1. **a String:** Write a program to reverse a given string without using built-in reverse functions.
2. **Palindrome Check:** Create a program that checks whether a given string is a palindrome or not (reads the same forwards and backwards).
3. **Anagram Check:** Write a program to check if two given strings are anagrams of each other (contain the same characters in a different order).
4. **Count Vowels and Consonants:** Develop a program that counts the number of vowels and consonants in a given string.
5. **String Compression:** Implement a program to compress a string by replacing consecutive repeating characters with the character followed by the count.
6. **Word Reversal:** Create a program that reverses the order of words in a given sentence.
7. **Longest Common Prefix:** Write a program to find the longest common prefix among an array of strings.
8. **Substring Frequency:** Implement a program to count the frequency of a given substring in a larger string.
9. **String Rotation:** Develop a program that checks whether one string is a rotation of another string.
10. **Remove Duplicate Characters:** Write a program that removes duplicate characters from a given string while maintaining the original order.
11. **String to Integer Conversion:** Create a program that converts a string containing digits into an integer.
12. **String Tokenization:** Implement a program that takes a sentence and breaks it into individual words using StringTokenizer or split function.
13. **Most Frequent Word:** Write a program to find the most frequent word in a given text.
14. **String Formatting:** Develop a program that takes user input for a name and formats it as "Last Name, First Name Middle Name".
15. **String Encryption:** Create a program that encrypts a string using a simple Caesar cipher (shifting characters by a fixed number of positions).
16.

    String Concatenation: Write a program to concatenate two strings and display the result.
17. **String Length:** Develop a program to calculate and display the length of a given string.
18. **Substring Extraction:** Create a program that extracts a substring from a given string based on start and end indexes.
19. **String Comparison:** Write a program that compares two strings for equality using the `equals()` method.
20. **String Case Conversion:** Implement a program that converts a given string to uppercase and lowercase.

21. **Trimming Whitespace:** Create a program that removes leading and trailing whitespace from a given string.
22. **Finding Characters:** Write a program that finds the index of a specific character within a given string.
23. **Replacing Characters:** Develop a program that replaces a specific character with another character in a given string.
24. **String Splitting:** Implement a program that splits a given string into an array of substrings based on a delimiter.
25. **Counting Occurrences:** Create a program that counts the occurrences of a specific character in a given string.
26. **String Formatting:** Write a program that formats and prints out a message using placeholders.
27. **String Conversion:** Develop a program that converts an integer to a string using the `Integer.toString()` method.
28. **Substring Check:** Implement a program that checks if a given substring exists within a larger string.
29. **String Empty Check:** Write a program that checks whether a given string is empty or not using the `isEmpty()` method.
30. **String Concatenation with Other Types:** Create a program that demonstrates concatenation of strings with other data types (e.g., numbers).
31. **Regular Expression Validation:** Write a program that validates if a given string matches a specific regular expression pattern (e.g., email, phone number).
32. **Levenshtein Distance Calculation:** Implement a program that calculates the Levenshtein distance between two strings (minimum number of edit operations to transform one string into another).
33. **Longest Palindromic Substring:** Create a program that finds the longest palindromic substring within a given string.
34. **String Permutations:** Write a program that generates all possible permutations of characters in a given string.
35. **Rabin-Karp Substring Search:** Implement the Rabin-Karp algorithm to efficiently search for a substring within a larger string.
36. **String Compression (Advanced):** Develop a program that performs more efficient string compression, such as using run-length encoding.
37. **Edit Distance and Alignment:** Create a program that computes the edit distance between two strings and outputs the aligned versions with edit operations marked.
38. **Longest Common Subsequence:** Write a program that finds the longest common subsequence between two strings.
39. **String Matching Algorithms:** Implement advanced string matching algorithms like KMP or Boyer-Moore to efficiently find occurrences of a substring in a larger string.
40. **Regular Expression Replacement:** Develop a program that performs replacement in a string based on a regular expression pattern.

41. **Unicode Manipulation:** Write a program that handles Unicode characters and performs operations like extracting code points and counting characters.
42. **String Interning and Memory Management:** Create a program that demonstrates string interning and explores memory management aspects of strings in Java.
43. **String Compression (Huffman Coding):** Implement a more advanced form of string compression using Huffman coding.
44. **Concurrent String Processing:** Develop a program that processes multiple strings concurrently using Java's multithreading or CompletableFuture.
45. **Custom String Class:** Create a custom string class that provides additional functionality beyond the standard Java String class, such as custom substring extraction or transformation methods.

Prepared By : Shiva Yannam