកត់សំគាល់ឃ្លាខាងក្រោមនេះ:

```
int done;
//....
if ( ! done )      ....// valid in C/C++
if ( done )        ....// but not valid in Java
```

នៅក្នុងភាសា Java ឃ្លានេះត្រូវសរសេរដូចខាងក្រោម:

```
if ( done == 0 )  ....//This is Java's style
if ( done != 0 )  ....
```

ហេតុផលគឺ Java មិនកំនត់តំលៃ True និង False ដូចគ្នានឹង C/C^{++} ទេ។ នៅក្នុង C/C^{++} true គឺជាតំលៃមិនសូន្យណាមួយ ហើយ fasle គឺ សូន្យ។ នៅក្នុង Java, true និង false គឺជាតំលៃមិនមែនជាលេខ ដែលមិនពាក់ព័ន្ធលេខសូន្យ ឬ មិនសូន្យនោះទេ។ ដូច្នេះដើម្បីផ្ទៀងផ្ទាត់លេខសូន្យ ឬ លេខមិនសូន្យនោះ យើងត្រូវប្រើសញ្ញាណនព្វន្តលេខធៀប មួយឬច្រើន ដោយបញ្ជាក់អោយច្បាស់។

៤. <u>សញ្ញាណនព្វន្តលេខបែបតក្កប្រភេទ **Boolean**</u>(*Relational operators*)**:**

សញ្ញាណនព្វន្តលេខបែបតក្កប្រភេទ boolean ដែលបានបង្ហាញនៅពេលនេះ ធ្វើប្រមាណវិធីតែចំពោះតួលេខ boolean ប៉ុណ្ណោះ។ គ្រប់សញ្ញាណនព្វន្តលេខបែបតក្កទាំងអស់ បន្សំពីរតំលៃប្រភេទ boolean ដើម្បីបង្កជាលទ្ធផលមួយ ដែលមានតំលៃប្រភេទជា boolean។

<u>**សញ្ញាណនព្វន្តលេខ**</u>	<u>**មុខងារ**</u>
&	Logical AND
|	Logical OR
^	Logical OR (Exclusive OR)
||	Short-curcuit OR
&&	Short-circuit AND
!	Logical unary NOT
&=	AND assignment
|=	OR assignment
^=	XOR assignment
==	Equal to
!=	Not equal to
?:	Ternary if-then-else

សញ្ញាណនព្វន្តលេខបែបតក្កប្រភេទ boolean គឺ &, |, និង ^ ធ្វើប្រមាណវិធីលើតំលៃ boolean ដូចគ្នាទៅនឹងប្រមាណវិធីលើ bits នៃចំនួនគត់មួយ។ សញ្ញាណនព្វន្តលេខបែបតក្ក ! បញ្ច្រស់សភាព *boolean: !true = false* និង *!false = true*។

នេះជាកម្មវិធីមួយ ដែលស្ទើរតែដូចកម្មវិធី BitLogic ដែលបានបង្ហាញកាលពីមុន តែវាធ្វើប្រមាណវិធីលើតំលៃតក្ក boolean ជំនួស bits លេខគោលពីរវិញ:

```
//Demonstrate the boolen logical operator
class BoolLogic{
        public static void main(String Args[]){
                boolean a = true;
                boolean b = false;
                boolean c = a | b;
                boolean d = a & b;
                boolean e = a ^ b;
                boolean f = ( !a & b ) ! ( a & !b );
                boolean g = !a;

                System.out.println("                    a = " + a);
                System.out.println("                    b = " + b);
                System.out.println("              a | b = " + c);
                System.out.println("              a & b = " + d);
                System.out.println("          a ^ b = " + e);
                System.out.println("( ! a & b ) | ( a & !B ) = " + f);
                System.out.println("                        !a = " + g);
        }
}
```

៥. <u>សញ្ញាណព្វន្តលេខកំនត់តំលៃ</u>(*The assignment operator*)**:**

សញ្ញាណនព្វន្តលេខកំនត់តំលៃ គឺជាសញ្ញាស្មើតែមួយ =។ ទំរង់ខាងក្រោមជាសញ្ញាណនព្វន្តលេខកំនត់តំលៃ:

var = expression;

ក្នុងនោះ ប្រភេទទិន្ន័យនៃ var ត្រូវតែស៊ីគ្នាជាមួយនឹងប្រភេទទិន្ន័យនៃកន្សោមលេខ។

សញ្ញាណនព្វន្តលេខកំនត់តំលៃមានលក្ខណៈមួយគួរអោយចាប់អារម្មណ៍ ហើយយើងគួរតែដឹងនោះគឺ វាអាចអោយយើងកំនត់តំលៃតគ្នា។ ឧទាហរណ៍ ចូរសង្កេតឃ្លាខាងក្រោមនេះ:

```
int x, y, z;
x = y = z = 100;          //set x , y, and z to 100
```

៦. <u>សញ្ញាណព្វន្តលេខ?</u>(*The ? operator*)**:**

Java បានបញ្ចូលសញ្ញាណនព្វន្តលេខពិសេសប្រភេទ ternary (three-way) ដែលអាចជំនួសអោយឃ្លា if-then-else។ សញ្ញាណនេះមាននិមិត្តសញ្ញា **?** ហើយវាតំណើរការនៅក្នុងភាសា Java ដូចគ្នាច្រើនទៅក្នុងភាសា C/C^{++}។ សញ្ញា ? នេះមានទំរង់ទូទៅ:

expression1 ? expression2 : expression3;

ក្នុងនោះ expression1 ជាកន្សោមមួយដែលអោយតំលៃជា boolean។ បើ expression1ពិត ពេលនោះexpression 2 ត្រូវរង្វាយតំលៃ, ផ្ទុយទៅវិញ expression3 ត្រូវរង្វាយតំលៃ។ លទ្ធផលនៃប្រមាណវិធី ? គឺជាកន្សោមវាយតំលៃនោះ។ expression2 និង expression3 ទាំងពីរត្រូវការអោយតំលៃក្នុងប្រភេទទិន្ន័យដូចគា តែមិនអាចជា void បានទេ។ ខាងក្រោមនេះជាឧទាហរណ៍នៃការប្រើ ?:

```
ratio = denom ==0 ? 0 : num / denom;
```

កម្មវិធីខាងក្រោមនេះ បង្ហាញពីការប្រើប្រាស់ ? ដើម្បីទទួលតំលៃដាច់ខាតនៃអញ្ញាត្តិ:

```
//Demonstrate ?:
class Ternary{
        public static void main(String Args[]){
                int i, k;

                i = 10;
                k = i < 0 ? -i : i;
                System.out.println("Absolute value of");
                System.out.println(i + " is " + k);

                i = -10;
                K = i < 0 ? -i : i;
                System.out.println("Absolute value of");
                System.out.println(i + " is " + k);
        }
}
```

៧. <u>ការប្រើសញ្ញាវង់ក្រចក</u>(*Using parentheses*):

សញ្ញាវង់ក្រចក លើកំពស់អទ្ធិភាពនៃសញ្ញាណនព្វន្តលេខដែលនៅក្នុងវា។ លក្ខណៈនេះមានសារៈសំខាន់ ដើម្បីបានលទ្ធផលដែលយើងត្រូវការ។ ឧទាហរណ៍ ចូរសង្កេតមើលកន្សោមលេខខាងក្រោម:

```
a >> b + 3;          a >> ( b + 2 )          ( a >> b ) + 3
```

ចំនុចមួយទៀត គឺសញ្ញាវង់ក្រចក (លើសលុប ឬ មិនលើសលុប) មិនបំបែកសកម្មភាពនៃកម្មវិធីរបស់យើងឡើយ។ ដូច្នេះបន្ថែមសញ្ញាវង់ក្រចក មិនធ្វើអោយប៉ះពាល់ដល់កម្មវិធីឡើយ ផ្ទុយទៅវិញវាជួយសំរួលដល់ការអាន។

អទ្ធិភាពខ្ពស់បំផុត			
()	**[]**	**.**	
++	--	~	!
*	/	%	
+	-		
>>	>>>	<<	
>	>=	<	<=
==	!=		
&			
^			
|			
&&			
||			
?:			
=	op=		
អទ្ធិភាពទាបបំផុត			

នេះជាតារាងបង្ហាញពីអទ្ធិភាពនៃសញ្ញាណនព្វន្តលេខរបស់ភាសា Java។