School of Computer Science and IT
JAIN (DEEMED-TO-BE UNIVERSITY)
Department of Bachelor of Computer Applications

Module 1 Introduction to Java Programming

Chapter 2 Operators and Looping Constructs

Operators in java

Operators in Java

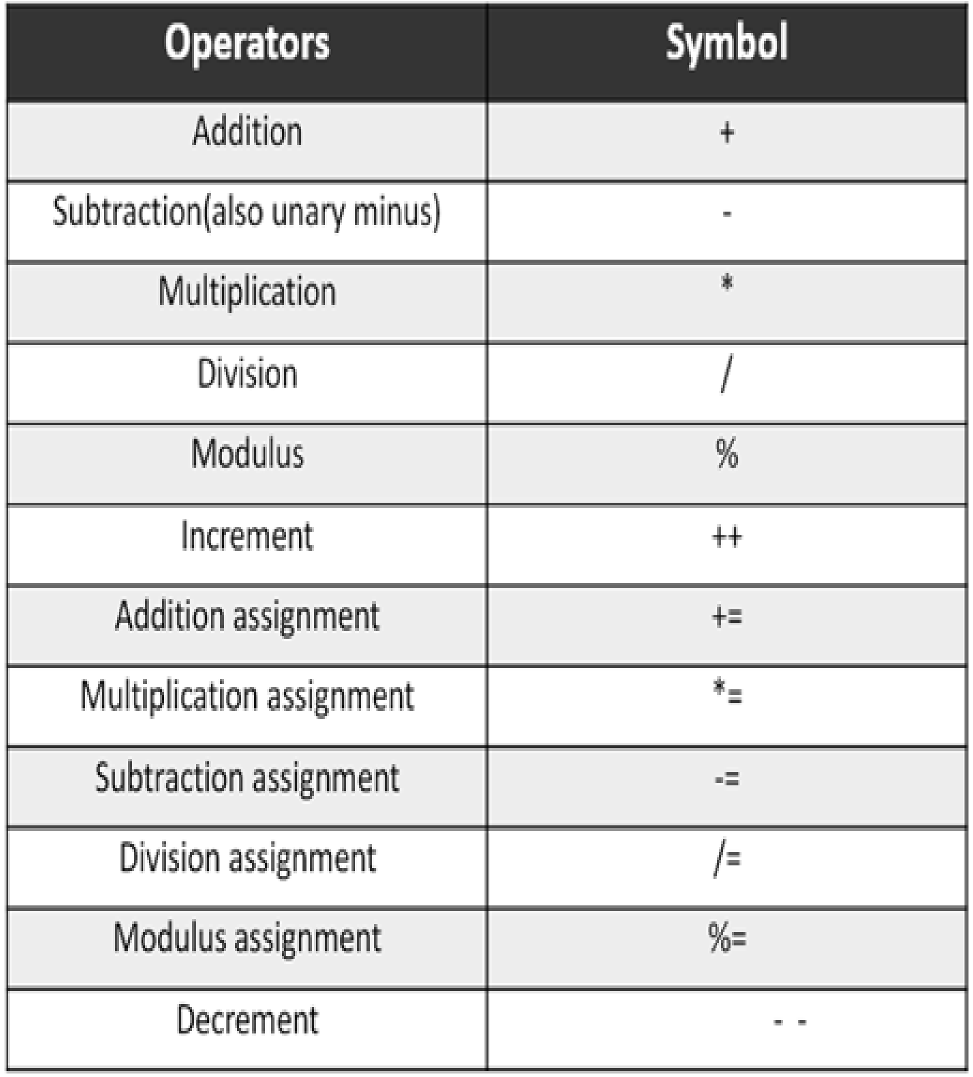
Operators in Java are used to perform arithmetic operations. They comprises of rich set of operators to manipulate variables. Following are some of the operators

- Arithmetic Operators
- Relational Operators
- Bitwise Operators
- Logical Operators
- Assignment Operators

a) The **operands** of the arithmetic operators necessity are of a numeric type because cannot use them on Boolean types, but you can use them on char types since the char type in Java is, essentially, a subset of int.Arithmetic Operators

Arithmetic Operators

Arithmetic operators are operators used to perform mathematical expression. It takes numerical expression as their operands and return a single value as a result. Following are the standard arithmetic operators



Operators	Symbol
Addition	+
Subtraction(also unary minus)	-
Multiplication	*
Division	/
Modulus	%
Increment	++
Addition assignment	+=
Multiplication assignment	*=
Subtraction assignment	-=
Division assignment	/=
Modulus assignment	%=
Decrement	- -

Arithmetic Operators Syntax and its Uses

Operators	Syntax	Uses
Addition	Operand 1 + operand 2	Sum two operands
Subtraction	Operand 1 – Operand 2	Provides difference of operands
Multiplication	Operand 1 * Operand 2	Product of two operands can be yielded
Division	Operand 1 / Operand 2	Produces the quotient of operands

Bitwise Operator

Bitwise operators - Binary operators treat their operands as a sequence of bits (zeroes and ones), rather than as decimal, hexadecimal, or octal numbers.

Java defines some bitwise operators that can be implemented to the integer types, long, int, short, char, and byte. These operators act upon the individual bits of their operands. They are compiled in table shown.

Operator	Result
~	Bitwise unary NOT
&	Bitwise AND
^	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

Relational Operator

In Java, relational operators are used to check relation between two variables or numbers. They are also called as comparison operators since the outcome will be of true or false (boolean) values. Relational operators are commonly used in conditional statement (if statement / looping statement) in order to check the conditions (true or false).

Operator	Outcome
= =	Equal to
! =	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Lesser than or equal to

Sample Program

The result of these operations is a **boolean value**. The relational operators are most commonly used in the definitions that control the if statement and the multiple loop statements. Any type in Java, containing integers, floating-point numbers, characters, and Booleans can be compared using the inequality test, ==, != equality test. Notice that in Java equality is denoted with two equal signs, not one. (Remember: a single equal sign is the distribution operator.) Only numeric types can be compared using the ordering operators. Means, only integer, floating-point, and character operands may be compared to see which is greater or less than the other. As stated, the result produced by a relational operator is a boolean value.

For example, the following code fragment is perfectly valid:

```
int  i = 5;
int  j = 4;
boolean k = i < j;
```

In this case, the result of i<j (which is false) is stored in "k".

Boolean Logical Operators

The Boolean logical operators displayed here operate only on Boolean operands. Each of the binary logical operators combines two Boolean values to determine a resultant Boolean value.

The Logical Boolean operators, &, |, and,^,! Operate on Boolean conditions in the similar way that operators work on the bits of an integer. The logical ! Operator inverts the Boolean state:!true == false and !false == true.

Operator	Result
&	Logical AND
|	Logical OR
^	Logical XOR (exclusive OR)
||	Short-circuit OR
&&	Short-circuit AND
!	Logical unary NOT
&=	AND assignment
|=	OR assignment
^=	XOR assignment
= =	Equal to
!=	Not equal to

Boolean Logical Operator

The following table shows the outcome of each logical operation

A	B	A|B	A&B	A^B	!A
False	False	False	False	False	True
True	False	True	False	True	False
False	True	True	False	True	True
True	True	True	True	False	False

Sample Program

```
class BooleanLogic{ public static void main(String args[])
{
boolean A = true;    boolean B = false;
 // these are boolean variables
System.out.println ("A|B = "+(A|B));
System.out.println ("A&B="+(A&B));
System.out.println ("!A = "+(!A));
System.out.println ("A^B = "+(A^B));
System.out.println ("(A|B)&A = "+((A|B)&A));
System.out.println("(A&&B)&A        =        "+((A&&B)&A));
System.out.println ("(A==B)&A = "+((A==B)&A));
} }
```

Output

```
A|B=true
A&B=false
!A=false
A^B=true
(A|B)&A=true
(A&&B)&A=false
(A==B)&A=false
```

Assignment Operators

The assignment operator is the individual equal sign, =. The assignment operator works in Java much as it does in any other computer language. It has this general form:

var = expression;

Here, the type of var must be compatible with the type of expression.

The assignment operator does have one interesting attribute that you may not be familiar with: it allows you to create a chain of assignments. For example, consider this fragment:

```
int i, j, k;
i = j = k = 200; // set i, j, and k to 200
```

This fragment sets the variables i, j, and k to 200 using a single statement. It works because the = is an operator that yields the value of the right-hand expression. Thus, the value of k = 200 is 200, which is then assigned to j, which in turn is assigned to i. Using a "chain of assignment" is a simple procedure to set a group of variables to a standard value

Operator Precedence

Precedence by name implies the order of code execution. For example, addition and subtraction has lower precedence than multiplication and division. Explicit parenthesis are used to override the precedence. Operator precedence in Java can be classified as

- Precedence order
- Associativity
- Precedence and Associativity of Java

Operator Precedence

Precedence Order

Operator with higher precedence goes first, when two operators share their operand.

For example:

1+4/2 is treated as 1+ (4 / 2)

Operator Precedence

Associativity

Associativity means the order in evaluating the expression with similar precedence.

Let's see an **example** of how associativity works.

a=b=c=15 can be written as a=(b=(c=15))

This happens because, **= operator** has right-to-left associativity and an assignment

statement always evaluates to the value on the right hand side.

Operator Precedence

Precedence and Associativity of Java

Let's have a look on the table below which shows all Java operators from highest to lowest precedence, along with their associativity.

Notify that the first row shows items that you may not usually think of as operators: parentheses, square brackets, and the dot operator. Technically, these are called separators, but they act like operators in an expression. Parentheses are used to alter the precedence of an operation. As you know from the previous chapter, the square brackets give array indexing

Operator Precedence

Operator	Description	Associativity
[]	access array element	Left to right
.	access object member	
()	invoke a method	
++	post-increment	
- -	post-decrement	
++	pre-increment	Right to left
- -	pre-decrement	
+	unary plus	
-	unary minus	
!	logical NOT	
~	bitwise NOT	
*	multiplicative	left to right
/		
%		

Operator Precedence

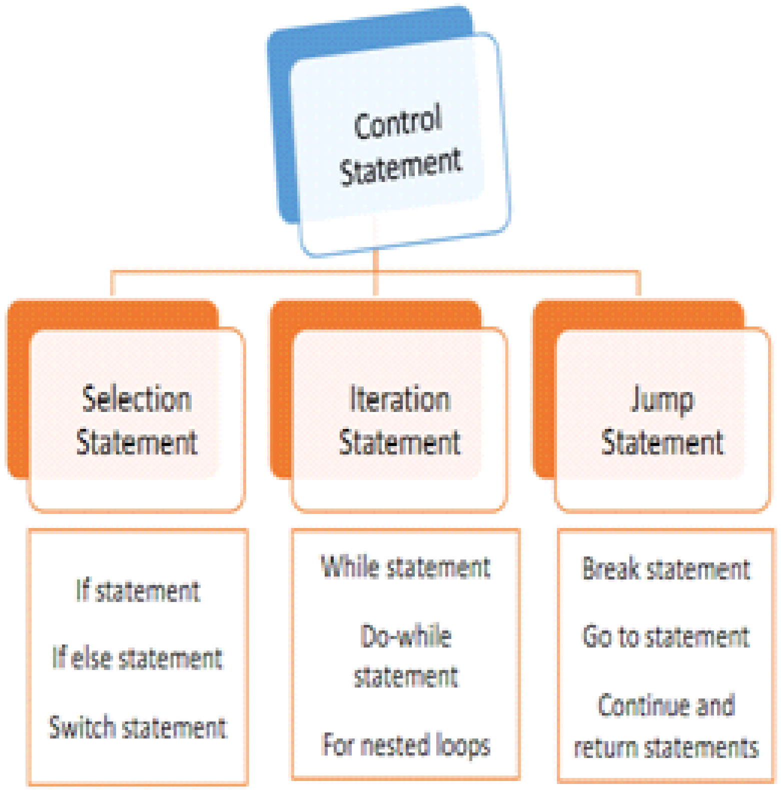
Operator	Description	Associativity
< <= > >=	relational type comparison	left to right
== !=	equality	right to left
&	bitwise AND	left to right
^	bitwise XOR	left to right
|	bitwise OR	left to right
&&	conditional AND	left to right
||	conditional OR	left to right
?:	conditional	left to right

Control Statements

Control Statements

In Java, control statements are used to control the order of execution of the program. They are split up into three categories

- Selection statement
- Iteration statement
- Jump statement



Selection Statement

In Java, selection statements are used to

- Find different path of execution based on the outcome of an expression or state of a variable.
- Control to the execution of the program.

If statement

If statements also called as conditional statement..If statement executes the statements associated with it only if the specified condition is true. Else it will skip the execution and continue with the rest of program, if **else** keyword is specified.

Selection Statement - Sample Program

```
import java.util.Scanner;
public class If
 {
public static void main(String  args[ ])
{
int age;
Scanner inputDevice = new Scanner (System.in);
System.out.println("Enter Your Age");
age = inputDevice.nextInt();
if(age>15)
System.out.println("above 15");
}}
```

Output

```
Enter Your Age   20
above 15
```

If else statement

In the preceding section, we used only a simple if statement. If a condition turns out to be true, then it will execute statements in the curly braces. What if that condition was false? The execution will go out of the if block to the next line of code. But, you may want to perform some action or show some message if the condition is false. The Java if-else statement can be used with the if statement when a condition is false.

Syntax for if and else statement

```
If(condition){
    Statement ;  // Code to be executed if condition is true
}
else{
    Statement ;}   // Code to be executed if condition is true
```

Switch Statement

The switch statement is a multiway branch statement. It provides an easy way to dispatch execution to different parts of your code based on the value of an expression. Such as, it often provides a better alternative than an extensive series of if-else-if statements.

General Syntax

```
switch(expression){
case value1:
//statement sequence
break;
case value1:
//statement sequence
break;
.
.
.
case valueN:
//statement sequence
break;
default:
// default statement sequence
}
```

Switch Statement

The expression must be the byte, char, int, or short; each of the values specified in the case statements must be of a type compatible with the expression. Each case value must be an incomparable literal (that is, it must be a constant, not a variable). Duplicate case values are not allowed.

Working of the switch statement :

The value of the expression is associated with each of the literal values in the case statements. If a match is detected, the code sequence following that case statement is done. If constants do not match with the value of the expression, then the default statement is done. Though, the default statement is arbitrary. If no case matches and also no default is being, then no more action is taken.

The break statement is applied within the switch statement to end a statement sequence. When a break statement is encountered, execution branches to the first line of code that follows the entire switch statement. It has the effect of "jumping out" of the switch.

Nested switch statements

You can apply a switch as part of the statement series of an external switch. It is called a nested switch. Since a switch statement defines its block, no conflicts arise between the case constants in the internal switch and those in the external switch. For example, the following the fragment is entirely valid

General Syntax:

```
switch(count) {
case 1:
switch(target) { // nested switch
case 0:
System.out.println("target is zero");
break;
case 1: // no conflicts with the external
switch
System.out.println("target is one");
break;
}
break;
case 2: // ...
```

Iteration Statement

Java's iteration statements are for "for ", "while ", and "do-while ". These statements create what we commonly call loops. As you reasonably know, a loop regularly executes the same set of guidance until a termination condition is met. As you will see, Java has a loop to fit any programming need.

There are three types of iteration statements. They are:

- while statement
- do-while statement
- for statement

While statement

Let’s see what does while statement do in Java programming.

- It uses a Boolean expression to control iteration.
- It executes as long as the boolean expression is true.
- Set of statements are repeated until the condition for termination is met.

Syntax

```
while (condition)
{
//body of the while loop
}
```

Sample Program

```
Class Example
{
public static void main (String  args[ ])
{
  int count =1;
  While (count<5)
{
 System.out.println (“count is:”+count);
count++;
  } }}
```

Output:

```
1 2 3 4
```

Do – While Statement

As you just noticed, if the conditional expression controlling a while loop is initially invalid, then the body of the loop will not be executed at all. Though, sometimes it is desirable to run the body of a loop at least once, despite if the conditional expression is wrong to begin with. In other words, when you would like to check the termination expression at the end of the loop rather than at the beginning. Fortunately, Java provides a circuit that does just that: the do-while. The do-while loop always executes its body at least once, because its conditional expression is at the bottom of the loop. Its general form is:

Syntax for do-while statement

```
do
{
  (statements);
}
while (expression);
```

Do – While Statement

Starting with JDK 5, there are two forms of the "for" loop. The first one is the old form that has been in use since the original version of Java. The second is the new "for-each" form. Both types of for loops are explained here, starting with the old form.

Here is the general form of the traditional for statement:

Syntax

```
for (initialization; condition; increment/decrement)
{
    Statements;
}
```

Nested Loop

Nested loops are very helpful in processing information. It can be described as, if one loop contains another then the second loop is said to be nested inside the first. In Java programming, any number of loops can be nested.

Syntax - nested loop statement

```
for(initialization; test condition;
increment/decrement)
{
statements;
while(expression)
{
statements;
do
{
statements;
}while(expression);
}
}
```

Nested Loop

Syntax – nested for statement

```
for(initialization; test condition;
increment/decrement)
{
statements;
for(initialization; test condition;
increment/decrement)
{
statements;
for(initialization; test condition;
increment/decrement)
{
statements; }}}
```

Syntax – nested while

```
While(expression)
{
statements; While(expression)
{
statements;
}}
```

Jump Statements

Java supports three jump statements: ***break***, ***continue***, and ***return***. Certain statements shift control to another part of the program. Each is examined here

- Break
- Continue
- Return

Break Statement

In Java Language, the break statement has three uses. First, as you have seen, it terminates a state In Java, the break statement has three purposes. First, as you have seen, it ends a statement sequence in a switch statement. Second, it can be used to exit a loop. Third, it can be utilized as a "civilized" form of goto. One statements are explained here.

Break statements are used to:

- Terminate the statement sequence.
- Transfer controls to other parts of the program.
- Exit from a loop, if any specified conditions evaluates to true, then the break statement can be used to terminate the loop instantly.

Syntax for break statement

```
break;
```

Go – To Statement

Goto statement is used to transfer the control to the user desired location. This statement refer the label in the same function only.

Syntax

```
goto label;
```

Continue Statement

Continue statements are used

- Inside the bounds of loop statement itself.
- To continue the conditional statements until it satisfies the specified condition.
- To skip the rest of the statements in the body of the loop and continue with the next iteration of the loop

Syntax

```
Continue;
```

Continue Statement

Continue statements are used

- Inside the bounds of loop statement itself.
- To continue the conditional statements until it satisfies the specified condition.
- To skip the rest of the statements in the body of the loop and continue with the next iteration of the loop

Syntax

```
Continue;
```

Difference between break and continue statements

break statement	continue statement
A break statement results in the termination of the statement to which it applies (switch , for , do , or while).	A continue statement is used to end the current loop iteration and return control to the loop statement.
The break statement results in the termination of the loop, it will come out of the loop and stops further iterations.	The continue statement stops the current execution of the iteration and proceeds to the next iteration. The return statement takes you out of the method. It stops executing the method and returns from the method execution.
A break statement when applied to a loop ends the statement.	A continue statement ends the iteration of the current loop and returns the control to the loop statement.
If the break keyword is followed by an identifier that is the label of a random enclosing statement, execution transfers out of that enclosing statement.	If the continue keyword is followed by an identifier that is the label of an enclosing loop, execution skips to the end of that loop instead.
a break statement discontinues the execution of the loop and gets the control out of the loop after meeting the condition	On the other hand the continue statement on meeting the condition, goes to the beginning of the loop and re-executes it.

Return Statement

In Java return statements are used to

- Return from a method.
- Transfer back the control to the caller of the method.
- Terminate the method in which it is performing currently

Syntax

```
return value;
```

Summary

✓ Operators and looping constructs directs the user how to perform, manipulate, control Java programs in its environment.

✓ Operators in Java are used to perform arithmetic operations.

✓ Relational operators are used to check relation between two variables or number.

✓ Precedence by name implies the order of code execution of a program.

✓ Selection statements allow the program to choose different ways of execution based upon the result of an expression or the state of a variable.

✓ Iteration statements allow program execution to repeat multiple statements (that is, iteration statements form loops).

✓ Jump statements allow your program to perform in a nonlinear fashion. All of Java control statements are examined here.

Summary

✓ In Java Language, the break statement has three uses. First, as you have seen, it terminates a state In Java, the break statement has three purposes. First, as you have seen, it ends a statement sequence in a switch statement. Second, it can be used to exit a loop. Third, it can be utilized as a “civilized” form of goto.

✓Java provides a circuit that does just that: the do-while. The do-while loop always executes its body at least once, because its conditional expression is at the bottom of the loop.