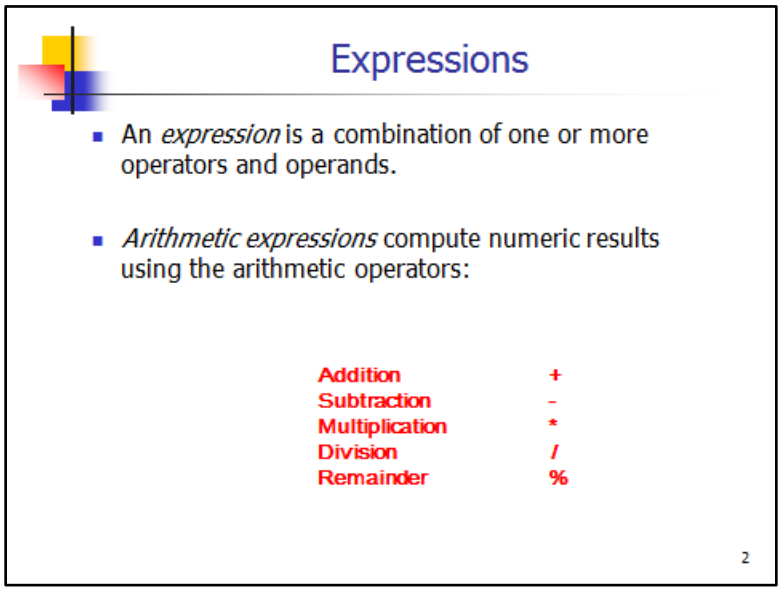
In this section of today's class, we will look at how we can combine multiple operators and operands to create *expressions*.

Most calculations that you will do in Java will be done with expressions.



To do numerical calcuations in a Java program you will write *arithmetic expressions*.

Arithmetic expressions use the five arithmetic operators:

plus sign for addition,
minus sign (or dash) for subtraction,
asterisk for multiplication,
forward slash for division,
percent sign for remainder.

The remainder operation is sometimes referred to as modulus.

a % b, where a and b are integers, returns the remainder from a divided by b.

Division and Remainder

- If both operands to the division operator (/) are integers, the result is an integer
  - The fractional part is discarded.
- Integer division results in truncation, not rounding.

`14 / 3`	**equals**	4
`8 / 12`	**equals**	0

- The remainder operator (%) returns the remainder from dividing the first operand by the second.

`14 % 3`	**equals**	2
`8 % 12`	**equals**	8

(No narration)

Operator Precedence

- Operators can be combined into complex expressions

```
result = total + count / max - offset;
```

- The order in which the operations are done is important.
- Multiplication, division, and remainder are evaluated prior to addition, subtraction, and string concatenation.
- Arithmetic operators with the same precedence are evaluated from left to right, but parentheses can be used to force the evaluation order.

4

Operators can be combined into complex expressions. The order in which operators are applied is important. The order is determined by the operators' *precedence*.

Higher precedence operations within an expression are evaluated before lower precedence operations. The lower precedence operations use the results of operations performed earlier.

Multiplication, division, and remainder are high precedence operations. These three operations have equal precedence.

Addition, subtraction, and string concatenation have equal lower precedence.

Operations with equal precedence are performed left to right within the expression.

If we want the operations to be applied in a different order, we can use parentheses to control the order of execution. Operations inside parentheses are always done before operations outside the parentheses. This is consistent with normal arithmetic notation as you probably learned in high school or junior high. Java, like most programming languages, simply adopted the rules that people already know.

Operator Precedence

- What is the order of evaluation in the following expressions?

```
a / (b + c) - d % e
  2    1     4   3

a / (b * (c + (d - e)))
  4    3    2    1
```

In PowerPoint, click on the slide to show sucessive results.

(No narration)

Exercises

Given the declarations below, find the result of each expression (in a Java program).

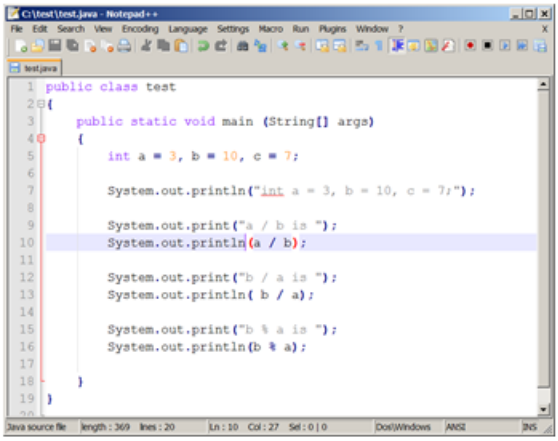
```
int a = 3, b = 10, c = 7;
```

- `a / b`
- `b / a`
- `b % a`

Answers on next slides.

(No narration)

Program to answer exercise



(No narration)

Exercise Answers

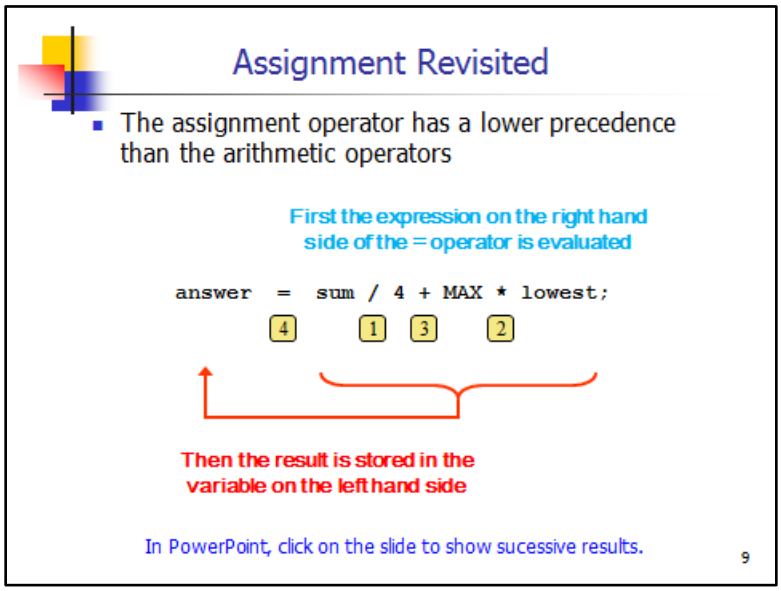
```
C:\Windows\system32\cmd.exe

C:\test>javac test.java

C:\test>java test
int a = 3, b = 10, c = 7;
a / b is 0
b / a is 3
b % a is 1

C:\test>
```


(No narration)



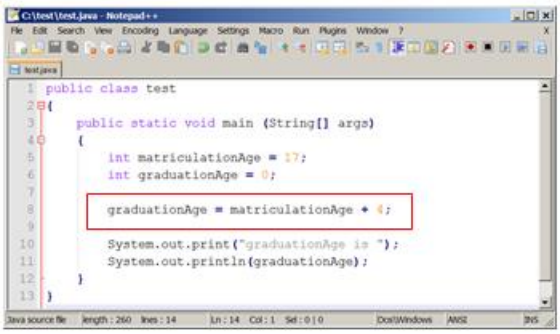
(No narration)

Exercise

Given two integer variables *matriculationAge* and *graduationAge*, write a statement that gives *graduationAge* a value that is 4 more than the value of *matriculationAge* .

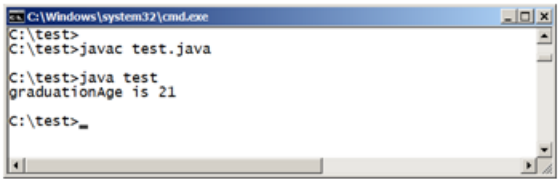
(No narration)

Program to answer exercise



(No narration)

Program to answer exercise running



(No narration)

Assignment Operators

- Often we perform an operation on a variable, and then store the result back into that variable. Java provides *assignment operators* to simplify that process
- For example, the statement

```
num += count;
```

is equivalent to

```
num = num + count;
```

13

Assignment operators combine assignment and an arithmetic operation into a single operator. The arithmetic operation is performed first, as if the equal sign were not present. Then the result is assigned to the variable on the left side of the operator.

These operators are part of Java's heritage from the C programming language. When the C language was created in the late 60s, keyboards were clunky and slow, and most programmers were not particularly good typists. So the designers of C went to great lengths to minimize keystrokes. These operators are good examples of that.

Minimizing keystrokes is not nearly so important now as it was when C was created, but we still have these language features. It is never necessary to use them. You can always write num = num + count rather than num += count. The choice is up to you.

Assignment Operators

- There are many assignment operators in Java, including the following:

Operator	Example	Equivalent To
`+=`	`x += y`	`x = x + y`
`-=`	`x -= y`	`x = x - y`
`*=`	`x *= y`	`x = x * y`
`/=`	`x /= y`	`x = x / y`
`%=`	`x %= y`	`x = x % y`

(No narration)

Assignment Operators

- The right hand side of an assignment operator can be a complex expression
- For example,

```
result /= (total-MIN) % num;
```

is equivalent to

```
result = result / ((total-MIN) % num);
```

15

The right hand side of an assignment operator can be a complex expression, just as it can be in a plain assignment statement. Anything that yields an appropriate value can be used on the right side.

The left hand side, however, must be a place to store the result, normally a variable name. It wouldn't make sense to put an expression on the left hand side, and if we did we would get a compile time error.