Implementation In-Class Exercise

By: David Dunlop, Elijah Cox, Nikki Clark, Steven Mullins

(Team Pop-Quiz)

Prompt: "Using Recursion; take two numbers in from the user (a human) and add them together then separate the least significant digit and add it the remaining digits and so on until you have a single digit answer.

EX: 87345 => 8734+5= 8739 => 873+9 = 882 => 88 + 2 = 90 => 9+0 =9"

Brainstorming Notes: The following notes were taken during the initial brainstorming process to keep track of potential requirements, restrictions, and the potential flow of the process for the app. These were taken using the sticky notes provided at the beginning of the course, put into this document. These are not 100% representative of our final thoughts and decisions, but show an insight into our problem-solving process:

- The program must mathematically add the numbers provided by the player (instead of concatenating)
- We're restricting the valid values the player can provide to just integers, so the program should throw errors when the player enters something that's not an integer
- If during calculations the number in the recursion process ends up less than or equal to 9 and greater than or equal to -9, the program must stop the process and return the current value.
- The least significant digit, when separated, will *always* be positive.
- When the GUI introduces the player to the program, it must prompt the player for their first integer before asking for the second (not at the same time).
- During the initial addition of the input, if the two numbers just result in a single digit, simply return that value instead of going through the recursion process at all.
- There should be a main function that takes in the two digits and adds them, and a separate function that handles the recursion process. The

main function will call the separate function and pass its results through a variable.
    - The CLI GUI aspects can be handled through the main function, or a separate function can be made just for the CLI GUI aspects (which would be cleaner, but more complex)
- **Possible procedure for the recursion process**:
    - Process starts with the full number from the addition process.
    - An integer variable is created.
    - The full number has a modulo 10 calculations run on it, and its returning value is stored in the new variable.
    - The full number is then divided by 10, and has any possible resulting decimals discarded.
    - This number then has the integer variable added to it and becomes the new “full number”.
    - This process repeats until any of the return conditions are met.

<u>Pseudocode</u>:

```
Main
{
    intro text with instructions
    I1 = user input int #1
    I2 = user input int #2
     X = I2 + I1
     X = recursive(x)
 }

recursive (int x)
{
    if ( x ≤ 9 and x ≥ -9)
    {
      return x
    }
    \\isolate least significant digit
    LSD = I3 % 10
    \\add a decimal point before LSD
    X = X ÷ 10
    \\ Round down to remove LSD
Likely something like Math.Floor → X = RoundDown (X)
    \\Add LSD to x since LSD fully isolated
    X = X + LSD
     recursive (x)
}
```

Step 1.

- Determine constraints and rules.
  - Integer inputs only.
  - Least significant digit is always treated as a positive.
  - Recursion continues only while the value is greater than 9 or less than −9.
  - Overflow must be detected and prevented.
  - Invalid input must be handled gracefully

Step 2.

- Determine basic functionality.
  - Main Function
    - Display instructions to user
    - Handling user input
    - Summation of both inputs
    - Check if sum is a single digit
    - Output sum if sum is a single digit, else continue
    - Calls recursive function
    - Output final result to user
    - Validate integer input and reject non-integer values
    - Prevent and detect overflow.
  - Recursive Function
    - Single digit sum check (before recursion)
    - Separate LSD
    - Remove LSD from the integer
    - Add LSD to the remaining digits
    - Repeat the process until a single digit is reached.
    - Return final value to main

Step 3.

- Determine required operations.
  - Use modulo to isolate the least significant digit.
  - Use integer division to remove the least significant digit.
  - Add the LSD to the remaining digits to form a new value.

    - Compare the new value to the single-digit end condition.
    - Pass the new value into the next recursive call.
    - Track each step for output.

Step 4.

- Determine supporting logic.
    - GetIntInput Method
        - Prompt the user with a message.
        - Read the input as a string.
        - Attempt to convert the string to an integer.
        - Reject invalid input and display an error message.
        - Continue prompting until a valid integer is entered.
        - Return validated integer to Main.
    - Overflow Method
        - Check whether two positive integers produce a negative sum.
        - Check whether two negative integers produce a positive sum.
        - Return true if overflow occurred, false otherwise.
        - Allow Main to handle overflow by warning the user and stopping execution.

Step 5.

- Determine test cases and strategy.
    - Positive multi-digit numbers to verify recursion.
    - Negative numbers to confirm LSD handling remains positive.
    - Single-digit inputs to ensure recursion does not run unnecessarily.
    - Test overflow scenarios to confirm detection works correctly.
    - Invalid inputs (letters, symbols, empty strings) to verify graceful handling.
    - Edge cases such as 0, 10, −10, and very large integers.