AP COMPUTER SCIENCE
JAVA CONCEPTS III: OPERATORS AND EXPRESSIONS

PAUL L. BAILEY

1. Operators

Operators are functions which are part of the language and appear in expressions as punctuation between values. Operators take one, two, or three typed values and return a single typed value. Examples include logical not (`!a`), arithmetic sum (`a+b`), or relational less than or equal to (`a<=b`).

Unary operators (like "not" or "negate") act on one value, *binary* operators (such as "plus" or "mod") act on two values, and *ternary* operators act on three values.

Expressions are sequences of constants and variables, possibly modified using unary operators, and joined by binary operators. Java has well-defined rules for specifying the order in which the operators in an expression are evaluated when the expression has several operators. For example, multiplication and division have a higher precedence than addition and subtraction. Precedence rules can be overridden by explicit parentheses.

Precedence determines the order in which operators are applied. When two operators share an operand the operator with the higher precedence goes first. For example, $1 + 2 * 3$ is treated as $1 + (2 * 3)$, whereas $1 * 2 + 3$ is treated as $(1 * 2) + 3$ since multiplication has a higher precedence than addition.

Associativity determines the placement of implied parentheses for operators of equal precedence. When two operators with the same precedence occur, the expression is evaluated according to its associativity. For example $x = y = z = 17$ is treated as $x = (y = (z = 17))$, leaving all three variables with the value 17, since the $=$ operator has right-to-left associativity (and an assignment statement evaluates to the value on the right hand side). On the other hand, $72 / 2 / 3$ is treated as $(72 / 2) / 3$ since the / operator has left-to-right associativity.

The table below shows all Java operators from highest to lowest precedence, along with their associativity. Most programmers do not memorize them all or the rules of precedence and associativity, and even those that do still use parentheses for clarity.

Date: September 21, 2019.

Operator	AP	Description	Level	Associativity
`++`	✓	post-increment	1	left
`--`	✓	post-decrement	1	left
`++`	✓	pre-increment	2	right
`--`	✓	pre-decrement	2	right
`+`	✓	unary plus	2	right
`-`	✓	unary minus	2	right
`!`	✓	logical NOT	2	right
`~`		bitwise NOT	2	right
`*`	✓	multiplication	4	left
`/`	✓	division	4	left
`%`	✓	modulo	4	left
`+`	✓	addition	5	left
`-`	✓	subtraction	5	left
`<<`		shift left	6	left
`>>`		shift right	6	left
`>>>`		shift right unsigned	6	left
`<`	✓	less than	7	left
`<=`	✓	less than or equal to	7	left
`>`	✓	greater than	7	left
`>=`	✓	greater than or equal to	7	left
`==`	✓	equals	8	left
`!=`	✓	not equals	8	left
`&`		bitwise AND	9	left
`^`		bitwise XOR	10	left
`|`		bitwise OR	11	left
`&&`	✓	conditional AND	12	left
`||`	✓	conditional OR	13	left
`? :`		ternary conditional	14	right
`=`	✓	assignment	15	right

The following additional assignment operators perform an operation, and assign the result.

`*=`	`/=`	`%=`
`+=`	`-=`	
`<<=`	`>>=`	`>>>=`
`&=`	`^=`	`|=`

2. Type Conversions

Every expression written in the Java programming language has a type that can be deduced from the structure of the expression and the types of the literals, variables, operators, and methods mentioned in the expression. It is possible, however, to write an expression in a context where the type of the expression is not appropriate. In some cases, this leads to an error at compile time. In other cases, the context may be able to accept a type that is related to the type of the expression; as a convenience, rather than requiring the programmer to indicate a type conversion explicitly, the Java programming language performs an implicit conversion from the type of the expression to a type acceptable for its surrounding context.

In every conversion context, only certain specific conversions are permitted. For convenience of description, the specific conversions that are possible in the Java programming language are grouped by the official Java documentation into several broad categories:

- Identity conversions
- Widening primitive conversions
- Narrowing primitive conversions
- Widening reference conversions
- Narrowing reference conversions
- Boxing conversions
- Unboxing conversions
- Unchecked conversions
- Capture conversions
- String conversions
- Value set conversions

The rules regarding type conversion are complex, but we mention the key points here.

2.1. **Widening Primitive Conversion.** 19 specific conversions on primitive types are called the widening primitive conversions:

- `byte` to `short`, `int`, `long`, `float`, or `double`
- `short` to `int`, `long`, `float`, or `double`
- `char` to `int`, `long`, `float`, or `double`
- `int` to `long`, `float`, or `double`
- `long` to `float` or `double`
- `float` to `double`

A widening primitive conversion does not lose information about the overall magnitude of a numeric value.

A widening primitive conversion from an integral type to another integral type, or (normally) from float to double, does not loose any precision at all; the numeric value is preserved exactly.

A widening conversion of an int or a long value to float, or of a long value to double, may result in loss of precision.

2.2. **Narrowing Primitive Conversion.** 22 specific conversions on primitive types are called the narrowing primitive conversions:

- `short` to `byte` or `char`
- `char` to `byte` or `short`
- `int` to `byte`, `short`, or `char`
- `long` to `byte`, `short`, `char`, or `int`
- `float` to `byte`, `short`, `char`, `int`, or `long`
- `double` to `byte`, `short`, `char`, `int`, `long`, or `float`

A narrowing primitive conversion may lose information about the overall magnitude of a numeric value and may also lose precision and range.

A narrowing primitive conversion from double to float may loose precision, but generally maintains the approximate value of the number.

A narrowing conversion of a signed integer or char to an integral type simply discards all but the lowest order bits.

2.3. **Boxing Conversion.** Boxing conversion converts expressions of primitive type to corresponding expressions of reference type. Specifically, the following nine conversions are called the boxing conversions:

- From type `boolean` to type `Boolean`
- From type `byte` to type `Byte`
- From type `short` to type `Short`
- From type `char` to type `Character`
- From type `int` to type `Integer`
- From type `long` to type `Long`
- From type `float` to type `Float`
- From type `double` to type `Double`
- From the `null` type to the `null` type

These types of conversions are necessary in Java because of the distinction between primitive types and classes. In particular, primitive types do not inherit from the class `Object`, so to pass primitive types to a method which takes an object as a parameter, Java "autoboxes" the value.

2.4. **Unboxing Conversion.** Unboxing conversion converts expressions of reference type to corresponding expressions of primitive type.

Reference: `http://docs.oracle.com/javase/tutorial/java/nutsandbolts/`
Reference: `http://docs.oracle.com/javase/specs/jls/se7/html/jls5.html`
Reference: `http://www.cs.umd.edu/~clin/MoreJava/Intro/varident.html`

Department of Mathematics, BASIS Scottsdale
Email address: `paul.bailey@basised.com`