Chapter

5

SECTION II: LANGUAGE BASICS

Operators

Chapter 04: Basic Fundamentals demonstrated declaring and initializing variables. This chapter depicts how to do something with them, using operators.

Operators are an important part of any programming language and their function is simple **i.e.** they allow manipulating data.

Operators are special symbols or keywords used to designate mathematical operations or some other types of operation that are performed on one or more values called **operands**.

Operators are indispensable components of Java. Java provides around 40 different operators. Operators are used for simple purposes such as**:**

- Assigning values to variables
- Working with strings
- Working with numbers
- Controlling program flow

  And so on…

Operators in Java, fall into the following different categories**:**

- Assignment
- Arithmetic
- Relational
- Logical
- Bitwise
- Compound assignment
- Conditional
- Type

Whilst exploring operators in Java, it's also prudent to know which operators have the highest precedence. Precedence determines in which order operators execute.

The operators in the following table are listed according to precedence order:

Operators	Precedence					
Postfix	expr++	expr--				
Unary	++expr	--expr	+expr	-expr	~	!
Multiplication	*	/	%			
Additive	+	-				
Shift	<<	>>	>>>			
Relational	<	>	<=	>=	instanceof	
Equality	==	!=				
Bitwise AND	&					
Bitwise Exclusive OR	^					
Bitwise Inclusive OR	|					
Logical AND	&&					
Logical OR	||					
Ternary	?	:				
Assignment	=	+=	-=	*=	/=	%=
	&=	^=	|=	<<=	>>=	>>>=

From the above table, it is clear that the operator **Postfix** has the **highest precedence**.

Operators with higher precedence are evaluated **before** operators with relatively lower precedence.

Operators on the same line have equal precedence. When operators of equal precedence appear in the same expression, a rule must govern which is evaluated first.

All binary operators except for the assignment operators are evaluated from left to right and the assignment operators are evaluated right to left.

Properties Of Operators

In Java, an **expression** carries out some operation(s) that are directed by a list of allowed **operators**.

An **operator** acts upon one, two or three **operands**. The following are some general properties of operators**:**

Operands

An operand can be**:**

- A numeric variable **i.e.** integer, floating point or character
- Any primitive type variable **i.e.** numeric and boolean
- Reference variable to an object
- A literal **i.e.** numeric value, boolean value or string
- An array element **i.e.** "a[2]"
- Char primitive, which in numeric operations is treated as an unsigned two byte integer

The operator is**:**

- Unary, if it acts on a single operand
- Binary, if it requires two operands

The **conditional operator** is the only **ternary** operator in Java.

A unary operator may appear before [prefix] its argument or after [postfix] its argument, whereas, a binary or ternary operator appears between its arguments.

Returned Value

After an operation completes, a value is **returned**.

Example

The following example uses the assignment operator **=** and the addition operator **+**.

Solution:

```
int x = 3;
int y = x+5;
```

Explanation:

The above example results into**:**

- **x** holding the value **3**
- **y** holding the value **8** [3+5]

The assignment operator **=** in the statement returns the value of **y** i.**e.** 8.

Example

The following example uses the above statement **y=x+5** in another expression.

Solution:

```
int x=3;
int y;
int z = (y=x+5) * 4;
```

Explanation:

The above example results into**:**

- **y** holding the value **8**
- **z** holding the value **32**

The assignment operator **=** in the statement **y=x+5** produces a new value for **y** and returns the value of **y** to be used in the expression for **z**.

Arithmetic Operators

There are five operators used to accomplish basic arithmetic in Java.

Arithmetic Operators that Java provides are addition, subtraction, multiplication and division, which are very similar to the counterparts in basic mathematics. The only additional symbol that looks new is %, which divides one operand by another and returns the remainder as its result.

The following are the arithmetic operators:

Operator	Meaning	Example	Result
+	Addition	x=3 x+3	 6
-	Subtraction	x=3 5-x	 2
*	Multiplication	x=5 x*5	 25
/	Division	20/2 7/2	10 3.5
%	Modulus	5%3 7%2 20%2	2 1 0

The arithmetic operators can be combined with the simple assignment operator to create compound assignments.

Example

The following example shows how the value of x is incremented by 5.

Solution:

```
x+=5;
```

Alternatively:

```
x=x+5;
```

The **+** operator can also be used for concatenating/joining two strings together.

Example

Solution:

```
1  class ConcatString {
2     public static void main(String[] args){
3        String firstString = "Welcome ";
4        String secondString = "to Sharanam Shah website.";
5        String thirdString = firstString + secondString;
6        System.out.println(thirdString);
7     }
8  }
```

Compile the above code spec as follows:

```
<System Prompt:/> javac ConcatString.java
```

Run the compiled code spec as follows:

```
<System Prompt:/> java ConcatString
```

Output:

```
Welcome to Sharanam Shah website.
```

This program using the assignment operator, makes appropriate string assignments to the first two variables and then using + concatenates the two values, assigns the concatenated value to a third variable which now contains "Welcome to Sharanam Shah website." and finally outputs the concatenated value.

Example

Solution:

```
 1  class  ArithmeticOperator {
 2     public static void main(String[] args) {
 3        int num1 = 38 + 26;
 4        System.out.println ("Addition of two numbers is: " + num1);
 5
 6        int num2 = 25 * 25;
 7        System.out.println ("Multiplication of two numbers is: " + num2);
 8
 9        int num3 = 25 * 25 - 35;
10        System.out.println ("Multiplication and Subtraction of numbers are: " + num3);
11     }
12  }
```

Compile the above code spec as follows:

```
<System Prompt:/> javac ArithmeticOperator.java
```

Run the compiled code spec as follows:

```
<System Prompt:/> java ArithmeticOperator
```

Output:

```
Addition of two numbers is: 64
Multiplication of two numbers is: 625
Multiplication and Subtraction of numbers are: 590
```

Assignment Operators

Variables were invented with an objective to hold something, which is usually data. That's where the assignment operator comes in.

To fill a variable with data, an assignment operator such as **=** [equal] symbol can be used.

Assignment operator enables writing a value to a variable. The first operand [the one on the left of the assignment operator] must be a variable. The value of an assignment is the final value assigned to the variable. This operator can be used to assign strings, integers even the results of functions

Syntax:

```
<variable> = <expression>
```

Example

Solution:

```
int var = 5;
String str = 'Sharanam Shah';
Object fn = MyFunction();
```

Explanation:

In the above code spec:

- The variable **var** is assigned the value **5**
- The variable **str** is assigned the value **Sharanam Shah**
- The variable **fn** is assigned the value **returned by the function MyFunction()**

The following table displays the expressions and its meaning:

Operators	Expression	Meaning
=	x=y	x=y
+=	x += y	x = x + y
-=	x -= y	x = x – y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

Example

Solution:

```
1  class AssignmentOperator{
2     public static void main(String[] args) {
3        int num1 = 40;
4        int num2 = num1;
5        System.out.println ("Second number is: " + num2);
6
7        num2 *= num1;
8        System.out.println ("Multiplication of two numbers is: " + num2);
9
10       num2 /= num1;
11       System.out.println ("Division of two numbers is: " + num2);
12    }
13 }
```

Compile the above code spec as follows:

```
<System Prompt:/> javac AssignmentOperator.java
```

Run the compiled code spec as follows:

```
<System Prompt:/> java AssignmentOperator
```

Output:

```
Second number is: 40
Multiplication of two numbers is: 1600
Division of two numbers is: 40
```

Increment and Decrement

There are operators which are unique i.e. they operate only on variables and not on any value. The reason for this is that in addition to calculating the result value, the value of the variable itself changes as well.

These special operators are called increment and decrement operators.

The increment operator is **++** and the decrement operator is **--**. These operators are placed immediately after or immediately before a variable name.

Increment and decrement operators are called *prefix* and *postfix* operators.

The following table shows the increment and decrement examples:

Example	Description	Result	Value of the Variable
var++	Post increment	var is incremented by 1	The previous value of var
++var	Pre increment	var is incremented by 1	The new value of var (incremented value)
var--	Post decrement	var is decremented by 1	The previous value of var
--var	Pre decrement	var is decremented by 1	The new value of var (decremented value)

Example

Consider the following example.

Solution:

```
1  class  IncrementDecrement {
2     public static void main(String[] args) {
3        int x, y, z;
4        x = 42;
5        y = x++;
6        System.out.println("Post Increment: " + y);
7        z = ++x;
8        System.out.println("Pre Increment: " + z);
9     }
10 }
```

Compile the above code spec as follows:

```
<System Prompt:/> javac IncrementDecrement.java
```

Run the compiled code spec as follows:

```
<System Prompt:/> java IncrementDecrement
```

Output:

```
Post Increment: 42
Pre Increment: 44
```

Explanation:

In the above code spec**:**

- The variable x is given the value 42
- The variable y is given the variable x's value i.e. 42 before it is incremented and then the value of x is increment to 43

This is the example of *post-increment*. Here, the assignment takes place first and increment takes place latter. Hence the value of y is 42.

- The value of x is incremented to 44 and then the value of z is given the variable x's value

  This is the example of *pre-increment*. Here, the increment takes place first and assignment takes place next. Hence, the value of z is 44 and not 43.

Relational Operators

Java has several operators, known as **relational operators**, which are used when making comparisons between variables and any other type of information in the program.

A relational operator compares two values and determines the relationship between them.

All relational operators are binary operators and their operands are numeric expressions.

Binary numeric promotion is applied to the operands of these operators. These operators are used in expressions that returns Boolean values of *true* or *false*, depending on whether the comparison being made is true or not. Relational operators have precedence lower than arithmetic operators, but higher than that of the assignment operators.

The following table shows a list of comparison operators:

Operator	Meaning	Example	Returns
= =	Equal	3 == 9	False
!=	Not equal	3 != 9	True
<	Less than	3 < 9	True
>	Greater than	15 > 18	False
<=	Less than or equal to	3 <= 9	True
>=	Greater than or equal to	15 >= 18	False

Example

Solution:

```
1  class  RelationalOperator {
2     public static void main(String[] args) {
3        int num1 = 40;
4        int num2 = 50;
5        if (num1 < num2) {
6           System.out.println(num1 + " is less than " + num2);
7        }
```

```
8     }
9  }
```

Compile the above code spec as follows:

```
<System Prompt:/> javac RelationalOperator.java
```

Run the compiled code spec as follows:

```
<System Prompt:/> java RelationalOperator
```

Output:

```
40 is less than 50
```

Logical Operators

Expressions that result in Boolean values such as relation operators can be combined to form more complex expressions. This is handled through logical operators. These operators are used for the logical combination AND, OR and the logical NOT.

Operator	Meaning	Description	Example	Returns
&&	AND	When two Boolean expressions are linked by the && operators, the combined expression returns a true only if both the Boolean expressions is true.	(A && B)	False
||	OR	When two Boolean expressions are linked by the || operators, the combined expression returns a true, if either of the Boolean expressions is true.	(A || B)	True
!	NOT	It reverses the value of a Boolean expression the same way that a minus symbol reverses the positive or negative sign on a number.	!(A && B)	True

Example

Solution:

```
1  class  LogicalOperator {
2     public static void main(String[] args) {
3        boolean a = true;
4        boolean b = false;
5        System.out.println("a && b = " + (a&&b) );
6        System.out.println("a || b = " + (a||b) );
7        System.out.println("!(a && b) = " + !(a && b) );
8     }
9  }
```

Compile the above code spec as follows:

```
<System Prompt:/> javac LogicalOperator.java
```

Run the compiled code spec as follows:

```
<System Prompt:/> java LogicalOperator
```

Output:

```
a && b = false
a || b = true
!(a && b) = true
```

Bitwise Operator

Java provides Bit wise operators to manipulate the contents of variables at the bit level.

Bitwise operators perform an operation on the bitwise representation of their arguments. Unless the arguments are strings, they are converted to their corresponding integer representation and the operation is then performed.

These variables must be of numeric data type such as char, short, int or long.

The following table displays the bitwise operators:

Operator	Meaning	Description
&	Bitwise AND	Compares all the bits in operand one against all the bits on operand two and returns a result with all the joint bits set.
¦, or, |	Bitwise OR	Compares all the bits in operand one against those in operand two and returns a result with all the bits set in either of them.
^	Bitwise XOR (Exclusive OR)	Compares all the bits in operand one against those in operand two and returns a result with all the bits set in either of them but **not both**.
~	Unary Bitwise Compliment	Flip each bit making every zeros a one and every one a zero. For example, a byte contains 8 bits, applying this operator to a value whose bit pattern is **00000000** would change to **11111111**
<<	Signed Left Shift	Shifts a bit to the left. High order bits lost. Zero bits fill in right bits.
>>	Signed Right Shift	Shifts a bit to the right. Low order bits lost. Same bits value as sign fills in the left bits. Zero for positive numbers and 1 for negative numbers.
>>>	Unsigned Right Shift	Shifts a zero into the leftmost position while the leftmost position after >> depends on sign extension.

Example

Solution:

```
1  class BitwiseOperator {
2     public static void main(String[] args) {
3        int num1 = 12 ^ 9;
4        int num2 = 12 & 9;
5        int num3 = 12 | 9;
6        int x = 0xFAEF;
7        int y = 0xF8E9;
8        System.out.println(num1);
9        System.out.println(num2);
10       System.out.println(num3);
11       System.out.println(x << y);
12       System.out.println(x >> y);
13       System.out.println(x >>> y);
14    }
15 }
```

Compile the above code spec as follows:

```
<System Prompt:/> javac BitwiseOperator.java
```

Run the compiled code spec as follows:

```
<System Prompt:/> java BitwiseOperator
```

Output:

```
5
8
13
32890368
125
125
```

Conditional Operator

The conditional operator **? :** can be thought of as shorthand for an if-then-else statement.

This operator is also known as the **ternary operator** because it uses three operands. It is the only ternary operator in Java.

The operator evaluates the first argument and, if true, evaluates the second argument. If the first argument evaluates to false, then the third argument is evaluated.

Syntax:

```
variable x = (expression) ? value if true : value if false
```

Example

Solution:

```
1  class ConditionalOperator {
2     public static void main(String[] args) {
3        int a , b;
4        a = 10;
5        b = (a == 1) ? 20: 30;
6        System.out.println( "Value of b is : " +  b );
7        b = (a == 10) ? 20: 30;
8        System.out.println( "Value of b is : " + b );
9     }
10 }
```

Compile the above code spec as follows:

```
<System Prompt:/> javac ConditionalOperator.java
```

Run the compiled code spec as follows:

```
<System Prompt:/> java ConditionalOperator
```

Output:

```
Value of b is : 30
Value of b is : 20
```

The instance Of Operator

The instanceOf operator is used for object reference variables. It checks whether an object is of a particular type.

The instanceOf operator is used to test if an object is an instance of a class, an instance of a subclass or an instance of a class that implements a particular interface.

Syntax:

```
(Object reference variable) instanceOf  (class/interface type)
```

If the object referred by the variable on the left side of the operator passes the **IS-A** check for the class/interface type on the right side then the result is true.

Example

Solution:

```
1  public class  InstanceOfOperator {
2     public static void main(String[] args) {
3        String s = "Sharanam Shah";
4        if (s instanceof java.lang.String) {
5          System.out.println("TRUE");
6        }
7        else {
8          System.out.println("FALSE");
9        }
10    }
11 }
```

Compile the above code spec as follows:

```
<System Prompt:/> javac InstanceOfOperator.java
```

Run the compiled code spec as follows:

```
<System Prompt:/> java InstanceOfOperator
```

Output:

```
TRUE
```

Example

Solution: [Parent.java]

```
1  class Parent {
2    public Parent() {
3
4    }
5  }
```

Solution: [Child.java]

```
1  class Child extends Parent {
2    public Child() {
3      super();
4    }
5  }
```

Solution: [MainClass.java]

```
1  public class MainClass {
2     public static void main(String[] a) {
3        Child child = new Child();
4        if (child instanceof Parent) {
5           System.out.println("true");
6        }
7     }
8  }
```

Compile the above code spec as follows:

```
<System Prompt:/> javac MainClass.java
```

Since this file is dependent on other .java files such as Parent.java and Child.java all of these are automatically compiled.

Run the compiled code spec as follows:

```
<System Prompt:/> java MainClass
```

Output:

```
true
```

HINT

When using the instanceOf operator, keep in mind that NULL is not an instance of anything.

The Book CDROM holds the complete application source code for the following applications:

- Codespecs \ Section 2 \ Chapter05_Cds \ **ConcatString.java**
- Codespecs \ Section 2 \ Chapter05_Cds \ **ArithmeticOperator.java**
- Codespecs \ Section 2 \ Chapter05_Cds \ **AssignmentOperator.java**
- Codespecs \ Section 2 \ Chapter05_Cds \ **IncrementDecrement.java**
- Codespecs \ Section 2 \ Chapter05_Cds \ **RelationalOperator.java**
- Codespecs \ Section 2 \ Chapter05_Cds \ **LogicalOperator.java**
- Codespecs \ Section 2 \ Chapter05_Cds \ **BitwiseOperator.java**
- Codespecs \ Section 2 \ Chapter05_Cds \ **InstanceOfOperator.java**
- Codespecs \ Section 2 \ Chapter05_Cds \ **Parent.java**
- Codespecs \ Section 2 \ Chapter05_Cds \ **Child.java**

- Codespecs \ Section 2 \ Chapter05_Cds \ **MainClass.java**

The application can be directly used by compiling and executing it.