Chapter 2: Java Fundamentals

Operators

Content

- Group of Operators
- Arithmetic Operators
- Assignment Operator
- Order of Precedence
- Increment/Decrement Operators
- Relational Operators
- Logical Operators

Operators

- Operators are special symbols used for:
  - mathematical functions
  - assignment statements
  - logical comparisons
- Examples of operators:
  - 3 + 5 // uses + operator
  - 14 + 5 – 4 * (5 – 3) // uses +, -, * operators
- Expressions: can be combinations of variables and operators that result in a value

Groups of Operators

- There are 5 different groups of operators:
  - Arithmetic Operators
  - Assignment Operator
  - Increment / Decrement Operators
  - Relational Operators
  - Logical Operators

Java Arithmetic Operators

Addition	+
Subtraction	–
Multiplication	*
Division	/
Remainder (modulus)	%

Arithmetic Operators

- The following table summarizes the arithmetic operators available in Java.

Operation	Java Operator	Example	Value (x = 10, y = 7, z = 2.5)
Addition	+	x + y	17
Subtraction	-	x - y	3
Multiplication	*	x * y	70
Division	/	x / y	1
		x / z	4.0
Modulo division (remainder)	%	x % y	3

This is an integer division where the fractional part is truncated.

Example

Example of division issues:

10 / 3 gives 3

10.0 / 3 gives 3.33333

As we can see,

- if we divide two integers we get an integer result.
- if one or both operands is a floating-point value we get a floating-point result.

Modulus

- Generates the remainder when you divide two integer values.

5%3 gives 2 5%4 gives 1

5%5 gives 0 5%10 gives 5

- Modulus operator is most commonly used with integer operands. If we attempt to use the modulus operator on floating-point values we will garbage!

Order of Precedence

( ) evaluated first, inside-out

*, /, or % evaluated second, left-to-right

+, – evaluated last, left-to-right

Basic Assignment Operator

- We assign a value to a variable using the basic *assignment operator (=)*.
- Assignment operator stores a value in memory.
- The syntax is

```
leftSide = rightSide ;
```

Allways it is a *variable identifier.*

It is either a *literal* | a *variable identifier* | an *expression.*

Examples:

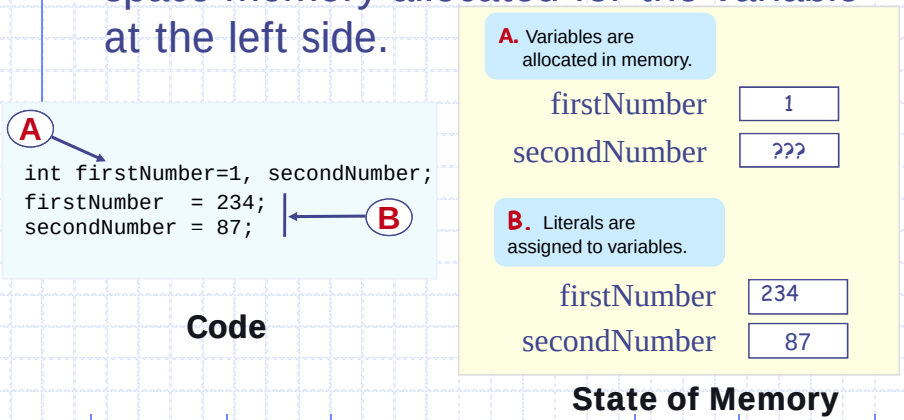
```
i = 1;
start = i;
sum = firstNumber + secondNumber;
avg = (one + two + three) / 3;
```

The Right Side of the Assignment Operator

- The Java assignment operator assigns the value on the right side of the operator to the variable appearing on the left side of the operator.
- The right side may be either:
  - Literal: ex. `i = 1;`
  - Variable identifier: ex. `start = i;`
  - Expression: ex. `sum = first + second;`

Assigning Literals

- In this case, the literal is stored in the space memory allocated for the variable at the left side.



Assigning Variables

- In this case, the value of the variable at the right side is stored in the space memory allocated for the variable at the left side.

A

```
int firstNumber=1, i;
firstNumber  = 234;
i = firstNumber;
```

B

Code

A. Variables are allocated in memory.

firstNumber	1
i	???

B. values are assigned to variables.

firstNumber	234
i	234

State of Memory

Assigning Expressions

- In this case, the result of the evaluation of the expression is stored in the space memory allocated for variable at the left side.

A

```
int first, second, sum;
first  = 234;
second = 87;
Sum = first + second
```

B

Code

A. Variables are allocated in memory.

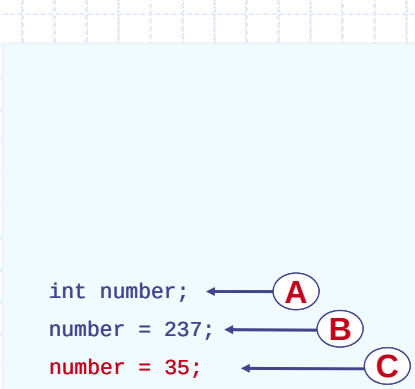
first	1	second	???
sum	???		

B. Values are assigned to variables.

first	234	second	87
sum	321		

State of Memory

Updating Data



A. The variable is allocated in memory.

number ???

B. The value 237 is assigned to number.

number 237

C. The value 35 overwrites the previous value 237.

number 35

Code **State of Memory**

Example: Sum of two integer

```
public class Sum {

 // main method
    public static void main( String args[] ){
         int a, b, sum;
         a = 20;
         b = 10;
         sum = a + b;
         System.out.println(a + " + " + b + "  = " + sum);

    } // end main

} // end class Sum
```

Arithmetic/Assignment Operators

Java allows combining arithmetic and assignment operators into a single operator:

Addition/assignment	**+ =**
Subtraction/assignment	**– =**
Multiplication/assignment	*** =**
Division/assignment	**/ =**
Remainder/assignment	**% =**

Arithmetic/Assignment Operators

- The syntax is

```
leftSide  Op= rightSide ;
```

It is either a *literal* | a *variable identifier* | an *expression*.

Allways it is a *variable identifier*.

It is an *arithmetic operator*.

- This is equivalent to:

```
leftSide = leftSide Op rightSide ;
```

- x%=5; ⇔ x = x % 5;
- x*=y+w*z; ⇔ x = x*(y+w*z);

Increment/Decrement Operators

Only use ++ or – – when a variable is being incremented/decremented as a statement by itself.

`x++;` is equivalent to `x = x+1;`

`x--;` is equivalent to `x = x-1;`

Relational Operators

- Relational operators compare two values
- They Produce a *boolean* value (**true** or **false)** depending on the relationship

Operation	Is true when
a >b	**a** is greater than **b**
a >=b	**a** is greater than or equal to **b**
a ==b	a is equal to **b**
a !=b	**a** is not equal to **b**
a <=b	**a** is less than or equal to **b**
a <b	**a** is less than **b**

Example

- int x = 3;
- int y = 5;
- boolean result;
  result = (x > y);
- now result is assigned the value false because 3 is not greater than 5

Logical Operators

Symbol	Name
&&	AND
||	OR
!	NOT

&&	T	F
T	T	F
F	F	F

||	T	F
T	T	T
F	T	F

Example

```
boolean x = true;
boolean y = false;
boolean result;

result = (x && y);
```

result is assigned the value false

```
result = ((x || y) && x);
```

(x || y) evaluates to true
(true && x) evaluates to true
result is then assigned the value true

Operators Precedence

Parentheses	(), inside-out
Increment/decrement	++, --, from left to right
Multiplicative	*, /, %, from left to right
Additive	+, -, from left to right
Relational	<, >, <=, >=, from left to right
Equality	==, !=, from left to right
Logical AND	&&
Logical OR	||
Assignment	=, +=, -=, *=, /=, %=