



Expressions and Operators

Bok, Jong Soon
javaexpert@nate.com

<https://github.com/swacademy/JavaSE>

Expression

- Is a statement that changes the program state.
- Ends in a semicolon.
- Includes assignments, pre- and post-increments, pre- and post-decrements, object creation, and method calls.

```
isValid = true;
```

```
count++;
```

```
Student sujan = new Student(70, 50, 90);
```

```
sujan.display();
```


Variables – local variables

Local variables are:

- Variables that are defined inside a method and are called *local*, *automatic*, *temporary*, or *stack* variables.
- Created when the method is executed and destroyed when the method is exited.
- Variables that *must* be initialized before they are used or compile time errors will occur.

Variables – class variables (Cont.)

Class variables are:

- Is declared using the **static** keyword and are called *class* or *static* variables.
- Is done when the class is loaded.
- Continues to exist for as long as the class exists.

Variables – instance variables (Cont.)

Instance variables are:

- Is declared without the **static** keyword.
- Are sometimes referred to as *member* variables because they are members of the class.
- Are created when the object is constructed using the **new** **Xxxx()** call.

Operators

- Performs a function on one, two, or three operands and returns a result.
- An operator that requires one operand is called a *unary operator*.
- An operator that requires two operands is a *binary operator*.
- A *ternary operator* is one that requires three operands.
- Contains precedence order(top to bottom) along with their associativity (left to right or right to left).

Operators in Order of Precedence

Operator	Kinds	Association
Prime Operators	. [] ()	
Unary Operators	++ -- + - ~ ! new (type)	R → L
Binary Operators	* / %	L → R
	+ -	L → R
	<< >> >>>	L → R
	< > <= >= instanceof	L → R
	== !=	L → R
	&	L → R
	^	L → R
		L → R
	&&	L → R
		L → R
Ternary Operators	? :	L → R
Assignment Operators	= *= /= %= += -= <<= >>= >>>= &= ^= =	R → L

Prime Operators

■ The `[]` Operator

- Use square brackets to declare arrays, to create arrays and to access a particular element in an array.

```
int [ ] arrayOfInts = new int[10];
```

■ The `.` Operator

- Accesses instance member of an object or class members of a class.

```
sujan.display();
```

■ The `()` Operator

- When declaring or calling a method, you list the method's arguments between (and)

```
4 * (8 + 10)
```


Unary Operators – Increment and Decrement Operators

- A more common requirement is to add or subtract 1 from a variable.
- Either ++ or -- can appear before (prefix) or after (postfix) its operand.

Purpose	Operator	Example
Pre - Increment	++	a = ++b;
Post - Increment	++	a = b++;
Pre - Decrement	--	a = --b;
Post - Decrement	--	a = b--;

Unary Operators

Purpose	Operator	Example
Unary plus	+	+4
Unary minus	-	-4
Bitwise complement	~	int su = ~ 5;
Boolean NOT	!	boolean isValid = ! true;
Create object	new	Test t = new Test();
Type cast	(type)	int su = (int) 89.5;

Binary Operators – Arithmetic Operators

Purpose	Operator	Example
Addition	+	sum = num1 + num2
Subtraction	-	diff = num1 - num2
Multiplication	*	prod = num1 * num2
Division	/	quot = num1 / num2
Modulus	%	mod = num1 % num2

Binary Operators – Shift Operators

Purpose	Operator	Example
Left shift	<<	64 << 3 returns $64 * 2^3 = 512$
Right shift ¹	>>	128 >> 1 returns $128/2^1 = 64$ 256 >> 4 returns $256/2^4 = 16$ -256 >> 4 returns $-256/2^4 = -16$
Unsigned right shift ²	>>>	-256 >>> 4 returns $(-256 / 2^4) * -1 = 16$

1. *Arithmetic* or *signed* right shift operator(>>)
 - The sign bit is copied during the shift.
2. A *logical* or *unsigned* right shift operator(>>>)
 - Not copied during the shift.

Binary Operators – Relational Operators

Purpose	Operator	Example
Less than	<	4 < 5 return true
Less than or equal to	<=	5 <= 5 return true
Greater than	>	4 > 5 return false
Greater than or equal to	>=	4 >= 4 return true
Type comparison	instanceof ¹	car instanceof Sonata
Equality	==	4 == 5 return false
Inequality	!=	4 != 5 return true

¹. Refer to Chapter 10. Object-Orientation-Third Story.pdf

Binary Operators – Bitwise Operators

& (Bitwise AND)			^ (Bitwise XOR)	(Bitwise OR)
1	1	1	0	1
1	0	0	1	1
0	1	0	1	1
0	0	0	0	0

Binary Operators – Short-Circuit Logical Operators

&& (Conditional AND)			(Conditional OR)
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

Ternary Operator

- The **? :** operator
- (Condition) **? true : false**
- e.g.

```
String am_pm = null;  
java.util.Calendar today = java.util.Calendar.getInstance();  
int hour = today.get(java.util.Calendar.HOUR_OF_DAY);  
am_pm = (hour >= 12) ? "PM" : "AM";  
System.out.println(am_pm);
```

Assignment Operators

- The `=` operator

```
int su = 5;
```

- The `+=`, `-=`, `*=`, `/=`, `%=` operator

```
int su = 5;          su += 8;
```

- The `&=`, `^=`, `|=` operator

```
int su = 5;          su |= 3;
```

- The `<<=`, `>>=`, `>>>=` operator

```
int su = -128;       su >>= 3;
```

String Concatenation With +

- The + operator:

- Performs **String** concatenation.
- Produces a new **String**:

```
String str = "Hello";  
String str1 = ", World";  
String newStr = str + str1;  
String newstr1 = 128 + newStr;
```

- One argument must be a **String** object.
- Non-strings are converted to **String** objects automatically.

Numeric Promotion of Primitive Types

- Promotion rules consist of both unary and binary promotion rules.
- Unary Numeric Promotion
 - If the operand is of type **byte**, **short**, or **char**, the type will be promoted to type **int**.
 - Otherwise, the type of the operand remains unchanged.
 - **+**, **-**, **~**

Numeric Promotion of Primitive Types (Cont.)

■ Binary Numeric Promotion

- If either operand is of type **double**, the non-double primitive is converted to type **double**.
- If either operand is of type **float**, the non-float primitive is converted to type **float**.
- If either operand is of type **long**, the non-long primitive is converted to type **long**.
- Otherwise, both operands are converted to **int**.

```
char ch = 65;
```

```
char ch1 = ch >> 3;    //compile error
```

```
int su = ch >> 3;
```


Special Cases for Conditional Operators

- If one operand is of type **byte** and the other is of type **short**, the conditional expression will be of type **short**.

```
short = true ? byte : short
```
- If one operand *R* is of type **byte**, **short**, or **char**, and the other is a constant expression of type **int** whose value is within range of *R*, the conditional expression is of type *R*.

```
short = (true ? short : 1967)
```
- Else, binary numeric promotion is applied and the conditional expression type will be that of the promoted type of the second and third operands.