

�� **PRACTICE PROBLEM 1: (Any 4)**

**Built-In String Methods - Basic Operations**

**Task:** Create a program that demonstrates common String methods for text analysis and manipulation.

public class StringBuiltInMethods {

public static void main(String[] args) {

String sampleText = " Java Programming is Fun and Challenging! ";

// TODO: Use built-in methods to perform the following operations: // 1. Display original string length including spaces

// 2. Remove leading and trailing spaces, show new length // 3. Find and display the character at index 5

// 4. Extract substring "Programming" from the text

// 5. Find the index of the word "Fun"

// 6. Check if the string contains "Java" (case-sensitive) // 7. Check if the string starts with "Java" (after trimming) // 8. Check if the string ends with an exclamation mark

// 9. Convert the entire string to uppercase

// 10. Convert the entire string to lowercase

// TODO: Create a method that counts vowels using charAt() // TODO: Create a method that finds all occurrences of a character // TODO: Display all results in a formatted manner

}

// TODO: Method to count vowels in a string

public static int countVowels(String text) {

// Your code here

}

// TODO: Method to find all positions of a character

public static void findAllOccurrences(String text, char target) { // Your code here

}

}

Program-

public class StringBuiltinMethods {

public static void main(String[] args) {

String sampleText = " Java Programming is Fun and Challenging! ";

// 1. Display original string length including spaces

System.out.println("Original String: '" + sampleText + "'");

System.out.println("Original Length (with spaces): " + sampleText.length());

// 2. Remove leading and trailing spaces, show new length

String trimmed = sampleText.trim();

System.out.println("Trimmed String: '" + trimmed + "'");

System.out.println("Length after trim: " + trimmed.length());

// 3. Find and display the character at index 5

char ch = sampleText.charAt(5);

System.out.println("Character at index 5: " + ch);

// 4. Extract substring "Programming" from the text

String sub = trimmed.substring(5, 16); // "Programming"

System.out.println("Extracted Substring: " + sub);

// --- Extra Tasks ---

// Count vowels

int vowelCount = countVowels(trimmed);

System.out.println("Total Vowels: " + vowelCount);

// Find all occurrences of a character (e.g., 'a')

System.out.println("Occurrences of 'a':");

findAllOccurrences(trimmed, 'a');

}

// Method to count vowels in a string

public static int countVowels(String text) {

int count = 0;

String vowels = "aeiouAEIOU";

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

if (vowels.indexOf(text.charAt(i)) != -1) {

count++;

}

}

return count;

}

// Method to find all positions of a character

public static void findAllOccurrences(String text, char target) {

boolean found = false;

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

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

System.out.println("Character '" + target + "' found at index: " + i);

found = true;

}

}

if (!found) {

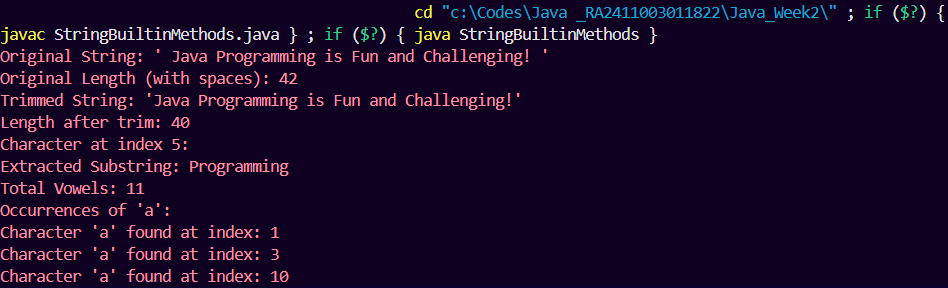
System.out.println("Character '" + target + "' not found.");

}

}

}

Output –



�� **PRACTICE PROBLEM 2:**

**String Manipulation Methods**

**Task:** Create a text processing utility that uses various string manipulation methods. import java.util.Scanner;

public class StringManipulation {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// TODO: Ask user to enter a sentence with mixed formatting // TODO: Process the input using the following methods: // 1. trim() - Remove extra spaces

// 2. replace() - Replace all spaces with underscores // 3. replaceAll() - Remove all digits using regex

// 4. split() - Split sentence into words array

// 5. join() - Rejoin words with " | " separator

// TODO: Create additional processing methods:

// - Remove all punctuation

// - Capitalize first letter of each word

// - Reverse the order of words

// - Count word frequency

scanner.close();

}

// TODO: Method to remove punctuation

public static String removePunctuation(String text) { // Your code here

}

// TODO: Method to capitalize each word

public static String capitalizeWords(String text) { // Your code here

}

// TODO: Method to reverse word order

2



public static String reverseWordOrder(String text) {

// Your code here

}

// TODO: Method to count word frequency

public static void countWordFrequency(String text) {

// Your code here

}

}

Program –

import java.util.\*;

public class StringManipulation {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Ask user to enter a sentence with mixed formatting

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

String input = scanner.nextLine();

// 1. trim() - Remove extra spaces

String trimmed = input.trim();

System.out.println("Trimmed: " + trimmed);

// 2. replace() - Replace all spaces with underscores

String replacedSpaces = trimmed.replace(" ", "\_");

System.out.println("Spaces replaced with underscores: " + replacedSpaces);

// 3. replaceAll() - Remove all digits using regex

String noDigits = trimmed.replaceAll("[0-9]", "");

System.out.println("Without digits: " + noDigits);

// 4. split() - Split sentence into words array

String[] words = trimmed.split("\\s+");

System.out.println("Words array: " + Arrays.toString(words));

// 5. join() - Rejoin words with " | " separator

String joined = String.join(" | ", words);

System.out.println("Rejoined with | : " + joined);

// --- Extra Processing ---

System.out.println("\n--- Extra Processing ---");

// Remove punctuation

String noPunct = removePunctuation(trimmed);

System.out.println("Without punctuation: " + noPunct);

// Capitalize first letter of each word

String capitalized = capitalizeWords(noPunct);

System.out.println("Capitalized Words: " + capitalized);

// Reverse word order

String reversed = reverseWordOrder(noPunct);

System.out.println("Reversed Word Order: " + reversed);

// Count word frequency

System.out.println("Word Frequency:");

countWordFrequency(noPunct);

scanner.close();

}

// Method to remove punctuation

public static String removePunctuation(String text) {

return text.replaceAll("[^a-zA-Z0-9\\s]", "");

}

// Method to capitalize each word

public static String capitalizeWords(String text) {

String[] words = text.split("\\s+");

String result = "";

for (String word : words) {

if (!word.isEmpty()) {

result += Character.toUpperCase(word.charAt(0))

+ word.substring(1).toLowerCase() + " ";

}

}

return result.trim();

}

// Method to reverse word order

public static String reverseWordOrder(String text) {

String[] words = text.split("\\s+");

String result = "";

for (int i = words.length - 1; i >= 0; i--) {

result += words[i] + " ";

}

return result.trim();

}

// Method to count word frequency (without Map, using array)

public static void countWordFrequency(String text) {

String[] words = text.toLowerCase().split("\\s+");

boolean[] counted = new boolean[words.length];

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

if (!counted[i]) {

int count = 1;

for (int j = i + 1; j < words.length; j++) {

if (words[i].equals(words[j])) {

count++;

counted[j] = true; // mark as counted

}

}

System.out.println(words[i] + " → " + count);

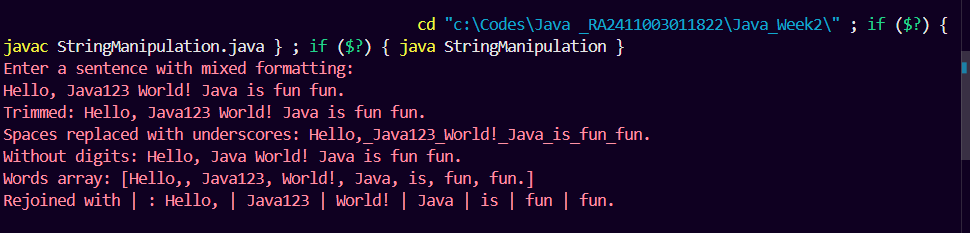
}

}

}

}

Output –



�� **PRACTICE PROBLEM 3:**

**ASCII Codes and Character Conversion**

**Task:** Create a program that demonstrates ASCII character manipulation and conversion. import java.util.Scanner;

public class ASCIIProcessor {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// TODO: Ask user to enter a string

// TODO: For each character in the string:

// 1. Display the character and its ASCII code

// 2. Determine if it's uppercase, lowercase, digit, or special character

// 3. If letter, show both upper and lower case versions with ASCII codes

// 4. Calculate the difference between upper and lower case ASCII values

// TODO: Create ASCII art using character codes

// TODO: Implement a simple Caesar cipher using ASCII manipulation

scanner.close();

}

3



// TODO: Method to classify character type

public static String classifyCharacter(char ch) {

// Return "Uppercase Letter", "Lowercase Letter", "Digit", or "Special Character"

// Your code here

}

// TODO: Method to convert case using ASCII manipulation public static char toggleCase(char ch) {

// Convert upper to lower and lower to upper using ASCII values // Your code here

}

// TODO: Method to implement Caesar cipher

public static String caesarCipher(String text, int shift) { // Shift each letter by 'shift' positions in ASCII

// Your code here

}

// TODO: Method to create ASCII table for a range

public static void displayASCIITable(int start, int end) { // Display ASCII codes and corresponding characters

// Your code here

}

// TODO: Method to convert string to ASCII array

public static int[] stringToASCII(String text) {

// Your code here

}

// TODO: Method to convert ASCII array back to string

public static String asciiToString(int[] asciiValues) { // Your code here

}

}

4



Program –

import java.util.\*;

public class ASCIIProcessor {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Ask user to enter a string

System.out.print("Enter a string: ");

String input = scanner.nextLine();

System.out.println("\n--- Character Analysis ---");

// For each character in the string

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

char ch = input.charAt(i);

int ascii = (int) ch;

// 1. Display character and ASCII code

System.out.println("Character: '" + ch + "' ASCII: " + ascii);

// 2. Determine type

String type = classifyCharacter(ch);

System.out.println("Type: " + type);

// 3. If letter, show both upper and lower case with ASCII codes

if (Character.isLetter(ch)) {

char upper = Character.toUpperCase(ch);

char lower = Character.toLowerCase(ch);

System.out.println("Uppercase: " + upper + " (ASCII: " + (int) upper + ")");

System.out.println("Lowercase: " + lower + " (ASCII: " + (int) lower + ")");

// 4. Difference between ASCII values

System.out.println("Difference (Upper - Lower): " + ((int) upper - (int) lower));

}

System.out.println();

}

// Create ASCII table for a range

System.out.println("\n--- ASCII Table (65 to 90) ---");

displayASCIITable(65, 90); // A–Z

// Convert string to ASCII array and back

int[] asciiArray = stringToASCII(input);

System.out.println("\nString to ASCII Array: " + Arrays.toString(asciiArray));

String backToString = asciiToString(asciiArray);

System.out.println("ASCII Array back to String: " + backToString);

// Simple Caesar cipher

System.out.println("\n--- Caesar Cipher (shift = 3) ---");

String ciphered = caesarCipher(input, 3);

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

String deciphered = caesarCipher(ciphered, -3);

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

scanner.close();

}

// Method to classify character type

public static String classifyCharacter(char ch) {

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

return "Uppercase Letter";

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

return "Lowercase Letter";

} else if (ch >= '0' && ch <= '9') {

return "Digit";

} else {

return "Special Character";

}

}

// Method to convert case using ASCII manipulation

public static char toggleCase(char ch) {

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

return (char) (ch + 32); // convert to lowercase

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

return (char) (ch - 32); // convert to uppercase

} else {

return ch; // unchanged

}

}

// Method to implement Caesar cipher

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

String result = "";

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

char ch = text.charAt(i);

if (Character.isUpperCase(ch)) {

char c = (char) ((ch - 'A' + shift + 26) % 26 + 'A');

result += c;

} else if (Character.isLowerCase(ch)) {

char c = (char) ((ch - 'a' + shift + 26) % 26 + 'a');

result += c;

} else {

result += ch; // keep digits and special chars same

}

}

return result;

}

// Method to create ASCII table for a range

public static void displayASCIITable(int start, int end) {

for (int i = start; i <= end; i++) {

System.out.println(i + " -> " + (char) i);

}

}

// Method to convert string to ASCII array

public static int[] stringToASCII(String text) {

int[] asciiValues = new int[text.length()];

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

asciiValues[i] = (int) text.charAt(i);

}

return asciiValues;

}

// Method to convert ASCII array back to string

public static String asciiToString(int[] asciiValues) {

String result = "";

for (int value : asciiValues) {

result += (char) value;

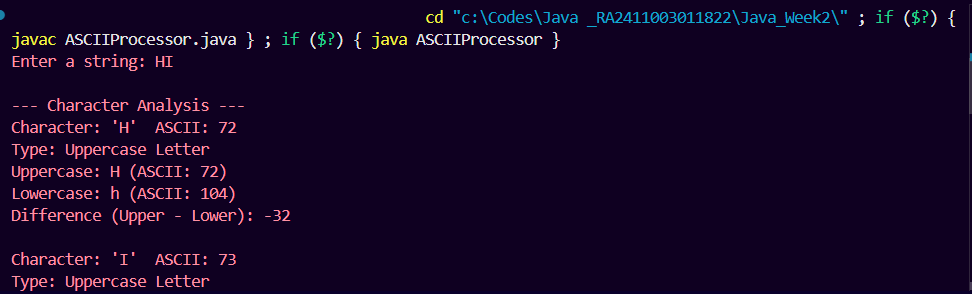
}

return result;

}

}

Output –



�� **PRACTICE PROBLEM 4:**

**StringBuilder, StringBuffer, and Performance**

**Task:** Create a performance comparison program that demonstrates the differences between String, StringBuilder, and StringBuffer.

public class StringPerformanceComparison {

public static void main(String[] args) {

// TODO: Implement performance tests for different approaches

// Test 1: String concatenation performance

System.out.println("=== PERFORMANCE COMPARISON ===");

// TODO: Test string concatenation with regular String (slow method)

long startTime = System.nanoTime();

String result1 = concatenateWithString(1000);

long endTime = System.nanoTime();

System.out.println("String concatenation time: " + (endTime - startTime) + " ns");

// TODO: Test string concatenation with StringBuilder (fast method) // TODO: Test string concatenation with StringBuffer (thread-safe method)

// TODO: Compare memory usage (approximate)

// TODO: Demonstrate thread safety differences

// TODO: Create practical examples showing when to use each approach

}

// TODO: Method using String concatenation (inefficient)

public static String concatenateWithString(int iterations) { String result = "";

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

result += "Java " + i + " ";

}

5



}

return result;

// TODO: Method using StringBuilder (efficient, not thread-safe) public static String concatenateWithStringBuilder(int iterations) { // Your code here

}

// TODO: Method using StringBuffer (efficient, thread-safe) public static String concatenateWithStringBuffer(int iterations) { // Your code here

}

// TODO: Method to demonstrate StringBuilder methods

public static void demonstrateStringBuilderMethods() { StringBuilder sb = new StringBuilder("Hello World");

// TODO: Use the following StringBuilder methods:

// 1. append() - Add text to end

// 2. insert() - Insert text at specific position

// 3. delete() - Remove characters from range

// 4. deleteCharAt() - Remove character at index

// 5. reverse() - Reverse the string

// 6. replace() - Replace substring

// 7. setCharAt() - Change character at index

// 8. capacity() - Show current capacity

// 9. ensureCapacity() - Set minimum capacity

// 10. trimToSize() - Reduce capacity to current length

// Your code here

}

// TODO: Method to demonstrate StringBuffer thread safety public static void demonstrateThreadSafety() {

// Create multiple threads that modify the same StringBuffer // Show that StringBuffer is thread-safe while StringBuilder is not // Your code here

}

// TODO: Method to compare string comparison methods

6



public static void compareStringComparisonMethods() {

String str1 = "Hello";

String str2 = "Hello";

String str3 = new String("Hello");

// TODO: Compare using:

// 1. == operator (reference comparison)

// 2. equals() method (content comparison)

// 3. equalsIgnoreCase() method

// 4. compareTo() method (lexicographic comparison)

// 5. compareToIgnoreCase() method

// TODO: Explain the differences and when to use each

// Your code here

}

// TODO: Method to demonstrate memory efficiency

public static void demonstrateMemoryEfficiency() {

// TODO: Show memory usage before and after different string operations

// TODO: Demonstrate string pool behavior

// TODO: Show StringBuilder capacity management

// Your code here

}

}

Program-

public class StringPerformanceComparison {

public static void main(String[] args) {

System.out.println("=== PERFORMANCE COMPARISON ===");

// Test 1: String concatenation (slow method)

long startTime = System.nanoTime();

String result1 = concatenateWithString(1000);

long endTime = System.nanoTime();

System.out.println("String concatenation time: " + (endTime - startTime) + " ns");

// Test 2: StringBuilder (faster method, not thread-safe)

startTime = System.nanoTime();

String result2 = concatenateWithStringBuilder(1000);

endTime = System.nanoTime();

System.out.println("StringBuilder concatenation time: " + (endTime - startTime) + " ns");

// Test 3: StringBuffer (fast and thread-safe)

startTime = System.nanoTime();

String result3 = concatenateWithStringBuffer(1000);

endTime = System.nanoTime();

System.out.println("StringBuffer concatenation time: " + (endTime - startTime) + " ns");

// Demonstrating StringBuilder methods

demonstrateStringBuilderMethods();

// Demonstrating StringBuffer thread safety

demonstrateThreadSafety();

// Demonstrating different string comparison methods

compareStringComparisonMethods();

// Demonstrating memory efficiency

demonstrateMemoryEfficiency();

}

// String concatenation using +

public static String concatenateWithString(int iterations) {

String result = "";

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

result += "Java " + i + " ";

}

return result;

}

// Using StringBuilder

public static String concatenateWithStringBuilder(int iterations) {

StringBuilder sb = new StringBuilder();

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

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

}

return sb.toString();

}

// Using StringBuffer

public static String concatenateWithStringBuffer(int iterations) {

StringBuffer sb = new StringBuffer();

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

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

}

return sb.toString();

}

// Demonstrating StringBuilder methods

public static void demonstrateStringBuilderMethods() {

System.out.println("\n=== StringBuilder Methods ===");

StringBuilder sb = new StringBuilder("Hello World");

sb.append("!!!"); // add at end

System.out.println("append: " + sb);

sb.insert(6, "Java "); // insert at index

System.out.println("insert: " + sb);

sb.delete(6, 11); // delete characters

System.out.println("delete: " + sb);

sb.deleteCharAt(0); // delete char at index

System.out.println("deleteCharAt: " + sb);

sb.reverse(); // reverse string

System.out.println("reverse: " + sb);

sb.reverse(); // reverse back

sb.replace(0, 5, "Hi"); // replace substring

System.out.println("replace: " + sb);

sb.setCharAt(0, 'h'); // change char

System.out.println("setCharAt: " + sb);

System.out.println("capacity: " + sb.capacity());

sb.ensureCapacity(50);

System.out.println("ensureCapacity(50): " + sb.capacity());

sb.trimToSize();

System.out.println("trimToSize: " + sb.capacity());

}

// Demonstrating thread safety

public static void demonstrateThreadSafety() {

System.out.println("\n=== Thread Safety Demo ===");

StringBuffer buffer = new StringBuffer("Start");

Runnable task = () -> {

for (int i = 0; i < 5; i++) {

buffer.append("X");

}

};

Thread t1 = new Thread(task);

Thread t2 = new Thread(task);

t1.start();

t2.start();

try {

t1.join();

t2.join();

} catch (InterruptedException e) {

e.printStackTrace();

}

System.out.println("Final StringBuffer content (thread-safe): " + buffer);

}

// Comparing string comparison methods

public static void compareStringComparisonMethods() {

System.out.println("\n=== String Comparison Methods ===");

String str1 = "Hello";

String str2 = "Hello";

String str3 = new String("Hello");

System.out.println("== comparison (str1 == str2): " + (str1 == str2));

System.out.println("== comparison (str1 == str3): " + (str1 == str3));

System.out.println("equals(): " + str1.equals(str3));

System.out.println("equalsIgnoreCase(): " + str1.equalsIgnoreCase("hello"));

System.out.println("compareTo(): " + str1.compareTo("World"));

System.out.println("compareToIgnoreCase(): " + str1.compareToIgnoreCase("HELLO"));

}

// Demonstrating memory efficiency

public static void demonstrateMemoryEfficiency() {

System.out.println("\n=== Memory Efficiency ===");

String s1 = "Java";

String s2 = "Java"; // same reference from string pool

String s3 = new String("Java"); // new object

System.out.println("s1 == s2 (pool): " + (s1 == s2));

System.out.println("s1 == s3 (different objects): " + (s1 == s3));

System.out.println("s1.equals(s3): " + s1.equals(s3));

StringBuilder sb = new StringBuilder("Hello");

System.out.println("Initial capacity: " + sb.capacity());

sb.append(" This is a long text to increase capacity...");

System.out.println("Increased capacity: " + sb.capacity());

}

}

Output –

