Multiple Methods

Java programs are organized into ***classes***, each of which has one or more ***methods***, each of which has one or more ***statements***. Writing methods allows you to break down a complex program into smaller blocks of reusable code.

Manager:		Recorder:	
Presenter:		Reflector:	

Content Learning Objectives

After completing this activity, students should be able to:

- Apply methods from the Math class based on their documentation.
- Explain the syntax of a method declaration (parameters and return type).
- Draw a diagram that shows the call stack at a given point of execution.

Process Skill Goals

During the activity, students should make progress toward:

- Tracing the execution of methods to determine contents of memory. (Critical Thinking)

Facilitation Notes

This activity will likely take students longer than they think it will. Remind managers to keep track of the time and presenters to be ready to report out. The recorder (or other team member) could read questions out loud to help with pacing.

Model 1 includes only void methods with no parameters, so that students may focus on the flow of execution. For #1 and #2, do a quick check of each team's answer before they get too far along. Some misread or misinterpret the question and/or don't get the difference between analyzing the code statically (i.e., how many lines) vs dynamically (i.e., how many times).

Model 2 introduces a number of terms. #14 and #15 are good for reporting out to reinforce the concepts of ***signature***, ***parameters***, and ***arguments***. In addition, they address a misconception: students often misuse `printf` by concatenating the arguments to the format string.

Model 3 builds on a previous technique for drawing diagrams and memory traces (from the Data Types activity). You might need to remind students that memory diagrams simply include a small box to the right of each variable name. Have multiple teams draw their diagram for #20 on the board to compare different ways students think about this question.

Key questions: #7, #10, #20

Source files: *Print.java*, *CtoF.java*, *Stack.java*

Model 1 Invoke and Return

Each statement in this program ***invokes*** (or calls) a method. At the end of a method, Java ***returns*** to where it was invoked. The list of events on the right illustrates how the program runs.

```
1  public class Print {
2
3      public static void main(String[] args) {
4          System.out.println("First line.");
5          threeLine();
6          System.out.println("Second line.");
7      }
8
9      public static void newLine() {
10         System.out.println();
11     }
12
13     public static void threeLine() {
14         newLine();
15         newLine();
16         newLine();
17     }
18
19 }
```

```
INVOKE println
RETURN to line 5
INVOKE threeLine
    INVOKE newLine
        INVOKE println
        RETURN to line 11
    RETURN to line 15
    INVOKE newLine
        INVOKE println
        RETURN to line 11
    RETURN to line 16
    INVOKE newLine
        INVOKE println
        RETURN to line 11
    RETURN to line 17
RETURN to line 6
INVOKE println
RETURN to line 7
```

Questions (10 min) Start time:

1. How many lines of source code invoke the `println` method? Three (lines 4, 6, 10)

2. How many times is `println` invoked when the program runs? Five times

3. For each `INVOKE` on the right, draw an arrow to the corresponding line of code. (Plan ahead or use different colors so that crossing lines will be legible.) There should be 9 lines total.

4. What is the output of the program? Please write `\n` to show each newline character.

```
First line.\n
\n
\n
\n
Second line.\n
```

5. When the Java compiler sees a name like `x`, `count`, or `newLine`, how can it tell whether it's a variable or a method? (Hint: syntax)

Methods have parentheses, and variables do not. Methods are similar to functions in math: when you see $g(x)$, you know that g is a function and x is a variable.

6. What is the difference between a method and a variable?

All computer programs, regardless of language, consist of ***code*** and ***data***. Methods contain code (statements or instructions), whereas variables contain data (references or values).

7. In your own words, describe what methods are for. Why not just write everything in `main`?

Methods help organize the code into separate parts. They also make it possible to write code once and use it multiple times.

Model 2 Math Methods

Consider the following methods defined in the `Math` class. (This list isn't exhaustive; the `Math` class has over 90 methods in total!)

Modifier and Type	Method and Description
`static int`	**`abs`**`(int a)` Returns the absolute value of an `int` value.
`static double`	**`log`**`(double a)` Returns the natural logarithm (base *e*) of a `double` value.
`static double`	**`pow`**`(double a, double b)` Returns the value of the first argument raised to the power of the second argument.
`static double`	**`random`**`()` Returns a `double` value with a positive sign, greater than or equal to `0.0` and less than `1.0`.
`static int`	**`subtractExact`**`(int x, int y)` Returns the difference of the arguments, throwing an exception if the result overflows an `int`.

The code for these methods is provided in a source file named *Math.java*. Here is what the definition of the `abs` method looks like:

```
public static int abs(int a) {
    // code omitted
}
```

To use a method from another source file (like *Math.java*), you must first specify the class name:

```
value = abs(-5);       // Error: cannot find symbol
value = Math.abs(-5);  // correct
```

Questions (20 min) Start time:

8. What type of value does `Math.random()` return? Give an example of what a random value might look like.

It returns a random `double` value greater than or equal to 0.0 and less than 1.0. For example, the value 0.7851637186510342.

9. When *defining* a method (like `abs` or `log`), what do you need to specify before the method name and after the method name?

Before the name, you need to specify what type of value the method will return. After the name, you need to specify what values are required to use the method.

10. Define a method named `average` that takes two integers named `x` and `y` and returns a `double`. Don't write any semicolons or braces.

```
public static double average(int x, int y)
```

11. When *using* a method, what do you need to specify before the method name and after the method name?

Before the name, you need to specify the class name (unless the method is in the same class). After the name, you need to specify the required values.

12. For each method in Model 2, write a Java statement that uses the method and assigns the result to a variable.

```
answer = Math.abs(-5);
length = Math.log(1.2);
amount = Math.pow(3.4, 5.6);
number = Math.random();
result = Math.subtractExact(-78, 90);
```

*What you wrote for Question #10 is called the method's **signature**. The variables declared inside the parentheses are called **parameters**. When invoking a method, the values you provide are called **arguments**. Since arguments will be assigned to parameters, their types must be compatible.*

13. In the table below, how many parameters and arguments does each method have? What is the relationship between the last two columns?

Method	# Params	# Args
`abs`	1	1
`log`	1	1
`pow`	2	2
`random`	0	0
`subtractExact`	2	2

14. Consider the statement `System.out.println("Price: " + price);` where the value of `price` is 9.99. What is the argument that `println` receives?

```
"Price: 9.99"
```

15. Consider the statement `System.out.printf("Price: %f", price);` where the value of `price` is 9.99. Why does `println` use *plus* and `printf` use *comma* to specify the arguments?

The `println` method requires only one argument, so plus in this context means concatenate. The `printf` method requires multiple arguments: one for the format string, and others for the values to substitute.

IMPORTANT: Never use + (string concatenation) with printf. You might accidentally add values to the format string itself, rather than substitute them.

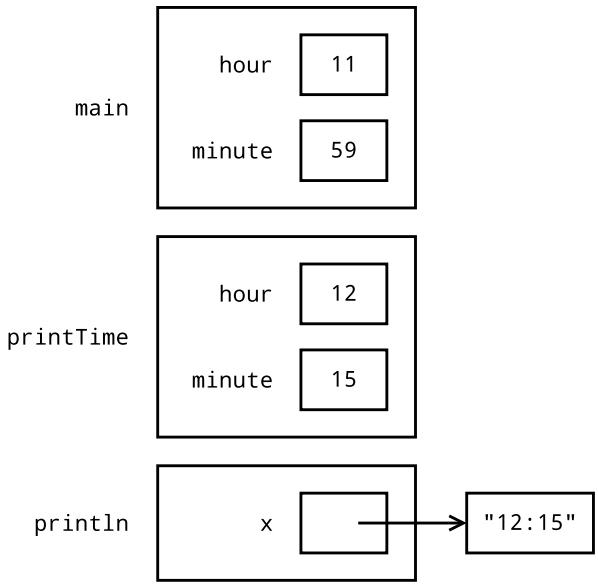
Model 3 Stack Diagrams

The following program displays the time `12:15` on the screen:

```
public static void printTime(int hour, int minute) {
    System.out.println(hour + ":" + minute);
}

public static void main(String[] args) {
    int hour = 11;
    int minute = 59;
    printTime(12, 15);
}
```

We can draw a ***stack diagram*** to visualize how the program works. Each method has its own area of memory to store parameters and other variables. For convenience, we stack method calls underneath each other (upside down).



Questions (15 min) Start time:

16. Based on the diagram, how many methods does the program use? Three

17. Based on the diagram, how many variables does the program have? Five

18. How are stack diagrams different from the memory diagrams we've seen previously?

There is a box for each method call, with the name of the method on the left.

Notice that reference types (like strings) are not stored "on the stack" — the variables are part of the method, but the actual data is stored outside of the method.

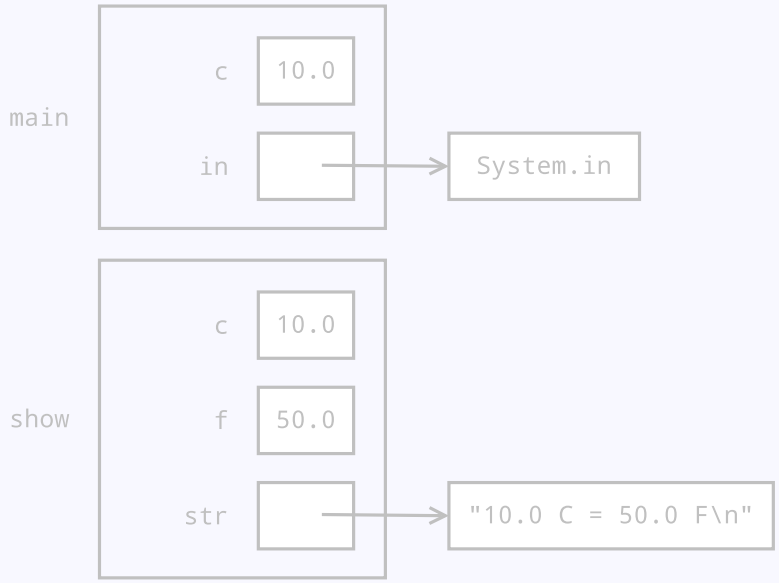
19. How is it possible that two variables with the same name can have different values?

Each method declares its own variables. Because they are stored in different memory locations, the values are independent from other methods.

20. Draw a stack diagram to show the state of memory just before `println` is called. Assume the user inputs the value 10. (You should be able to do this kind of math without a calculator.)

```
public static void show(double c) {
    double f;
    String str;
    f = c * 1.8 + 32;
    str = String.format("%.1f C = %.1f F\n", c, f);
    System.out.println(str);
}

public static void main(String[] args) {
    double c;
    Scanner in = new Scanner(System.in);
    System.out.print("Enter temperature in Celsius: ");
    c = in.nextDouble();
    show(c);
}
```



21. What is the difference between the `String.format` method (used in the previous question) and `System.out.printf`?

They both substitute values based on format specifiers. `String.format` returns a new string, whereas `System.out.printf` displays it on the screen. (In fact, `System.out.printf` invokes the `String.format` method.)