Model 1 Factorial

”In mathematics, the ***factorial*** of a non-negative integer n, denoted by $n!$, is the product of all positive integers less than or equal to n. For example, $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$.”

Source: https://en.wikipedia.org/wiki/Factorial

n	$n!$
0	1
1	1
2	2
3	6
4	24
5	120

Questions (25 min) Start time:

1. Consider how to calculate $4! = 24$.

a) Write out all the numbers that need to be multiplied:

$4! =$ 4 * 3 * 2 * 1

b) Rewrite the expression using 3! instead of $3 \times 2 \times 1$:

$4! =$ 4 * 3!

2. Write an expression similar to #1b showing how each factorial can be calculated in terms of a simpler factorial.

a) $3! =$ 3 * 2!

b) $2! =$ 2 * 1!

c) $100! =$ 100 * 99!

d) $n! =$ $n * (n - 1)!$

3. What is the value of 0! based on Model 1? Does it make sense to define 0! in terms of a simpler factorial? Why or why not?

0! is 1 by convention for an empty product. We can’t say $0 \times -1!$, because factorial is only defined for non-negative integers. At some point we need to define the solution in concrete terms, without referencing itself.

If we repeatedly break down a problem into smaller versions of itself, we eventually reach a basic problem that can't be broken down any further. Such a problem, like 0!, is referred to as the ***base case****.*

4. Assume you already have a working method named `factorial(int n)` that returns $n!$ for any positive integer.

a) Review your answer to #2c that shows how to compute 100! using a simpler factorial. Convert this expression to Java by using the `factorial` method instead of the `!` operator.

```
100 * factorial(99)
```

b) Now rewrite your answer to #2d in Java using the variable `n`.

```
n * factorial(n - 1)
```

5. Here is a `factorial` method that includes output for debugging:

```
1  public static int factorial(int n) {
2      System.out.println("n is " + n);
3      if (n == 0) {
4          return 1;  // base case
5      } else {
6          System.out.printf("need factorial of %d\n", n - 1);
7          int answer = factorial(n - 1);
8          System.out.printf("factorial of %d is %d\n", n - 1, answer);
9          return n * answer;
10     }
11 }
12
13 public static void main(String[] args) {
14     System.out.println(factorial(3));
15 }
```

a) What specific method is invoked on line 7?

The `factorial` method invokes itself (with a smaller argument).

b) Why is the `if` statement required on line 3?

Without the base case, it would invoke itself forever (until running out of memory).

6. A method that invokes itself is called **recursive**. What two steps were necessary to define the `factorial` method? How were these steps implemented in Java?

1. The base case, which was implemented using an if statement.
2. The recursive case, which as implemented using a method call.

7. How many distinct method calls would be made to `factorial` to compute the factorial of 3? Identify the value of the parameter n for each of these separate calls.

Four method calls: `factorial(3)` → `factorial(2)` → `factorial(1)` → `factorial(0)`.

8. Here is the complete output from the program in #5. Identify which distinct method call printed each line. In other words, which lines were printed by `factorial(3)`, which lines were printed by `factorial(2)`, and so on.

```
n is 3                 factorial(3)
need factorial of 2    factorial(3)
n is 2                 factorial(2)
need factorial of 1    factorial(2)
n is 1                 factorial(1)
need factorial of 0    factorial(1)
n is 0                 factorial(0)
factorial of 0 is 1    factorial(1)
factorial of 1 is 1    factorial(2)
factorial of 2 is 2    factorial(3)
6                      main
```

9. What happens if you try to calculate the factorial of a negative number? How could you prevent this behavior in the `factorial` method?

The recursion would repeat until the program runs out of memory (`StackOverflowError`). To fix this bug, you could add an if statement that checks for `n < 0` and returns -1.

10. Trivia question: What is the largest factorial you can compute in Java when using `int` as the data type? If you don't know, how could you find out?

$12! = 479,001,600$. Anything larger exceeds the 32-bit range. 20! is the largest for `long` integers. You can find this out by trail and error (e.g., using JShell).