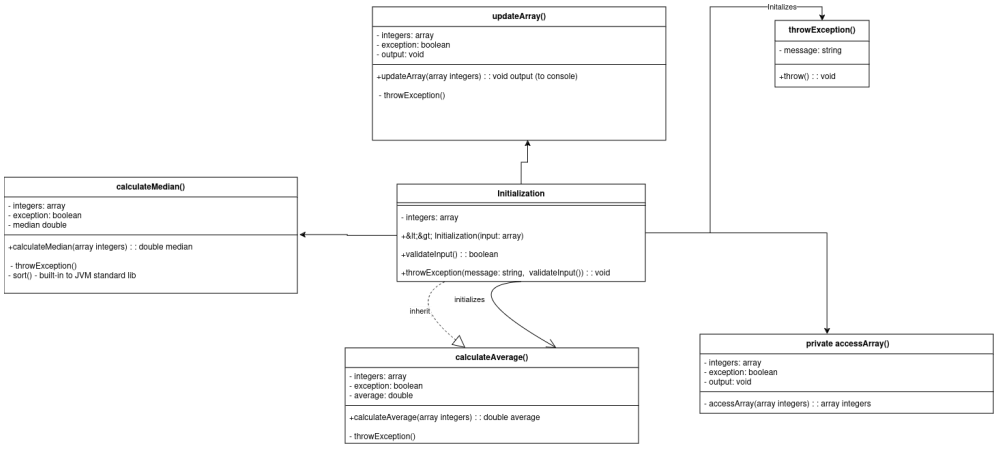
Assignment04-text

1. UML Diagram below



Question 6:

- **Encapsulation:** By keeping existing classes closed for modification, the internal implementation details are encapsulated. This promotes a clear separation of concerns and improves the overall design of the software.
- **Flexibility:** The open-closed principle allows for flexible and scalable designs. New features can be added with minimal impact on existing code, making it easier to adapt to changing requirements.
- **Maintainability:** When new functionality is needed, it can be added by extending existing classes rather than modifying them. This minimizes the risk of introducing bugs into existing code and makes the codebase easier to maintain.
- **Code Reusability:** By designing classes that are open for extension but closed for modification, existing code can be reused in new contexts without changes. This reduces duplication and promotes consistency.

Benefits of Applying the Open-Closed Principle:

1. **TextAnalyzer Class (Question 5):**

- The `TextAnalyzer` class further extended the functionality of `StringAnalyzer` by adding more advanced analysis methods. Again, instead of modifying the `StringAnalyzer` class, the `TextAnalyzer` class reused its methods and added new functionality. This allowed the `TextAnalyzer` class to build upon the existing functionality without altering the original classes.

2. **StringAnalyzer Class (Question 4):**
    - The `StringAnalyzer` class extended the functionality of `StringCleaner` by adding methods for string analysis. Instead of modifying the `StringCleaner` class, the `StringAnalyzer` class reused its methods and added new functionality. This followed the open-closed principle by extending the behavior of string processing without changing the original `StringCleaner` class.
3. **StringCleaner Class (Question 3):**
    - The `StringCleaner` class was designed to handle basic string preprocessing tasks. It was implemented in a way that allowed other classes to use its methods without modifying its source code. This made it easy to reuse the `StringCleaner` class in other contexts.

Application in Questions 3, 4, and 5:

The open-closed principle (OCP) states that software entities (such as classes, modules, and functions) should be open for extension but closed for modification. This means that the behavior of a module can be extended without modifying its source code, promoting code reusability and maintainability.

Question 7:

By breaking down the overall string analysis problem into smaller, focused methods, the code became more modular, easier to understand, and easier to maintain. Each method could be developed, tested, and debugged independently, leading to a more robust and flexible solution.

1. **TextAnalyzer Class (Question 5):**
    - The `TextAnalyzer` class further extended the functionality of `StringAnalyzer` by adding methods to calculate the average word length, find the most frequently occurring letter, and print words in alphabetical order. Each of

these tasks was implemented as a separate method (`calculateAverageWordLengthAndMostFrequentLetter`, `getWordsInAlphabeticalOrder`), following the divide-and-conquer approach.

2. **StringAnalyzer Class (Question 4):**
    - The `StringAnalyzer` class extended the functionality of `StringCleaner` by adding methods to analyze the cleaned string, such as calculating word frequencies, finding the longest word, and checking for palindromes. Each analysis task was implemented as a separate method (`getWordFrequencies`, `getLongestWord`, `isPalindrome`), allowing each task to be handled independently.
3. **StringCleaner Class (Question 3):**
    - The `StringCleaner` class was designed to handle basic string preprocessing tasks such as removing punctuation, converting to lowercase, and removing extra spaces. Each of these tasks was implemented as a separate method (`removePunctuation`, `convertToLowercase`, `removeExtraSpaces`), making the code modular and easier to manage.