String problems

Problem: Count the Vowels in a String

Objective:

Write a Java program that counts the number of vowels in a given string.

Requirements:

- The program should define a method `countVowels(String input)` that takes a string as input and returns the number of vowels (a, e, i, o, u) in the string.
- The method should be case-insensitive, meaning it should count both uppercase and lowercase vowels (e.g., 'A' and 'a').
- The program should also handle an empty string input by returning 0.

Example Usage:

- `countVowels("Hello World")` should return `3`.
- `countVowels("Java Programming")` should return `5`.
- `countVowels("Aeiou")` should return `5`.
- `countVowels("")` should return `0`.

Remove Vowels from a String

- **Objective:** Write a method that removes all vowels from a given string.
- **Requirements:**
  - The method `removeVowels(String input)` should return the string without any vowels.
- **Example Usage:**
  - `removeVowels("Hello World")` should return `"Hll Wrld"`.
  - `removeVowels("Java Programming")` should return `"Jv Prgrmmng"`.
- **Hints:**
  - Use a loop to iterate through the string and build a new string that excludes vowels.
  -

Find the Longest Word in a String

- **Objective:** Write a method that finds and returns the longest word in a string.
- **Requirements:**
  - The method `findLongestWord(String input)` should return the longest word in the string.
  - If there are multiple words of the same length, return the first one.

- **Example Usage:**
  - `findLongestWord("I love programming in Java")` should return `"programming"`.
  - `findLongestWord("The quick brown fox jumps over the lazy dog")` should return `"jumps"`.
- **Hints:**
  - Use the `split(" ")` method to split the string into words and then iterate through to find the longest one.

Capitalize the First Letter of Each Word

- **Objective:** Write a method that capitalizes the first letter of each word in a string.
- **Requirements:**
  - The method `capitalizeWords(String input)` should return the string with each word's first letter capitalized.
- **Example Usage:**
  - `capitalizeWords("hello world")` should return `"Hello World"`.
  - `capitalizeWords("java programming language")` should return `"Java Programming Language"`.
- **Hints:**
  - Split the string into words, capitalize each word, and then join them back together.
-

Array problems

Check if an Array is Sorted

- **Objective:** Write a method that checks if an array is sorted in ascending order.
- **Requirements:**
  - The method `isSorted(int[] array)` should return `true` if the array is sorted and `false` otherwise.
- **Example Usage:**
  - `isSorted(new int[]{1, 2, 3, 4})` should return `true`.
  - `isSorted(new int[]{4, 3, 2, 1})` should return `false`.
- **Hints:**
  - Compare each element with the next one to ensure the order is non-decreasing.

Merge Two Arrays

- **Objective:** Write a method that merges two arrays into one.
- **Requirements:**
  - The method `mergeArrays(int[] array1, int[] array2)` should return a new array containing all elements of `array1` followed by all elements of `array2`.
- **Example Usage:**
  - `mergeArrays(new int[]{1, 2}, new int[]{3, 4})` should return `[1, 2, 3, 4]`.
  - `mergeArrays(new int[]{10, 20}, new int[]{30})` should return `[10, 20, 30]`.
- **Hints:**
  - Create a new array of the combined length and copy elements from both arrays into it.

Find the Duplicate Elements in an Array

- **Objective:** Write a method that finds and returns any duplicate elements in an array.
- **Requirements:**
  - The method `findDuplicates(int[] array)` should return an array of duplicates found in the input array.
- **Example Usage:**
  - `findDuplicates(new int[]{1, 2, 3, 2, 4, 3})` should return `[2, 3]`.
  - `findDuplicates(new int[]{5, 5, 5, 5})` should return `[5]`.
- **Hints:**
  - Use nested loops to compare each element with every other element.

Shift Elements in an Array

- **Objective:** Write a method that shifts all elements of an array to the right by a specified number of positions.
- **Requirements:**
  - The method `shiftArray(int[] array, int positions)` should shift the array elements and handle wrap-around.
- **Example Usage:**
  - `shiftArray(new int[]{1, 2, 3, 4}, 2)` should modify the array to `[3, 4, 1, 2]`.
  - `shiftArray(new int[]{10, 20, 30}, 1)` should modify the array to `[30, 10, 20]`.
- **Hints:**
  - Consider how to handle the wrap-around when shifting elements.

Find the Smallest and Largest Elements in an Array

- **Objective:** Write a method that finds both the smallest and largest elements in an array.
- **Requirements:**
  - The method `findMinMax(int[] array)` should return an array where the first element is the smallest and the second element is the largest.
- **Example Usage:**
  - `findMinMax(new int[]{1, 2, 3, 4, 5})` should return `[1, 5]`.
  - `findMinMax(new int[]{10, -3, 7, 2})` should return `[-3, 10]`.
- **Hints:**
  - Iterate through the array, keeping track of the smallest and largest values encountered.

Simple Calculator with String Expressions and Memory Operations Using Arrays

1. Basic Arithmetic Operations with String Expressions

- **Objective:** Implement basic arithmetic operations using a single string expression that includes numbers and operators.
- **Requirements:**
  - Implement a method `calculate(String expression)` that takes a string expression like `"3.5 + 2.1"` and returns the result.
  - The method should:
    - Parse the string to identify the numbers and the operator.
    - Perform the corresponding arithmetic operation based on the operator (`+`, `-`, `*`, `/`).
    - Return the result as a `double` or an appropriate error message if the operation is invalid.
  - Handle basic operations:
    - `calculate("3.5 + 2.1")` should return `5.6`.
    - `calculate("10 - 4")` should return `6.0`.
    - `calculate("6 * 7")` should return `42.0`.
    - `calculate("8 / 2")` should return `4.0`.
  - If the expression is invalid (e.g., `"10 / 0"`), return an error message like `"Division by zero is not allowed."` or `"Invalid expression."`.

2. Handling Multiple Operations in a Single String

- **Objective:** Allow the calculator to process a more complex string containing multiple arithmetic operations.
- **Requirements:**
  - Extend the `calculate(String expression)` method to handle expressions with multiple operations, like `"3 + 5 * 2 - 4 / 2"`.

- Parse the expression and break it down into individual numbers and operators.
- Handle operator precedence, ensuring that multiplication and division are performed before addition and subtraction.
- Example:
  - `calculate("3 + 5 * 2 - 4 / 2")` should return `10.0`.

3. Support for Parentheses

- **Objective:** Enhance the calculator to support operations involving parentheses.
- **Requirements:**
  - Modify the `calculate(String expression)` method to correctly evaluate expressions with parentheses, such as `"3 + (2 * 4) - 5"`.
  - Ensure that operations within parentheses are evaluated first.
  - Example:
    - `calculate("3 + (2 * 4) - 5")` should return `6.0`.

4. Advanced Mathematical Operations with String Expressions

- **Objective:** Extend the calculator to support advanced operations using string expressions.
- **Requirements:**
  - Implement support for functions like square root and power within string expressions.
  - Example:
    - `calculate("sqrt(16)")` should return `4.0`.
    - `calculate("2 ^ 3")` should return `8.0`.
    - Use `"^"` for exponentiation and `"sqrt"` for square root in the string expressions.

5. Memory Functionality

- **Objective:** Implement memory functions to store, recall, and clear values, including the ability to work with string expressions.
- **Requirements:**
  - Implement the following methods:
    - **Store a value in memory:** `storeInMemory(double value)`: Stores the provided `value` in a static variable.
    - **Store the result of an expression:** Modify the `calculate(String expression)` method to optionally store the result in memory if a specific command is given (e.g., `"M+"` at the end of the expression).
      - Example: `calculate("3 + 5 M+")` should calculate the result `8.0` and store it in memory.
    - **Recall the stored value:** `recallMemory()`: Returns the value stored in memory.
    - **Clear the stored value:** `clearMemory()`: Clears the value stored in memory.

- Implement a method `recallAllMemory()` if multiple memory slots are used:
  - Store up to 5 recent values in memory using an array, and return them as a string when requested.
  - Example: `recallAllMemory()` could return `"Stored values: 8.0, 15.5, 42.0"`.

6. Error Handling and Input Validation

- **Objective:** Ensure robust error handling and input validation for string expressions and memory operations.
- **Requirements:**
  - If the string contains invalid syntax (e.g., `"3 + * 2"`), return an error message like `"Invalid expression."`.
  - If the operation is not supported (e.g., `"2 ^ -3"`), return a message like `"Operation not supported."`.
  - For division by zero, square roots of negative numbers, or any invalid memory operation (e.g., recalling memory when empty), return appropriate error messages.
  - Example:
    - `calculate("10 / 0")` should return `"Division by zero is not allowed."`
    - `calculate("sqrt(-9)")` should return `"Square root of a negative number is not allowed."`
    - `recallMemory()` when memory is empty should return `"No value stored in memory."`

Example Usage

- `calculate("3 + 5")` → `8.0`
- `calculate("10 - 2 * 3")` → `4.0`
- `calculate("(10 + 2) * 3")` → `36.0`
- `calculate("sqrt(25) + 3 ^ 2")` → `14.0`
- `calculate("10 / 0")` → `"Division by zero is not allowed."`
- `calculate("3 + 5 M+")` → Stores `8.0` in memory
- `recallMemory()` → Returns the stored value, e.g., `8.0`
- `clearMemory()` → Clears the stored value

Summary

This enhanced **Simple Calculator** will:

- Use **String expressions** to perform operations, allowing users to input full arithmetic expressions like `"3 + 5 * 2"`.
- Handle **operator precedence** and **parentheses** to ensure correct calculation order.
- Incorporate **memory functionality**, allowing users to store, recall, and clear values, as well as perform operations that automatically store results.

- Provide **error handling** and **input validation** to manage incorrect or unsupported operations and ensure smooth operation.