Chapter 2 - Operators and Statements

Bob

May 11, 2022

1 Java Operators

Java Operator : A Java operator is a special symbol that can be applied to a set of variables, values, or literals - referred to as operands - and that returns a result.

Types of Java Operators : There are three flavors of operators available in Java:

- unary
- binary
- ternary

1.1 Order of Operation

Unless overridden with parentheses, Java operators follow *order of operation*, listed in this table by decreasing order of *operator precedence*. If two operators have the same level of precedence, then Java guarantees left-to-right evaluation.

Operator	Symbols and examples
Post-unary operators	expression++, expression–
Pre-unary operators	++expression, –expression
Other unary operators	,+,-,!
Multiplication/Division/Modulus	*,/,%
Addition/Subtraction	+,-
Shift operators	¡¡,¿¿,¿¿¿
Relational operators	¡,¿,¡=,¿=, instanceof
Equal to/not equal to	==, !=
Logical operators	&,ˆ,—
Short-circuit logical operators	&&,——
Ternary operators	boolean expression ? expression1 : expression2
Assignment operators	=,+=,-=,*=,/=,%=,&=, ˆ=, —=, ¡¡=, ¿¿=, ¿¿¿=

1.2 Binary Operators

1.2.1 Arithmetic Operators

All of the arithmetic operators may be applied to any Java primitives, except *boolean* and *String*. Furthermore, only the addition operators + and += may be applied to String values, whcih results in String concatenation.

1.2.2 Numeric Promotion

1. If two values have different data types, Java will automatically promote one of the values to the larger of the two data types.
2. If one of the values is integral and the other is floating-point, Java will automatically promote the integral value to the floating-point value's data type.
3. Smaller data types, namely *byte*, *short* and *char*, are first promoted to *int* any time they're used with a Java binary arithmetic operator, even if neither of the operands is *int*.
4. After all promotion has occurred and the operands have the same data type, the resulting value will have the same data type as its promoted operands.

Note: Unary operators are excluded from this rule.

1.3 Unary Operators

Unary Operator : A unary operator is onen that requires exactly one operand, or variable, to function.

Unary Operator	Description
+	Indicates a number is positive, although numbers are assumed to be positive in Java
-	Indicates a literal number is negative or negate an expression
++	Increments a value by 1
–	Decrements a value by 1
!	Inverts a Boolean's logical value

Negation Operator (-) : Reverses the sign of a numeric expression. **Logical Complement Operator (!)** : Flips the value of a boolean expression.

Note: Unlike some other programming languages, in Java *1* and *true* are not related in any way, just as *0* and *false* are not related.

Increment and Decrement Operators (++, –) : Increment and decrement operators, ++ and –, respectively, can be appied to numeric operands and have the higher order or precedence, as compared to binary operators. In other words, they often get applied first to an expression.

1.4 Additional Binary Operators

1.4.1 Assignment Operator

An *assignment operator* is a binary operator that modifies, or assigns, the variable on the left-hand side of the operator, with the result of the value on the right-hand side of the equation.

Note: **Java will automatically promote from smaller to larger data types**, but it will throw a compiler exception if it detects you are trying to convert from larger to smaller data types.

1.4.2 Casting Primitive Values

Casting primitives is required any time you are going from a larger numerical data type to a smaller numerical data type, or converting from a floating-point number to an integral value. By performing this **explicit cast** of a larger value into a smaller dat type, you are instructing the compiler to ignore its default behavior. You are telling the compiler that you have taken additional steps to prevent overflow or underflow.

Overflow and Underflow : *Overflow* is when a number is so large that it will no longer fit within the data type, so the system "wraps around" to the next lowest value and counts up from there. There's also an analogous *underflow*, when the number is too low to fit in the data type.

1.4.3 Compound Assignment Operators

Besides the simple assignment operator, =, there are also numerous *compound assignment operators*. Complex operators are really just glorified forms of the simple assignment operator, with a built-in arithmetic or logical operation that applies the left- and right-hand sides of the statement and stores the resulting value in a variable in the left-hand side of the statement. The lef-hand side of the compound operator can only be applied to a variable that is already defined and cannot be used to declare a new variable.

1.4.4 Relational Operators

Relational operators compare two expressions and return a *boolean* value. If the two numeric operands are not of the same data type, the smaller one is promoted in the manner as previously discussed.

Operator	Description
¡	Strictly less than
¡=	Less than or equal to
¿	Strictly greater than
¿=	Greater than or equal to

1.4.5 Logical Operators (&, — and ^)

The *logical operators*, (&), (—), and (^), may be applied to both *numeric* and *boolean* data types, they're referred to as *logical operators*. Alternatively, when they're applied to *boolean* data types, they're referred to as *bitwise operators*, as they perform bitwise comparisons of the bits that compose the number.

1.4.6 Truth Tables

x & y (AND)

—	y = true	y = false
x = true	true	false
x = false	false	false

x — y (INCLUSIVE OR)

—	y = true	y = false
x = true	true	true
x = false	true	false

x ŷ (EXCLUSIVE OR)

—	y = true	y = false
x = true	false	true
x = false	true	false

1.5 Shortcut Operators (&& and ——)

The short-circuit operators are nearly identical to the logical operators, & and —, respectively, except that **the right-hand side of the expression may never be evaluated if the final result can be determined by the left-hand side of the expression.**

1.6 Equality Operators

Determining equality in Java can be a nontrivial endeavor as ther's a semantic difference between "two objects are the same" and "two objects are equivalent". It is further complicated by the fact that for numeric and boolean primitives, there is no such distinction.

The *equals* operator == and *not equals* operator *!=*. Like the relational operators, they compare two operands and return a boolean value about whether the expressions or values are equal, or not equal, respectively. The equality operators are used in one of three scenarios:

1. Comparing two numeric primitive types. If the numeric values are of different data types, the values are automatically promoted.

2. Comparing two *boolean* values.

3. Comparing two objects, including *null* and *String* values.

Note: Pay close attention to the data types when you see an equality operator on the exam. (= vs ==)

For object comparison, the equality operator is applied to the references to the objects, not the objects they point to. Two references are equal if and only

if they point to the same object, or both point to *null*. (objectA == objectB vs objectA.equals(objectB).

2 Java Statements

Java Statement : is a complete unit of execution in Java, terminated with a semicolon (;).

Control Flow Statement : Break up the flow of execution byusing decision making, looping, and branching, allowing the application to selectively execute particular segments of code.

Block of Code : is a group of zero or more statements between balanced braces (), and can be used anywhere a single statement is allowed.

Decision making control structures:

- if-then
- if-then-else
- switch

Repetition control structures:

- for
- for-each
- while
- do-while

2.1 The if-then statement

The *if-then* statement execute a block of code under certain circumstances, by allowing our application to execute a particular block of code if and only if a *boolean* expression evalueates to true at runtime.

```
if(booleanExpression) {
   // Branch if true
}
```

For readability, it is considered good coding practice to put blocks around the execution component of *if-then* statements, as well as many other control flow statements, although it is not required.

Note: In Java tabs are just whitespace and are not evaluated as part of the execution.

2.2 The if-then-else statemnt

```
if(booleanExpression) {
   // Branch if true
} else {
   // Branch if false
}
```

2.3 The switch statement

2.4 The while statement

2.5 The do-while statement

2.6 The for statement

2.7 The for-each statement