Building Java Programs

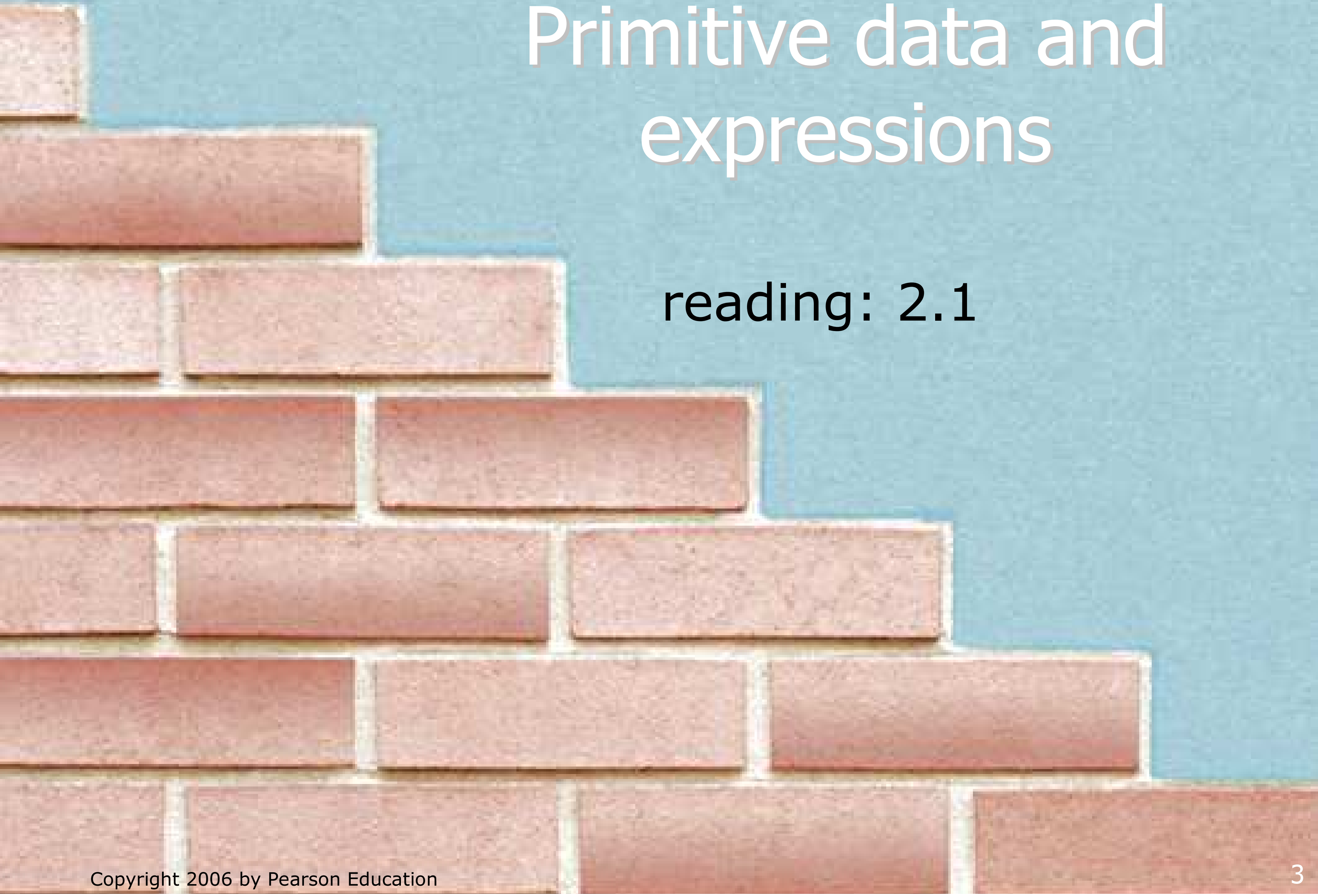
Chapter 2: Primitive Data and Definite Loops

Lecture outline

- data concepts
  - primitive types, expressions, and precedence
  - variables: declaration, initialization, assignment
  - mixing types: casting, string concatenation
  - modify-and-reassign operators
  - `System.out.print`

Primitive data and expressions

reading: 2.1



Programs that examine data

- We have printed text with `println` and strings:

```
    System.out.println("Hello, world!");
```

- Now we will learn how to print and manipulate other kinds of data, such as numbers:

```
                                         // OUTPUT:
    System.out.println(42);              // 42
    System.out.println(3 + 5 * 7);       // 38
    System.out.println(12.5 / 8.0);      // 1.5625
```

Data types

- **type**: A category or set of data values.
  - Many languages have a notion of data *types* and ask the programmer to specify what type of data is being manipulated.
  - Examples: integer, real number, string.

- Internally, the computer stores all data as 0s and 1s.
  - examples:
    ```
    42     →  101010
    "hi"   →  0110100001101001
    ```

Java's primitive types

- **primitive types**: Java's built-in simple data types for numbers, text characters, and logic.
  - Java has eight primitive types.
  - Types that are not primitive are called *object* types. (seen later)

- Four primitive types we will use:

Name	**Description**	**Examples**
`int`	integers (whole numbers)	`42, -3, 0, 926394`
`double`	real numbers	`3.1, -0.25, 4.0, 9.4e3`
`char`	single text characters	`'a', 'X', '?', '\n'`
`boolean`	logical values	`true, false`

Expressions

- **expression**: A data value, or a set of operations that compute a data value.

  Example: `1 + 4 * 3`

  - The simplest expression is a *literal value*.
  - A complex expression can use *operators* and parentheses.
    - The values to which an operator applies are called *operands*.

- Five arithmetic operators we will use:

+	addition
-	subtraction or negation
*	multiplication
/	division
%	modulus, a.k.a. remainder

Evaluating expressions

- As your Java program executes:
  - When a line with an expression is reached, the expression is *evaluated* (its value is computed).
    - `1 + 1` is evaluated to `2`
  - `System.out.println(3 * 4);` prints `12`
    (How would we print the text `3 * 4` ?)
- When an expression contains more than one operator of the same kind, it is evaluated left-to-right.
  - `1 + 2 + 3` is `(1 + 2) + 3` which is `6`
  - `1 - 2 - 3` is `(1 - 2) - 3` which is `-4`

Integer division with /

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

```
     3                    4                   52
4 ) 14              10 ) 45            27 ) 1425
    12                   40                 135
     2                    5                   75
                                              54
                                              21
```

  - More examples:
    - 1425 / 27 is 52
    - 35 / 5 is 7
    - 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 a division of two integers.
  - `14 % 4 is 2`
  - `218 % 5 is 3`

```
      3                          43
4 ) 14                   5 ) 218
    12                       20
     2                        18
                              15
                               3
```

- What are the results of the following expressions?

```
45 % 6
2 % 2
8 % 20
11 % 0
```

Applications of % operator

- Obtains the last digit (units place) of a number:
  - Example: From 230857, obtain the 7.

- Obtain the last 4 digits of a Social Security Number:
  - Example: From 658236489, obtain 6489.

- Obtains a number's second-to-last digit (tens place):
  - Example: From 7342, obtain the 4.

- Use the % operator to see whether a number is odd:
  - Can it help us determine whether a number is divisible by 3?

Operator precedence

- **precedence**: Order in which operations are computed.
  - `* / %` have a higher level of precedence than `+ -`

    `1 + `**`3 * 4`** is 13

  - Parentheses can be used to 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

```
1 * 2 + 3 * 5 / 4          1 + 2 / 3 * 5 - 4
 \_/                            \_/
  |                              |
  2   + 3 * 5 / 4          1 +   0   * 5 - 4
         \_/                      \___/
          |                         |
  2   +  15   / 4          1 +      0    - 4
           \___/            \______/
             |                 |
  2   +      3                 1         - 4
   \________/                   \_________/
       |                             |
       5                             -3
```

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))`

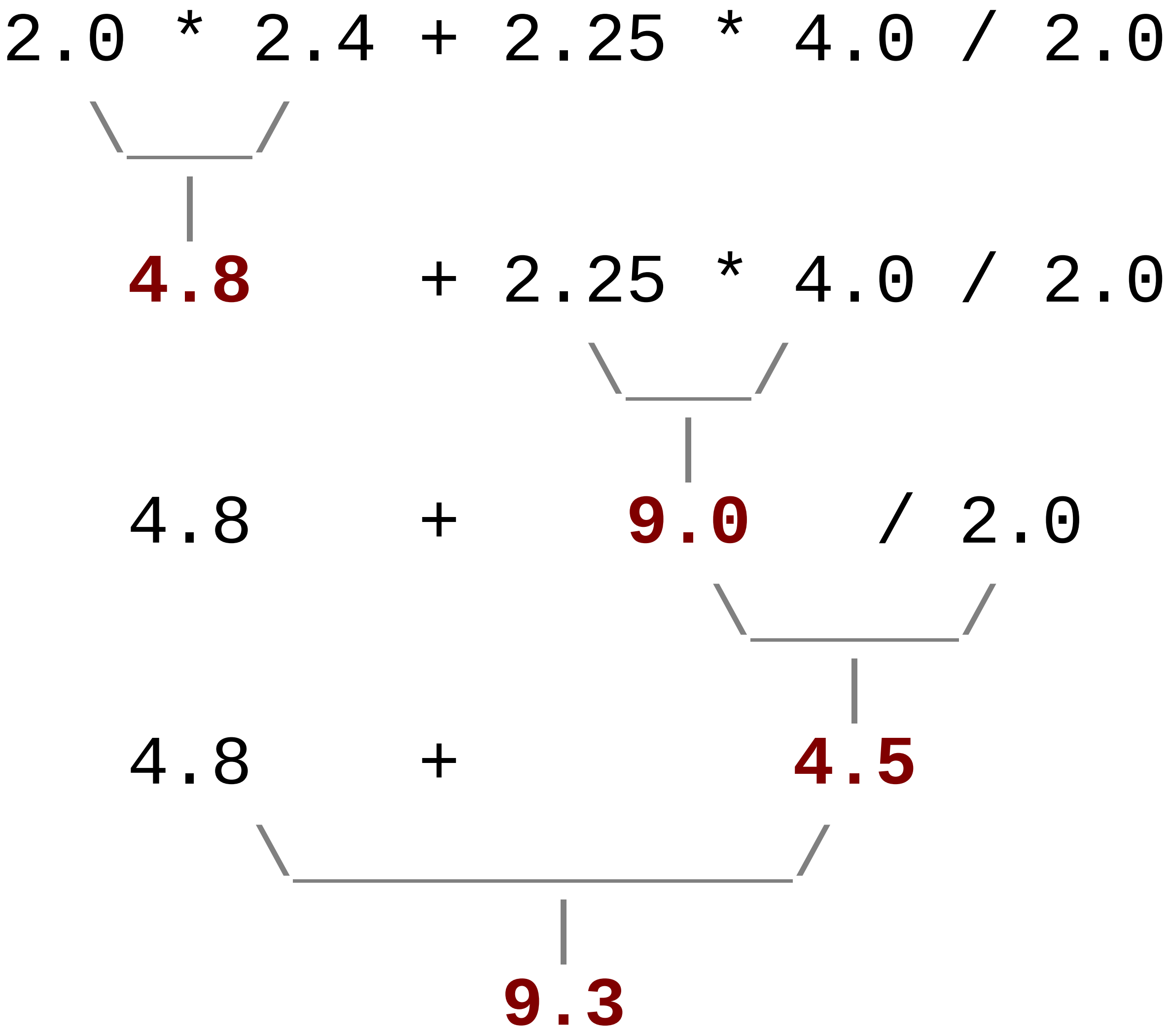
Real numbers (`double`)

- Java can also manipulate real numbers (type `double`).
  - Examples: `6.022`   `-15.9997`   `42.0`   `2.143e17`

- The operators `+ - * / % ( )` all work for real numbers.
  - The `/` produces an exact answer when used on real numbers.
    `15.0 / 2.0` is `7.5`

- The same rules of precedence that apply to integers also apply to real numbers.
  - Evaluate `( )` before `* / %` before `+ -`

Real number example



```
2.0 * 2.4 + 2.25 * 4.0 / 2.0

   4.8    + 2.25 * 4.0 / 2.0

   4.8    +    9.0   / 2.0

   4.8    +        4.5

         9.3
```

Real number precision

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

- Example:

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

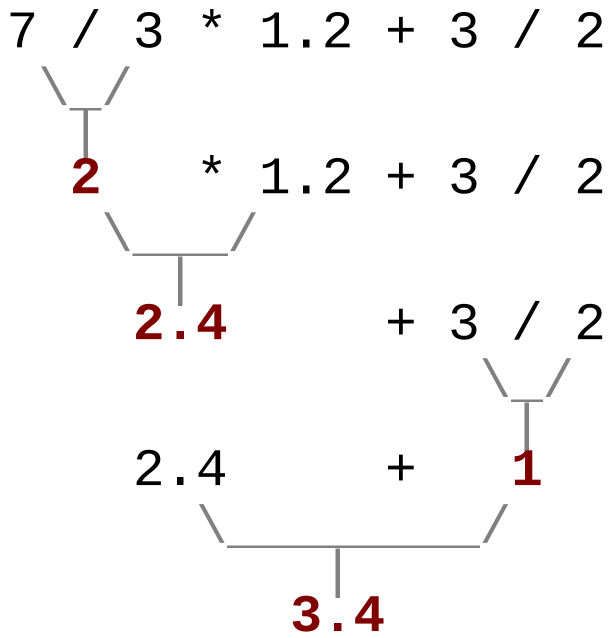
  - The mathematically correct answer should be 0.3
  - Instead, the output is 0.30000000000000004

- Later we will learn some ways to produce a better output for examples like the above.

Mixing integers and reals

- When a Java operator is used on an integer and a real number, the result is a real number.
  - `4.2 * 3` is `12.6`
  - `1 / 2.0` is `0.5`

- The conversion occurs on a per-operator basis. It affects only its two operands.



```
7 / 3 * 1.2 + 3 / 2
  2   * 1.2 + 3 / 2
    2.4     + 3 / 2
    2.4     +   1
        3.4
```

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

Mixed types example

```
2.0 + 10 / 3 * 2.5 - 6 / 4
       \___/
         |
2.0 +    3   * 2.5 - 6 / 4
         \_____/
            |
2.0 +      7.5     - 6 / 4
                      \_/
                       |
2.0 +      7.5     -   1
 \_________/
      |
     9.5           -   1
       \______________/
               |
              8.5
```

Variables

reading: 2.2

The computer's memory

- Expressions are like using the computer as a calculator.

- Calculators have memory keys to store/retrieve values.
  - When is this useful?

  - We'd like the ability to save and restore values in our Java programs, like the memory keys on the calculator.

Variables

- **variable**: A piece of your computer's memory that is given a name and type and can store a value.
  - Usage:
    - compute an expression's result,
    - store that result into a variable,
    - and use that variable later in the program.
  - Unlike with a calculator, we can declare as many variables as we want.
- Variables are a bit like preset stations on a car stereo.

Declaring variables

- **variable declaration statement**: A Java statement that creates a new variable of a given type.
  - A variable is *declared* in a statement with its type and name.
  - Variables must be declared before they can be used.

- Declaration syntax:

  <type> <name> ;

  - `int x;`
  - `double myGPA;`

  - The name can be any identifier.

More on declaring variables

- Declaring a variable sets aside a piece of memory in which you can store a value.
  - `int x;`
  - `int y;`

  - Part of the computer's memory:

(The memory has no values in it yet.)

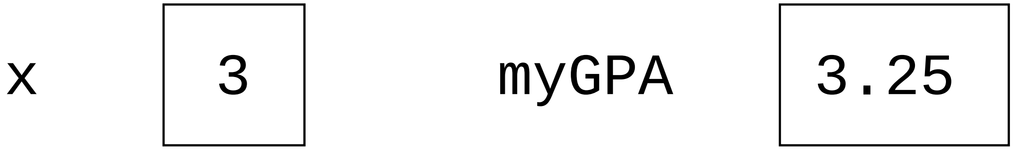
Assignment statements

- **assignment statement**: A statement that stores a value into a variable's memory.
  - Variables must be declared before they can be assigned a value.

- Assignment statement syntax:

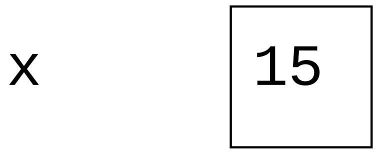
  <name> `=` ***<value>*** `;`

  - `x = 3;`
  - `myGPA = 3.25;`



More about assignment

- The ***<value>*** assigned can be a complex expression.
  - The expression is evaluated; the variable stores the result.
  - `x = (2 + 8) / 3 * 5;`



- A variable can be assigned a value more than once.
  - Example:

```
int x;
x = 3;
System.out.println(x);   // 3

x = 4 + 7;
System.out.println(x);   // 11
```

Using variables' values

- Once a variable has been assigned a value, it can be used in an expression, just like a literal value.

```
int x;
x = 3;
System.out.println(x * 5 - 1);
```

  - The above has output equivalent to:

```
System.out.println(3 * 5 - 1);
```

Assignment and algebra

- Though the assignment statement uses the = character, it is not an algebraic equation.
  - = means, "store the value on the right in the variable on the left"
  - Some people read `x = 3;` as, "x becomes 3" or, "x gets 3"
  - We would not say `3 = 1 + 2;` because `3` is not a variable.

- What happens when a variable is used on both sides of an assignment statement?

  -
    ```
    int x;
    x = 3;
    x = x + 2;   // what happens?
    ```

  - The above wouldn't make any sense in algebra...

Some errors

- A compiler error will result if you declare a variable twice, or declare two variables with the same name.

  -
    ```
    int x;
    int x;                        // ERROR: x already exists
    ```

- A variable that has not been assigned a value cannot be used in an expression or `println` statement.

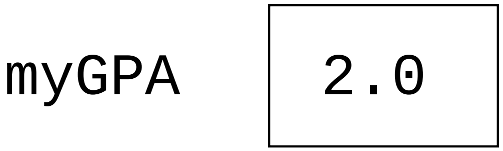
  -
    ```
    int x;
    System.out.println(x);    // ERROR: x has no value
    ```

Assignment and types

- A variable can only store a value of its own type.

  -
    ```
    int x;
    x = 2.5;    // ERROR: x can only store int
    ```
- An `int` value can be stored in a `double` variable.
  - The value is converted into the equivalent real number.

  -
    ```
    double myGPA;
    myGPA = 2;
    ```



Assignment examples

- What is the output of the following Java code?

```
int number;
number = 2 + 3 * 4;
System.out.println(number - 1);

number = 16 % 6;
System.out.println(2 * number);
```

- What is the output of the following Java code?

```
double average;
average = (11 + 8) / 2;
System.out.println(average);

average = (5 + average * 2) / 2;
System.out.println(average);
```

Declaration/initialization

- A variable can be declared and assigned an initial value in the same statement.

- Declaration/initialization statement syntax:

  <type> ***<name>*** = ***<value>*** ;

  - `double myGPA = 3.95;`
  - `int x = (11 % 3) + 12;`

same effect as:

```
double myGPA;
myGPA = 3.95;

int x;
x = (11 % 3) + 12;
```

Multiple declaration error

- The compiler will fail if you try to declare-and-initialize a variable twice.

  -
    ```
    int x = 3;
    System.out.println(x);

    int x = 5;       // ERROR: variable x already exists
    System.out.println(x);
    ```
  - This is the same as trying to declare x twice.

- How can the code be fixed?

Multiple declarations per line

- It is legal to declare multiple variables on one line:

  <type> ***<name>***, ***<name>***, ..., ***<name>*** ;

  - `int a, b, c;`
  - `double x, y;`

- It is legal to declare/initialize several at once:

  <type> ***<name>*** = ***<value>*** , ..., ***<name>*** = ***<value>*** ;

  - `int a = 2, b = 3, c = -4;`
  - `double grade = 3.5, delta = 0.1;`

- The variables must be of the same type.

Integer or real number?

- Categorize each of the following quantities by whether an `int` or `double` variable would best to store it:

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

Type casting

- **type cast**: A conversion from one type to another.

  Common uses:
  - To promote an `int` into a `double` to achieve exact division.
  - To truncate a `double` from a real number to an integer.

- type cast syntax:

  (***<type>***) ***<expression>***

  Examples:
  - double result = **(double)** 19 / 5;   // 3.8
  - int result2 = **(int)** result;   // 3

More about type casting

- Type casting has high precedence and only casts the item immediately next to it.
  - double x = **(double)** 1 + 1 / 2;   // 1
  - double y = 1 + **(double)** 1 / 2;   // 1.5

- You can use parentheses to force evaluation order.
  - double average = **(double)** (a + b + c) / 3;

- A conversion to `double` can be achieved in other ways.
  - double average = 1.0 * (a + b + c) / 3;

String concatenation

- **string concatenation**: Using the + operator between a String and another value to make a longer String.
  - Examples:
    - Recall: Precedence of + operator is below * / %

```
"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"

"abc" + 4 - 1    causes a compiler error... why?
```

Printing String expressions

- String expressions with + are useful so that we can print complicated messages that involve computed values.

  -
    ```
    double grade = (95.1 + 71.9 + 82.6) / 3.0;
    System.out.println("Your grade was " + grade);

    int students = 11 + 17 + 4 + 19 + 14;
    System.out.println("There are " + students +
                     " students in the course.");
    ```

Output:

```
Your grade was 83.2
There are 65 students in the course.
```

Example variable exercise

- Write a Java program that stores the following data:
  - Section AA has 17 students.
  - Section AB has 8 students.
  - Section AC has 11 students.
  - Section AD has 23 students.
  - Section AE has 24 students.
  - Section AF has 7 students.
  - The average number of students per section.

and prints the following:

```
There are 24 students in Section AE.
There are an average of 15 students per section.
```

Increment and decrement

- The *increment* and *decrement* operators increase or decrease a variable's value by 1.

Shorthand	Equivalent longer version
<variable> ++ ;	***<variable>*** = ***<variable>*** + 1;
<variable> -- ;	***<variable>*** = ***<variable>*** - 1;

- Examples:

```
int x = 2;
x++;                    // x = x + 1;
                        // x now stores 3

double gpa = 2.5;
gpa--;                  // gpa = gpa - 1;
                        // gpa now stores 1.5
```

Modify-and-assign operators

- Java has several shortcut operators that allow you to quickly modify a variable's value:

Shorthand	Equivalent longer version
<variable> += ***<value>*** ;	***<variable>*** = ***<variable>*** + ***<value>*** ;
<variable> -= ***<value>*** ;	***<variable>*** = ***<variable>*** - ***<value>*** ;
<variable> *= ***<value>*** ;	***<variable>*** = ***<variable>*** * ***<value>*** ;
<variable> /= ***<value>*** ;	***<variable>*** = ***<variable>*** / ***<value>*** ;
<variable> %= ***<value>*** ;	***<variable>*** = ***<variable>*** % ***<value>*** ;

- Examples:
  - `x += 3;`   `// x = x + 3;`
  - `gpa -= 0.5;`   `// gpa = gpa - 0.5;`
  - `number *= 2;`   `// number = number * 2;`

System.out.print command

- Recall: `System.out.println` prints a line of output and then advances to a new line.
- `System.out.print` prints without moving to a new line.
  - This allows you to print partial messages on the same line.
- Example:

  -
    ```
    System.out.print("Kind of");
    System.out.print("Like a cloud,");
    System.out.println("I was up");
    System.out.print("Way up ");
    System.out.println("in the sky");
    ```

    Output:

    ```
    Kind ofLike a cloud,I was up
    Way up in the sky
    ```