

**Java String Operations and Performance Lab Practice Problems**

**Best Programming Practices**

1. Use Variables including for Fixed, User Inputs, and Results

2. Use Methods instead of writing code in the main() function

3. Proper naming conventions for all variables and methods

4. Proper Program Name and Class Name

5. Handle checked and Unchecked Exceptions wherever possible

6. Proper Method Name, which indicates action-taking inputs and provides rthe esult

**Lab Practice Programs (Any Six)**

**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 a boolean

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

Program-

import java.util.\*;

public class Problem1 {

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;

}

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

StringBuilder result = new StringBuilder();

int i = 0;

while (i < text.length()) {

if (i <= text.length() - find.length() && text.substring(i, i + find.length()).equals(find)) {

result.append(replace);

i += find.length();

} else {

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

i++;

}

}

return result.toString();

}

public static boolean compareWithBuiltIn(String manual, String text, String find, String replace) {

return manual.equals(text.replace(find, replace));

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

String text = sc.nextLine();

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

String find = sc.nextLine();

System.out.print("Enter replacement: ");

String replace = sc.nextLine();

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

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

boolean isSame = compareWithBuiltIn(manualResult, text, find, replace);

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

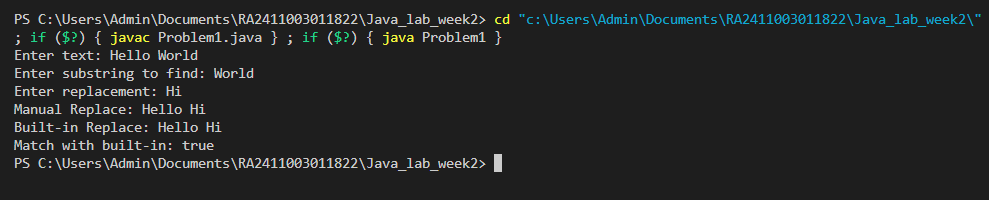
System.out.println("Built-in Replace: " + text.replace(find, replace));

System.out.println("Match with built-in: " + isSame);

}

}

Output –



1



**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

Program –

import java.util.\*;

public class Problem2 {

public static char toUpper(char c) {

return (c >= 'a' && c <= 'z') ? (char)(c - 32) : c;

}

public static char toLower(char c) {

return (c >= 'A' && c <= 'Z') ? (char)(c + 32) : c;

}

public static String toUpperCase(String str) {

StringBuilder sb = new StringBuilder();

for (char c : str.toCharArray()) sb.append(toUpper(c));

return sb.toString();

}

public static String toLowerCase(String str) {

StringBuilder sb = new StringBuilder();

for (char c : str.toCharArray()) sb.append(toLower(c));

return sb.toString();

}

public static String toTitleCase(String str) {

StringBuilder sb = new StringBuilder();

boolean capitalize = true;

for (char c : str.toCharArray()) {

if (c == ' ') {

sb.append(c);

capitalize = true;

} else {

sb.append(capitalize ? toUpper(c) : toLower(c));

capitalize = false;

}

}

return sb.toString();

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

String text = sc.nextLine();

String upper = toUpperCase(text);

String lower = toLowerCase(text);

String title = toTitleCase(text);

System.out.println("Custom\t\tBuilt-in");

System.out.println("Upper: " + upper + "\t" + text.toUpperCase());

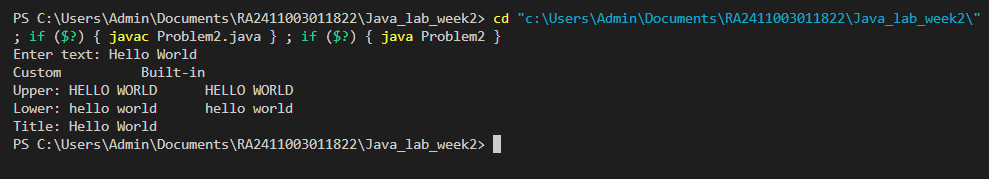
System.out.println("Lower: " + lower + "\t" + text.toLowerCase());

System.out.println("Title: " + title);

}

}

Output –



**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 2



● 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

Program –

import java.util.\*;

public class Problem3 {

public static long stringConcat(int n) {

long start = System.currentTimeMillis();

String s = "";

for (int i = 0; i < n; i++) s += "a";

return System.currentTimeMillis() - start;

}

public static long stringBuilderConcat(int n) {

long start = System.currentTimeMillis();

StringBuilder sb = new StringBuilder();

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

return System.currentTimeMillis() - start;

}

public static long stringBufferConcat(int n) {

long start = System.currentTimeMillis();

StringBuffer sb = new StringBuffer();

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

return System.currentTimeMillis() - start;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter iterations: ");

int n = sc.nextInt();

long t1 = stringConcat(n);

long t2 = stringBuilderConcat(n);

long t3 = stringBufferConcat(n);

System.out.println("Method\t\tTime(ms)");

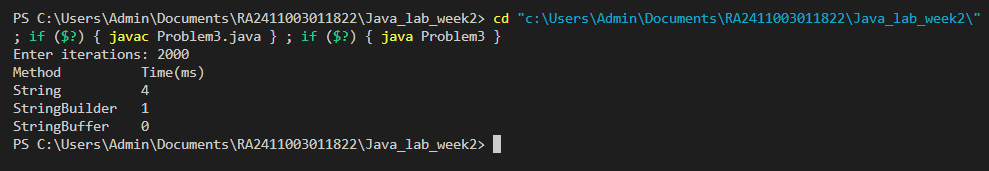
System.out.println("String\t\t" + t1);

System.out.println("StringBuilder\t" + t2);

System.out.println("StringBuffer\t" + t3);

}}

Output-



**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

3



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

Program-

import java.util.\*;

public class Problem4 {

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

        StringBuilder sb = new StringBuilder();

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

            if (Character.isLetter(c)) {

                char base = Character.isUpperCase(c) ? 'A' : 'a';

                sb.append((char)((c - base + shift + 26) % 26 + base));

            } else sb.append(c);

        }

        return sb.toString();

    }

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

        return encrypt(text, -shift);

    }

    public static void displayAscii(String original, String encrypted) {

        System.out.println("Char\tASCII(original)\tASCII(encrypted)");

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

            System.out.println(original.charAt(i) + "\t" + (int)original.charAt(i) + "\t\t" + (int)encrypted.charAt(i));

        }

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

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

        String text = sc.nextLine();

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

        int shift = sc.nextInt();

        String encrypted = encrypt(text, shift);

        String decrypted = decrypt(encrypted, shift);

        displayAscii(text, encrypted);

        System.out.println("Encrypted: " + encrypted);

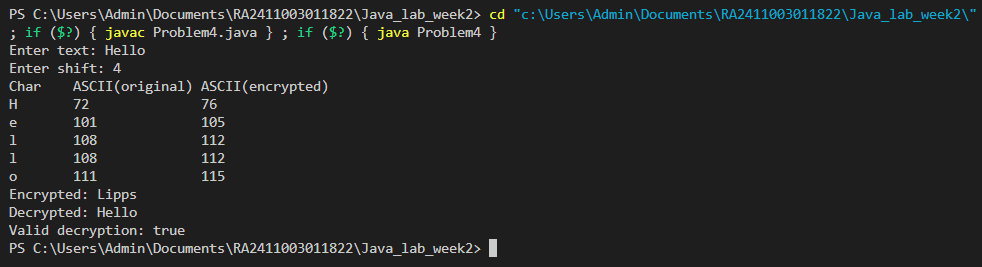
        System.out.println("Decrypted: " + decrypted);

        System.out.println("Valid decryption: " + text.equals(decrypted));

    }

}

Output –



**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

Program –

import java.util.\*;

public class EmailAnalyzer {

// Validate email format

public static boolean isValidEmail(String email) {

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

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

if (atIndex == -1 || atIndex != lastAtIndex) return false; // exactly one '@'

if (atIndex == 0 || atIndex == email.length() - 1) return false; // username or domain empty

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

if (!domain.contains(".")) return false; // at least one '.' after '@'

// Check domain name and extension not empty

int dotIndex = domain.indexOf('.');

if (dotIndex == 0 || dotIndex == domain.length() - 1) return false;

return true;

}

// Extract email components

public static String[] extractComponents(String email) {

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

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

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

int dotIndex = domain.indexOf('.');

String domainName = domain.substring(0, dotIndex);

String extension = domain.substring(dotIndex + 1);

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

}

// Analyze email statistics

public static void analyzeEmails(List<String> emails, List<Boolean> valids, List<String[]> components) {

int validCount = 0;

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

int totalUsernameLength = 0;

for (int i = 0; i < emails.size(); i++) {

if (valids.get(i)) {

validCount++;

String domainName = components.get(i)[1]; // full domain

domainCount.put(domainName, domainCount.getOrDefault(domainName, 0) + 1);

totalUsernameLength += components.get(i)[0].length();

}

}

int invalidCount = emails.size() - validCount;

// Find most common domain

String mostCommonDomain = null;

int maxCount = 0;

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

if (entry.getValue() > maxCount) {

maxCount = entry.getValue();

mostCommonDomain = entry.getKey();

}

}

double avgUsernameLength = validCount > 0 ? (double) totalUsernameLength / validCount : 0;

System.out.println("\n--- Email Statistics ---");

System.out.println("Total valid emails: " + validCount);

System.out.println("Total invalid emails: " + invalidCount);

System.out.println("Most common domain: " + (mostCommonDomain == null ? "N/A" : mostCommonDomain));

System.out.printf("Average username length: %.2f\n", avgUsernameLength);

}

// Display results in tabular format

public static void displayResults(List<String> emails, List<Boolean> valids, List<String[]> components) {

System.out.println("\nEmail\t\tUsername\tDomain\t\tDomain Name\tExtension\tValid/Invalid");

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

for (int i = 0; i < emails.size(); i++) {

String validStr = valids.get(i) ? "Valid" : "Invalid";

if (valids.get(i)) {

String[] comp = components.get(i);

System.out.printf("%-16s %-12s %-16s %-12s %-10s %s\n",

emails.get(i), comp[0], comp[1], comp[2], comp[3], validStr);

} else {

System.out.printf("%-16s %-12s %-16s %-12s %-10s %s\n",

emails.get(i), "-", "-", "-", "-", validStr);

}

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int n = sc.nextInt();

sc.nextLine(); // consume newline

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

List<Boolean> valids = new ArrayList<>();

List<String[]> components = new ArrayList<>();

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

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

String email = sc.nextLine();

emails.add(email);

boolean valid = isValidEmail(email);

valids.add(valid);

if (valid) {

components.add(extractComponents(email));

} else {

components.add(new String[]{"-", "-", "-", "-"});

}

}

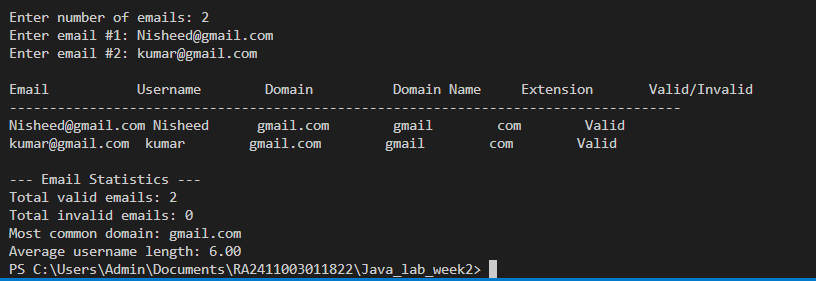
displayResults(emails, valids, components);

analyzeEmails(emails, valids, components);

}

}

Output-



**Problem 6: Write a program to create a text formatter that justifies text to a specified width using StringBuilder for efficient string manipulation**

4



**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

Program-

import java.util.\*;

public class TextFormatter {

// Split text into words without using split()

public static String[] splitWords(String text) {

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

int start = 0;

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

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

if (start < i) {

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

}

start = i + 1;

}

}

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

}

// Justify text using StringBuilder

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

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

int index = 0;

while (index < words.length) {

int count = words[index].length();

int last = index + 1;

while (last < words.length) {

if (count + 1 + words[last].length() > width) break;

count += 1 + words[last].length();

last++;

}

StringBuilder sb = new StringBuilder();

int gaps = last - index - 1;

// If last line or line with one word - left justify

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

for (int i = index; i < last; i++) {

sb.append(words[i]);

if (i != last - 1) sb.append(' ');

}

// Fill remaining spaces

for (int i = sb.length(); i < width; i++) {

sb.append(' ');

}

} else {

// Distribute spaces evenly

int totalSpaces = width - count + gaps; // count already includes minimum spaces

int spacePerGap = totalSpaces / gaps;

int extraSpaces = totalSpaces % gaps;

for (int i = index; i < last - 1; i++) {

sb.append(words[i]);

sb.append(' ');

for (int s = 0; s < spacePerGap; s++) sb.append(' ');

if (extraSpaces > 0) {

sb.append(' ');

extraSpaces--;

}

}

sb.append(words[last - 1]);

}

lines.add(sb.toString());

index = last;

}

return lines;

}

// Center align text

public static List<String> centerAlign(List<String> lines, int width) {

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

for (String line : lines) {

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

StringBuilder sb = new StringBuilder();

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

sb.append(line.trim());

// Add remaining spaces if needed to make width equal

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

centered.add(sb.toString());

}

return centered;

}

// Format using string concatenation (for performance comparison)

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

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

int index = 0;

while (index < words.length) {

int count = words[index].length();

int last = index + 1;

while (last < words.length) {

if (count + 1 + words[last].length() > width) break;

count += 1 + words[last].length();

last++;

}

String line = "";

int gaps = last - index - 1;

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

for (int i = index; i < last; i++) {

line += words[i];

if (i != last - 1) line += " ";

}

while (line.length() < width) line += " ";

} else {

int totalSpaces = width - count + gaps;

int spacePerGap = totalSpaces / gaps;

int extraSpaces = totalSpaces % gaps;

for (int i = index; i < last - 1; i++) {

line += words[i] + " ";

for (int s = 0; s < spacePerGap; s++) line += " ";

if (extraSpaces > 0) {

line += " ";

extraSpaces--;

}

}

line += words[last - 1];

}

lines.add(line);

index = last;

}

return lines;

}

// Display formatted text with line numbers and char counts

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

System.out.println("\n--- " + title + " ---");

for (int i = 0; i < lines.size(); i++) {

System.out.printf("%2d: %s (%d chars)\n", i + 1, lines.get(i), lines.get(i).length());

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

String text = sc.nextLine();

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

int width = sc.nextInt();

sc.nextLine(); // consume newline

String[] words = splitWords(text);

// Justify with StringBuilder

long startTime = System.nanoTime();

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

long endTime = System.nanoTime();

long durationBuilder = endTime - startTime;

// Justify with String concatenation

startTime = System.nanoTime();

List<String> justifiedConcat = justifyTextConcat(words, width);

endTime = System.nanoTime();

long durationConcat = endTime - startTime;

// Center align based on justified text

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

System.out.println("\nOriginal Text:\n" + text);

displayFormatted(justified, "Left Justified Text");

displayFormatted(centered, "Center Aligned Text");

System.out.println("\nPerformance:");

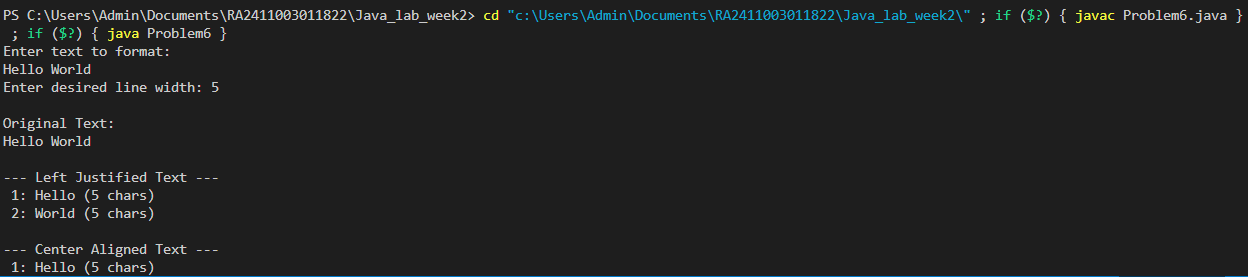
System.out.println("StringBuilder justify time: " + durationBuilder + " ns");

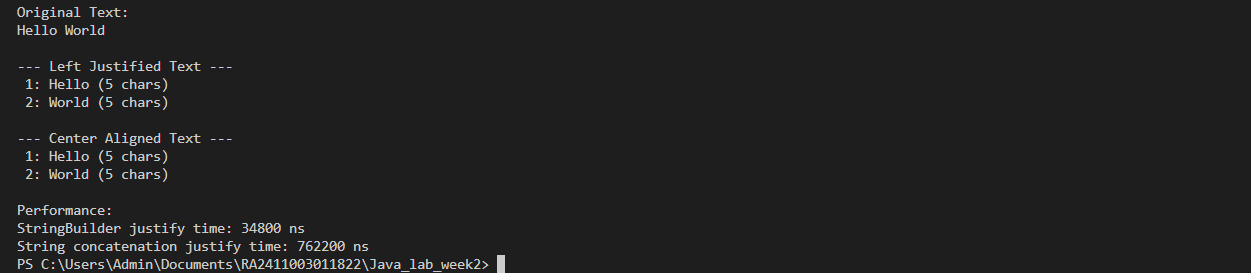
System.out.println("String concatenation justify time: " + durationConcat + " ns");

}

}

Output –





5