Arithmetic Operators

DSCE
Dept. of Information Science & Engg

```
// Demonstrate the basic 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 doubles
System.out.println("\nFloating Point Arithmetic");
```

Cntd..,

DSCE
Dept. of Information Science & Engg

```
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);
}
}
```

Output:

Integer Arithmetic a = 2
b = 6
c = 1
d = -1
e = 1
Floating Point Arithmetic
da = 2.0
db = 6.0
dc = 1.5
dd = -0.5
de = 0.5

The Modulus Operator

DSCE
Dept. of Information Science & Engg

- The modulus operator, **%**, returns the remainder of a division operation.
- It can be applied to floating-point types as well as integer types.

Example:
```
class Modulus{
    public static void main(String args[])
{       int x = 42;
        double y = 42.25;
        System.out.println("x mod 10 = " + x % 10);
        System.out.println("y mod 10 = " + y % 10);
}
}
```

output:

x mod 10 = 2

y mod 10 = 2.25

The Modulus Operator

- The modulus operator, **%**, returns the remainder of a division operation.
- It can be applied to floating-point types as well as integer types.

Example:
```
class Modulus{
    public static void main(String args[])
{       int x = 42;
        double y = 42.25;
        System.out.println("x mod 10 = " + x % 10);
        System.out.println("y mod 10 = " + y % 10);
}
}
```

output:

x mod 10 = 2

y mod 10 = 2.25

Arithmetic Compound Assignment Operators

- Java provides special operators that can be used to combine an arithmetic operation with an assignment.

  Example 1: a = a + 4;

- In Java, you can rewrite this statement as shown here:

  a += 4;

  Example 2: a = a % 2;

- It can be expressed as a %= 2;
- Thus, any statement of the form

  var = var op expression;

- The above can be rewritten as

  var op= expression;

Arithmetic Compound Assignment Operators cntd..

- The compound assignment operators provide two benefits.
  - First, they save you a bit of typing, because they are "shorthand" for their equivalent long forms.
  - Second, they are implemented more efficiently by the Java run-time system than are their equivalent long forms.

Arithmetic Compound Assignment Operators cntd..

Example:

```
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);
}
}
```

Output:

a = 6
b = 8
c = 3

Increment and Decrement cntd..

DSCE
Dept. of Information Science & Engg

- Example 1: x = 42;
  y = ++x;
  y=43; x=43
- y = ++x; is equivalent to these two statements:
  x = x + 1;
  y = x;
- Example 2: x = 42;
  y = x++;
  y=42; x=43
- y = x++; is equivalent to these two statements:
  y = x;
  x = x + 1;

Increment and Decrement cntd..

DSCE
Dept. of Information Science & Engg

```
// Demonstrate ++.
class IncDec {
public static 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);
```

The output of this program follows:

a = 2
b = 3
c = 4
d = 1

The Bitwise Operators

DSCE
Dept. of Information Science & En

- Java defines several bitwise operators that can be applied to the integer types, long, int, short, char, and byte.

Operator	Result
~	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

The Bitwise Operators

1. The Bitwise NOT

Also called the bitwise complement, the unary NOT operator, ~, inverts all of the bits of its operand. For example, the number 42, which has the following bit pattern: 00101010 becomes 11010101 after the NOT operator is applied.

2. The Bitwise AND

The AND operator, &, produces a 1 bit if both operands are also 1. A zero is produced in all other cases. Here is an example:

00101010 42

&

00001111 15

00001010 10

The Bitwise Operators

Dept. of Information Science & Engg

3. The Bitwise OR

The OR operator, |, combines bits such that if either of the bits in the operands is a 1, then the resultant bit is a 1, as shown

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

4. The Bitwise XOR

The XOR operator, ^, combines bits such that if exactly one operand is 1, then the result is 1. Otherwise, the result is zero.

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

DSCE
Dept. of Information Science & Engg

The Bitwise Operators

5. The Left Shift

- The left shift operator, << shifts all of the bits in a value to the left a specified number of times. It has this general form:
- value << num
- Here, num specifies the number of positions to left-shift the value in value.
- That is, the << moves all of the bits in the spe
- cified value to the left by the number of bit positions specified by num. For each shift left, the high-order bit is shifted out (and lost), and a zero is brought in on the right.

The Bitwise Operators

6. The Right Shift

- The right shift operator, >>, shifts all of the bits in a value to the right a specified number of times. Its general form is :

  value >> num

- Here, num specifies the number of positions to right-shift the value in value. That is, the >> moves all of the bits in the specified value to the right the number of bit positions specified by num.
- The following code fragment shifts the value 32 to the right by two positions, resulting in a being set to 8:

Example: int a = 32;

a = a >> 2; // a now contains 8

00100011	35
>> 2	
00001000	8

The Bitwise Operators

6. The Unsigned Right Shift, >>>

- If you are shifting something that does not represent a numeric value, you may not want sign extension to take place
- . In these cases, you will generally want to shift a zero into the high-order bit no matter what its initial value was. This is known as an unsigned shift.
- To accomplish this, you will use Java's unsigned, shift-right operator,

>>>, which always shifts zeros into the high-order bit.

Example:

int a = -1;

a = a >>> 24;

Here is the same operation in binary form to further illustrate what is happening:

11111111 11111111 11111111 11111111 –1 in binary as an int

>>>24

00000000 00000000 00000000 11111111 255 in binary as an int

The Bitwise Operators

- Bitwise operator works on bits and performs bit-by-bit operation. Assume if a = 60; and b = 13;

a = 0011 1100

b = 0000 1101

a&b = 0000 1100

a|b = 0011 1101

a^b = 0011 0001

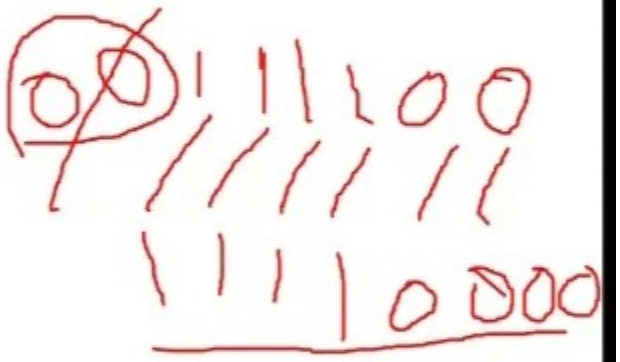
~a = 1100 0011

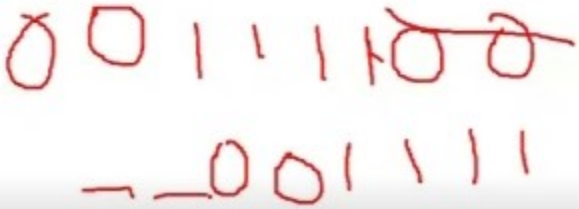
The Bitwise Operators cntd..

DSCE
Dept. of Information Science & Engg

- Assume integer variable A holds 60 and variable B holds 13 then:

```
public class Test
 {
 public static void main(String args[])
{     int a = 60;      /* 60 = 0011 1100 */
      int b = 13;      /* 13 = 0000 1101 */
     int c = 0;
  c = a & b;        /* 12 = 0000 1100 */
     System.out.println("a & b = " + c );
  c = a | b;        /* 61 = 0011 1101 */
   System.out.println("a | b = " + c );
 c = a ^ b;        /* 49 = 0011 0001 */
  System.out.println("a ^ b = " + c );
```





DSC
Dept. of Information Science

The Bitwise Operators

- **The Bitwise Logical Operators**
- The bitwise logical operators are &, |, ^, and ~.

A	B	A | B	A & B	A ^ B	~A
0	0	0	0	0	1
1	0	1	0	1	0
0	1	1	0	1	1
1	1	1	1	0	0

Assignment operators:

- The assignment operator works in
- Java much as it does in any other computer language. It has this general form:

var = expression;

- **For example**, consider this fragment:

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

DSCE
Dept. of Information Science & Engg

The ? Operator

It is also known as Conditional operator or the **ternary operator.**

This operator consists of three operands and is used to evaluate Boolean expressions.

The goal of the operator is to decide which value should be assigned to the variable. The operator is written as:

variable x = (expression) ? value if true : value if false

The ? Operator

Following is the **Example**:

```
public class Test {
  public static void main(String args[])
{     int a , b;
        a = 10;
  b = (a == 1) ? 20: 30;
System.out.println( "Value of b is : " +  b );
    b = (a == 10) ? 20: 30;
  System.out.println( "Value of b is : " + b );
  }}
```

This would produce the following result:

Value of b is : 30**Value of b is : 20**

Operator Precedence

DSC
Dept. of Information Science

Highest			
( )	[ ]	.	
++ .	– –	~	!
*	/	%	
+	–		
>>	>>>	<<	
>	>=	<	<=
==	!=		
&			
^			
|			
&&			
||			
?:			
=	op=		
Lowest			

Operator Precedence

DSCE
Dept. of Information Science & Engg

- **Using Parentheses**
  - *Parentheses raise the precedence of the operations that are inside them*
  - For example, consider the following expression:

    a >> b + 3
  - That is, this expression can be rewritten using redundant parentheses like this:

    a >> (b + 3)
  - However, if you want to first shift **a right by b positions and then add 3 to that result,** you will need to parenthesize the expression like this:

    (a >> b) + 3
  - Parentheses can sometimes be used to help clarify the meaning of an expression.
  - For example, which of the following expressions is easier to read?

    (i) a | 4 + c >> b & 7

    ii) (a | (((4 + c) >> b) & 7))

Control Statements

- Programming language uses *control statements to cause the flow of execution to* advance and branch based on changes to the state of a program.
- Java's programcontrol statements can be put into the following categories:
- selection,
- iteration, and
- jump.

Java's Selection Statements

2. Nested ifs

- A *nested* ***if is an if statement that is the target of another if or else.***
- ***Nested ifs are very common*** in programming.
- **Here is an example:**

```
if(i == 10) {
if(j < 20) a = b;
if(k > 100) c = d; // this if is
else a = c; // associated with this else
}
else a = d; // this else refers to if(i == 10)
```

Java's Selection Statements

3. The if-else-if Ladder

- A common programming construct that is based upon a sequence of nested **ifs is the** *if-else-if ladder. It looks like this:*

```
if(condition)
statement;
else if(condition)
statement;
else if(condition)
statement;
...
else
statement;
```