**Problem 1: Write a program to find and replace all occurrences of a**

**substring in a text without using the replace() method**

**Hint =>**

**a. Take user input using the Scanner nextLine() method for the main text and the substring**

**to find and replace**

**b. Create a method to find all occurrences of the substring using indexOf() method in a loop**

**and store the starting positions in an array**

**c. Create a method to replace the substring manually by:**

**● i. Building a new string character by character using charAt() method**

**● ii. Skip the characters of the original substring and insert the replacement substring**

**d. Create a method to compare the result with the built-in replace() method and return**

**boolean**

**e. The main function calls all user-defined methods and displays both results along with the**

**comparison**

import java.util.\*;

public class ManualSubstringReplace {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

// Step a: Take inputs

System.out.println("Enter the main text: ");

String text = sc.nextLine();

System.out.println("Enter the substring to find: ");

String find = sc.nextLine();

System.out.println("Enter the replacement substring: ");

String replace = sc.nextLine();

// Step b: Find all occurrences

List<Integer> positions = findOccurrences(text, find);

System.out.println("Occurrences found at positions: " + positions);

// Step c: Manual replace

String manualResult = manualReplace(text, find, replace);

// Step d: Compare with built-in replace()

String builtInResult = text.replace(find, replace);

boolean isSame = compareResults(manualResult, builtInResult);

// Step e: Display results

System.out.println("\n=== RESULTS ===");

System.out.println("Original Text: " + text);

System.out.println("Manual Replace Result: " + manualResult);

System.out.println("Built-in Replace Result: " + builtInResult);

System.out.println("Do both results match? " + isSame);

sc.close();

}

// Step b: Find all occurrences using indexOf in a loop

public static List<Integer> findOccurrences(String text, String find) {

List<Integer> positions = new ArrayList<>();

int index = text.indexOf(find);

while (index != -1) {

positions.add(index);

index = text.indexOf(find, index + find.length());

}

return positions;

}

// Step c: Manual replace using charAt

public static String manualReplace(String text, String find, String replace) {

StringBuilder result = new StringBuilder();

int i = 0;

while (i < text.length()) {

// if substring matches

if (i <= text.length() - find.length() &&

text.substring(i, i + find.length()).equals(find)) {

result.append(replace); // insert replacement

i += find.length(); // skip original substring

} else {

result.append(text.charAt(i)); // copy original char

i++;

}

}

return result.toString();

}

// Step d: Compare manual vs built-in

public static boolean compareResults(String manual, String builtIn) {

return manual.equals(builtIn);

}

}

**Problem 2: Write a program to convert text between different cases**

**(uppercase, lowercase, title case) using ASCII values without using built-in**

**case conversion methods**

**Hint =>**

**a. Take user input using the Scanner nextLine() method**

**b. Create a method to convert a character to uppercase using ASCII values:**

**● i. Check if the character is a lowercase letter (ASCII 97-122)**

**● ii. Convert by subtracting 32 from the ASCII value**

**c. Create a method to convert a character to lowercase using ASCII values:**

**● i. Check if the character is an uppercase letter (ASCII 65-90)**

**● ii. Convert by adding 32 to the ASCII value**

**d. Create a method for title case conversion:**

**● i. Convert the first character of each word to uppercase**

**● ii. Convert all other characters to lowercase**

**e. Create a method to compare results with built-in methods (toUpperCase(),**

**toLowerCase())**

**f. The main function calls all methods and displays the results in a tabular format**

import java.util.Scanner;

public class ASCIICaseConversion {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

// Step a: Input text

System.out.println("Enter a sentence: ");

String text = sc.nextLine();

// Convert using custom methods

String upperManual = toUpperCaseManual(text);

String lowerManual = toLowerCaseManual(text);

String titleManual = toTitleCaseManual(text);

// Convert using built-in methods (for comparison)

String upperBuiltIn = text.toUpperCase();

String lowerBuiltIn = text.toLowerCase();

// Step e: Compare results

boolean upperMatch = upperManual.equals(upperBuiltIn);

boolean lowerMatch = lowerManual.equals(lowerBuiltIn);

// Step f: Display results in a formatted table

System.out.println("\n=== CASE CONVERSION RESULTS ===");

System.out.printf("%-20s | %-50s\n", "Operation", "Result");

System.out.println("---------------------------------------------------------------");

System.out.printf("%-20s | %-50s\n", "Original Text", text);

System.out.printf("%-20s | %-50s\n", "Manual Uppercase", upperManual);

System.out.printf("%-20s | %-50s\n", "Built-in Uppercase", upperBuiltIn);

System.out.printf("%-20s | %-50s\n", "Match?", upperMatch);

System.out.printf("%-20s | %-50s\n", "Manual Lowercase", lowerManual);

System.out.printf("%-20s | %-50s\n", "Built-in Lowercase", lowerBuiltIn);

System.out.printf("%-20s | %-50s\n", "Match?", lowerMatch);

System.out.printf("%-20s | %-50s\n", "Manual Title Case", titleManual);

sc.close();

}

// Step b: Convert character to uppercase using ASCII

public static char toUpperChar(char ch) {

if (ch >= 'a' && ch <= 'z') {

return (char) (ch - 32); // subtract 32 from ASCII

}

return ch;

}

// Step c: Convert character to lowercase using ASCII

public static char toLowerChar(char ch) {

if (ch >= 'A' && ch <= 'Z') {

return (char) (ch + 32); // add 32 to ASCII

}

return ch;

}

// Convert entire string to uppercase manually

public static String toUpperCaseManual(String text) {

StringBuilder result = new StringBuilder();

for (int i = 0; i < text.length(); i++) {

result.append(toUpperChar(text.charAt(i)));

}

return result.toString();

}

// Convert entire string to lowercase manually

public static String toLowerCaseManual(String text) {

StringBuilder result = new StringBuilder();

for (int i = 0; i < text.length(); i++) {

result.append(toLowerChar(text.charAt(i)));

}

return result.toString();

}

// Step d: Title Case conversion

public static String toTitleCaseManual(String text) {

StringBuilder result = new StringBuilder();

boolean newWord = true;

for (int i = 0; i < text.length(); i++) {

char ch = text.charAt(i);

if (ch == ' ') {

result.append(ch);

newWord = true;

} else {

if (newWord) {

result.append(toUpperChar(ch));

newWord = false;

} else {

result.append(toLowerChar(ch));

}

}

}

return result.toString();

}

}

**Problem 3: Write a program to analyze and compare the performance of**

**String concatenation vs StringBuilder vs StringBuffer for building large**

**strings**

**Hint =>**

**a. Take user input for the number of iterations (e.g., 1000, 10000, 100000)**

**b. Create a method to perform String concatenation in a loop:**

**● i. Use System.currentTimeMillis() to measure start and end time**

**● ii. Concatenate a sample string multiple times using the + operator**

**● iii. Return the time taken and final string length**

**c. Create a method to perform StringBuilder operations:**

**● i. Use StringBuilder.append() method in a loop**

**● ii. Measure the time taken and return results**

**d. Create a method to perform StringBuffer operations:**

**● i. Use StringBuffer.append() method in a loop**

**● ii. Measure the time taken and return results**

**e. Create a method to display performance comparison in a tabular format showing:**

**● i. Method used, Time taken (milliseconds), Memory efficiency**

**f. The main function calls all methods and displays the performance analysis**

import java.util.Scanner;

public class StringPerformanceAnalysis {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

// Step a: Take user input for iterations

System.out.print("Enter number of iterations (e.g., 1000, 10000, 100000): ");

int iterations = sc.nextInt();

// Run tests

Result stringResult = testStringConcat(iterations);

Result builderResult = testStringBuilder(iterations);

Result bufferResult = testStringBuffer(iterations);

// Step e: Display comparison in tabular format

System.out.println("\n=== PERFORMANCE ANALYSIS ===");

System.out.printf("%-15s | %-15s | %-15s\n", "Method", "Time (ms)", "Final Length");

System.out.println("-----------------------------------------------------");

System.out.printf("%-15s | %-15d | %-15d\n", "String (+)", stringResult.timeTaken, stringResult.finalLength);

System.out.printf("%-15s | %-15d | %-15d\n", "StringBuilder", builderResult.timeTaken, builderResult.finalLength);

System.out.printf("%-15s | %-15d | %-15d\n", "StringBuffer", bufferResult.timeTaken, bufferResult.finalLength);

sc.close();

}

// Helper class to store results

static class Result {

long timeTaken;

int finalLength;

Result(long timeTaken, int finalLength) {

this.timeTaken = timeTaken;

this.finalLength = finalLength;

}

}

// Step b: String concatenation using + operator

public static Result testStringConcat(int iterations) {

long start = System.currentTimeMillis();

String str = "";

for (int i = 0; i < iterations; i++) {

str += "Java" + i;

}

long end = System.currentTimeMillis();

return new Result(end - start, str.length());

}

// Step c: StringBuilder operations

public static Result testStringBuilder(int iterations) {

long start = System.currentTimeMillis();

StringBuilder sb = new StringBuilder();

for (int i = 0; i < iterations; i++) {

sb.append("Java").append(i);

}

long end = System.currentTimeMillis();

return new Result(end - start, sb.length());

}

// Step d: StringBuffer operations

public static Result testStringBuffer(int iterations) {

long start = System.currentTimeMillis();

StringBuffer sb = new StringBuffer();

for (int i = 0; i < iterations; i++) {

sb.append("Java").append(i);

}

long end = System.currentTimeMillis();

return new Result(end - start, sb.length());

}

}

**Problem 4: Write a program to create a simple encryption and decryption**

**system using ASCII character shifting (Caesar Cipher implementation)**

**Hint =>**

**a. Take user input for the text to encrypt and the shift value**

**b. Create a method to encrypt text using ASCII values:**

**● i. For each character, get its ASCII value using (int) casting**

**● ii. Shift the ASCII value by the given amount**

**● iii. Handle wrap-around for alphabetic characters (A-Z, a-z)**

**● iv. Keep non-alphabetic characters unchanged**

**c. Create a method to decrypt text:**

**● i. Reverse the shifting process**

**● ii. Handle negative shifts properly**

**d. Create a method to display ASCII values of characters before and after encryption**

**e. Create a method to validate that decryption returns the original text**

**f. The main function takes inputs, calls encryption/decryption methods, and displays:**

**● i. Original text with ASCII values**

**● ii. Encrypted text with ASCII values**

**● iii. Decrypted text with validation result**

import java.util.Scanner;

public class CaesarCipher {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

// Step a: Take user input

System.out.print("Enter text to encrypt: ");

String text = sc.nextLine();

System.out.print("Enter shift value: ");

int shift = sc.nextInt();

// Encrypt the text

String encrypted = encrypt(text, shift);

// Decrypt the text

String decrypted = decrypt(encrypted, shift);

// Step f: Display results

System.out.println("\n=== CAESAR CIPHER DEMO ===");

System.out.println("Original Text : " + text);

displayASCII(text);

System.out.println("\nEncrypted Text : " + encrypted);

displayASCII(encrypted);

System.out.println("\nDecrypted Text : " + decrypted);

displayASCII(decrypted);

// Step e: Validation

if (text.equals(decrypted)) {

System.out.println("\n✅ Decryption successful (matches original text)");

} else {

System.out.println("\n❌ Decryption failed (does not match original text)");

}

sc.close();

}

// Step b: Encryption

public static String encrypt(String text, int shift) {

StringBuilder result = new StringBuilder();

for (char ch : text.toCharArray()) {

if (ch >= 'A' && ch <= 'Z') {

result.append((char) ((ch - 'A' + shift) % 26 + 'A'));

} else if (ch >= 'a' && ch <= 'z') {

result.append((char) ((ch - 'a' + shift) % 26 + 'a'));

} else {

result.append(ch); // keep non-alphabetic unchanged

}

}

return result.toString();

}

// Step c: Decryption

public static String decrypt(String text, int shift) {

return encrypt(text, 26 - (shift % 26)); // reverse the shift

}

// Step d: Display ASCII values

public static void displayASCII(String text) {

for (char ch : text.toCharArray()) {

System.out.println("'" + ch + "' -> ASCII: " + (int) ch);

}

}

}

**Problem 5: Write a program to extract and analyze different parts of an**

**email address using substring() and indexOf() methods**

**Hint =>**

**a. Take user input for multiple email addresses using Scanner**

**b. Create a method to validate email format:**

**● i. Check for exactly one '@' symbol using indexOf() and lastIndexOf()**

**● ii. Check for at least one '.' after '@' symbol**

**● iii. Validate that username and domain are not empty**

**c. Create a method to extract email components:**

**● i. Extract username using substring() from start to '@' position**

**● ii. Extract domain using substring() from '@' position to end**

**● iii. Extract domain name and extension separately**

**d. Create a method to analyze email statistics:**

**● i. Count total valid/invalid emails**

**● ii. Find most common domain**

**● iii. Calculate average username length**

**e. Create a method to display results in tabular format showing:**

**● i. Email, Username, Domain, Domain Name, Extension, Valid/Invalid**

**f. The main function processes multiple emails and displays analysis results**

import java.util.\*;

public class EmailAnalyzer {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

// Step a: Take user input

System.out.print("Enter number of emails: ");

int n = sc.nextInt();

sc.nextLine(); // consume newline

String[] emails = new String[n];

for (int i = 0; i < n; i++) {

System.out.print("Enter email " + (i + 1) + ": ");

emails[i] = sc.nextLine();

}

// Stats

int validCount = 0, invalidCount = 0;

int totalUsernameLength = 0;

Map<String, Integer> domainFrequency = new HashMap<>();

System.out.println("\n=== EMAIL ANALYSIS RESULTS ===");

System.out.printf("%-25s %-15s %-20s %-15s %-10s %-10s\n",

"Email", "Username", "Domain", "Domain Name", "Extension", "Valid?");

for (String email : emails) {

boolean valid = validateEmail(email);

if (valid) {

validCount++;

String[] parts = extractComponents(email);

String username = parts[0];

String domain = parts[1];

String domainName = parts[2];

String extension = parts[3];

totalUsernameLength += username.length();

// track domain frequency

domainFrequency.put(domain, domainFrequency.getOrDefault(domain, 0) + 1);

System.out.printf("%-25s %-15s %-20s %-15s %-10s %-10s\n",

email, username, domain, domainName, extension, "Valid");

} else {

invalidCount++;

System.out.printf("%-25s %-15s %-20s %-15s %-10s %-10s\n",

email, "-", "-", "-", "-", "Invalid");

}

}

// Step d: Analysis

String mostCommonDomain = "-";

int maxCount = 0;

for (Map.Entry<String, Integer> entry : domainFrequency.entrySet()) {

if (entry.getValue() > maxCount) {

mostCommonDomain = entry.getKey();

maxCount = entry.getValue();

}

}

double avgUsernameLength = (validCount > 0)

? (double) totalUsernameLength / validCount : 0.0;

System.out.println("\n=== STATISTICS ===");

System.out.println("Total Emails : " + n);

System.out.println("Valid Emails : " + validCount);

System.out.println("Invalid Emails : " + invalidCount);

System.out.println("Most Common Domain : " + mostCommonDomain);

System.out.printf("Avg Username Length: %.2f\n", avgUsernameLength);

sc.close();

}

// Step b: Validate email format

public static boolean validateEmail(String email) {

int atPos = email.indexOf('@');

int lastAtPos = email.lastIndexOf('@');

if (atPos == -1 || atPos != lastAtPos) return false; // not exactly one '@'

if (atPos == 0) return false; // username empty

if (atPos == email.length() - 1) return false; // domain empty

int dotPos = email.indexOf('.', atPos);

if (dotPos == -1 || dotPos == email.length() - 1) return false; // no '.' in domain or no extension

return true;

}

// Step c: Extract components

public static String[] extractComponents(String email) {

int atPos = email.indexOf('@');

int dotPos = email.lastIndexOf('.');

String username = email.substring(0, atPos);

String domain = email.substring(atPos + 1);

String domainName = email.substring(atPos + 1, dotPos);

String extension = email.substring(dotPos + 1);

return new String[]{username, domain, domainName, extension};

}

}

**Problem 6: Write a program to create a text formatter that justifies text to a**

**specified width using StringBuilder for efficient string manipulation**

**Hint =>**

**a. Take user input for the text to format and desired line width**

**b. Create a method to split text into words without using split():**

**● i. Use charAt() to identify spaces**

**● ii. Extract words using substring() method**

**● iii. Store words in an array**

**c. Create a method using StringBuilder to justify text:**

**● i. Add words to current line until width limit is reached**

**● ii. Distribute extra spaces evenly between words**

**● iii. Handle last line separately (left-aligned only)**

**d. Create a method to center-align text:**

**● i. Calculate padding needed on both sides**

**● ii. Use StringBuilder to build centered lines**

**e. Create a method to compare performance:**

**● i. Implement the same formatting using String concatenation**

**● ii. Measure time difference using System.nanoTime()**

**f. Create a method to display the formatted text with:**

**● i. Line numbers**

**● ii. Character count per line**

**● iii. Performance comparison results**

**g. The main function calls all methods and displays:**

**● i. Original text**

**● ii. Left-justified text**

**● iii. Center-aligned text**

**● iv. Performance analysis**

import java.util.\*;

public class TextFormatter {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

// Step a: Take input

System.out.println("Enter the text to format:");

String text = sc.nextLine();

System.out.print("Enter desired line width: ");

int width = sc.nextInt();

// Split words manually

String[] words = splitWords(text);

// Justified text

long start = System.nanoTime();

List<String> justified = justifyText(words, width);

long end = System.nanoTime();

long builderTime = end - start;

// Centered text

List<String> centered = centerAlign(words, width);

// Performance comparison with String concatenation

long stringTime = performanceWithStrings(words, width);

// Display results

System.out.println("\n=== ORIGINAL TEXT ===");

System.out.println(text);

System.out.println("\n=== JUSTIFIED TEXT ===");

displayFormatted(justified);

System.out.println("\n=== CENTER-ALIGNED TEXT ===");

displayFormatted(centered);

System.out.println("\n=== PERFORMANCE ANALYSIS ===");

System.out.println("StringBuilder Time : " + builderTime + " ns");

System.out.println("String Concatenation Time : " + stringTime + " ns");

sc.close();

}

// Step b: Split words without split()

public static String[] splitWords(String text) {

List<String> words = new ArrayList<>();

int start = 0;

for (int i = 0; i < text.length(); i++) {

if (text.charAt(i) == ' ') {

if (start < i) {

words.add(text.substring(start, i));

}

start = i + 1;

}

}

if (start < text.length()) {

words.add(text.substring(start));

}

return words.toArray(new String[0]);

}

// Step c: Justify text using StringBuilder

public static List<String> justifyText(String[] words, int width) {

List<String> result = new ArrayList<>();

int i = 0;

while (i < words.length) {

int lineLen = words[i].length();

int j = i + 1;

while (j < words.length && lineLen + words[j].length() + (j - i) <= width) {

lineLen += words[j].length();

j++;

}

int gaps = j - i - 1;

StringBuilder sb = new StringBuilder();

if (j == words.length || gaps == 0) {

// Last line or single word

for (int k = i; k < j; k++) {

sb.append(words[k]);

if (k < j - 1) sb.append(" ");

}

while (sb.length() < width) sb.append(" ");

} else {

int spaces = (width - lineLen) / gaps;

int extra = (width - lineLen) % gaps;

for (int k = i; k < j; k++) {

sb.append(words[k]);

if (k < j - 1) {

for (int s = 0; s < spaces; s++) sb.append(" ");

if (extra-- > 0) sb.append(" ");

}

}

}

result.add(sb.toString());

i = j;

}

return result;

}

// Step d: Center align

public static List<String> centerAlign(String[] words, int width) {

List<String> result = new ArrayList<>();

StringBuilder line = new StringBuilder();

for (String word : words) {

if (line.length() + word.length() + 1 > width) {

result.add(centerLine(line.toString().trim(), width));

line = new StringBuilder();

}

line.append(word).append(" ");

}

if (line.length() > 0) {

result.add(centerLine(line.toString().trim(), width));

}

return result;

}

private static String centerLine(String line, int width) {

int padding = (width - line.length()) / 2;

StringBuilder sb = new StringBuilder();

for (int i = 0; i < padding; i++) sb.append(" ");

sb.append(line);

while (sb.length() < width) sb.append(" ");

return sb.toString();

}

// Step e: Performance with String concatenation

public static long performanceWithStrings(String[] words, int width) {

long start = System.nanoTime();

String line = "";

for (String word : words) {

if (line.length() + word.length() + 1 > width) {

line = "";

}

line += word + " "; // inefficient concatenation

}

return System.nanoTime() - start;

}

// Step f: Display with line numbers & char count

public static void displayFormatted(List<String> lines) {

int lineNum = 1;

for (String line : lines) {

System.out.printf("%2d | %-s | (%d chars)\n", lineNum++, line, line.length());

}

}

}