



Java™ Object-Oriented Programming



Arithmetic Operators

Operator	Description	Example
+	add	x = x + 1;
-	subtract	x = x - 1;
*	multiply	x = x * 10;
/	divide	x = x / 5;
%	modulus – returns remainder	x = x % 3;



Operator Precedence

Java's Mathematical Order of Precedence

- The operator precedence defines what operators take precedence over other operators, and hence get executed first.
- Expressions contained within parenthesis are evaluated first.
- The order of precedence for the mathematical operators is from **left to right in the following order**:
 - Multiplication and Division
 - Addition and Subtraction



Operator Precedence

Example	Explanation
<pre>int x = 2; int y = 10; int z = 3; int r; Example 1: r = x + y * z; // result is 32 Example 2: r = (x + y) * z; // result is 36</pre>	<p>Rule: Parenthesis determine 1st order of mathematical operations.</p> <p>Example 1: Multiplication first; then addition</p> <p>Example 2: Math proceeds first with the parenthesis then the multiplication takes place.</p>



Additional Operators

Operator	Description	Example
<	less than	<code>if (x < y)</code>
>	greater than	<code>if (x > y)</code>
>=	greater than or equal	<code>if (x >= y)</code>
<=	less than or equal	<code>if (x <= y)</code>
==	equality (notice there are TWO = signs here)	<code>if (x == y)</code>
=	assignment	<code>x = y;</code>
!=	not equal	<code>if (x != y)</code>
!	negation "not"	<code>if (!false)</code>



Equality Operator

- '==' is very different from '='.
- Rule:
 - '=' is used to compare two variables.
 - '=' is used **only** for assignment.

```
if (a = 0)    // compiler error!
```

```
if (a == 0)  // compiler happy!
```



Increment/Decrement Operators

Keyword	Description	Code	Equivalent
<code>++var</code>	Pre-Increment	<code>++x;</code>	<code>x = x + 1;</code>
<code>var++</code>	Post-Increment	<code>x++;</code>	<code>x = x + 1;</code>
<code>--var</code>	Pre-Decrement	<code>--x;</code>	<code>x = x - 1;</code>
<code>var--</code>	Post-Decrement	<code>x--;</code>	<code>x = x - 1;</code>

- ❑ With the "Pre" operators the variable is inc/decremented BEFORE it is used.
- ❑ With the "Post" operators the variable is inc/decremented AFTER it is used.



Pre-Post Increment/Decrement Example

Initial x	Expression	Final y	Final x
3	$y = x++$	3	4
3	$y = ++x$	4	4
3	$y = x--$	3	2
3	$y = --x$	2	2



Shorthand Operators in Java

Java Shorthand	Equivalent Expanded Java
<code>x += 2; or x+=2;</code>	<code>x = x + 2;</code>
<code>x -= 3; or x-=3;</code>	<code>x = x - 3;</code>
<code>x *= 4; or x*=4;</code>	<code>x = x * 4;</code>
<code>x /= 5; or x/=5;</code>	<code>x = x / 5;</code>
<code>x %= 6; or x%=6;</code>	<code>x = x % 6;</code>

- You may code either way, code performance is not affected.
-



Logical Operators

Operator	Example	Description
&	<code>((x==5)&(y==3))</code>	& (Logical AND) or (Logical OR) used in a conditional statement means every condition is evaluated <u>BEFORE</u> a final decision is made. These SHOULD RARELY be used in conditional statements.
	<code>((x==5) (y==3))</code>	
&&	<code>((x==5)&&(y==3))</code>	Conditional statements using && (Logical AND) or (Logical OR) are only evaluated as much as needed to make a <u>DEFINITE</u> decision. These are known as short-circuit operators, and should USUALLY be used in conditional statements.
	<code>((x==5) (y==3))</code>	



& 'AND' Example

```
int x = 5;  
int y = 0;  
  
if ((y > 0) & ((x / y) > 6))  
    x = x + 1;
```

□ Why is this a problem?



Short Circuit 'AND' Example

```
int x = 5;
int y = 0;

if ((y > 0) && ((x / y) > 6))
    x = x + 1;
```

□ Why does this work better?



| 'OR' Example

```
boolean cash = true;  
boolean credit = false;  
boolean check = true;  
  
if (cash | credit | check)  
    //do something
```

□ Why is this inefficient?



Short Circuit 'OR' Example

```
boolean cash = true;
boolean credit = false;
boolean check = true;

if (cash || credit || check)
    //do something
```

□ Why is this more efficient?



Lab

- Write new class named operTest
- Declare variables x and y.
- Set x to 1.
- Set `y = ++x;`
- Print `"y = ++x is" + y.`
- Set `x = 1.`
- Set `y = x++;`
- Print `"y = x++ is" + y.`



Block

- A block is zero or more Java statements
- Blocks are defined by { }

```
{  
    int x = 5;  
    int y = 0;  
}
```



Scope Rules

- ❑ A variable is available only within its 'scope'.
- ❑ Scope is the defining block and all contained blocks.

```
{
    {
        int x = 5;
        System.out.println(x);
    }
    System.out.println(x); // error!
}
```



Scope Rules - Example 2

```
{
    int x = 10;
    {
        int x = 5;
        System.out.println(x);
        //What is output?
    }
    System.out.println(x);
    //What is output?
}
```



if Statements

- "if" statements are used to control code execution by making decisions.

```
int x = 5;
boolean isGreaterThanTen = false;

if ( x > 10 ) {
    isGreaterThanTen = true;
} // end of if block "scope".
```



Equivalent if Statements

```
if ( x > 10 ) {  
    isGreaterThanTen = true;  
} // end of if block "scope".
```

```
if ( x > 10 )  
    isGreaterThanTen = true;
```

□ Why is the 2nd example a bad idea?



if/else Statements-Example 1

```
int x = 5;
boolean isGreaterThanTen = false;

if ( x > 10 ) {
    isGreaterThanTen = true;
} else {
    isGreaterThanTen = false;
}
```



if/else Statements-Example

```
int x = 5;

if ( x > 10 ) {
    System.out.println("x > 10");
} else if ( x > 5 ) {
    System.out.println("x > 5");
} else {
    System.out.println("x < 5!");
}
```

- The else can be combined with if. The last else becomes the default condition.



for Loops

- A Java 'for' loop repeats a block of statements a fixed number of times.

```
for (int i = 0; i < 4; i++ ) {  
    System.out.println(i);  
}
```

Output:

0
1
2
3

What is the value of 'i' after the for loop?



while Loops

- ❑ A Java 'while' loop continually executes a block of statements while a condition remains true.
- ❑ A while loop executes zero or more times.

```
int i = 0;
while (i < 5) {
    System.out.println(i++);
}
```



break Statement and Loops

- The 'break' statement immediately ends a loop. What is the output?

```
int i = 0
while(i < 5) {
    if (i == 2) {
        break;
    }
    i++;
    System.out.println(i);
}
```



continue Statement and Loops

- A 'continue' statement allows one to skip the rest of the current loop iteration. What is the output?

```
for (int i = 0; i < 5; i++ ) {  
    if (i == 2) {  
        continue;  
    }  
    System.out.println(i);  
}
```



Lab Practice – if/else

- Let's practice using if then/else and loops. We'll test integers from 1 to 10 to determine if the number is even or odd and print it. HINT: We need to use % operator.

Output:

1 is odd

2 is even

3 is odd

...

10 is even