2D Arrays and Recursion

Today, we start with 2D arrays (a natural extension of 1D arrays). We then explore recursion, a powerful technique for solving problems by breaking them down into smaller subproblems. Today's coverage corresponds to the MBL must-have skill MH11: Recursion Level 1. (We will revisit recursion later in the course, leading to IMP04: Recursion Level 2.)
We'll work in teams for Model 1, do Model 2 together as a class, then work in pairs for Model 3.

Manager:

Recorder:

Presenter:

Reflector:

Content Learning Targets

After completing this activity, you should be able to say:

- I can create and use two-dimensional (2D) arrays in Java.
- I can explain the concept of recursion and identify base and recursive cases.
- I can trace the execution of a recursive method using stack frames.
- I can solve small problems using recursion.

Process Skill Goals

During the activity, you should make progress toward:

- Visualizing 2D arrays as grids and understanding their structure. (Critical Thinking)
- Visualizing and tracing recursive method calls using stack frames. (Critical Thinking)

Model 1 2D Arrays

We have worked with one-dimensional (1D) arrays, which are like lists of values. Java also lets us create two-dimensional (2D) arrays, which are like ***grids*** of values. See the ***slides*** folder for supplemental resources. Here's a first example of creating and using a 2D array:

```
public class Array2DExample {
    public static void main(String[] args) {
        // Create a 2D array with 3 rows and 4 columns
        int[][] grid = new int[3][4];

        // Fill the array with values
        for (int row = 0; row < grid.length; row++) {
            for (int col = 0; col < grid[row].length; col++) {
                grid[row][col] = row * col;
            }
        }

        // Print the array
        for (int row = 0; row < grid.length; row++) {
            for (int col = 0; col < grid[row].length; col++) {
                System.out.print(grid[row][col] + "\t");
            }
            System.out.println();
        }
    }
}
```

Questions (30 min) Start time:

1. In the above example, notice how we declare and instantiate a 2D array. What are the similarities and differences compared to a 1D array?

2. How do we access elements in a 2D array? How does this compare to accessing elements in a 1D array?

3. Notice how we can use nested loops to iterate over all elements in the 2D array. What does `grid.length` represent? What about `grid[row].length`?

4. Predict the output of running the above program. Then, run it (***src/Array2DExample.java***) to check your prediction.

5. Modify the program to produce this 6 × 3 array. (**Hint**: what's the pattern?)

```
0 -1 -2
1 0 -1
2 1 0
3 2 1
4 3 2
5 4 3
```

6. **Split into pairs** and use pair programming to work through ***src/Practice2DArrayProblems.java***. After each method, switch navigator/driver roles.

Model 2 Introduction to Recursion

Recursion is a technique in which a method calls itself. At first, that might seem silly or inefficient. What's the point? By modifying inputs in the ***recursive*** call of itself, the method solves a smaller ***subproblem***, then uses that result to solve the original problem.
We will start with a short lecture and then have time for practice in pairs.

Questions (20 min) Start time:

7. Open the ***Triangle*** class in ***src/firstRecursion***. This class represents a triangular tower built from square blocks. The `width` field is the number of blocks in the bottom row.

8. Suppose we define a mathematical function/sequence $A(n)$ that denotes the number of blocks (or equivalently, the front-facing surface area) in a triangular tower of width n, for $n = 0, 1, 2, 3, \ldots$. What are the first six values of $A(n)$?

9. We can define $A(n)$ recursively as follows:

$$A(n) = \begin{cases} 0 & \text{if } n = 0 \\ n + A(n-1) & \text{if } n > 0 \end{cases}$$

What is the ***base case*** in this definition? What is the ***recursive case***?

10. Compare the above recursive definition to the `computeAreaRecursively(int inputWidth)` method in the `Triangle` class. How do they correspond?

11. Trace the execution of `computeAreaRecursively(3)`. What calls are made, and what values are returned at each step?

12. If time allows, we will work together through the recursion example in the ***Inception*** class. Note how the recursive call stack appeared in IntelliJ, and how we used the debugger to step through the calls.

Model 3 Recursion Practice

In pairs, work through additional recursion examples and problems.

- ***FactorialCalculator.java*** shows how to compute the factorial of a number both iteratively and recursively.
- ***SimplePalindrome.java*** has two recursion problems: one for checking if a String is a palindrome (i.e., the same forwards and backwards), and one for checking if a list of integers is a palindrome.
- CodingBat has many short recursion problems for practice: https://codingbat.com/java/Recursion-1. Try solving `bunnyEars`, `bunnyEars2`, `count7`, `fibonacci`, and `noX`. For solutions, see the `recursion.pptx` slides.
- For an advanced challenge combining recursion and Swing graphics, implement the Koch snowflake in ***src/fractals***.