# Overview

In this lab, you will use recursive backtracking to solve a simple puzzle-type problem in which you have to assign values from a given list of numbers to each index in an array. The values you assign must follow the rule that to be assigned as the **n**th item in the array, the value must be a multiple of **n**. For example, the first item (index 0) must be a multiple of 1, the second item (index 1) must be a multiple of 2, the third item a multiple of 3, and so on. For this task, you will need to write some additional helper methods. For the purpose of testing, all methods should be public.

# Objectives

* Practice creating a class
* Writing helper methods for a larger task
* Using recursive backtracking to solve a problem

# Steps

1. Create a folder on your local machine for your Java program, you can name it whatever you like
2. Start Visual Studio Code (VS Code)
3. In VS Code, Open that newly created folder.
4. Download the starter code (**Multiples.java**) from the course public folder ([public/14L](https://cs.unh.edu/~cs416/public/14L)) and save it in the src directory of your new project.
5. Add a method named isMultiple which takes two integer parameters, num and div. It should return true if num is a multiple of div, and false otherwise. The mod operator (%) will likely be helpful in writing this method.
6. Add a method named assignValue which takes two integer parameters, n and i. It should get the value at index i from numbers, and assign it to index n in result. It should then remove the value at index i from numbers. This will allow your program to only use a number once when finding a solution to the problem.
7. Add a method named unassignValue which takes a single integer parameter, n. It should add the value at index n from results back into numbers (at index 0.) Then it should assign 0 to index n in result to clear out the value from the results. This will allow your program to "undo" a choice it made when it needs to backtrack and try other possibilities.
8. Finally, you must now work on the recursive backtracking method solve( int n ).  
   1. First, add a base case to the method. Because it will start at index 0 and assign values before moving on to the next index, the method will assume that any index before n has already been given a value in a previous call to the method. So your base case will be when n is past the last valid index for result, and should return true to indicate that the problem was solved.
   2. Next, you need to handle the recursive case. **For each number** in numbers, the method should check if that number is valid for index n (meaning it is a multiple of n+1). If it is valid, then it should assign that value at index n. Then recursively call the method to solve the rest of the problem, but make sure to store the return value of that recursive call in a boolean variable. If the recursive call returned true, this means the rest of the problem was solved by the recursive call, and no more solving is needed (the method can return true.) For now, have the method return false if the recursive call returned false. **If you run or submit your code at this point, you will see that it solves the problems which don't require backtracking.**
   3. To allow backtracking, we need the method not to quit when a single recursive call fails, so delete the part of the code that returns false when the recursive call returned false. Instead, what it should do is un-assign the value at n when the recursive call failed to find a solution (to undo the choice of value, because we found there was no solution possible with that choice of value.) The method should return false only if the for loop finishes without finding a solution, so return false outside the for loop (nowhere else!) **If you run or submit your code at this point, it should solve the problems which do require backtracking and correctly report when no solution is possible.**
9. Set a breakpoint in IntelliJ at the line in solve() where solve(0) is called. Run in debug mode. Step through the code (step into solve(0) and then use step over from then onward) and watch as it solves each of the problems (or discovers no solution is possible. This will give you a clearer idea of how the recursion finds solutions and how the backtracking allows it to try other options when it fails to find a solution.