Lecture 5: Basic Java Syntax

AP Computer Science Principles

Data Types and Operations

Data types

- **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

- Internally, computers store everything as 1s and 0s

```
104    → 01101000
"hi"   → 0110100 0110101
```

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, 9263`
`float`	**real numbers (up to 10^{38})**	`3.1, -0.25, 9.4e3`
`boolean`	`logical values`	`true, false`

Arithmetic operators

- **operator**: Combines multiple values or expressions.

+	addition
–	subtraction (or negation)
*	multiplication
/	division
%	modulus (a.k.a. remainder)

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`

`2 % 2`

`8 % 20`

`11 % 0`

Answers:

3

0

8

undefined

- 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`

Exact Change

Find the exact change for 137 cents using quarters, dimes, nickels and cents. Use the least number of coins.

How many quarters? 137 / 25 = 5 quarters (Integer Division!)

What's leftover? 137 % 25 = 12 cents

How many dimes? 12 / 10 = 1 dime

What's leftover? 12 % 10 = 2 cents

How many nickels? 2 / 5 = 0 nickels.

What's leftover? 2 % 5 = 2 cents.

Precedence

- **precedence**: Order in which operators are evaluated.
  - Generally operators evaluate left-to-right for operators with the same level of precedence. For example, + and -.

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

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

    `1 + 3 * 4` is `13`

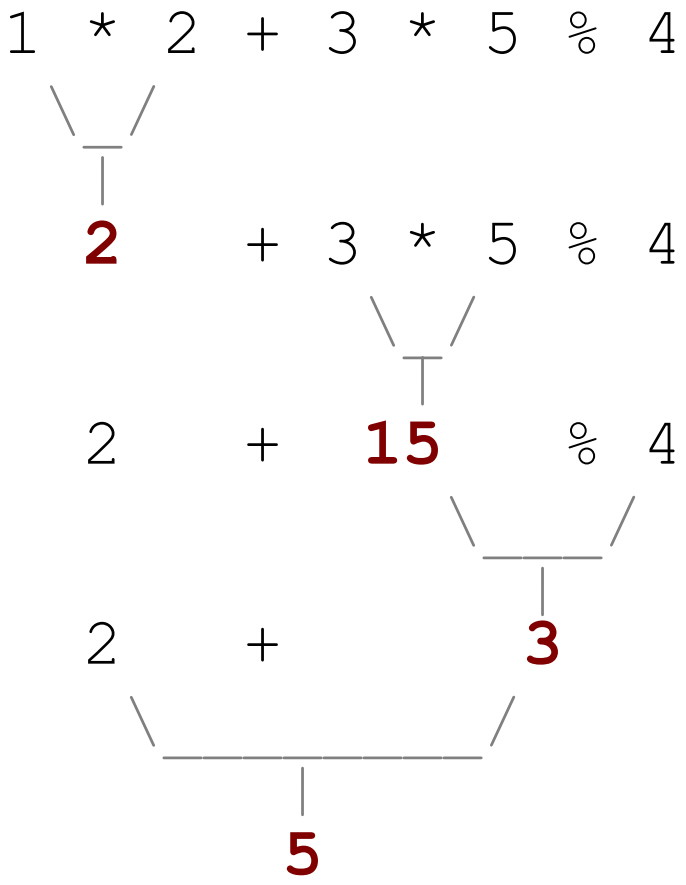
    ```
    6 + 8 / 2 * 3
    6 +   4   * 3
    6 +     12
    ```

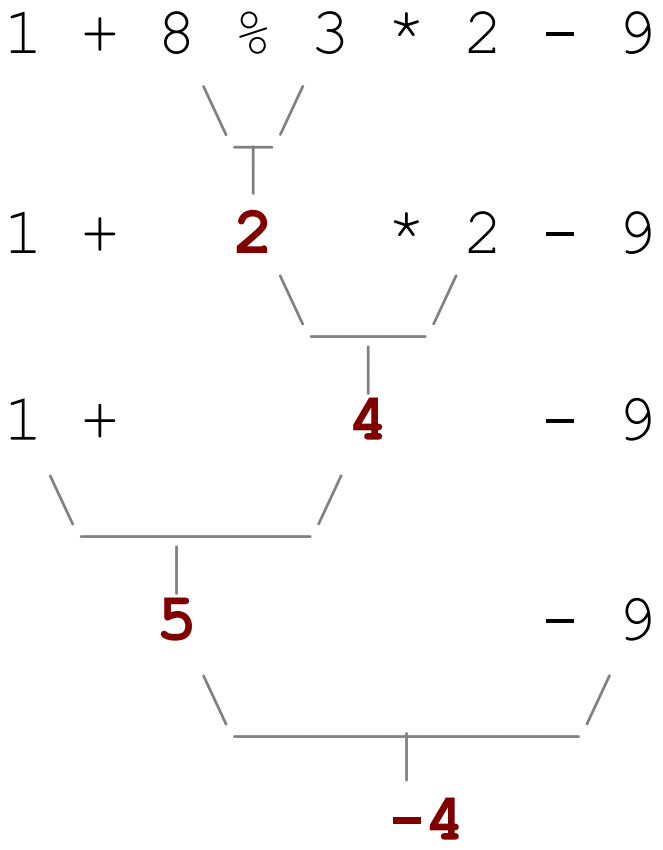
    is `18`
  - Parentheses can force a certain order of evaluation:

    `(1 + 3) * 4` is `16`
  - Spacing does not affect order of evaluation

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

Precedence examples

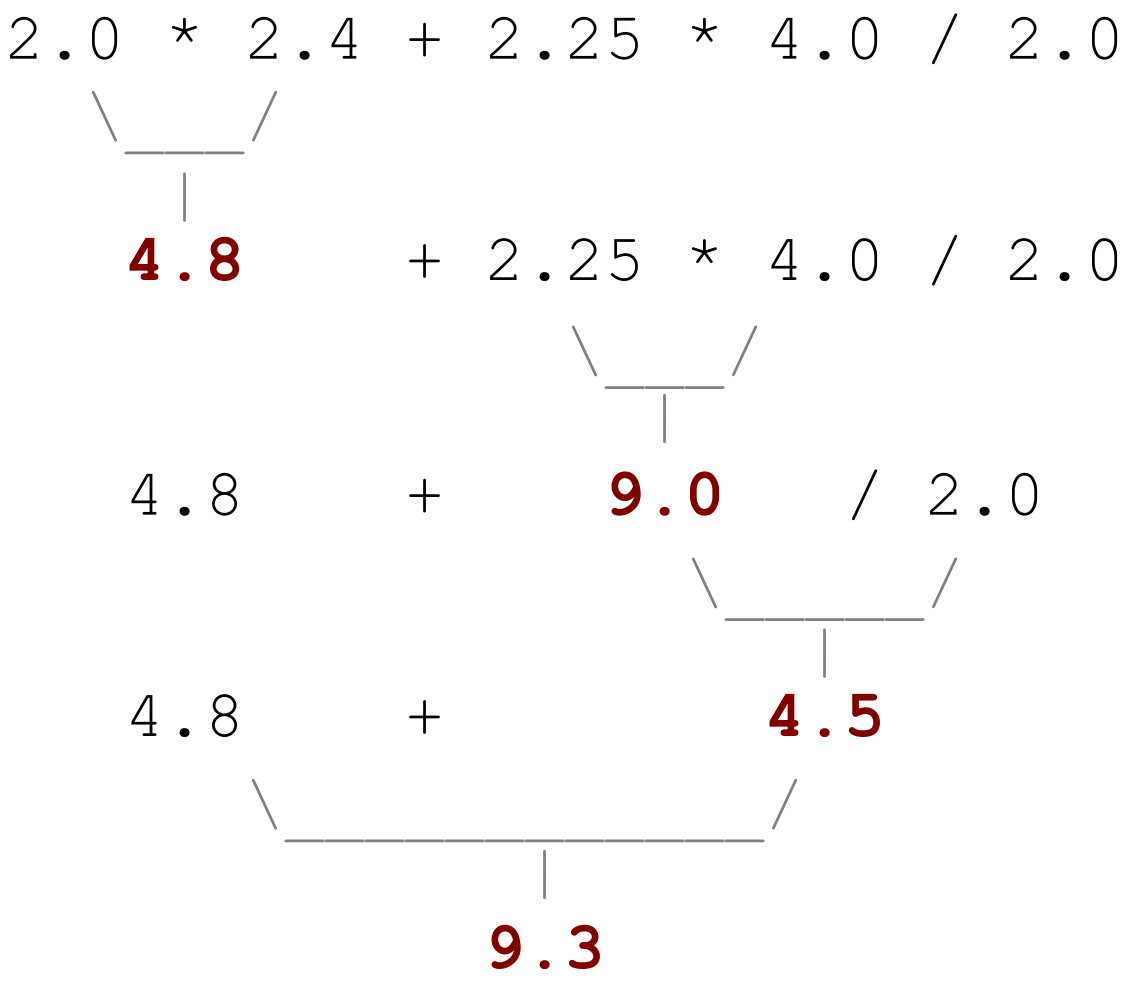




Real numbers (type `float`)

- Examples: `6.022`, `-42.0`, `2.143`
  - Placing `.0` or `.` after an integer makes it a `float`.
- The operators `+ - * / % ()` all still work with `float`.
  - `/` produces an exact answer: `15.0 / 2.0` is `7.5`
  - Precedence is the same: `()` before `* / %` before `+ -`
- When `int` and `float` are mixed, the result is a `float`.
  - `4.2 * 3` is `12.6`

Real number example



Strings

- **string**: A sequence of characters to be printed.
  - Starts and ends with a " quote " character.
  - Examples:

```
"hello"
"This is a string.  It's very long!"
```

- Restrictions:
  - May not span multiple lines.

```
"This is not
a legal String."
```

  - May not contain a " character.

```
"This is not a "legal" String either."
```

String concatenation

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

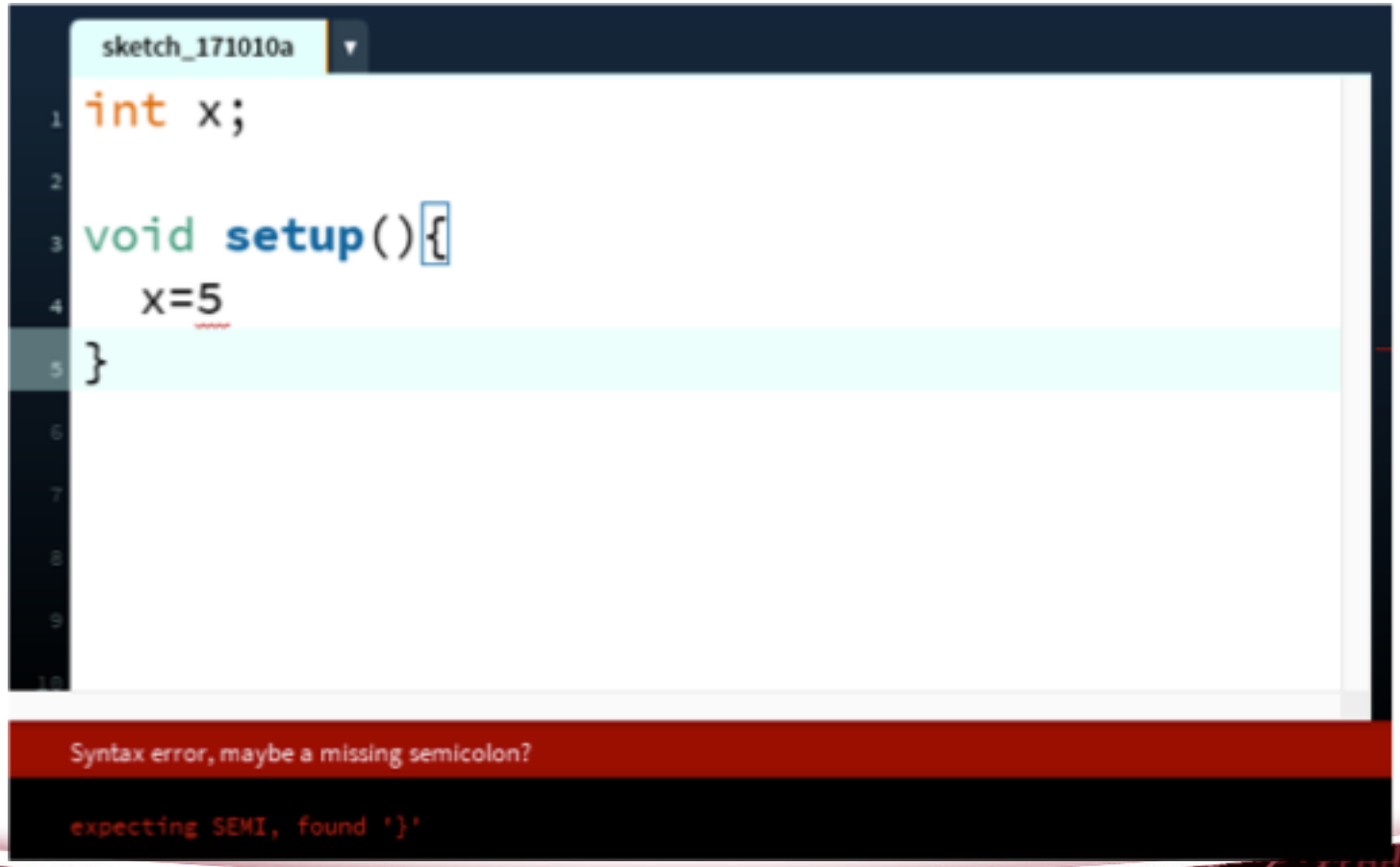
```
"hello" + 42          "hello42"
1 + "abc" + 2         "1abc2"
"abc" + 1 + 2         "abc12"
1 + 2 + "abc"         "3abc"
"abc" + 9 * 3         "abc27"
"1" + 1               "11"
("abc"+(2+3))+4*5     "abc520"
```

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

    Output:
    - `Grade: 83.5`

The Console

- The **console** refers to the box at the bottom of Processing. It is generally used to print error messages in the code.



The Console

Messages can be printed on the console(for error-checking purposes, etc..) by using the command **`println().`**

println(4); // 4

println(4+3/2); // 5

println(“Hello, world”); // Hello, world(no quotation in output)

Variables

Variables

- **variable**: A piece of the computer's memory that is given a name and type, and can store a value.
  - Like preset stations on a car stereo, or cell phone speed dial:

  - Steps for using a variable:
    - *Declare* it - state its name and type
    - *Initialize* it - store a value into it
    - *Use* it - print it or use it as part of an expression

Declaration

- **variable declaration**: Sets aside memory for storing a value.
  - Variables must be declared before they can be used.

- Syntax:

  type name;

  - The name is an *identifier*.

  - `int x;`

  - `float myGPA;`

Assignment

- **assignment**: Stores a value into a variable.
  - The value can be an expression; the variable stores its result.

- Syntax:

```
name = expression;
```

- `int x;`
  `x = 3;`

x	3

- `float myGPA;`
  `myGPA = 1.0 + 2.25;`

myGPA	3.25

Using variables

- Once given a value, a variable can be used in expressions:

```
int x;
x = 3;
println("x is " + x);      // x is 3

println(5 * x - 1);        // 14
```

- You can assign a value more than once:

x	11

```
int x;
x = 3;
println(x + " here");      // 3 here

x = 4 + 7;
println("now x is " + x); // now x is 11
```

Declaration/initialization

- A variable can be declared/initialized in one statement.

- Syntax:

```
type name = value;
```

- `float myGPA = 3.95;`

myGPA	3.95

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

x	14

Assignment and algebra

- Assignment uses = , but it is not an algebraic equation.

  =means, *"store the value at right in variable at left"*

  - The right side expression is evaluated first, and then its result is stored in the variable at left.

- What happens here?

```
int x = 3;
x = x + 2;    // ???
```

Multiple Variables

- Multiple variables of the same type can be declared and initialized at the same time.

- Syntax:

type name1, name 2, name3;

type name1 = value1, name2 = value2, name3 = value3;

```
int x, y, z;  // declare three integers.

int a = 1, b = 2, c = 3; // declare and initialize three
                         // integers.
```

Assignment and types

- A variable can only store a value of its own type.
  - `int x = 2.5;` **`// ERROR: incompatible types`**
- An `int` value can be stored in a `float` variable.
  - The value is converted into the equivalent real number.
  - `float myGPA = 4;`

myGPA	4.0

  - `float avg = 11 / 2;`

avg	**5.0**

    - Why does `avg` store `5.0` and not `5.5` ?

Compiler errors

- Order matters.
  - ```
    int x;
    7=x; // ERROR: should be x=7;
    ```
- A variable can't be used until it is assigned a value.
  - ```
    int x;
    println(x);   // ERROR: x has no value
    ```
- You may not declare the same variable twice.
  - ```
    int x;
    int x;                    // ERROR: x already exists
    ```
  - ```
    int x = 3;
    int x = 5;                // ERROR: x already exists
    ```
    - How can this code be fixed? (remove the second int)

Declaring Strings

A string can be stored in a variable with type **String**. String is not one of the 8 primitive types; it is an **object type**.

```
String x = "Hello!"; // declare and initialize x
x = x + " Mike!"; // "Hello! Mike!"
x = x + 2 * 4 + 5; //"Hello! Mike!85"


println(x);
// Hello! Mike!85
println("x");
 // x
```

Declaring Strings

```
String beans;
beans = “Programming”;

String cheese = “in”;
String quartet;
println(quartet); //error, quartet not initialized
quartet = beans + cheese + “Java”;

println(quartet); //ProgramminginJava

String theory = beans + ” “ + cheese;
println(theory + “ ” +“Java”);
//Programming in Java
```

Printing a variable's value

- Use + to print a string and a variable's value on one line.

```
float grade = (95.1 + 71.9 + 82.6) / 3.0;
println("Your grade was " + grade);

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

- Output:

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

Comments

- **comment**: A note written in source code by the programmer to describe or clarify the code.
  - Comments are not executed when your program runs.
- Syntax:

```
// comment text, on one line
```

or,

```
/* comment text; may span multiple lines */
```

- Examples:

```
// This is a one-line comment.

/* This is a very long
   multi-line
   comment. */
```

Lab 1: Exact Change

Reread the Exact Change example in this lecture. This program will convert that example into code. Download the distribution code from the website. It will guide you through writing a program that display the exact change from a given total.

```
Total:122 cents

4 quarters 2 dimes 0 nickels 2 pennies
```

Lab 1: Exact Change

The method `text(str,x,y)` will display the the String str at given (x,y) position. You can set the font size with `textSize(size)`.

The following display “Hello” at (200,300).

```
textSize(32);
text(“Hello”,200,300);
```

Lab 2: Stopwatch

The example will help you write a stopwatch program, similar to the stopwatch app on your phone. The math is similar to the exact change lab.

Hint: By default `draw()` runs 60 times a second and `frameCount` is a reserved variable that keep track of the current frame; at the end of the first draw iteration, `frameCount` is 1, at the end of the second, it is 2 etc…

Lab 2: Stopwatch

Use frameCount to write the stopwatch program that display a running stopwatch. Here's a sample frame:

0:1:35.95

The above output denotes 0 hrs: 1 mins: 35.95 seconds.

Lab 2: Stopwatch

The distribution code, Stopwatch.pde, for this program has been created for you with comments that guides you through the process of writing the program. Download it from the website.

Homework

1) Read and reread these lecture notes.

2) Complete the problem set.

3) Complete the two labs.

References

Part of this lecture is taken from the following book.

Stuart Reges and Marty Stepp. Building Java Programs: A Back to Basics Approach. Pearson Education. 2008.