# Week 2 - S2 - Lab Problem

**D.Purna Sankar Reddy**

**RA2411028010116**

**Java String Operations and Performance**

**Lab Practice Programs**

**Problem 1: Write a program to find and replace all occurrences of a substring in a text without using the replace() method**

**Hint =>**

1. Take user input using the Scanner nextLine() method for the main text and the substring to find and replace
2. Create a method to find all occurrences of the substring using indexOf() method in a loop and store the starting positions in an array
3. 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
4. Create a method to compare the result with the built-in replace() method and return a boolean
5. The main function calls all user-defined methods and displays both results along with the comparison

**Program:**

import java.util.\*;

public class ReplaceSubstring {

public static int[] findOccurrences(String text, String find) {

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

int index = text.indexOf(find);

while (index >= 0) {

list.add(index);

index = text.indexOf(find, index + 1);

}

return list.stream().mapToInt(i -> i).toArray();

}

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

StringBuilder result = new StringBuilder();

for (int i = 0; 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 compareResults(String original, String manual, String find, String replace) {

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

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

String text = sc.nextLine();

String find = sc.nextLine();

String replace = sc.nextLine();

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

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

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

System.out.println("Match: " + compareResults(text, manual, find, replace));

}

}

**Output:**

hello world hello SRM

hello

hi

Manual Result: hi world hi SRM

Built-in Result: hi world hi SRM

Match: true



**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 =>**

1. Take user input using the Scanner nextLine() method
2. 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
3. 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
4. Create a method for title case conversion:
   * i. Convert the first character of each word to uppercase
   * ii. Convert all other characters to lowercase
5. Create a method to compare results with built-in methods (toUpperCase(), toLowerCase())
6. The main function calls all methods and displays the results in a tabular format

**Program:**

import java.util.\*;

public class CaseConverter {

public static char toUpper(char c) {

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

return c;

}

public static char toLower(char c) {

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

return c;

}

public static String toUpperCaseManual(String s) {

StringBuilder sb = new StringBuilder();

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

return sb.toString();

}

public static String toLowerCaseManual(String s) {

StringBuilder sb = new StringBuilder();

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

return sb.toString();

}

public static String toTitleCase(String s) {

StringBuilder sb = new StringBuilder();

boolean start = true;

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

if (c == ' ') {

sb.append(c);

start = true;

} else if (start) {

sb.append(toUpper(c));

start = false;

} else sb.append(toLower(c));

}

return sb.toString();

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

String text = sc.nextLine();

String upper = toUpperCaseManual(text);

String lower = toLowerCaseManual(text);

String title = toTitleCase(text);

System.out.println("Text\tManual\tBuilt-in");

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

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

System.out.println("Title\t" + title);

}

}

**Output:**

hello SRM

Text Manual Built-in

Upper HELLO SRM HELLO SRM

Lower hello srm hello srm

Title Hello Srm



**Problem 3: Write a program to analyze and compare the performance of String concatenation vs StringBuilder vs StringBuffer for building large strings**

**Hint =>**

1. Take user input for the number of iterations (e.g., 1000, 10000, 100000)
2. 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
3. Create a method to perform StringBuilder operations:
   * i. Use StringBuilder.append() method in a loop
   * ii. Measure the time taken and return results
4. Create a method to perform StringBuffer operations:
   * i. Use StringBuffer.append() method in a loop
   * ii. Measure the time taken and return results
5. Create a method to display performance comparison in a tabular format showing:
   * i. Method used, Time taken (milliseconds), Memory efficiency
6. The main function calls all methods and displays the performance analysis

**Program:**

import java.util.\*;

public class StringPerformance {

public static long concatString(int n) {

long start = System.currentTimeMillis();

String s = "";

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

return System.currentTimeMillis() - start;

}

public static long concatStringBuilder(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 concatStringBuffer(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);

int n = sc.nextInt();

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

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

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

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

}

}

**Output:**

10000

Method Time(ms)

String 31

StringBuilder 0

StringBuffer 1



**Problem 4: Write a program to create a simple encryption and decryption system using ASCII character shifting (Caesar Cipher implementation)**

**Hint =>**

1. Take user input for the text to encrypt and the shift value
2. 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
3. Create a method to decrypt text:
   * i. Reverse the shifting process
   * ii. Handle negative shifts properly
4. Create a method to display ASCII values of characters before and after encryption
5. Create a method to validate that decryption returns the original text
6. 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 CaesarCipher {

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

StringBuilder sb = new StringBuilder();

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

if (Character.isUpperCase(c))

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

else if (Character.isLowerCase(c))

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

else sb.append(c);

}

return sb.toString();

}

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

return encrypt(text, -shift);

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

String text = sc.nextLine();

int shift = sc.nextInt();

String encrypted = encrypt(text, shift);

String decrypted = decrypt(encrypted, shift);

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

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

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

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

}

}

**Output:**

Hello SRM

2

Original: Hello SRM

Encrypted: Jgnnq UTO

Decrypted: Hello SRM

Valid: true



**Problem 5: Write a program to extract and analyze different parts of an email address using substring() and indexOf() methods**

**Hint =>**

1. Take user input for multiple email addresses using Scanner
2. 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
3. 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
4. Create a method to analyze email statistics:
   * i. Count total valid/invalid emails
   * ii. Find most common domain
   * iii. Calculate average username length
5. Create a method to display results in tabular format showing:
   * i. Email, Username, Domain, Domain Name, Extension, Valid/Invalid
6. The main function processes multiple emails and displays analysis results

**Program:**

import java.util.\*;

public class EmailAnalyzer {

public static boolean isValidEmail(String email) {

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

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

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

return at > 0 && at == lastAt && dot > at + 1 && dot < email.length() - 1;

}

public static String[] extractParts(String email) {

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

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

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

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

String domainName = email.substring(at + 1, dot);

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

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

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int n = sc.nextInt();

sc.nextLine();

for (int i = 0; i < n; i++) emails.add(sc.nextLine());

int valid = 0, invalid = 0, totalLen = 0;

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

System.out.println("Email\tUsername\tDomain\tDomainName\tExtension\tValid/Invalid");

for (String e : emails) {

if (isValidEmail(e)) {

valid++;

String[] parts = extractParts(e);

totalLen += parts[0].length();

domainCount.put(parts[1], domainCount.getOrDefault(parts[1], 0) + 1);

System.out.println(e + "\t" + parts[0] + "\t" + parts[1] + "\t" + parts[2] + "\t" + parts[3] + "\tValid");

} else {

invalid++;

System.out.println(e + "\t-\t-\t-\t-\tInvalid");

}

}

String mostCommonDomain = domainCount.entrySet().stream().max(Map.Entry.comparingByValue()).map(Map.Entry::getKey).orElse("None");

System.out.println("Valid: " + valid + ", Invalid: " + invalid);

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

if (valid > 0) System.out.println("Average Username Length: " + (totalLen / valid));

}

}

**Output:**

2

purna@gmail.com

pd@srmist.edu

Email Username Domain DomainName Extension Valid/Invalid

purna@gmail.com purna gmail.com gmail com Valid

pd@srmist.edu pd srmist.edu srmist edu Valid

Valid: 2, Invalid: 0

Most Common Domain: srmist.edu

Average Username Length: 3



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

**Hint =>**

1. Take user input for the text to format and desired line width
2. 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
3. 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)
4. Create a method to center-align text:
   * i. Calculate padding needed on both sides
   * ii. Use StringBuilder to build centered lines
5. Create a method to compare performance:
   * i. Implement the same formatting using String concatenation ● ii. Measure time difference using System.nanoTime()
6. Create a method to display the formatted text with:
   * i. Line numbers
   * ii. Character count per line
   * iii. Performance comparison results
7. 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 {

public static List<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 (i > start) words.add(text.substring(start, i));

start = i + 1;

}

}

if (start < text.length()) words.add(text.substring(start));

return words;

}

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

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

int i = 0;

while (i < words.size()) {

int j = i, lineLen = 0;

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

lineLen += words.get(j).length();

j++;

}

int gaps = j - i - 1;

StringBuilder sb = new StringBuilder();

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

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

sb.append(words.get(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.get(k));

if (k < j - 1) {

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

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

}

}

}

lines.add(sb.toString());

i = j;

}

return lines;

}

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

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

StringBuilder sb = new StringBuilder();

for (String word : words) {

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

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

lines.add(" ".repeat(padding) + sb.toString());

sb = new StringBuilder();

}

if (sb.length() > 0) sb.append(" ");

sb.append(word);

}

if (sb.length() > 0) {

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

lines.add(" ".repeat(padding) + sb.toString());

}

return lines;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

String text = sc.nextLine();

int width = sc.nextInt();

List<String> words = splitWords(text);

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

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

System.out.println("Justified:");

int lineNum = 1;

for (String line : justified) System.out.println(lineNum++ + ": " + line + " (" + line.length() + ")");

System.out.println("Centered:");

lineNum = 1;

for (String line : centered) System.out.println(lineNum++ + ": " + line + " (" + line.length() + ")");

}

}

**Output:**

Hello world welcome to SRM

10

Justified:

1: Hello (10)

2: world (10)

3: welcome to (10)

4: SRM (10)

Centered:

1: Hello (7)

2: world (7)

3: welcome to (10)

4: SRM (6)