

# Act21: Recursion Revisited

Today, we look at recursion again, but this time with a focus on how to design *recursive helper methods* when we need additional parameters to keep track of state.

Our next homework assignment will offer a chance to practice with recursive helper methods. This will be an important skill for those of you continuing to CSSE 230 (and eventually CSSE 304, which uses a language called Scheme which is specifically designed to prioritize recursion).

## Content Learning Targets

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

- I can explain how recursive helper methods work, including base case(s) and recursive case(s).
- I can identify when a recursive helper method is needed.
- I can design and implement a recursive helper method to solve a problem.

## Process Skill Goals

*During the activity, you should make progress toward:*

- N/A

## Facilitation Notes

### First Hour:

- Brief recursion review
- Model problems involving recursion with helper methods
- Practice problems: not clear whether helper methods are needed, and if so, how to design them

### Second Hour: Project work time.



Copyright © 2025 Ian Ludden, based on [prior work](#) of Mayfield et al. This work is under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

## Model 1 Recursion Review

As we learned in Act12, recursion is a technique in which a method calls itself to solve a smaller subproblem, then uses the result of to help solve the original problem. Some key terms:

**Base case:** A condition under which the method can return a result without making any further recursive calls. This prevents infinite recursion (remember `StackOverflowError`?).

**Recursive case:** The part of the method that makes a recursive call to itself with modified arguments that bring it closer to the base case.

**Recursive call stack:** Each time a method is called (including recursive calls), a new *frame* is added to the call stack. Each frame contains its own parameters and local variables. When a method returns, its frame leaves the stack, and control returns to the previous frame.

## Model 2 Recursive Helper Methods

Sometimes, a method just isn't set up to support straightforward recursion.

1. **Live coding:** In `RecursiveHelperFunctions.java`, let's solve `sumWholeArray(int[] array)`.
2. How does adding a second method, called a *recursive helper method*, help us here?

A helper method can have additional parameters to keep track of state needed for the recursion (in this case, the current index in the array). The public method can call the private helper method with appropriate initial values for those parameters.

3. In pairs, practice designing recursive helper methods for the remaining problems.
4. Why is it best practice to make recursive helper methods `private`?

Helper methods are implementation details that users of the class don't need to know about. Making them `private` encapsulates them and prevents accidental misuse. It keeps the publicly available methods (also called the class's API, or Application Programming Interface) clean and easy to understand.

## Model 3 Project Work Time

Continue working toward your Milestone 1 goals. A few reminders:

- Update UML design diagrams as needed.
- Revise your plan for milestones M2-M4 as needed based on any design changes.
- Clean up code for the M1 commit and clearly label it "Final M1 commit" or similar.
- Push all changes to the repository. Confirm your latest commit is visible in GitHub.