Java Fundamentals

Siddharth Sharma

Operators:

```
class modulus
{
   public static void main(String args[])
   {
      float x= 5.50;
      float y= 2.25;
      System.out.println("x mod y: "+x%y);

   }
}
```

OUTPUT:

```
prog.java:5: error: incompatible types: possible lossy
conversion from double to float
        float x= 5.50;
                 ^
prog.java:6: error: incompatible types: possible lossy
conversion from double to float
        float y= 2.25;
                 ^
2 errors
```

```
class modulus
{
   public static void main(String args[])
   {
      float x= 5.50f;  f or F
      float y= 2.25f;
      System.out.println("x mod y: "+x%y);

   }
}
```

OUTPUT:

```
x mod y: 1.0
```

Why did this discrepancy occur?

This is because in Java **double is a default data type** to represent decimal numbers. To use floating point numbers, put a **F or f** in front of the number.

- **Modulus Operator:**

```
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);
       System.out.println("y mod x = " + y %x);
    }
}
```

OUTPUT:

```
x mod 10 = 2
y mod 10 = 2.25
y mod x = 0.25
```

- **Arithmetic Compound Assignment Operators:**

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

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.

Examples:

a=a%6 can be represented using a%=6

a=a+4 can be represented using a+=4

var = var op expression; can be represented using *var op= expression;*

• Increment and Decrement Operators:

x=13;	x=13;
y=++x;	y=x++;
y=14	y=13
x becomes 14 in both cases	
x=x+1;	y=x;
y=x;	x=x+1;

• BITWISE OPERATORS:

Java defines several bitwise operators that can be applied to the **integer types, long, int, short, char, and byte**. These operators act upon the individual bits of their operands.

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

OUTPUT:

```
a= 0011
b= 0110
a|b= 0111
a&b= 0010
a^b= 0101
```

```
~a&b|a&~b = 0101
~a = 1100
~b = 1001
```

- **Left Shift Operator :**

```
class introduction
{
   public static void main(String args[])
   {
      byte a=64,b;// declaring byte
      int i;
      i=a<<2; /* Type promotion: byte is promoted to int and then shifting
            is done */
      b=(byte)(a<<2) ;  /* int is type casted to byte back again, here only
                  the rightmost bit is considered for evaluation */
      System.out.println(a);   //( 0000 ....0100 0000)
      System.out.println(i);// (0000... 1 0000 0000)
      System.out.println(b);// (0000 0000)
   }
}
```

OUTPUT:

```
64
256
0
```