Object-Oriented Programming I

Expressions In-Depth

Slides by Magdin Stoica
Updates by Georg Feil

Learning Outcomes

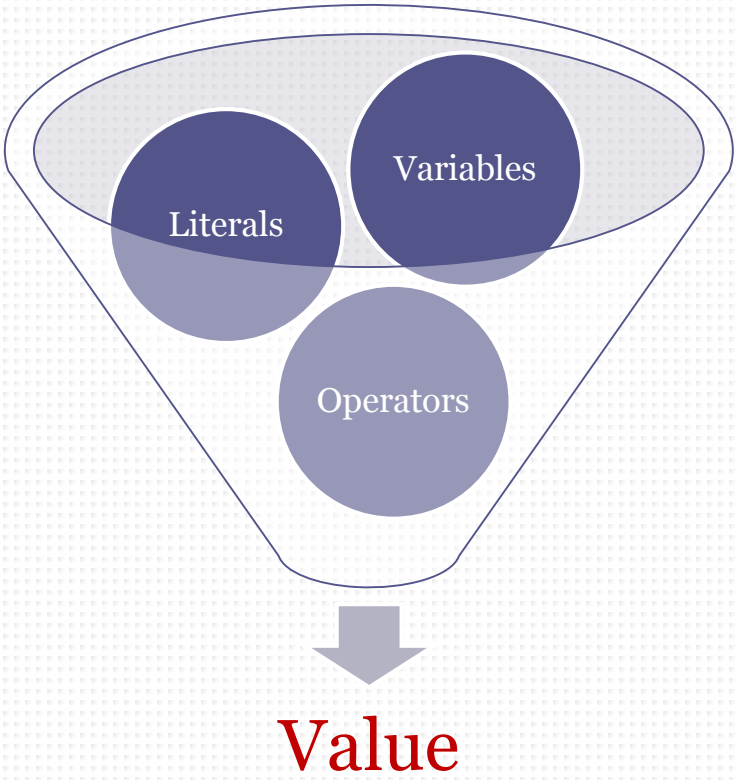
1. Analyze the components and role of expressions in program development
2. Identify types of operators based on number of operands, and types of expressions
3. Evaluate expressions that use arithmetic operators, logic operators, comparison operators and others.
4. Analyze various assignment operators and their use in formulating assignment expressions

Reading Assignment

- Introduction to Java Programming (required)
  - Chapter 2: Elementary Programming, Sections 2.13 to 2.15
  - Chapter 3: , Sections 3.11 to 3.13

Expressions

- Variables and literals (values) are combined using operators in expressions
- An expression always has a value
  - The resulting value of the combination
- Operators act on values, and may require 1, 2 or 3 values
- Operators are classified into different groups depending on the type of values they can combine and the type of the result
  - Arithmetic
  - Comparison
  - Logical
  - Assignment



Operators and Operands

- In an expression, operands are the variables and literals that are combined using operators
- Every operator has a precise number of operands that it requires
  - Unary operators only require one operand
  - Binary operators require two operands
  - Ternary operators require three operands
- An operand can be another expression leading to composite expressions.
- An expression can only be correctly formed if operators are used with the proper number and type of operands it expects:
  - A comparison can only be done between two operands of similar types (e.g. numbers of any kind of can be compared, a number and a string cannot)
  - A logical operator can only use boolean operands

Arithmetic Operators – Binary

- + is the addition operator that adds two operands. The resulting value of the expression is the sum of the operands
- - is the subtraction operator that subtract one operand from another. The resulting value if the difference between the two operands
- * is the multiplication operator that multiplies two operands. The resulting value is the product of the two operands
- / is the division operator that divides two operands.
  - The resulting value is the real division if at least one operand is a floating type
  - The resulting value is the integer division if both operands are integral types.
- % is the remainder or modulo operator. The resulting value is remainder of the integer division
- The resulting value is a number, integral or floating-point depending on the operands

Arithmetic Operators - Unary

- + and – are both unary and binary operators
  - This means they can be used to act on one value (operand), or two
- The minus unary operator is used to negate the value of an operand, e.g.
  - -5
  - int x = 5;
  - int y = -x; // 'y' will be the negative of 'x'
- The plus unary operator can be used if you want but doesn't do much, it leaves the value the same, e.g.
  - +87.3
  - int x = 5;
  - int y = +x; // 'y' will be the same as 'x'

Examples

```
> int num1 = 5;
> int num2 = 10;
> num1 + num2
15
> num1 - num2
-5
> num1 * num2
50
> num1 / num2
0
> num1
5
> num2
10
```

The result of the "num1 + num2" expression

The "num1" and "num2" variables have not been changed.

Arithmetic operators do not change the operands in any way.

Assignment operators do.

Examples

```
> int x = 5
> int y = -x
> y
-5
> -y
5
> +y
-5
>
```

Assignment Operator (review)

- The assignment operator is a binary operator whose result is the value being assigned:
  - The assignment "x = 5" is an expression with value 5
- Assignment expressions have a side effect of changing the variable that is being assigned, the left operand
  - In the example above variable x is now 5
- Understanding x = y = 5;
  - Read this as x = (y=5);
  - y = 5 is an assignment expression that uses the assignment operator and has the value 5. y variable is also assigned the value 5 in the process
  - x is assigned the value of the expression (y = 5) which is 5
  - x and y both become 5

Shorthand Assignment Operators

- Assignment operators can be combined with arithmetic operators to perform both an arithmetic operation followed by an assignment
- Binary short-hand assignment operators
  - += is the addition assignment operator
  - -= is the subtraction assignment operator
  - *= is the multiplication assignment operator
  - /= is the division assignment operator
  - %= is the remainder assignment operator
- Unary short-hand assignment operators
  - ++ is the increment operator, increments a numeric variable by 1
  - -- is the decrement operator, decrements a numeric variable by 1

Assignment Operators

- **+=** (addition assignment operator) adds the left and the right operand and assigns the result to the left operand
  - e.g. sum += 7; *is the same as* sum = sum + 7;
- **-=** (subtraction assignment operator) subtracts the right from the left operand and assigns the result to the left operand
  - e.g. x -= 1; *is the same as* x= x - 1;
- ***=** (multiplication assignment operator) multiplies the left and the right operand and assigns the result to the left operand
  - e.g. factor *= 100.0; *is the same as* factor = factor * 100.0;
- **/=** (division assignment operator) divides the left and the right operand and assigns the result to the left operand
  - e.g. div /= 2; *is the same as* div = div / 2;
- **%=** (remainder assignment operator) calculates the remainder from dividing the left by the right operand and assigns the result to the left operand
  - e.g. r %= 11; *is the same as* r = r % 11;

<variable> += <expression>;

is the same as

<variable> = <variable> + <expression>;

<variable> -= <expression>;

is the same as

<variable> = <variable> - <expression>;

<variable> *= <expression>;

is the same as

<variable> = <variable> * <expression>;

<variable> /= <expression>;

is the same as

<variable> = <variable> / <expression>;

<variable> %= <expression>;

is the same as

<variable> = <variable> % <expression>;

Examples

```
> int num1 = 5;
> num1 = num1 + 10;
> num1
15
```

```
> int num1 = 5;
> num1 += 10;
> num1
15
```

This is the same thing as num1 = num1 + 10;

Examples

```
> int m = 10;
> m *= 5;
> m

50

> int div = 7;
> div /= 3;
> div

2

> double dd = 7;
> dd /= 3;
> dd

2.3333333333333335

> int rem = 7;
> rem %= 3;
> rem

1
```

Examples

```
> int j;
> int k;
> j = k = 129;

> j

129

> k

129

> k = 77;

> k

77

> j

129
```

Pre and Post Increment / Decrement

- ++ operator increments the value of a variable by one
  - <variable>++ is equivalent to <variable> = <variable> + 1;
  - <variable>++ is also equivalent to <variable>+=1;
- -- operator decrements the value of a variable by one
  - <variable>-- is the equivalent to <variable> = <variable> - 1;
  - <variable>-- is also equivalent to <variable>-=1;
- The value of the entire increment/ decrement expression is determined by the position of the increment / decrement operator
  - When used as a prefix, before the variable, the increment/ decrement is performed before the value of the expression is determined and therefore affects the value
  - When used as a suffix, after the variable, the increment/ decrement is performed after the value of the expression is determined

Pre and Post

```
> int num1 = 5;
> int value = ++num1;
> value
6
> num1
6
```

Perform num1 = num1 + 1 and THEN assign the result to value

Both "value" and "num1" are now 6|

```
> int num1 = 5;
> int value = num1++;
> value
5
> num1
6
```

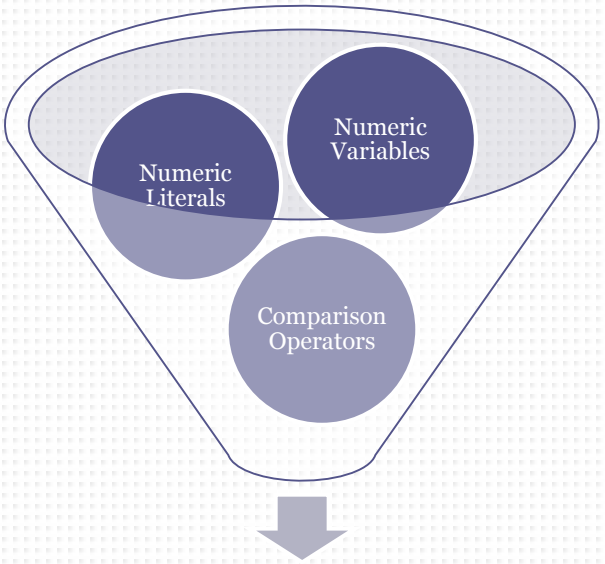
Perform value = num1 first and THEN num1 = num1 + 1

The increment was performed AFTER the expression was evaluated and therefore value = 5, the previous value for num1

num1 variable was incremented as before, it was just increment after its OLD value was remembered into the "value" variable|

Comparison Operators - Binary

- Two values of a related type (e.g. two numbers, two characters) can be compared using binary comparison operators or relational operators
- The result of using a comparison operator is always a Boolean value, true or false
  - < is the "less than" operator.
  - <= is the "less than or equal to" operator.
  - == is the "equal to" operator.
  - != is the "not equal to" operator.
  - > is the "greater than" operator.
  - >= is the "greater than or equal to" operator



Comparison Operators – Binary (cont.)

- < The resulting value is true if the left operand is less than the right operand and false otherwise
- <= The resulting value is true if the left operand is less than OR equal to the right operand and false otherwise
- == The resulting value is true if the left operand has the same value as the right operand and false otherwise.
  - DO NOT MISTAKE with the assignment operator which uses just one equal sign
- != The resulting value is true if the left operand is NOT the same value as the right operand and false otherwise.
- > The resulting value is true if the left operand is greater than the right operand and false otherwise
- >= The resulting value is true if the left operand is greater than OR equal to the right operand and false otherwise

= (assignment operator)

vs.

== (equals to operator)

Examples

```
> int a = 4;
> int b = 38;
> a == b

false

> a > b

false

> a < b

true

> char c = 'Z';
> char d = 'D';
> c != d

true

> c = 'D';
> c >= d

true
```

Logical Operators

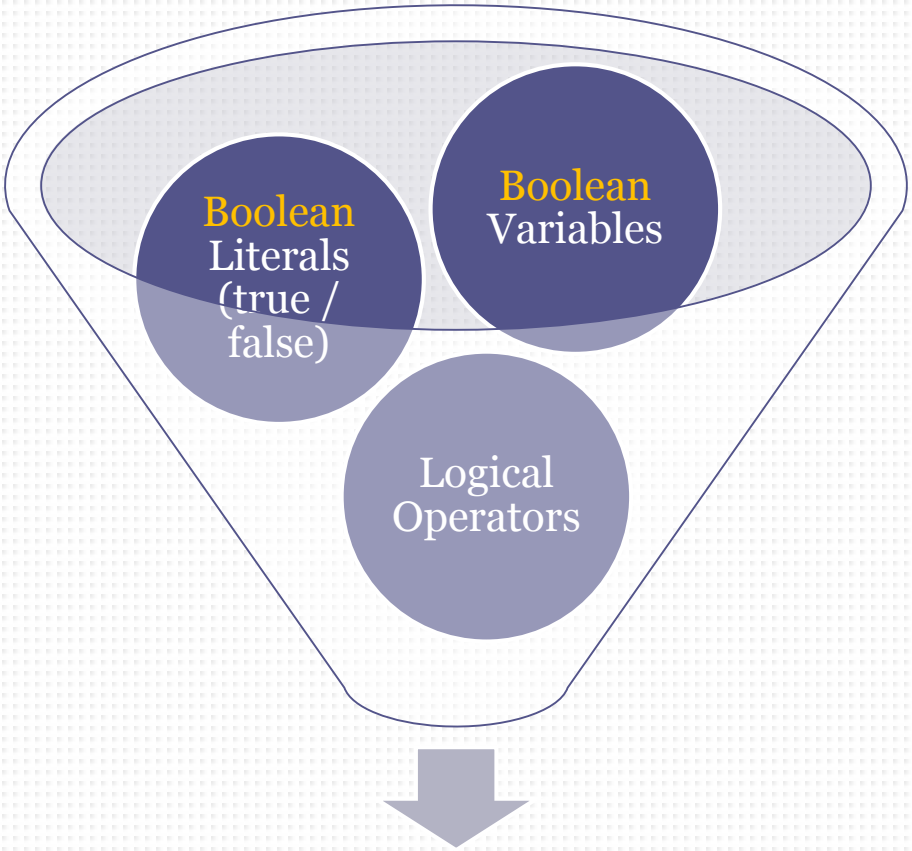
- Boolean values can be combined into Boolean expressions using logical operators or Boolean operators
  - Boolean expressions can be combined into longer and longer expressions

- The result of using a logical operator is always a Boolean value: true or false

Logical Operators

- Binary logical operators
  - && is the logical "and" operator which returns true if both operands are true and false otherwise
  - || is the logical "or" operator which returns true if at least one of the operands is true
  - ^ is the logical "exclusive or" operator which returns true if the one of the operands is true and the other one is false.
- Unary logical operator
  - ! is the logical "not" operator which returns the opposite of the operand. Returns true if the operand is false and false if the operand is true

Logical Expressions



true / false (Boolean)

Exercise 1

- Ask the user to enter an integer. If the value is not in the range (0-100) print “The value is less than 0 or greater than 100”

- Ask the user to enter two integers. If both numbers are greater than or equal to 100 multiply the two numbers together and print the result and “Both numbers are greater than 100”