Java Operators

Table of Contents

Introduction to Java Operators

- Operator – symbol(s) representing operation that can be performed on constants and variables
- Types of Operators
  - Assignment
  - Arithmetic
  - Increment and decrement
  - Relational
  - Logical
  - Conditional

Assignment Operator

Assignment Operator

- Uses an equal (=) symbol
- Stores or assigns a value from the right-hand side to the left-hand side
- Assignment operation is also called assignment statement, so it should end in a semicolon

```
<variable name> = <expression>;
```

Assignment Operator

- Left-hand side "gets the value" of the right-hand side
- Do not use "equal to" as it is not interchangeable
- It is also possible to assign a value of a variable to another variable of a compatible data type

```
X = 5;
Y = X;    // Y = 5
Z = Y;    // Z = 5
```

Assignment – Storing Values with Different Data Types

`int` to `double/float` and vice versa

- `int` to `double / float` – any decimal is dropped and the whole number is only retained; no rounding off

```
int i = 14. 50987;          // i = 14
```

- `double / float` to `int` – zeros are added according to the data type

```
double d = 14;          // d = 14.0
```

Arithmetic Operators

Arithmetic Operators

- Also known as mathematical operators

Symbol	Meaning	Compatible Data Types	Remarks
+	Addition	int, float, double	
-	Subtraction		
*	Multiplication		
/	Division		For `int`, any decimal is dropped
%	Modulo	int	

Arithmetic – Precedence and Associativity Rule

- Operators with higher precedence must be solved first
- Operators with the same precedence are evaluated from left to right

Operator	Associativity
*, / , %	Left to Right
+, -	Left to Right

Arithmetic – Precedence and Associativity Rule

```
X = 3 + 6 * 2 - 5 + 10 / 2 * 8 /2 - 3
X = 3 + 12 - 5 + 10 / 2 * 8 /2 - 3
X = 3 + 12 - 5 + 5 * 8 /2 - 3
X = 3 + 12 - 5 + 40 / 2 - 3
X = 3 + 12 - 5 + 20 - 3
X = 15 - 5 + 20 - 3
X = 10 + 20 - 3
X = 30 - 3
X = 27
```

Increment and Decrement Operators

Pre-Increment Operator:

- If an Increment operator is used in front of an operand, it is called a Pre-Increment operator.

- ++x : which increments the value by 1 of ‘x’ variable.

x = 10;

y = ++x;

y = ?

Post Increment Operator:

- If an Increment operator is used after an operand, then is called Post Increment operator.

- x++ : which increase the value by 1 of variable ‘x’.

x = 10;

y = x++;

y = ?

Pre Decrement Operator:

- If a decrement operator is used in front of an operand, then it is called Pre decrement operator.

- –x : which decrease the value by 1 of variable ‘x’ .

x = 10;

y = --x;

Y = ?

Post Decrement Operator:

- If a decrement operator is used after an operand, then it is called Post decrement operator.

- x- – : which decrease the value by 1 of variable ‘x’ .

x = 10;

y = x--;

Y = ?

Relational Operators

Relational Operators

- Checks association of the left-hand side to right-hand side
- Yields a `TRUE` or `FALSE` condition only

Symbol	Meaning
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Relational Operators

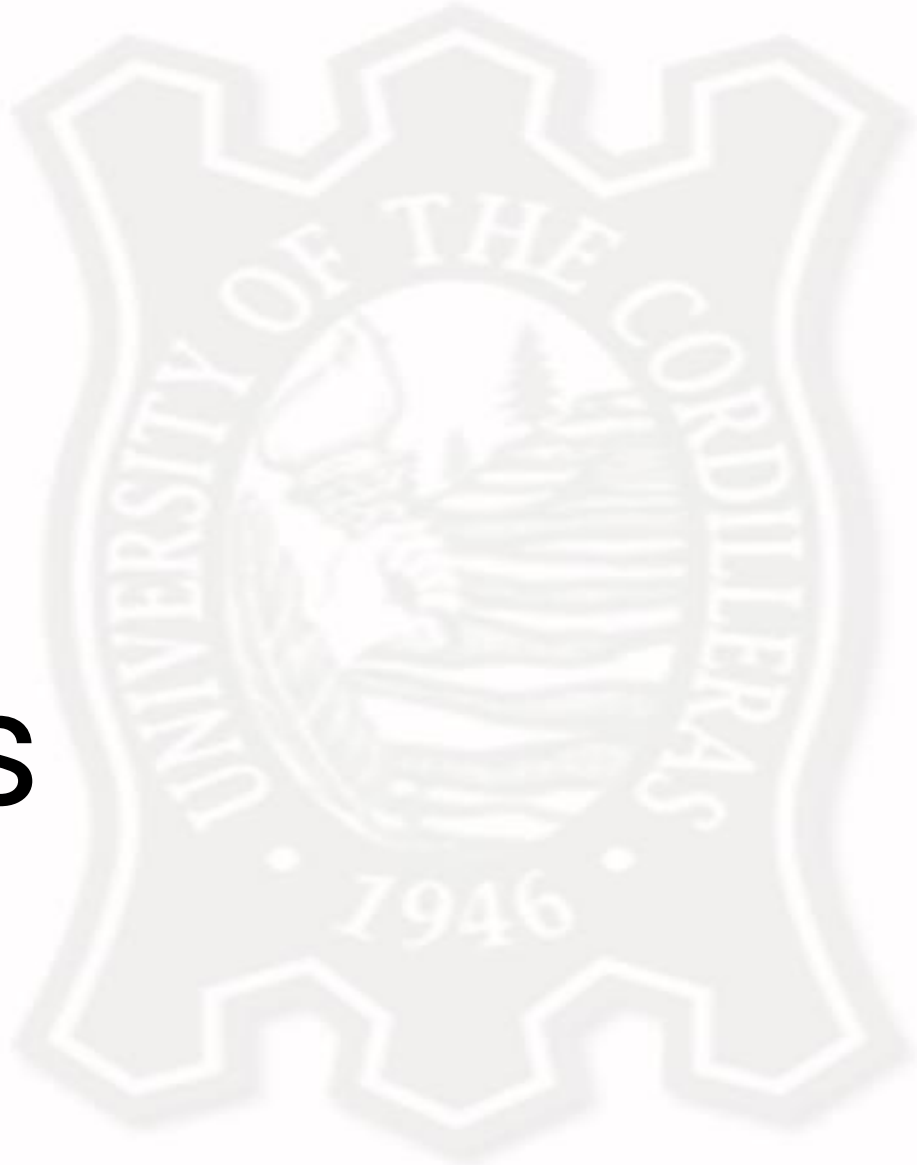
- Can be performed on all basic data types
- Can also be used in conjunction with other operators (conditional and logical)
- Must be enclosed in parenthesis
- Take note of the difference of the assignment (=) and relational (==) operators

Relational Operators

```
x = 8;
y = 13;

a = (x == y)          //result is FALSE
b = (x != y)    //result is TRUE
c = (x > y)           //result is FALSE
d = (x < y)           //result is TRUE
e = (x >= y)    //result is FALSE
f = (x <= y)    //result is TRUE
```

Logical Operators



Logical Operators

- Used to test multiple conditions
- Normally used in conjunction with relational operators

Symbol	Meaning
!	Logical NOT
&&	Logical AND
&	boolean Logical AND
||	Logical OR
|	boolean Logical Inclusive OR
^	boolean Logical Exclusive OR

Logical Operators

- Logical (NOT, AND, OR) supports short-circuit evaluations
- Boolen Logical (AND, Inclusive OR, Exclusive OR) evaluates every expression before a result is given
- General statement:

```
<expr1> <Logical Operator> <expr2>
```

```
(x == y) ! (a > b)
(x != y) && (a < b)
(x >= y) || (a <= b)
```

&& (Logical AND) and & (boolean Logical AND)

- The AND operator combines two expressions (or conditions) together into one condition group. Both expressions are tested separately by JVM and then && operator compares the result of both.

- If the conditions on both sides of && operator are true, the logical && operator returns true. If one or both conditions on either side of the operator are false, then the operator returns false.

&& (Logical AND) and & (boolean Logical AND)

- All expressions evaluated should be correct to be considered `TRUE`; otherwise the statement is `FALSE`

expr1	expr2	Result
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

&& (Logical AND) and & (boolean Logical AND)

- Example:

```
if(x > y && y < z)
  System.out.println("Hello Java");
```

expr1	expr2	Result
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

|| (Logical OR) and | (boolean Logical inclusive OR)

- The logical OR operator in Java combines two or more expressions or conditions together into a single condition group.

- The OR operator returns true if either one or both conditions returns true. If the conditions on both sides of the operator are false, the logical OR operator returns false.

|| (Logical OR) and | (boolean Logical inclusive OR)

- At least one expression evaluated should be correct for it to be considered `TRUE`; otherwise the statement is `FALSE`

expr1	expr2	Result
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

|| (Logical OR) and | (boolean Logical inclusive OR)

- Example:

```
if(x == 1 || y == 1 || z == 1)
   System.out.println("Hello");
Syntax error..
```

expr1	expr2	Result
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

! (Logical NOT)

- The NOT operator is used to reverse the logic state of its operand. If the condition is correct, the logical NOT operator returns false. If the condition is false, the operator returns true.

! (Logical NOT)

- Unary operator
- Negates or gets the opposite of a certain result in a relational operation

expr	Result
TRUE	FALSE
FALSE	TRUE

! (Logical NOT)

- Example:

```
if(!( x > y ))
   System.out.println("Hello Java");
```

expr	Result
TRUE	FALSE
FALSE	TRUE

^ (boolean Logical exclusive OR)

- One statement must be `TRUE` and the other statement must be `FALSE`

expr1	expr2	Result
TRUE	TRUE	FALSE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

Conditional Operators

Conditional Operator

- Conditional operators check the condition and decides the desired result based on both conditions.

Conditional Operator

Ternary operator (uses three arguments)

```
(Condition) ? expr1 : expr2
```

- `conditon` – boolean statement that determines what expression shall be used
- `expr1` – statement used if `expr1` is `TRUE`
- `expr2` – statement used if `expr1` is `FALSE`

Conditional Operator

- Example:

```
x = 20;
y = (x == 1) ? 61: 90;
System.out.println("Value of y is: " +  y);
y = (x == 20) ? 61: 90;
System.out.println("Value of y is: " + y);
```

- 1.) 2 * (3 + 4) = (2 * 3) + (2 * 4)
- 2.) 5 y + 6 * 10 / 8 * 9 + 1 – 4 + 7
- 3.) 8 – 10y / 6 * 10 + 34 – 20 * 2
- 4.) (6 x 4) ÷ 12 + 72 ÷ 8 – 9
- 5.) What is the “Motto” of UC