# LAB 6: STRING AND COLLECTION

## Exercise 1:

Create a class called EmailValidator with a method called validateEmail(String email) that checks if an email address is valid. Implement the validation logic using regular expressions and String operations. Test the EmailValidator class by creating an instance, calling the validateEmail method with different email addresses, and checking the validation results.

## Exercise 2:

Create a class called TextFormatter with a method called formatText(String text) that formats a given text by removing excess whitespace and converting it to lowercase. Implement the formatting logic using String operations. Test the TextFormatter class by creating an instance, calling the formatText method with different texts, and observing the formatted output.

## Exercise 3:

Create a class called NameInitialsGenerator with a method called generateInitials(String fullName) that generates the initials of a given full name. Implement the logic to extract initials using String operations and StringBuilder. Test the NameInitialsGenerator class by creating an instance, calling the generateInitials method with different full names, and checking the generated initials.

## Exercise 4:

Create a class called PalindromeChecker with a method called isPalindrome(String word) that checks if a given word is a palindrome (reads the same forward and backward). Implement the palindrome checking logic using String operations and StringBuilder. Test the PalindromeChecker class by creating an instance, calling the isPalindrome method with different words, and checking the palindrome results.

## Exercise 5:

Create a class called StringReverser with a method called reverseString(String text) that reverses a given text. Implement the string reversal logic using StringBuilder. Test the StringReverser class by creating an instance, calling the reverseString method with different texts, and observing the reversed output.

## Exercise 6:

Create a class called WordCounter with a method called countWords(String sentence) that counts the number of words in a given sentence. Implement the word counting logic using String operations and StringBuilder. Test the WordCounter class by creating an instance, calling the countWords method with different sentences, and checking the word count results.

## Exercise 7:

Create a class called StringAnalyzer with methods to analyze strings, such as countOccurrences(String text, String pattern), findLongestWord(String sentence), and capitalizeFirstLetter(String text). Implement the string analysis logic using String operations and StringBuilder. Test the StringAnalyzer class by creating an instance, calling the analysis methods with different inputs, and observing the analysis results.

## Exercise 8:

Create a class called StringEncryptor with methods to encrypt and decrypt strings. Implement encryption and decryption algorithms using String operations and StringBuilder. Test the StringEncryptor class by creating an instance, encrypting and decrypting different strings, and verifying the encryption-decryption results.

## Exercise 9:

Create a class called SentenceGenerator with a method called generateRandomSentence(int length) that generates a random sentence of a given length. Implement the sentence generation logic using StringBuilder and random string generation techniques. Test the SentenceGenerator class by creating an instance, calling the generateRandomSentence method with different lengths, and observing the generated sentences.

## Exercise 10:

Create a class called TextAnalyzer with methods to analyze and manipulate text, such as removeDuplicates(String text), countVowels(String text), and replaceSubstring(String text, String substring, String replacement). Implement the text analysis and manipulation logic using String operations and StringBuilder. Test the TextAnalyzer class by creating an instance, calling the analysis and manipulation methods with different inputs, and observing the results.

## Exercise 11:

Create a class called PasswordGenerator with a method called generatePassword(int length) that generates a random password of a given length. Implement the password generation logic using StringBuilder and random character selection. Test the PasswordGenerator class by creating an instance, calling the generatePassword method with different lengths, and observing the generated passwords.

## Exercise 12:

Create a class called StringCompressor with a method called compressString(String text) that compresses a given string by replacing repeated characters with a number indicating the repetition count. Implement the string compression logic using StringBuilder. Test the StringCompressor class by creating an instance, calling the compressString method with different texts, and observing the compressed output.

## Exercise 13:

Create a class called WordFrequencyCounter with a method called countWordFrequencies(String text) that counts the frequencies of words in a given text. Implement the word frequency counting logic using String operations, StringBuilder, and a HashMap to store the word-frequency pairs. Test

the WordFrequencyCounter class by creating an instance, calling the countWordFrequencies method with different texts, and observing the word frequency results.

## Exercise 14:

Create a class called CharacterCounter with a method called countCharacterOccurrences(String text, char character) that counts the occurrences of a specific character in a given text. Implement the character counting logic using String operations and StringBuilder. Test the CharacterCounter class by creating an instance, calling the countCharacterOccurrences method with different texts and characters, and observing the occurrence counts.

## Exercise 15:

Create a class called StringShuffler with a method called shuffleString(String text) that shuffles the characters of a given string randomly. Implement the string shuffling logic using StringBuilder and random character swapping techniques. Test the StringShuffler class by creating an instance, calling the shuffleString method with different texts, and observing the shuffled output.

## Exercise 16:

Create a class called TextAnalyzer with a method called reverseWordOrder(String text) that reverses the order of words in a given text. Implement the word order reversal logic using String operations, StringBuilder, and splitting/joining techniques. Test the TextAnalyzer class by creating an instance, calling the reverseWordOrder method with different texts, and observing the reversed word order.

## Exercise 17:

Create a class called StringAnalyzer with a method called findLongestCommonSubstring(String text1, String text2) that finds the longest common substring between two given strings. Implement the common substring finding logic using String operations and StringBuilder. Test the StringAnalyzer class by creating an instance, calling the findLongestCommonSubstring method with different texts, and observing the longest common substrings.

## Exercise 18:

Create a class called SentenceCapitalizer with a method called capitalizeSentences(String text) that capitalizes the first letter of each sentence in a given text. Implement the sentence capitalization logic using String operations, StringBuilder, and regular expressions. Test the SentenceCapitalizer class by creating an instance, calling the capitalizeSentences method with different texts, and observing the capitalized sentences.

## Exercise 19:

Create a class called StringJoiner with a method called joinStrings(String[] strings, String delimiter) that joins an array of strings into a single string using a specified delimiter. Implement the string joining logic using StringBuilder. Test the StringJoiner class by creating an instance, calling the joinStrings method with different string arrays and delimiters, and observing the joined strings.

## Exercise 20:

Create a class called AnagramChecker with a method called isAnagram(String word1, String word2) that checks if two given words are anagrams (contain the same characters in a different

order). Implement the anagram checking logic using String operations and StringBuilder. Test the AnagramChecker class by creating an instance, calling the isAnagram method with different word pairs, and observing the anagram results.

---

## Exercise 21:

Write a Java program to find the kth element from the end of a LinkedList. Implement a method called `findKthElement` that takes a LinkedList and an integer k as input, and returns the kth element from the end of the LinkedList.

---

## Exercise 22:

Write a Java program to sort a LinkedList in ascending order. Implement a method called `sortLinkedList` that takes a LinkedList as input and modifies it to sort its elements in ascending order.

---

## Exercise 23:

Write a Java program to merge two sorted LinkedLists into a single sorted LinkedList. Implement a method called `mergeSortedLists` that takes two sorted LinkedLists as input and returns a new LinkedList that contains all the elements from both LinkedLists in sorted order.

---

## Exercise 24:

Write a Java program to find the nth node from the end of a LinkedList without using the size of the LinkedList. Implement a method called `findNthFromEnd` that takes a LinkedList and an integer n as input, and returns the nth node from the end of the LinkedList.

---

## Exercise 25:

Write a Java program to detect a cycle in a LinkedList. Implement a method called `hasCycle` that takes a LinkedList as input and returns true if there is a cycle in the LinkedList, and false otherwise.

---

## Exercise 26:

Create a class called `Employee` with attributes like `id`, `name`, and `salary`. Implement a LinkedList to store instances of the `Employee` class. Write a program to perform various operations on the LinkedList, such as adding employees, removing employees, finding employees by ID, and displaying employee details.

---

## Exercise 27:

Create a class called `Student` with attributes like `rollNumber`, `name`, and `marks`. Implement a LinkedList to store instances of the `Student` class. Write a program to perform various operations on the LinkedList, such as adding students, removing students, calculating the average marks, finding students with the highest marks, and displaying student details.

---

## Exercise 28:

Create a class called `Product` with attributes like `id`, `name`, and `quantity`. Implement a LinkedList to store instances of the `Product` class. Write a program to perform various

operations on the LinkedList, such as adding products, removing products, updating product quantities, finding products with low quantities, and displaying product details.

---

## Exercise 29:

Create a class called `Library` with attributes like `name` and a LinkedList of `Book` objects. Each `Book` should have attributes like `title`, `author`, and `year`. Implement methods in the `Library` class to add books, remove books, search for books by title or author, and display the list of books.

---

## Exercise 30:

Create a class called `Course` with attributes like `name` and a LinkedList of `Student` objects. Each `Student` should have attributes like `name`, `age`, and `grade`. Implement methods in the `Course` class to add students, remove students, calculate the average grade, find students with the highest grade, and display student information.

---

## Exercise 31:

Create a class called `TravelAgency` with attributes like `name` and a LinkedList of `Trip` objects. Each `Trip` should have attributes like `destination`, `duration`, and `price`. Implement methods in the `TravelAgency` class to add trips, remove trips, find trips within a price range, sort trips by duration, and display trip details.

---

## Exercise 32:

Create a class called `ShoppingCart` with attributes like `customerName` and a LinkedList of `Product` objects. Each `Product` should have attributes like `name`, `price`, and `quantity`. Implement methods in the `ShoppingCart` class to add products, remove products, calculate the total price, find products with a specific name, and display the contents of the cart.

---

## Exercise 33:

Create a class called `Bank` with attributes like `name` and a LinkedList of `Account` objects. Each `Account` should have attributes like `accountNumber`, `holderName`, and `balance`. Implement methods in the `Bank` class to add accounts, remove accounts, search for accounts by holder name, calculate the total balance, and display account information.

---

## Exercise 34:

Create a class called `Inventory` with attributes like `name` and a LinkedList of `Item` objects. Each `Item` should have attributes like `name`, `quantity`, and `price`. Implement methods in the `Inventory` class to add items, remove items, search for items by name, calculate the total value of the inventory, and display the inventory list.

---

## Exercise 35:

Create a class called `EmployeeManagement` with attributes like `department` and a LinkedList of `Employee` objects. Each `Employee` should have attributes like `name`, `age`, and `salary`. Implement methods in the `EmployeeManagement` class to add employees, remove

employees, search for employees by department, calculate the average salary, and display employee information.

---

## Exercise 36:

Create a class called `School` with attributes like `name` and a LinkedList of `Student` objects. Each `Student` should have attributes like `name`, `age`, and a LinkedList of `Subject` objects. Each `Subject` should have attributes like `name` and `grade`. Implement methods in the `School` class to add students, remove students, search for students by name, calculate the average grade for a specific subject, and display student information.

---

## Exercise 37:

Create a class called `ChatRoom` with attributes like `name` and a LinkedList of `Message` objects. Each `Message` should have attributes like `sender`, `receiver`, and `content`. Implement methods in the `ChatRoom` class to add messages, remove messages, search for messages by sender or receiver, and display the chat history.