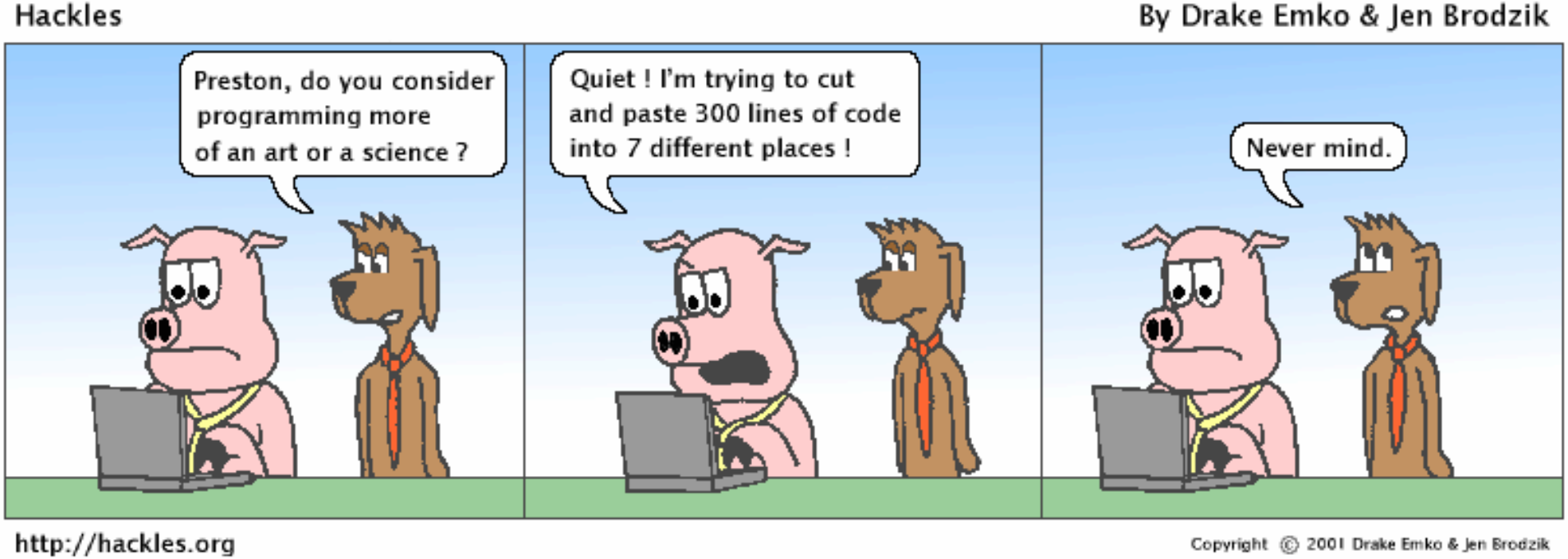
# Building Java Programs

Chapter 2

Lecture 2-1: Expressions and Variables

**reading: 2.1**


Hackles
By Drake Emko & Jen Brodzik
Preston, do you consider programming more of an art or a science ?
Quiet ! I'm trying to cut and paste 300 lines of code into 7 different places !
Never mind.
http://hackles.org
Copyright © 2001 Drake Emko & Jen Brodzik

# Data and expressions

**reading: 2.1**

# The computer's view

- Internally, computers store everything as 1's and 0's
  - Example:

```
h      → 0110100
"hi"   → 01101000110101
104    → 0110100
```

- How can the computer tell the difference between an `h` and `104`?

- **type**: A category or set of data values.
  - Constrains the operations that can be performed on data
  - Many languages ask the programmer to specify types
  - Examples: integer, real number, string

# Java's primitive types

- **primitive types**: 8 simple types for numbers, text, etc.
  - Java also has **object types**, which we'll talk about later

| Name | Description | | Examples |
|---|---|---|---|
| `int` | integers | (up to $2^{31} - 1$) | `42, -3, 0, 926394` |
| `double` | real numbers | (up to $10^{308}$) | `3.1, -0.25, 9.4e3` |
| `char` | single text characters | | `'a', 'X', '?', '\n'` |
| `boolean` | logical values | | `true, false` |

  - Why does Java distinguish integers vs. real numbers?

# Integer or real number?

- Which category is more appropriate?

| integer (`int`) | real number (`double`) |
|---|---|
| | |

1. Temperature in degrees Celsius
2. The population of lemmings
3. Your grade point average
4. A person's age in years
5. A person's weight in pounds
6. A person's height in meters
7. Number of miles traveled
8. Number of dry days in the past month
9. Your locker number
10. Number of seconds left in a game
11. The sum of a group of integers
12. The average of a group of integers

- credit: Kate Deibel

# Expressions

- **expression**: A value or operation that computes a value.
  - Examples:

```
1 + 4 * 5
(7 + 2) * 6 / 3
42
"Hello, world!"
```

  - The simplest expression is a *literal value*.
  - A complex expression can use operators and parentheses.

# Arithmetic operators

- **operator**: Combines multiple values or expressions.
  - `+` addition
  - `-` subtraction (or negation)
  - `*` multiplication
  - `/` division
  - `%` modulus (a.k.a. remainder)

- As a program runs, its expressions are *evaluated*.
  - `1 + 1` evaluates to `2`
  - `System.out.println(3 * 4);` prints `12`
    - How would we print the text `3 * 4` ?

# Integer division with /

- When we divide integers, the quotient is also an integer.
  - `14 / 4` is `3`, not `3.5`

```
   3
4 ) 14
   12
    2
```

```
   4
10 ) 45
   40
    5
```

```
   52
27 ) 1425
   135
    75
    54
    21
```

- More examples:
  - `32 / 5` is `6`
  - `84 / 10` is `8`
  - `156 / 100` is `1`

  - Dividing by 0 causes an error when your program runs.

# Integer remainder with %

- The % operator computes the remainder from integer division.
  - `14 % 4` is `2`
  - `218 % 5` is `3`

```
    3
4 ) 14
    12
     2
```

```
   43
5 ) 218
   20
    18
    15
     3
```

What is the result?

```
45 % 6  = 3
2 % 2   = 0
8 % 20  = 8
11 % 0 (div by 0!)
```

# Integer remainder with %

- The % operator computes the remainder from integer division.
  - `14 % 4` is `2`
  - `218 % 5` is `3`

```
    3
4 ) 14
   12
    2
```

```
   43
5 ) 218
   20
   18
   15
    3
```

What is the result?

```
45 % 6  = 3
2 % 2   = 0
8 % 20  = 8
11 % 0 (div by 0!)
```

- Applications of % operator:
  - Obtain last digit of a number: `230857 % 10` is `7`
  - Obtain last 4 digits: `658236489 % 10000` is `6489`
  - See whether a number is odd: `7 % 2` is `1`, `42 % 2` is `0`

# Remember PEMDAS?

- **precedence**: Order in which operators are evaluated.
  - Generally operators evaluate left-to-right.

    `1 - 2 - 3` is `(1 - 2) - 3` which is `-4`

  - But `* / %` have a higher level of precedence than `+ -`

    `1 + 3 * 4` is `13`

    ```
    6 + 8 / 2 * 3
    6 +   4   * 3
    6 +     12        is 18
    ```

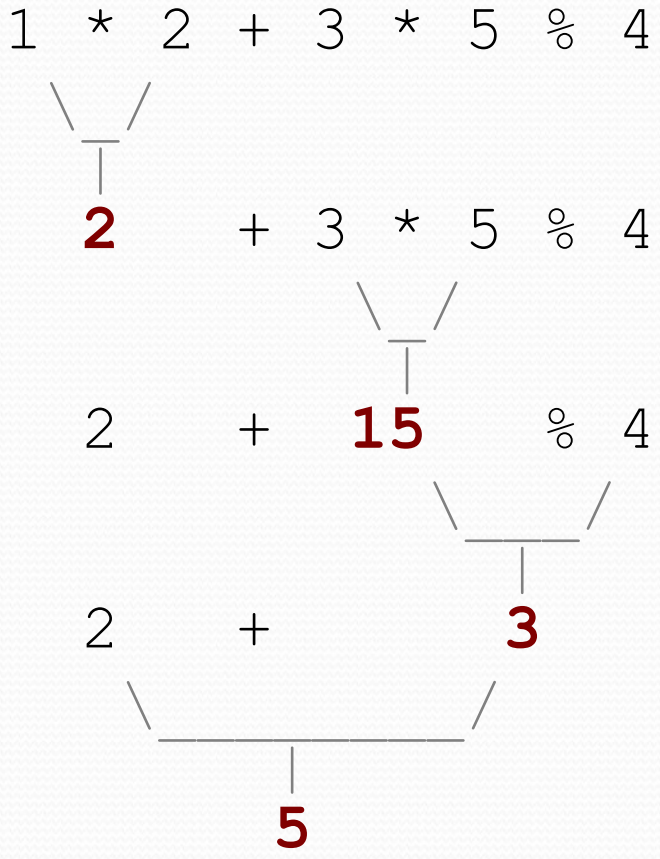
  - Parentheses can force a certain order of evaluation:

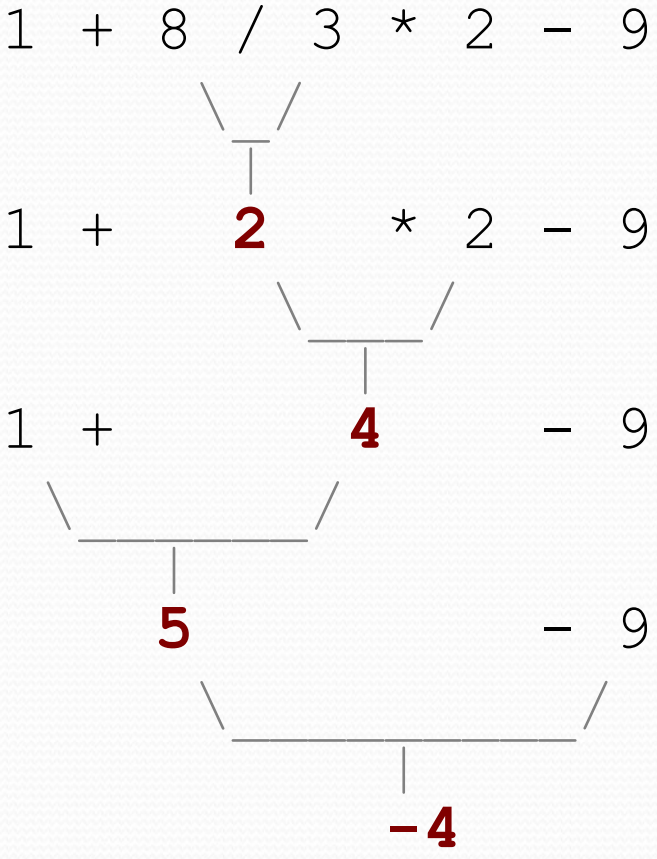
    `(1 + 3) * 4` is `16`

  - Spacing does not affect order of evaluation

    `1+3 * 4-2` is `11`

# Precedence examples



# Precedence questions

- What values result from the following expressions?
  - `9 / 5`
  - `695 % 20`
  - `7 + 6 * 5`
  - `7 * 6 + 5`
  - `248 % 100 / 5`
  - `6 * 3 - 9 / 4`
  - `(5 - 7) * 4`
  - `6 + (18 % (17 - 12))`

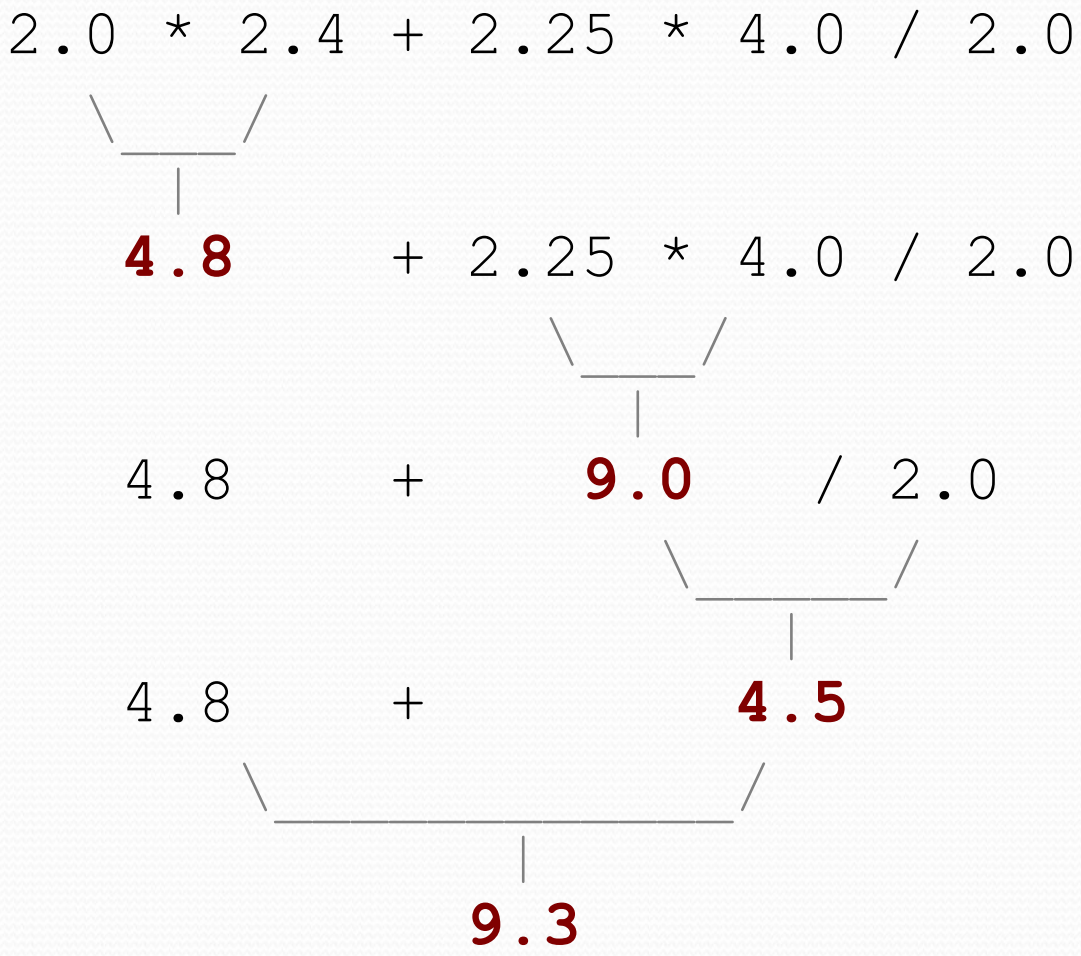
# Precedence questions

- What values result from the following expressions?
  - `9 / 5 = 1`
  - `695 % 20 = 15`
  - `7 + 6 * 5 = 37`
  - `7 * 6 + 5 = 47`
  - `248 % 100 / 5 = 9`
  - `6 * 3 - 9 / 4 = 16`
  - `(5 - 7) * 4 = -8`
  - `6 + (18 % (17 - 12)) = 9`

# Real numbers (type `double`)

- Examples: `6.022`, `-42.0`, `2.143e17`
  - Placing `.0` or `.` after an integer makes it a `double`.
- The operators `+ - * / % ()` all still work with `double`.
  - `/` produces an exact answer: `15.0 / 2.0` is `7.5`
  - Precedence is the same: `()` before `* / %` before `+ -`

# Real number example



# Precision in real numbers

- The computer internally represents real numbers in an imprecise way.

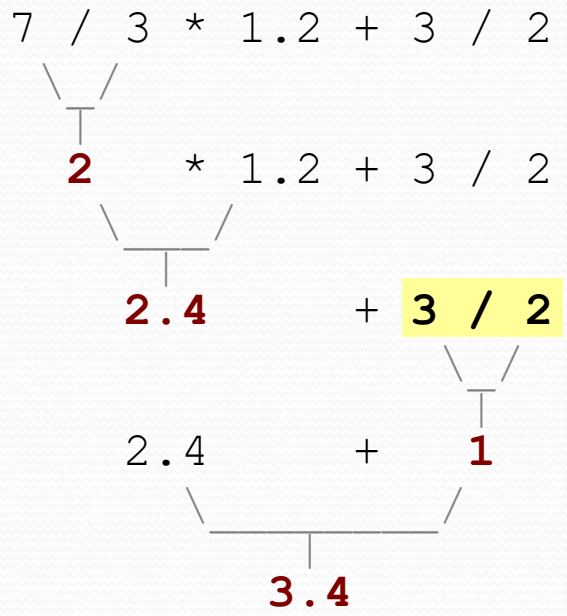
- Example:

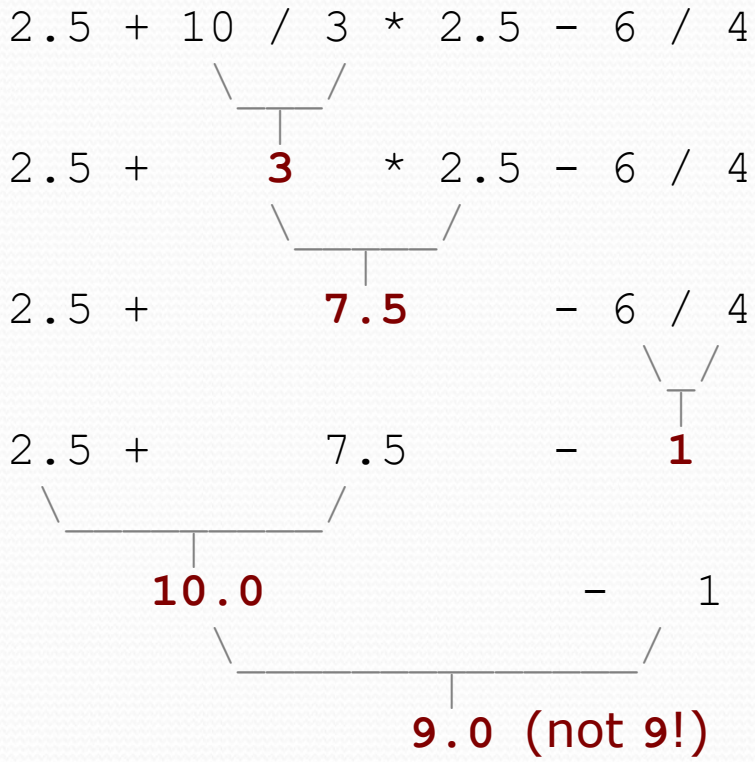
```
System.out.println(0.1 + 0.2);
```

  - The output is `0.30000000000000004`!

# Mixing types

- When `int` and `double` are mixed, the result is a `double`.
  - `4.2 * 3` is `12.6`
- The conversion is per-operator, affecting only its operands.




  - `3 / 2` is `1` above, not `1.5`.

# String concatenation

- **string concatenation**: Using + between a string and another value to make a longer string.

```
"hello" + 42     is "hello42"
1 + "abc" + 2    is "1abc2"
"abc" + 1 + 2    is "abc12"
1 + 2 + "abc"    is "3abc"
"abc" + 9 * 3    is "abc27"
"1" + 1          is "11"
4 - 1 + "abc"    is "3abc"
```

- Use + to print a string and an expression's value together.
  - `System.out.println("Grade: " + (95.1 + 71.9) / 2);`
  - Output: `Grade: 83.5`