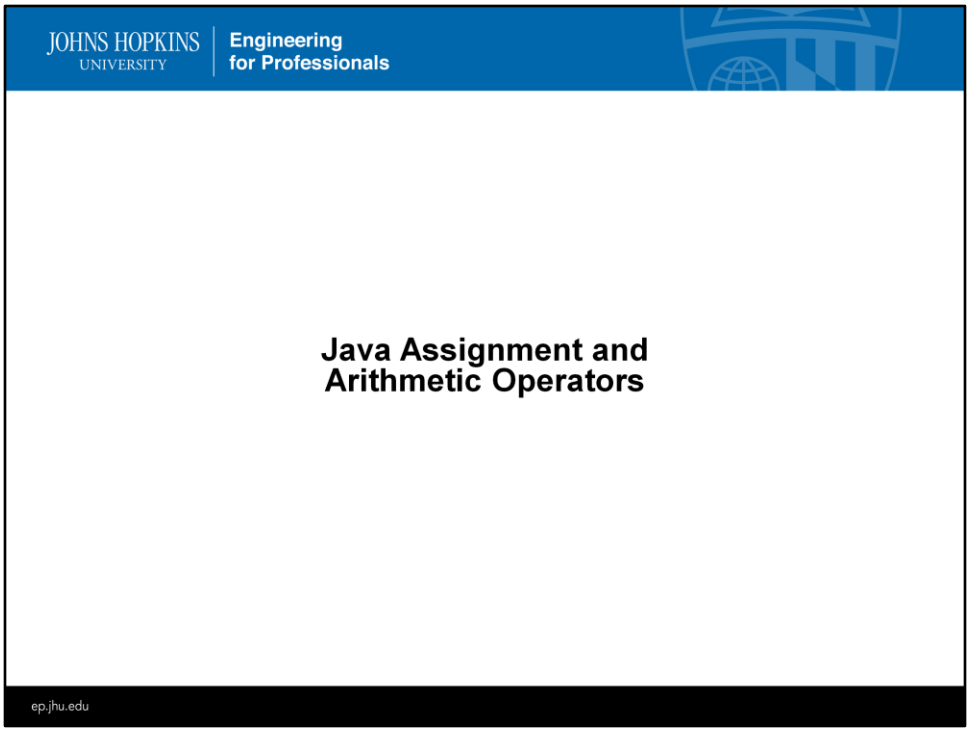


Welcome viewers!

This presentation discusses basic Java assignment and arithmetic operators. We will start with discussing the simple but essential assignment operator.

So lets get to it!

Topics

Assignment Operator

Arithmetic Operators

As mentioned earlier, this presentation has two major topics, assignment operator and arithmetic operators.

It actuality, some arithmetic operators include assignment operator functionality which can save some typing but may also require more thinking, unless you are an assembly language programmer.

Lets look at the assignment-only operator to get started.

Assignment (=)

Moves calculated results into a variable

- Changes variable on left, right elements unchanged
  - However right elements could be changed by *side effects*

Example

- c = a + b
  - C changed with results of a + b calculation
  - A and B unchanged after operation
  - What about: a = a + b ?

The assignment operator moves some calculated result into a variable. Notice the syntax has the variable receiving the value to be on the left and the "calculation" to be on the right. The calculation on the right can be a complex formula or as simple as a constant number or letter. Left of the assignment operator must be a variable for example "a" or "lineNumber". It can not be a constant value which makes sense since the purpose of a constant is to maintain a "constant" value. However there are many types of variables which for now we will keep the discussion to simple variables for example an integer, character, etc. More complex assignments will be discussed in later presentations.

In the example here, the results of the calculation of a + b is copied or *assigned* to the variable c. The previous contents of variable c is overwritten with the results of the calculation. However, the values of a and b are unchanged by this operation.

So what if we have a = a + b? What happens to a? What happens to b? What happens to c?

Well a is on the left side of the assignment operator so its value is reset to the results of the calculation. But a is also on the right side of the calculation. What happens is the calculation of a + b happens *before* the assignment. Once the calculation is complete, the assignment takes place.

Assigning the results of a calculation to one of the variables involved in the calculation turns out to be a fairly common operation. So much so Java supplies operators to support this type of operation.

Arithmetic Operators

Operator	Action
+	Addition (also unary plus)
-	Subtraction (also unary minus)
*	Multiplication
/	Division
%	Modulus
++	Increment
+=	Addition with assignment
-=	Subtraction with assignment
*=	Multiplication with assignment
/=	Division with assignment
%=	Modulus with assignment
--	Decrement

Schildt, Herbert Java The Complete Reference, 9th Edition, pp. 61

Now on to Java arithmetic operators. As you can see from the chart here, Java supplies several arithmetic operators. Some I am sure you have seen before and others which may be new to you.

Lets take a closer look.

Addition, Subtraction, Multiplication, and Division

Normal arithmetic operations

Two operands

- a + b, c / d, etc.
- Neither operand modified via operation

Divide by zero illegal

- Produces a run-time error (exception)

Integer division truncates results

- 10 / 4 = 2
- 10.0 / 4.0 = 2.5

Java supplies the normal add, subtract, multiply, and division operations represented by well-known symbols except for possibly division which uses the right-slash. Addition, subtraction, and multiplication work as you would expect, taking the operands on either side symbol and perform the requested operation, supplying a result.

There are two properties of Java division I wish to point out here. First division by zero is "illegal". An attempt to divide by zero results in a run-time error (exception).

The second Java division property is for *integer* values (int, long, etc) any remainder is truncated. This is not true for floating-point type variables as shown in the example here where the results will be as precise as the data type allows.

Modulus (%)

Provides the remainder of a division operation

- Ex. 10 % 6 – results is 4 (10/6 = 1 (remainder 4)
- x % 0 – results in a divide-by-zero error (exception)

Useful in determining if variable evenly divided by a value

- Ex. if (x % 2 == 0) // Is x even?
- Ex. while (lineNumber % 20 > 0) // 20 lines per page

The modulus operator is in interest one. It performs a division but instead of supplying the result of the division it supplies the remainder of the division. Since it is performing a division, a right operand (which is the divisor) of zero will result in a divide-by-zero error.

The modulus operator can be quite useful. A couple of examples are shown here.

Note the examples show some decision syntax not covered yet. We will get to decision constructs very soon so at this point don't worry about it.

One very common use of the modulus operator is to determine if an integer-type variable is an even number. The first example demonstrates this. Assuming x is an integer. If the result is zero, x is an even number since the value of x is evenly divisible by 2.

The second example demonstrates how to use the modulus operator to determine when to do a page break on a report. Here if the current line number is evenly divisible by 20, we take some action to force a page break. If not, we will just print a line of information.

I am sure you will find many other uses for the modulus operator in your Java programming endeavors.

Increment (++) and Decrement (--)

Compact way to increase or decrease variable value by one.

- Increment (++) increase by one
- Decrement (--) decrease by one

Pre and post operations

- ++/-- Placement determines when the action takes place
  - a = 1; b = a++; // Value of a and b after this?
  - a = 1; b = ++a; // What about this?
  - a = 1; a++; // And this? What if I did ++a instead?

The increment and decrement operators are a compact way to represent increasing or decreasing a value of a variable by one respectively. This is a fairly straight-ahead concept.

The interesting part of these operators is you can specify if the action takes place before or after the next action. What the next action will be is dependent on the context of the statement involved but here are a couple of examples (note proper code formatting is relaxed to provide a compact representation on the slide).

In the first example, the increment operator is doing a post operation. This means the increment of the variable a is AFTER the assignment to b. What takes place is the value of a is set to one, then the current value of a, which is one, is copied to b' Then the value of a is increased by one making a = 2. At the end, b will be equal to 1 and a will equal 2.

In the second example, the increment operator is doing a pre operation. This means the increment of the variable a is BEFORE the assignment operation. What takes place is the value of a is set to one, then the current value of a, which is one, is increased by one because of the increment operation making a = 2. Then the value of a is copied into the variable b make b equal two as well. At the end, both a and b are equal to 2.

The increment/decrement operators can be stand-alone statements which the third example illustrates. Here a is set to one then the next statement increments the value of a. An interesting question is; does it matter if I do a pre or post operation in this case?

Operation With Assignment

Shortcut to update variable with operation

X += 10; is the same as X = X + 10;

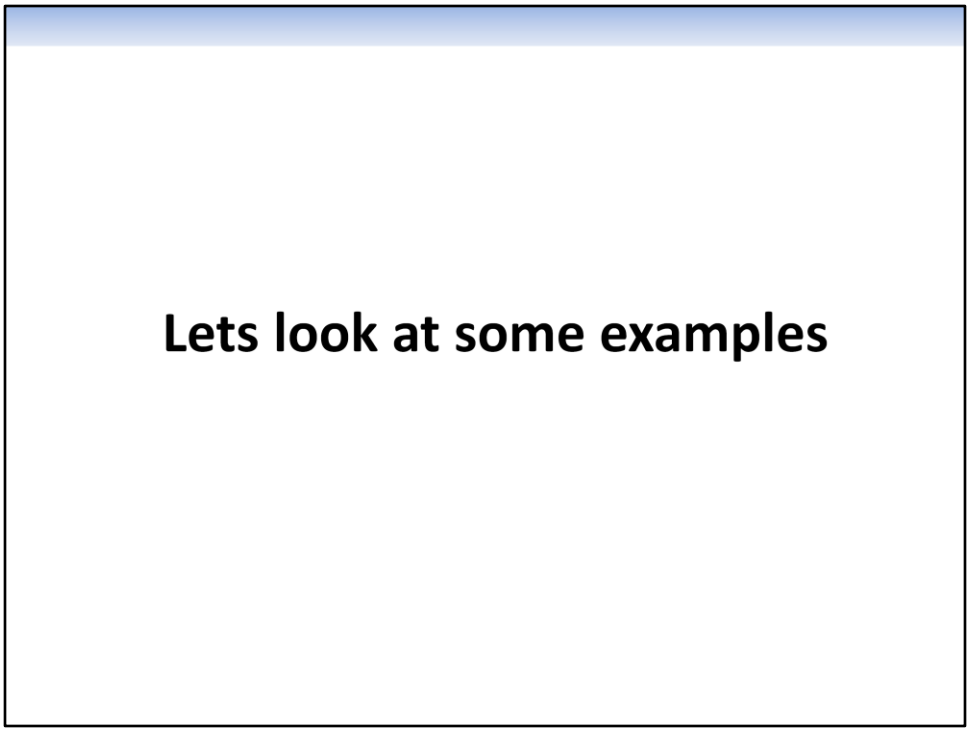
Also available for subtraction, multiplication, division, and bit operations (-=, *=, /=, &=, etc.)

Examples:

```
b += a + 3; // b = b + a + 3;
b /= a – 1;  // Any danger here?
```

Java supplies short cuts if you are going to overwrite a variable which you are using in a calculation. The symbols used for these operations combine the arithmetic and bit operators with the assignment operator. Some of the available combinations are shown here. Bit operators are discussed in a different presentation so don't worry about them now.

All these operators do is imply the variable on the left to be also placed at the beginning of the calculation on the right with the specified operation after it as shown in these examples. Think about the last example. Do you see any danger with this statement?



Let’s look at some examples of using Java arithmetic operators.

Replace with BasicMath Project

Main Points

Add, Subtract, Multiply, Divide work as expected

Modulus calculates the remainder of a division operation

Increment/decrement operations have pre and post operations

Operation with assignment operators provide variable update short-cut.

Here are the main points of this presentation.

Add, Subtract, Multiply, and Divide work as expected. The only one to watch out for is integer division. Remember the result is an integer which can not represent fractional results. Any remainder is truncated.

Modulus calculates the remainder of a division operation. If the operands are integers, an integer is returned. If the operands are floating-point types, a floating point type will be returned.

Increment and decrement operations provide a short-cut way to increase or decrease a numeric variable's value by one. There are pre and post versions of each. Care should be taken when using increment and decrement operations in formulas and as parameters to methods.

Java also supplies operations which combine arithmetic operations with the assignment operator. These provide short-cuts when using the variable on the left of the assignment operator as part of the calculation on the right side of the assignment operator.

That's It!

And that's it!