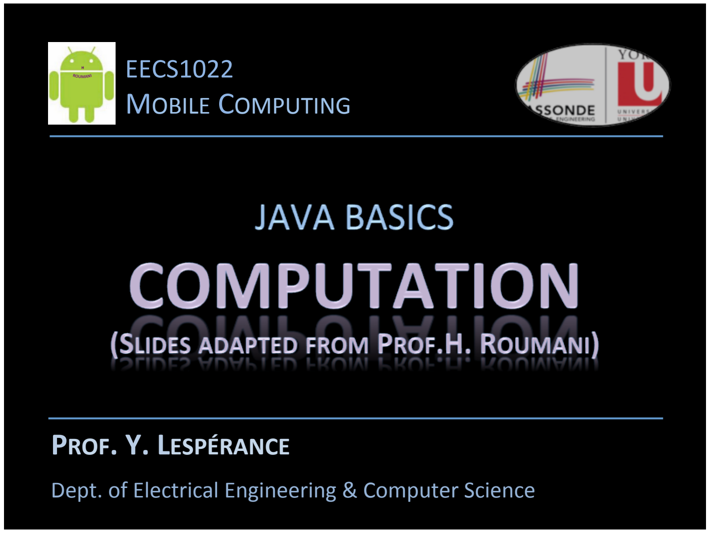


EECS1022
MOBILE COMPUTING

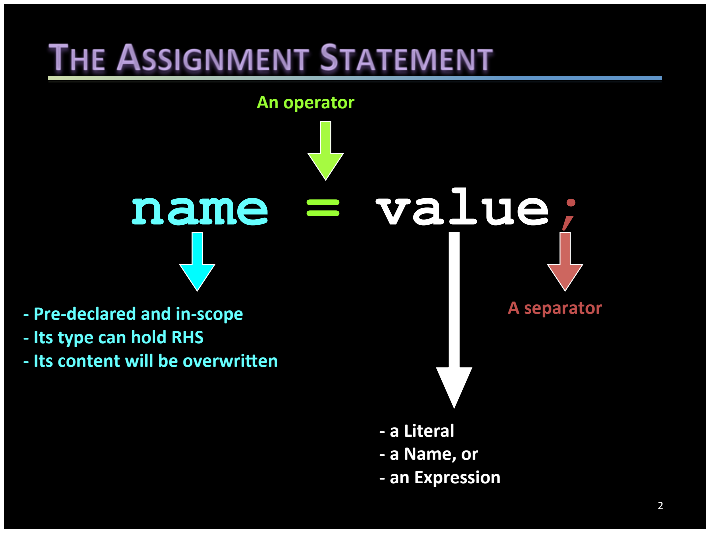
JAVA BASICS

COMPUTATION

(SLIDES ADAPTED FROM PROF.H. ROUMANI)

PROF. Y. LESPÉRANCE

Dept. of Electrical Engineering & Computer Science



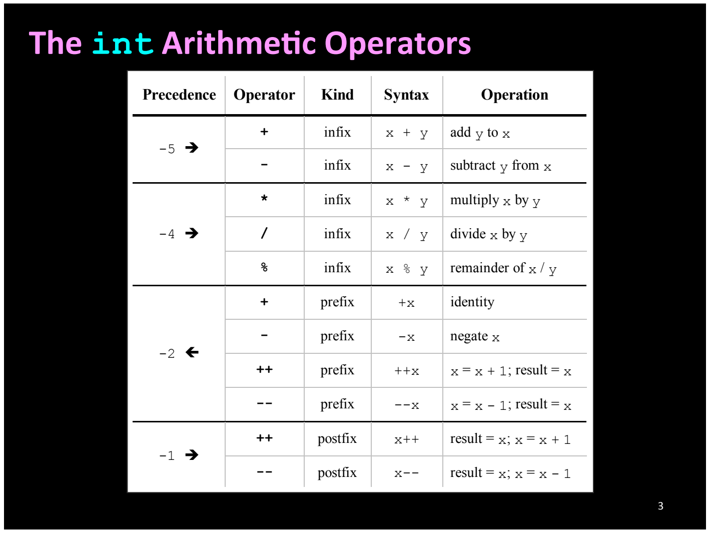
THE ASSIGNMENT STATEMENT

```
name = value;
```

- Pre-declared and in-scope
- Its type can hold RHS
- Its content will be overwritten

- a Literal
- a Name, or
- an Expression

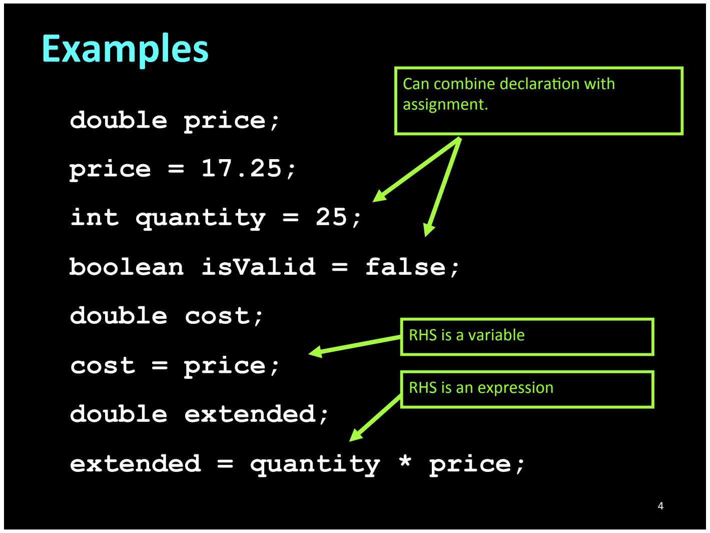
2



The int Arithmetic Operators

Precedence	Operator	Kind	Syntax	Operation
-5 →	+	infix	x + y	add y to x
	-	infix	x - y	subtract y from x
-4 →	*	infix	x * y	multiply x by y
	/	infix	x / y	divide x by y
	%	infix	x % y	remainder of x / y
-2 ←	+	prefix	+x	identity
	-	prefix	-x	negate x
	++	prefix	++x	x = x + 1; result = x
	--	prefix	--x	x = x - 1; result = x
-1 →	++	postfix	x++	result = x; x = x + 1
	--	postfix	x--	result = x; x = x - 1

3

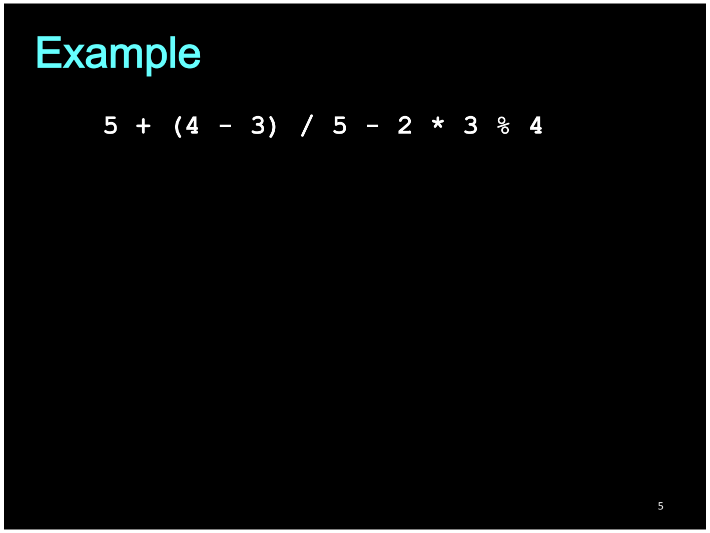


Examples

```
double price;
price = 17.25;
int quantity = 25;
boolean isValid = false;
double cost;
cost = price;
double extended;
extended = quantity * price;
```

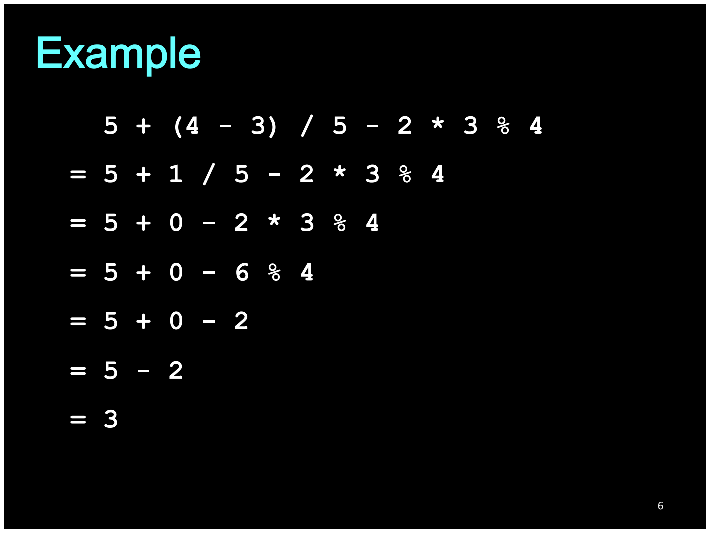
Can combine declaration with assignment.
RHS is a variable
RHS is an expression

4



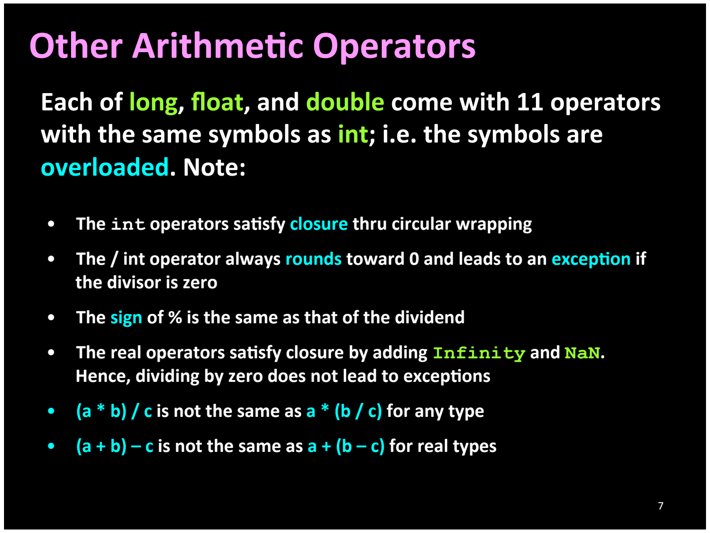
Example

```
5 + (4 - 3) / 5 - 2 * 3 % 4
```



Example

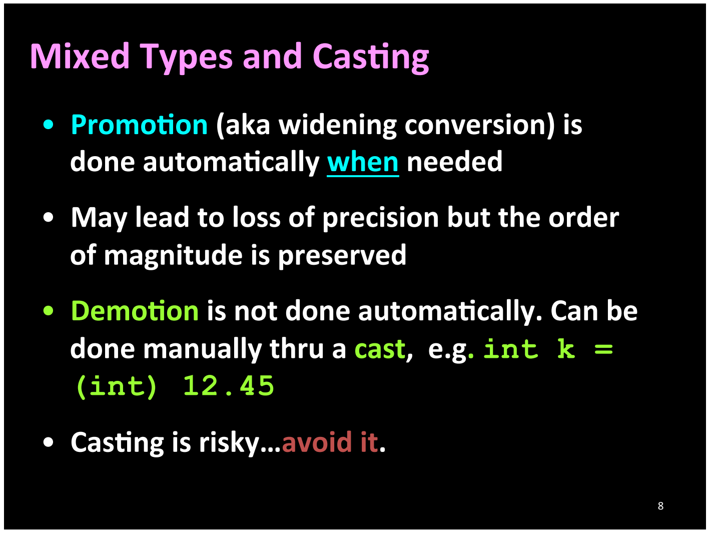
```
  5 + (4 - 3) / 5 - 2 * 3 % 4
= 5 + 1 / 5 - 2 * 3 % 4
= 5 + 0 - 2 * 3 % 4
= 5 + 0 - 6 % 4
= 5 + 0 - 2
= 5 - 2
= 3
```



Other Arithmetic Operators

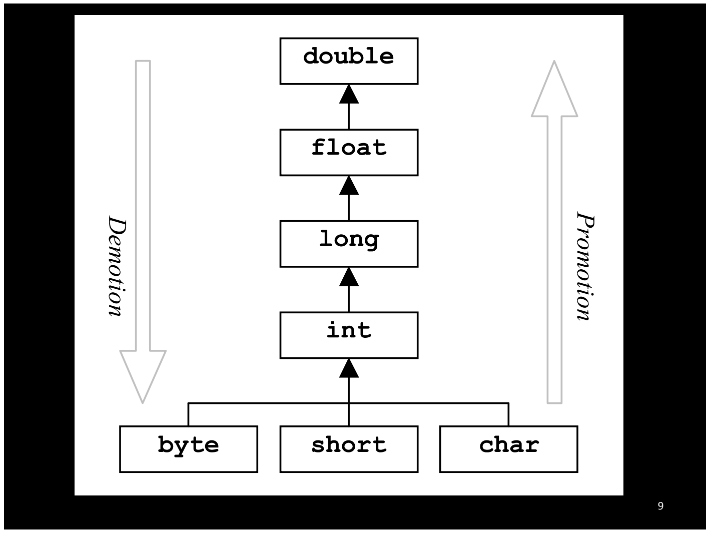
Each of long, float, and double come with 11 operators with the same symbols as int; i.e. the symbols are overloaded. Note:

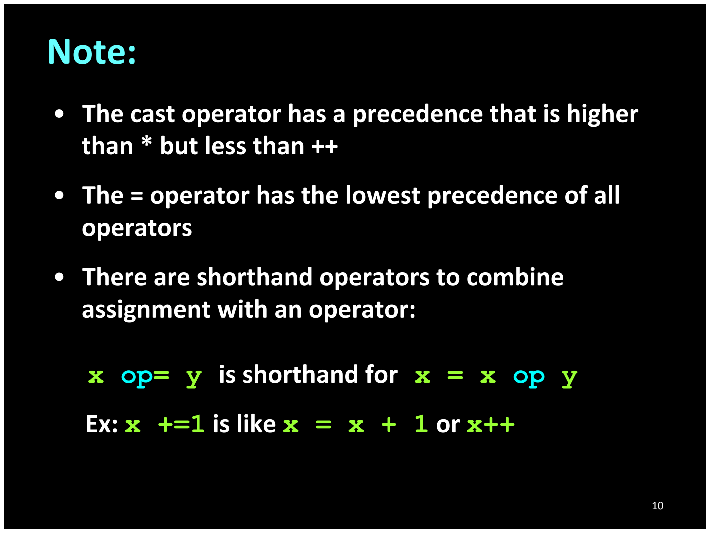
- The int operators satisfy closure thru circular wrapping
- The / int operator always rounds toward 0 and leads to an exception if the divisor is zero
- The sign of % is the same as that of the dividend
- The real operators satisfy closure by adding Infinity and NaN. Hence, dividing by zero does not lead to exceptions
- (a * b) / c is not the same as a * (b / c) for any type
- (a + b) – c is not the same as a + (b – c) for real types



Mixed Types and Casting

- Promotion (aka widening conversion) is done automatically when needed
- May lead to loss of precision but the order of magnitude is preserved
- Demotion is not done automatically. Can be done manually thru a cast, e.g. `int k = (int) 12.45`
- Casting is risky…avoid it.



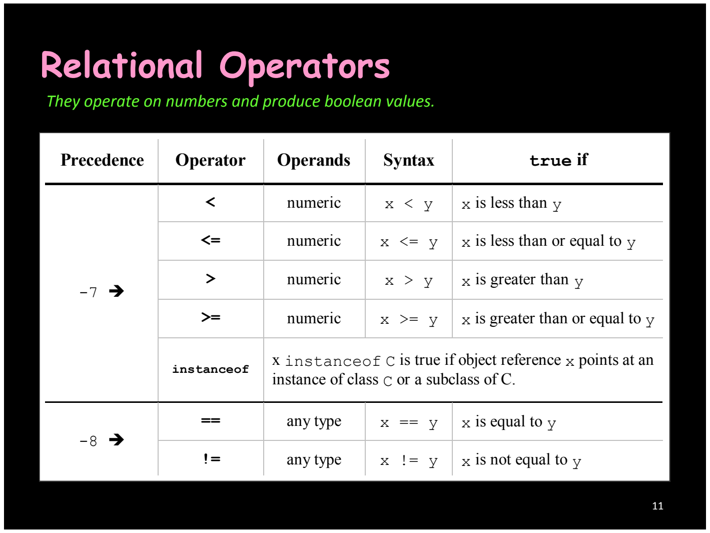


Note:

- The cast operator has a precedence that is higher than * but less than ++
- The = operator has the lowest precedence of all operators
- There are shorthand operators to combine assignment with an operator:

`x op= y` is shorthand for `x = x op y`

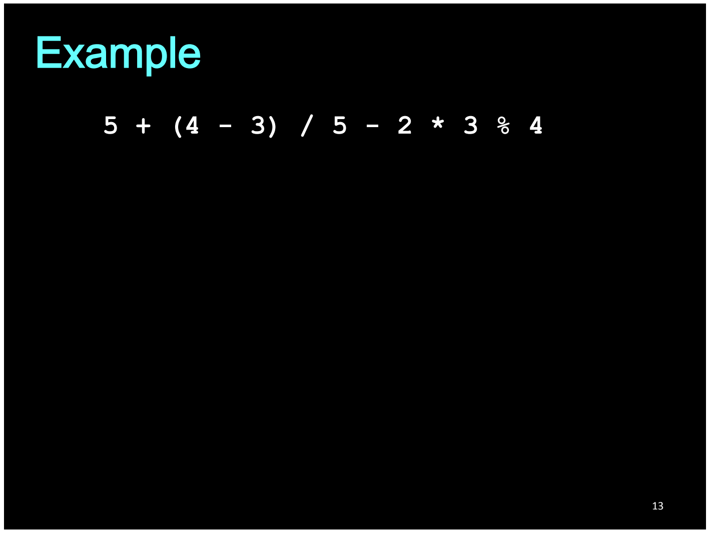
Ex: `x +=1` is like `x = x + 1` or `x++`



Relational Operators

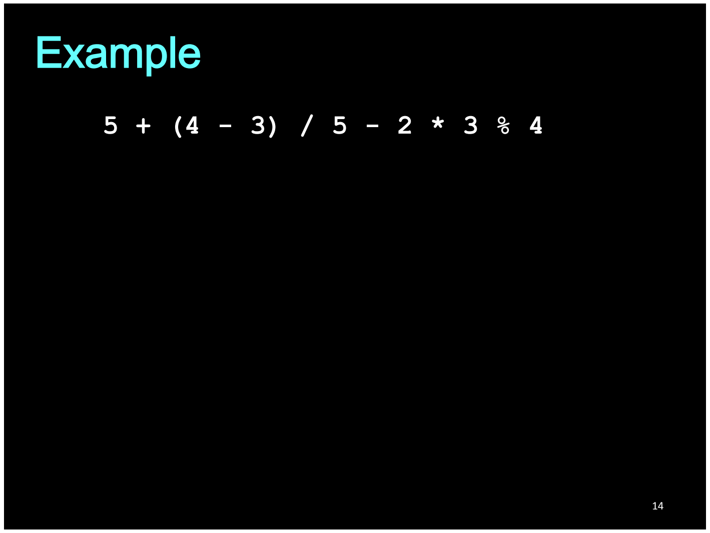
They operate on numbers and produce boolean values.

Precedence	Operator	Operands	Syntax	`true` if
-7 ➔	<	numeric	`x < y`	x is less than y
	<=	numeric	`x <= y`	x is less than or equal to y
	>	numeric	`x > y`	x is greater than y
	>=	numeric	`x >= y`	x is greater than or equal to y
	instanceof	x `instanceof C` is true if object reference x points at an instance of class C or a subclass of C.		
-8 ➔	==	any type	`x == y`	x is equal to y
	!=	any type	`x != y`	x is not equal to y



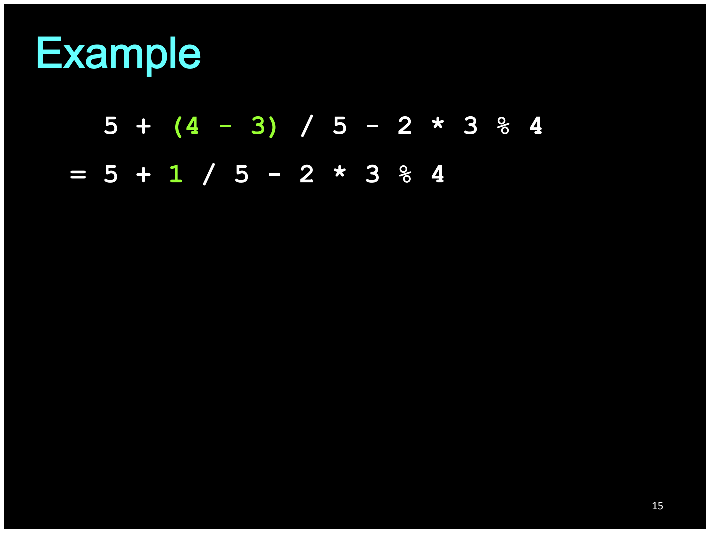
Example

```
  5 + (4 - 3) / 5 - 2 * 3 % 4
```



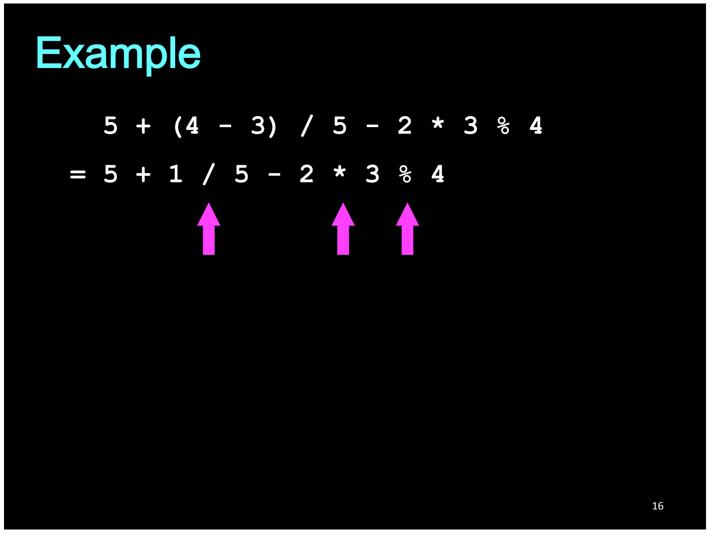
Example

```
  5 + (4 - 3) / 5 - 2 * 3 % 4
```



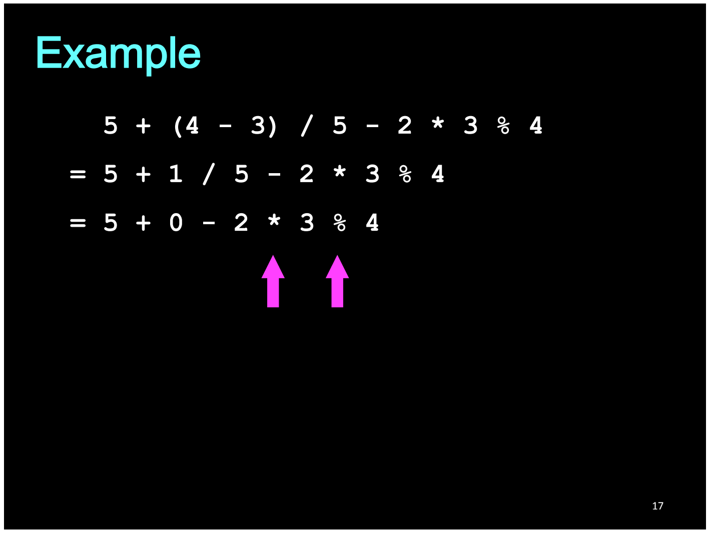
Example

```
  5 + (4 - 3) / 5 - 2 * 3 % 4
= 5 + 1 / 5 - 2 * 3 % 4
```



Example

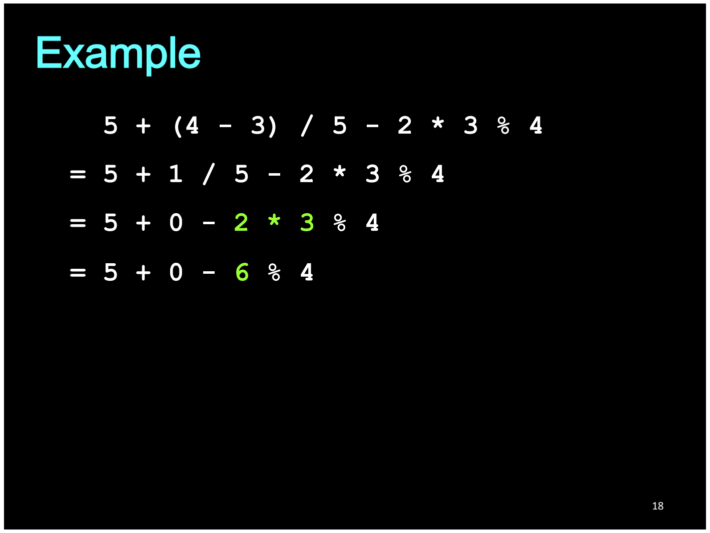
```
  5 + (4 - 3) / 5 - 2 * 3 % 4
= 5 + 1 / 5 - 2 * 3 % 4
```



Example

```
  5 + (4 - 3) / 5 - 2 * 3 % 4
= 5 + 1 / 5 - 2 * 3 % 4
= 5 + 0 - 2 * 3 % 4
```

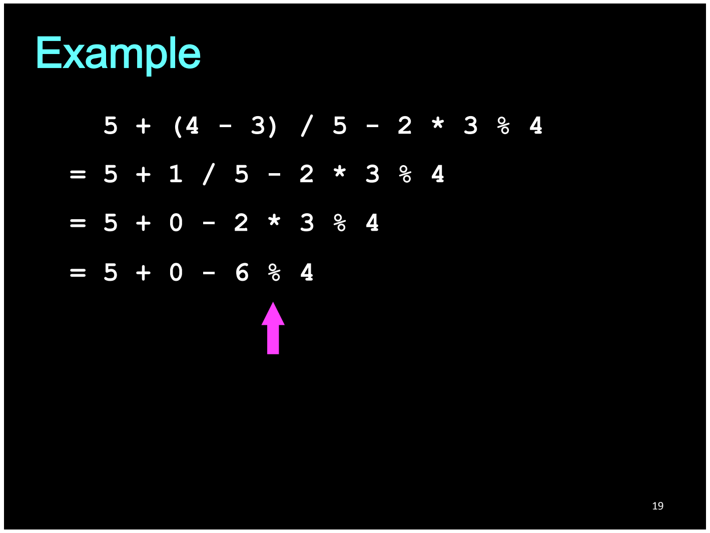
17



Example

```
  5 + (4 - 3) / 5 - 2 * 3 % 4
= 5 + 1 / 5 - 2 * 3 % 4
= 5 + 0 - 2 * 3 % 4
= 5 + 0 - 6 % 4
```

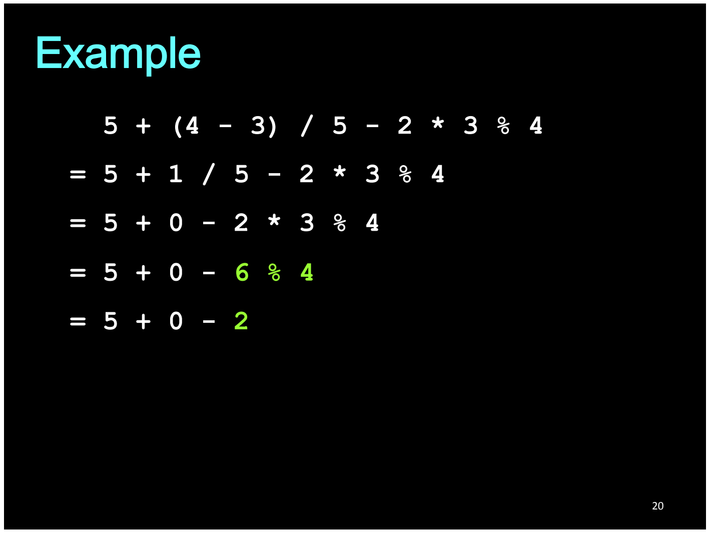
18



Example

```
  5 + (4 - 3) / 5 - 2 * 3 % 4
= 5 + 1 / 5 - 2 * 3 % 4
= 5 + 0 - 2 * 3 % 4
= 5 + 0 - 6 % 4
```

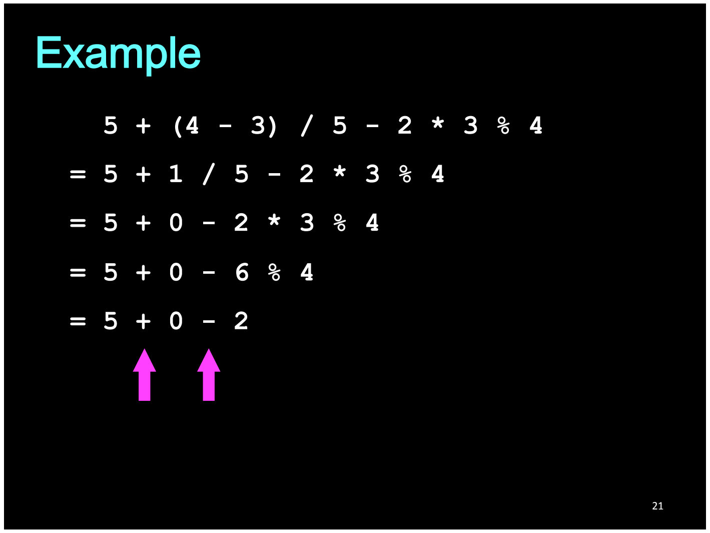
19



Example

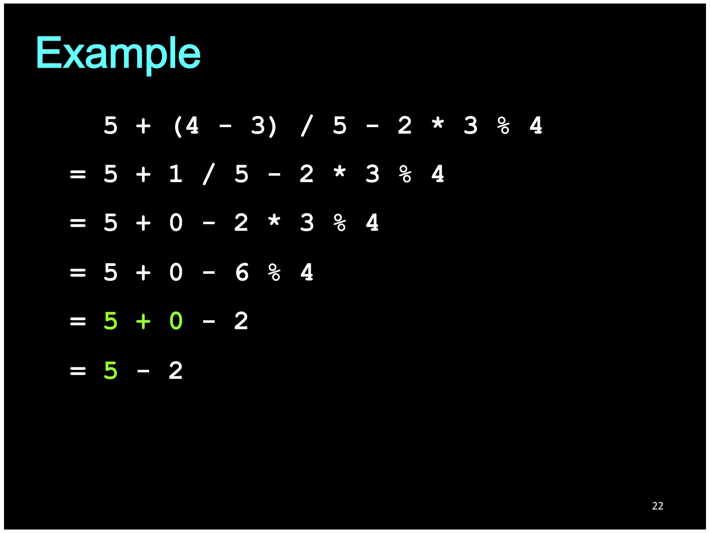
```
  5 + (4 - 3) / 5 - 2 * 3 % 4
= 5 + 1 / 5 - 2 * 3 % 4
= 5 + 0 - 2 * 3 % 4
= 5 + 0 - 6 % 4
= 5 + 0 - 2
```

20



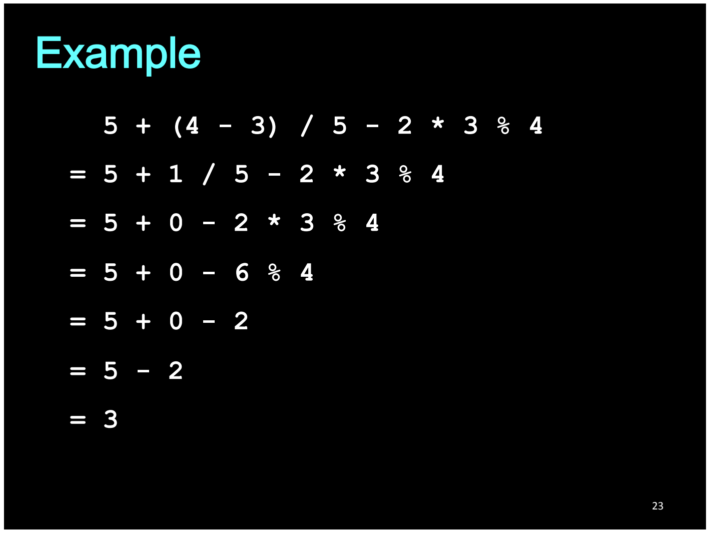
Example

```
  5 + (4 - 3) / 5 - 2 * 3 % 4
= 5 + 1 / 5 - 2 * 3 % 4
= 5 + 0 - 2 * 3 % 4
= 5 + 0 - 6 % 4
= 5 + 0 - 2
```



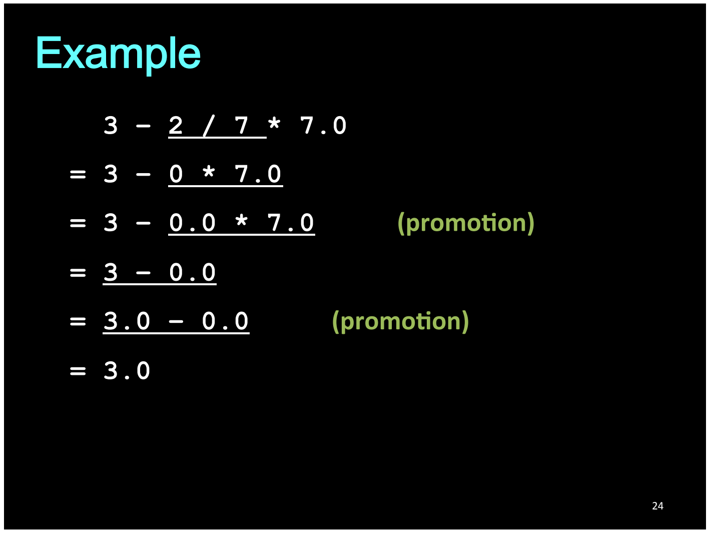
Example

```
  5 + (4 - 3) / 5 - 2 * 3 % 4
= 5 + 1 / 5 - 2 * 3 % 4
= 5 + 0 - 2 * 3 % 4
= 5 + 0 - 6 % 4
= 5 + 0 - 2
= 5 - 2
```



Example

```
  5 + (4 - 3) / 5 - 2 * 3 % 4
= 5 + 1 / 5 - 2 * 3 % 4
= 5 + 0 - 2 * 3 % 4
= 5 + 0 - 6 % 4
= 5 + 0 - 2
= 5 - 2
= 3
```



Example

  3 − $\underline{\texttt{2 / 7 }}$* 7.0

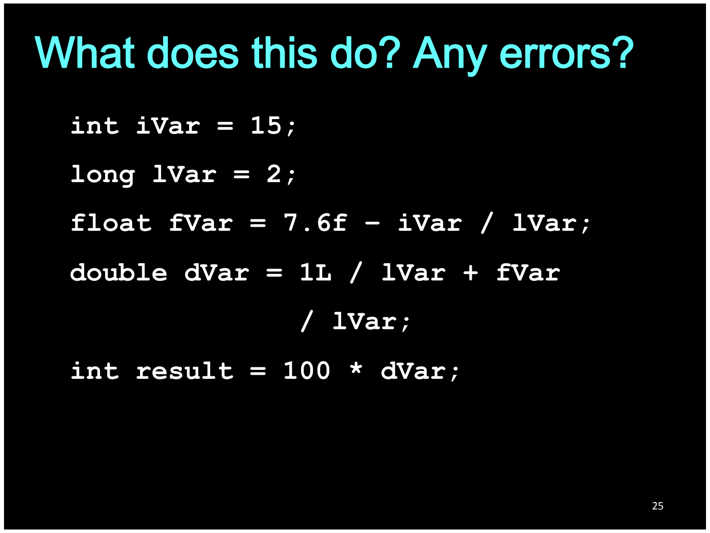
= 3 − $\underline{\texttt{0 * 7.0}}$

= 3 − $\underline{\texttt{0.0 * 7.0}}$ (promotion)

= $\underline{\texttt{3 - 0.0}}$

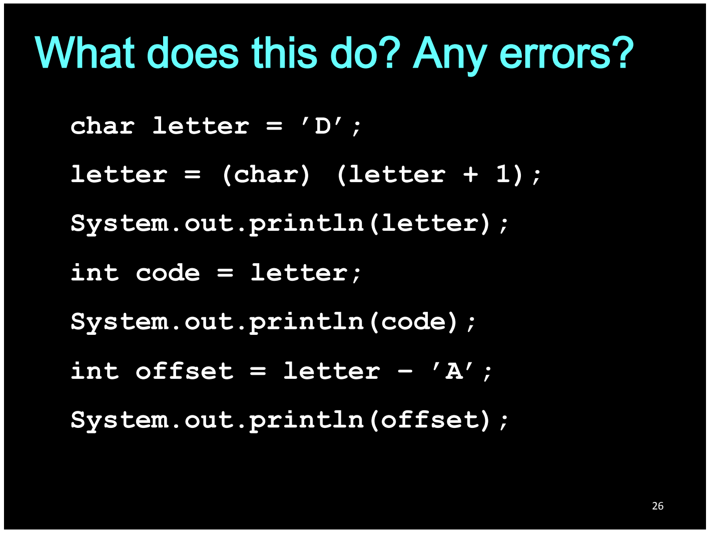
= $\underline{\texttt{3.0 - 0.0}}$ (promotion)

= 3.0



What does this do? Any errors?

```
int iVar = 15;
long lVar = 2;
float fVar = 7.6f – iVar / lVar;
double dVar = 1L / lVar + fVar
              / lVar;
int result = 100 * dVar;
```



What does this do? Any errors?

```
char letter = ’D’;
letter = (char) (letter + 1);
System.out.println(letter);
int code = letter;
System.out.println(code);
int offset = letter – ’A’;
System.out.println(offset);
```