Activity 6: Recursion

Sometimes when solving a problem, we can use the solution of a simpler version of the *same* problem. Eventually we reach the most basic problem, for which the answer is trivial.

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 (20 min)   **Start time:** _____

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

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

      $4! =$

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

      $4! =$

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

   a) $3! =$

   b) $2! =$

   c) $100! =$

   d) $n! =$

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

> *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.

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

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

```java
public static int factorial(int n) {
    System.out.println("n is " + n);
    if (n == 0) {
        return 1;  // base case
    } else {
        System.out.printf("need factorial of %d\n", n - 1);
        int answer = factorial(n - 1);
        System.out.printf("factorial of %d is %d\n", n - 1, answer);
        return n * answer;
    }
}

public static void main(String[] args) {
    System.out.println(factorial(3));
}
```

   a) What specific method is invoked on line 7?

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

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

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.

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
need factorial of 2
n is 2
need factorial of 1
n is 1
need factorial of 0
n is 0
factorial of 0 is 1
factorial of 1 is 1
factorial of 2 is 2
6
```

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

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?

Model 2   Summation

"In mathematics, *summation* (capital Greek sigma symbol: Σ) is the addition of a sequence of numbers; the result is their sum or total."

$$\sum_{i=1}^{100} i = 1 + 2 + 3 + \ldots + 100 = 5050$$

Questions (20 min)   Start time: _____

11. Consider how to calculate $\sum_{i=1}^{4} i = 10$.

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

$$\sum_{i=1}^{4} i =$$

b) Showing how this sum can be calculated in terms of a smaller summation.

$$\sum_{i=1}^{4} i =$$

12.   Write an expression similar to #11b showing how any summation of n integers can be calculated in terms of a smaller summation.

$$\sum_{i=1}^{n} i =$$

13. What is the base case of the summation? (Write the complete formula, not just the value.)

14. Implement a recursive method `summation` that takes a single parameter `n` and returns the sum $1 + 2 + \ldots + n$. It should only have an `if` statement and two `return` statements.

15. Discuss how the `factorial` method below uses temporary variables. What lines would you have to change to implement the `summation` method instead?

```java
public static int factorial(int n) {
    if (n == 0) {
        return 1;   // base case
    }
    int recurse = factorial(n - 1);
    int result = n * recurse;
    return result;
}
```

16. Here is a stack diagram of `factorial(3)` when invoked from `main`. Draw a similar diagram for `summation(3)` as described in the previous question.
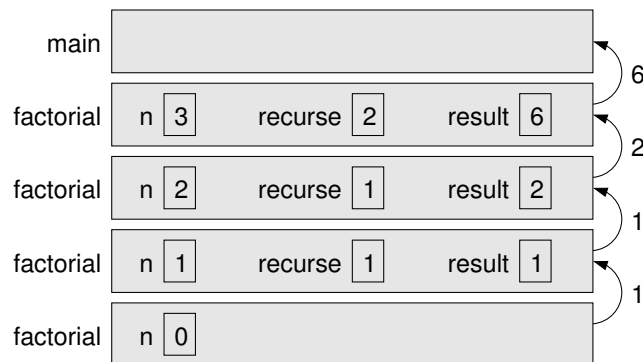
17. Why are there no values for `recurse` and `result` in the stack diagram for the last call to `factorial` (when n == 0)?

18. Looking at the stack diagram, how is it possible that the parameter `n` can have multiple values in memory at the same time?