CS139 Algorithm Development
Activity 05B: Introduction to Methods

Objectives
At the end of this activity, students will be able to:

- Define void methods with optional parameters
- Break down a large program into several methods
- Draw a stack diagram that illustrates method calls

Part 1 – Warm-up

Today you will practice writing your own methods. Methods help organize and simplify your programs by:

1. hiding multiple lines of code behind a single statement.
2. using an English name in place of complex Java code.
3. allowing you to reuse code without copying and pasting.
4. making your code easier to read, understand, and debug.

Consider the following example: (JavaDoc comments have been removed to save space)

```
public class Part1 {

    public static void main(String[] args) {
        System.out.println("First line.");
        threeLine();
        System.out.println("Second line.");
    }

    public static void newLine() {
        System.out.println();
    }

    public static void threeLine() {
        newLine();
        newLine();
        newLine();
    }

}
```

How many lines of code call the `System.out.println()` method? _____

How many times is the `println` method called when the program is run? _____

In your own words, what is a method?

Part 2 – Order of Execution

When you look at a class definition that contains several methods, it may be tempting to read it from top to bottom. But that is likely to be confusing, because "top to bottom" is not the *order of execution* of the program. Execution always begins at the first statement of `main`, regardless where it is located in the source code. Statements are then executed one at a time until you reach a method call. At this point, instead of going to the next statement you go to the first line of the invoked method, execute all the statements there, and then come back and pick up again where you left off.

BOARD – What is the output of the following program? Be precise about where there are spaces and where there are newlines.

```
public class Part2 {

    public static void baffle() {
        System.out.print("wug");
        ping();
    }

    public static void main(String[] args) {
        System.out.print("No, I ");
        zoop();
        System.out.print("I ");
        baffle();
    }

    public static void ping() {
        System.out.println(".");
    }

    public static void zoop() {
        baffle();
        System.out.print("You wugga ");
        baffle();
    }

}
```

(Credit: http://www.greenteapress.com/thinkapjava/)

WAIT HERE

Part 3 – Divide and Conquer

"Methods are commonly used to break a problem into small, manageable pieces." (Gaddis 5/e, p. 273) Your goal is to write a Java program that prints out the entire text that your instructor will give you. As a contest, let's see who can accomplish this task using the fewest number of `println` statements!

Rules:

1. Each team member must write at least one method (including `main`).
2. Each line of text (including blank lines) must have its own `System.out.println()`.
3. You may NOT join multiple lines of text using the `\n` character.
4. You may NOT use loops, decisions, or other Java syntax we haven't yet covered in class.
5. To simplify writing code by hand, you MAY define constants with abbreviated names.

BOARD – How many `println` statements did you write? How many methods did you write (including `main`)?

Part 4 – Parameters and Arguments

Some methods we used in PA1 require *arguments*, or values that you provide when you call the method. For example, `println` can take a `String` as an argument. Some methods take more than one argument. For example, `printf` takes two or more arguments: one for the format string, and others for the values to be formatted.

When you write a method, you specify a list of *parameters*. A parameter is a variable that stores an argument. The parameter list indicates what arguments are required. For example, `printTwice` specifies a single parameter:

```
public static void printTwice(String line) {
    System.out.println(line);
    System.out.println(line);
}
```

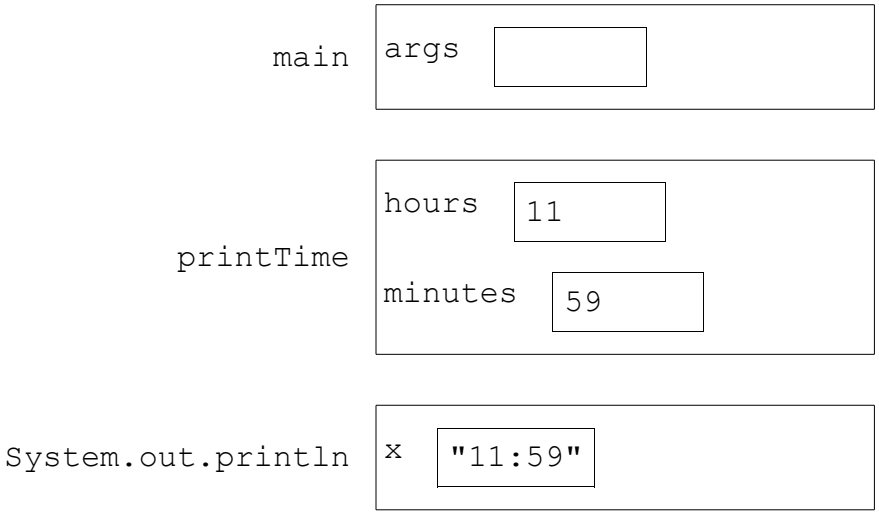
Write a method `showLength` that describes how long a string is, e.g, `showLength("CS139")` should print the message `CS139 is 5 characters long` followed by a newline.

Write a method `printTime` that takes two integers, `hours` and `minutes`, and prints them out separated by a colon. For example, `printTime(11, 59)` should print the text `11:59` followed by a newline. Note that multiple parameters are separated by commas in the method header.

Part 5 – Stack Diagrams

Parameters and other variables only exist inside their own methods. Continuing the `printTime` example, there is no such thing as `hours` or `minutes` within the scope of `main`. If you refer to them in `main`, the compiler will complain. Similarly, within `printTime` there is no such variable `args` as defined by `main`.

One way to keep track of where each variable is defined is with a *stack diagram*. When `println` is called in the `printTime` example, the stack diagram looks like this:



For each method there is a box called a *frame* that contains the method's parameters and variables. The name of the method appears to the left of the frame. As usual, the value of each variable is drawn inside a box with the name of the variable beside it.

Draw a stack diagram for your `showLength` method from Part 4 when `println` is called.

Draw the longest stack diagram you can using your program from Part 3.