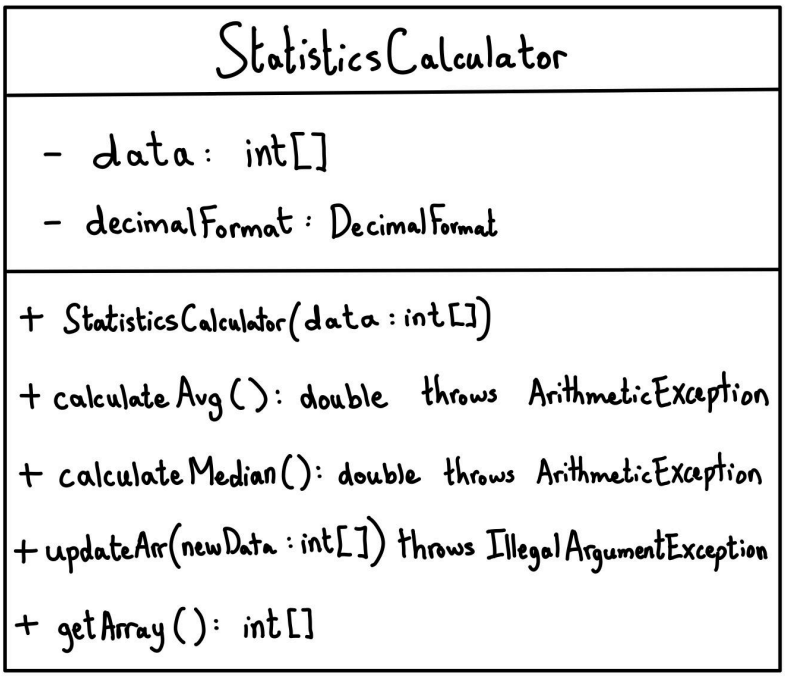
Assignment04-Q1:



Class:
The “StatisticsCalculator” class is responsible for performing statistical operations such as calculating the average and median on an array of integers, while handling invalid inputs and empty arrays through exception handling. It also provides methods to update and safely access the array without external modifications.

Attributes:
(- data : int[]): Private integer array stores the data that the StatisticsCalculator operates on, it’s also encapsulated to prevent external modifications.

(- decimalFormat : DecimalFormat): DecimalFormat object used for formatting double values to two decimal places.

Constructor:
(+ StatisticsCalculator(int[] data)): Constructor initializes the data array with the provided integer array. It validates that the input array is neither null nor empty, in which it throws an IllegalArgumentException if the input is invalid.

Methods:
(+ calculateAvg() : double): This method calculates the average of the elements in the data array and returns it as a double, formatted to two decimal places. It does so by iterating through each element, summing them up, and then dividing the total by the number of elements. The

result is formatted to two decimal places before being returned as a double. It throws an IllegalStateException if the data array is empty.

(+ calculateMedian() : double): This method calculates the median of the elements in the data array, in which it sorts the array prior to calculating the median. If the array has an odd number of elements, the median is the middle element. If there are an even number of elements, the median is the average of the two middle elements. The result is formatted to two decimal places before being returned. It also throws an IllegalStateException if the data array is empty.

(+ updateArr(newData : int[])): This method allows updating the data array with a new integer array. It validates the newData array to ensure it is not null and not empty, as these would be invalid inputs. If invalid, it throws an IllegalArgumentException and if it's valid, the data array is updated with the new values.

(+ getArray() : int[]): This method returns a copy of the current data array, preventing direct modification of the array from outside the class. This ensures that the integrity of the data array is maintained, allowing external code to access, but not modify it.

Assignment04-Q6:

In Q3, Q4, and Q5, I applied the divide-and-conquer principle by breaking down larger and more complex string processing tasks into smaller, more manageable methods.

When looking at the StringCleaner class, I divided the cleaning operations into separate methods, such that each method carried out a specific operation. For instance, removePunct removes punctuation, convertToLowerCase converts all characters to lowercase, and removeExtraSpaces removes any extra spaces in the given text. By giving each task a separate method and therefore encapsulating the operations, the code becomes modular, such that each method is independent and focused on a single purpose. This approach makes it easier to understand how different parts of the code work, and so locating bugs or making specific adjustments becomes a faster job to complete due to the code's readability.

Secondly, the StringAnalyzer class is a more advanced task that is divided into separate methods. Particularly, wordFrequency calculates the word frequency in the text, findLongestWord finds the longest word in the text, and isPalindrome checks for palindromes in the text. This way of separation allows each method to focus on a single responsibility, which enhances the code's readability and allows each function to be tested separately.

Lastly, in the TextAnalyzer class, the same principle is extended with additional functionalities. Specifically, it includes calculating the average word length and finding the most frequent letter in a given text. Each of these tasks is handled by specialized methods such as wordLenAndLetterFreq and alphabeticalOrder, which apply the divide-and-conquer principle as they divide each complex operation. This code structure is organized and modular as each

class focuses on specific aspects of text processing, where TextAnalyzer extends StringAnalyzer and StringAnalyzer imports StringCleaner.

Assignment04-Q7:

In Q3, Q4, and Q5, the open-closed principle is applied by creating separate classes for different string operations such as StringCleaner, StringAnalyzer, and TextAnalyzer, in which each class is responsible for a specific functionality. Instead of modifying existing classes, other classes are created to extend the functionality for the string operations. Specifically, StringAnalyzer extends StringCleaner's functionality by analyzing cleaned text, TextAnalyzer extends StringAnalyzer to add more advanced analysis features. Therefore, each class can be enhanced without changing internal code, making the system easier to maintain and extend. The benefit of this principle in this scenario is that each class remains focused on its core functionality. Following this modular structure makes it easy to add or adjust existing functionality without harming the program with errors or bugs. This structure improves code readability and reusability, allowing for new classes or methods to be added in the program with minimal risk of harming functionality.