**Week 2 – S2 – Lab Solution**

**Name: Ramesh Harisabapathi Chettiar**

**Roll Number: RA2411030010263**

**Course: Networking and Communications**

**Semester: 3**

**Date of Submission: 23/08/2025**

**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**

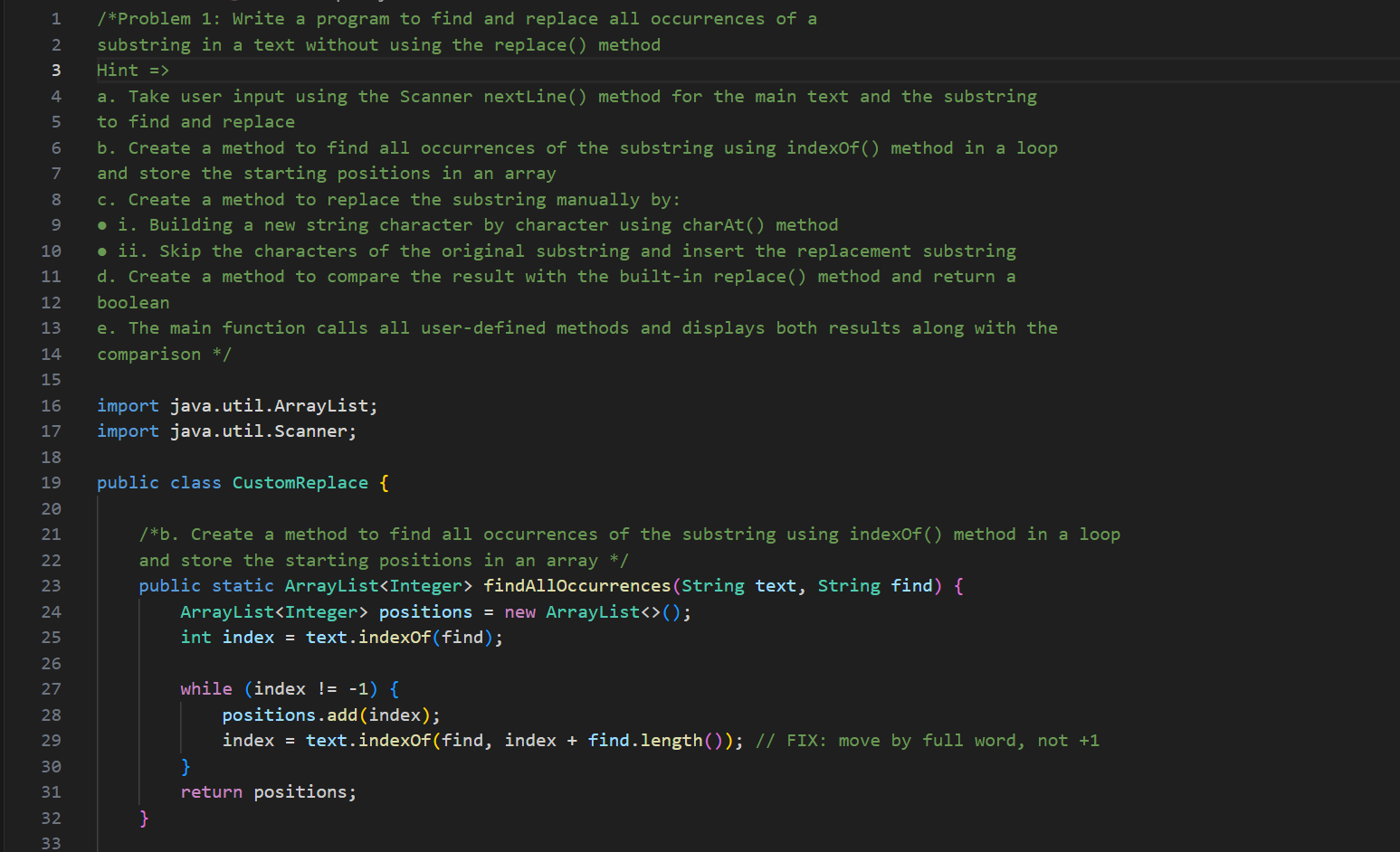
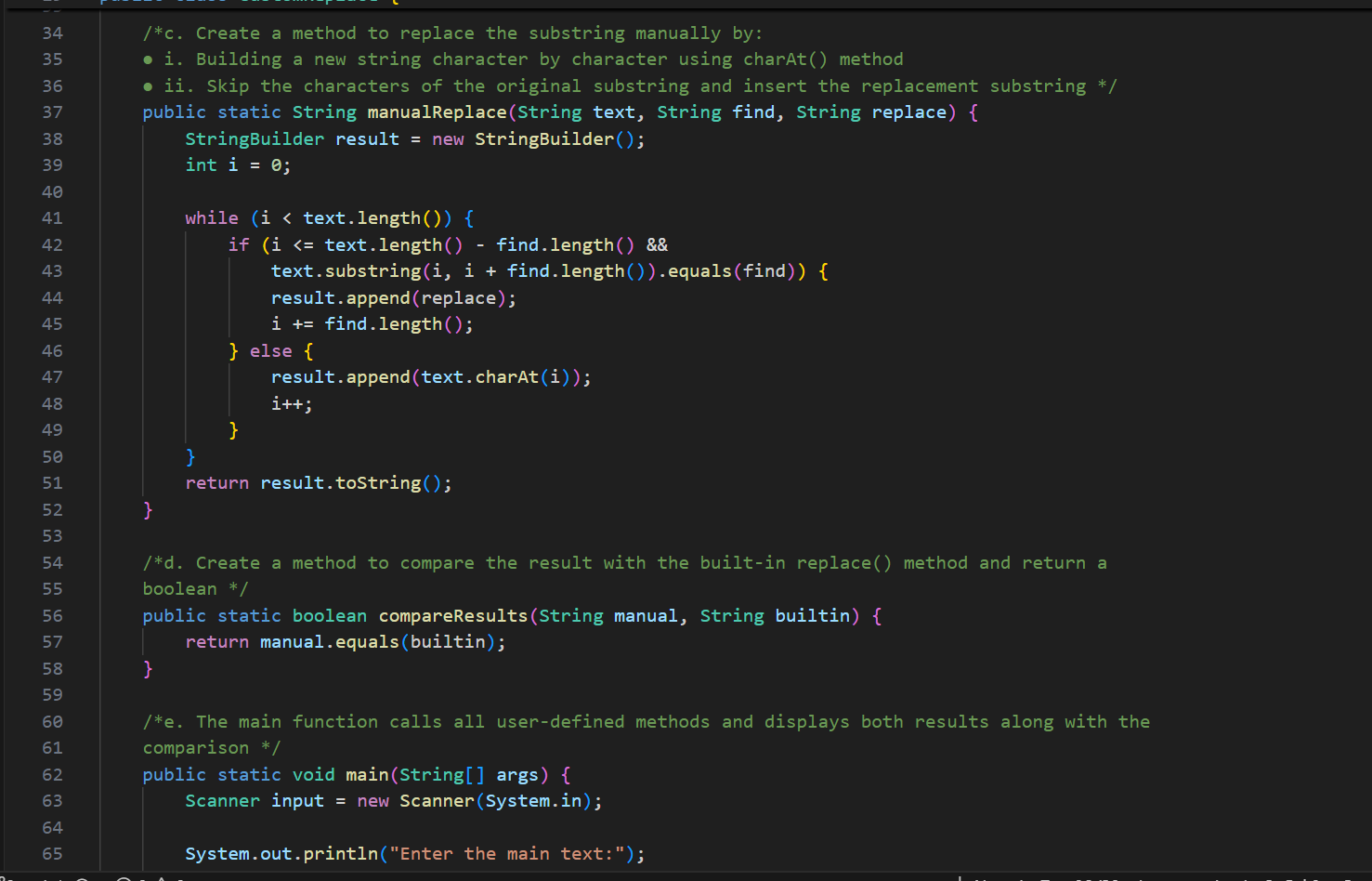
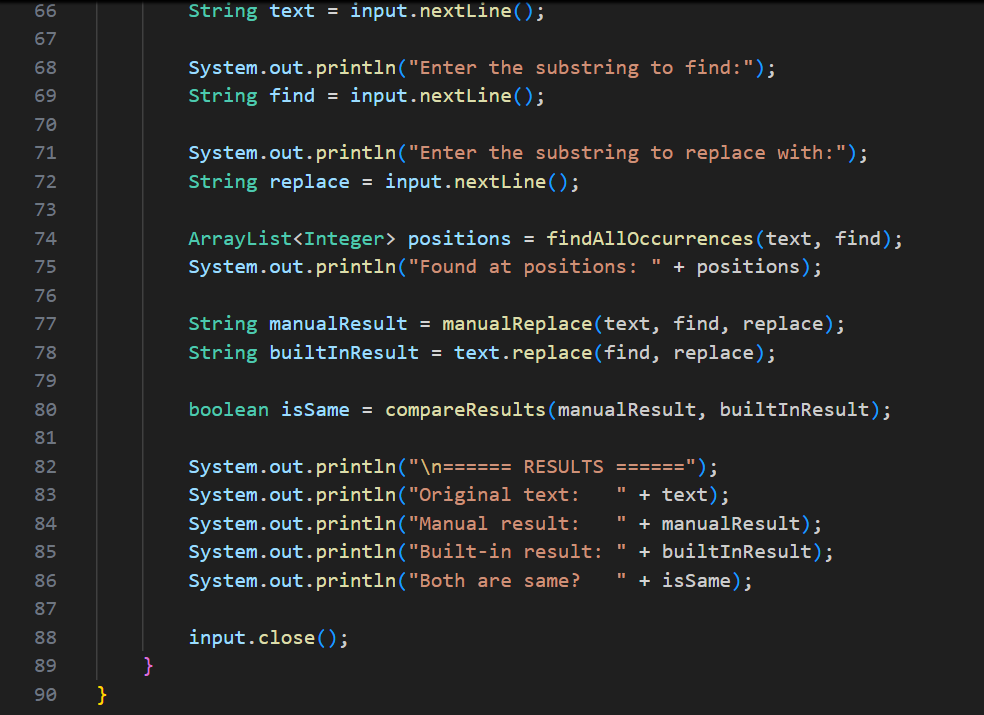
**● ii. Skip the characters of the original substring and insert the replacement substring**

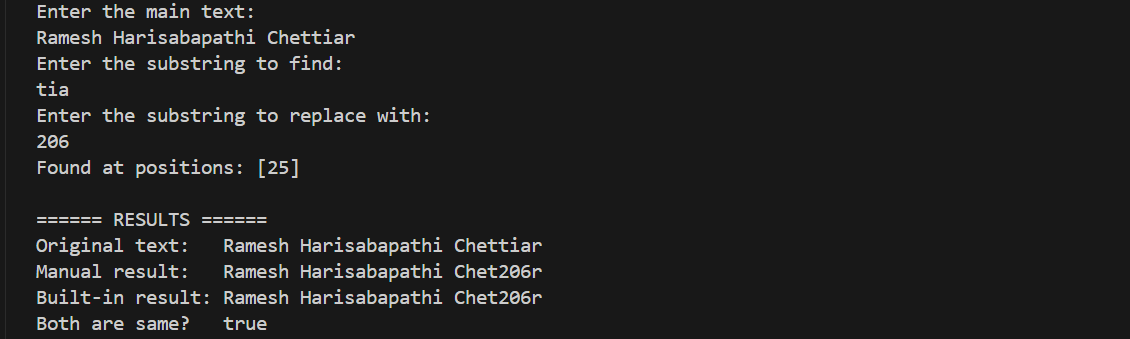
**d. Create a method to compare the result with the built-in replace() method and return a**

**boolean**

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

**comparison**

**** **** 

**OUTPUT🡪** ****

**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:**

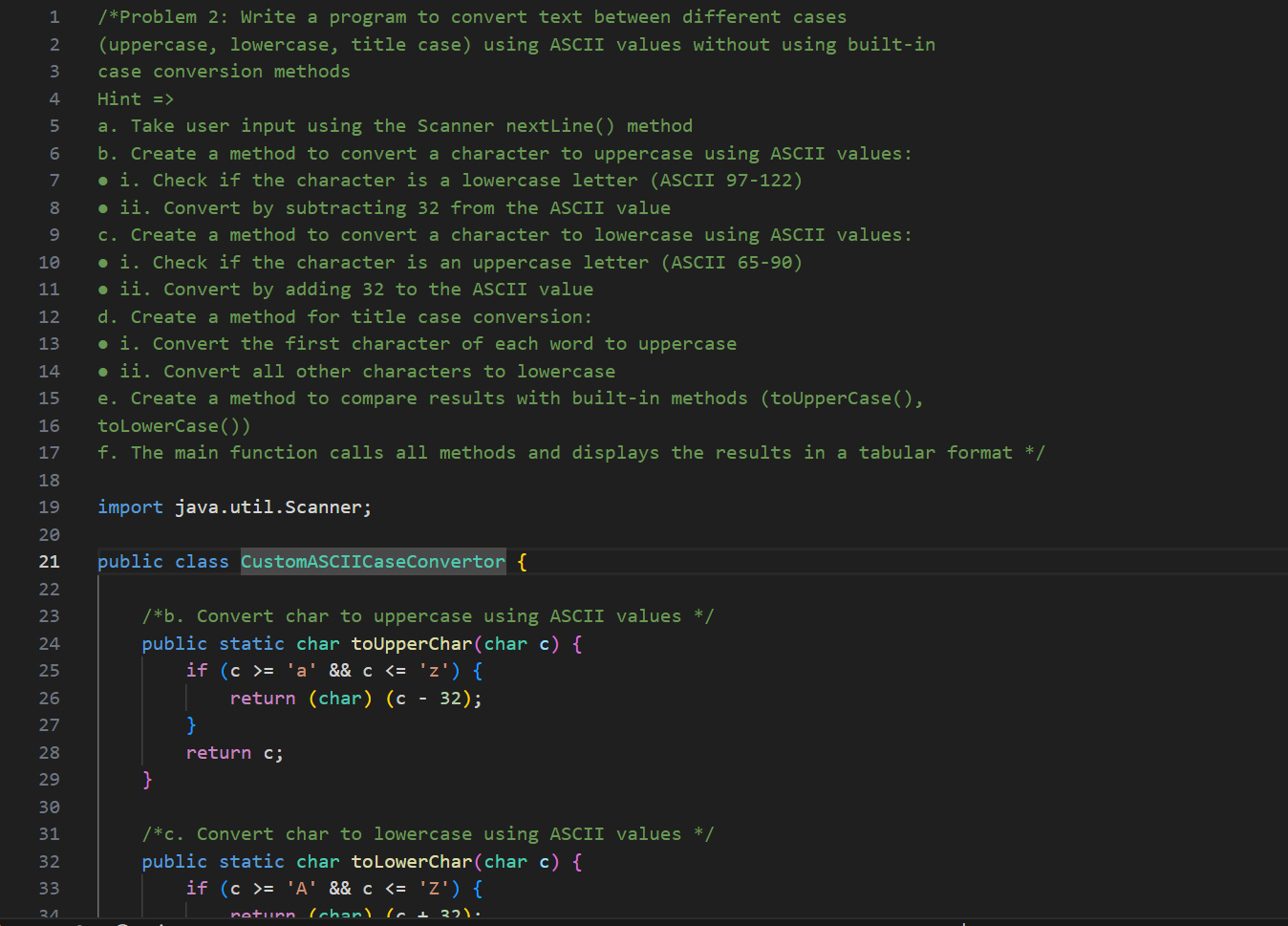
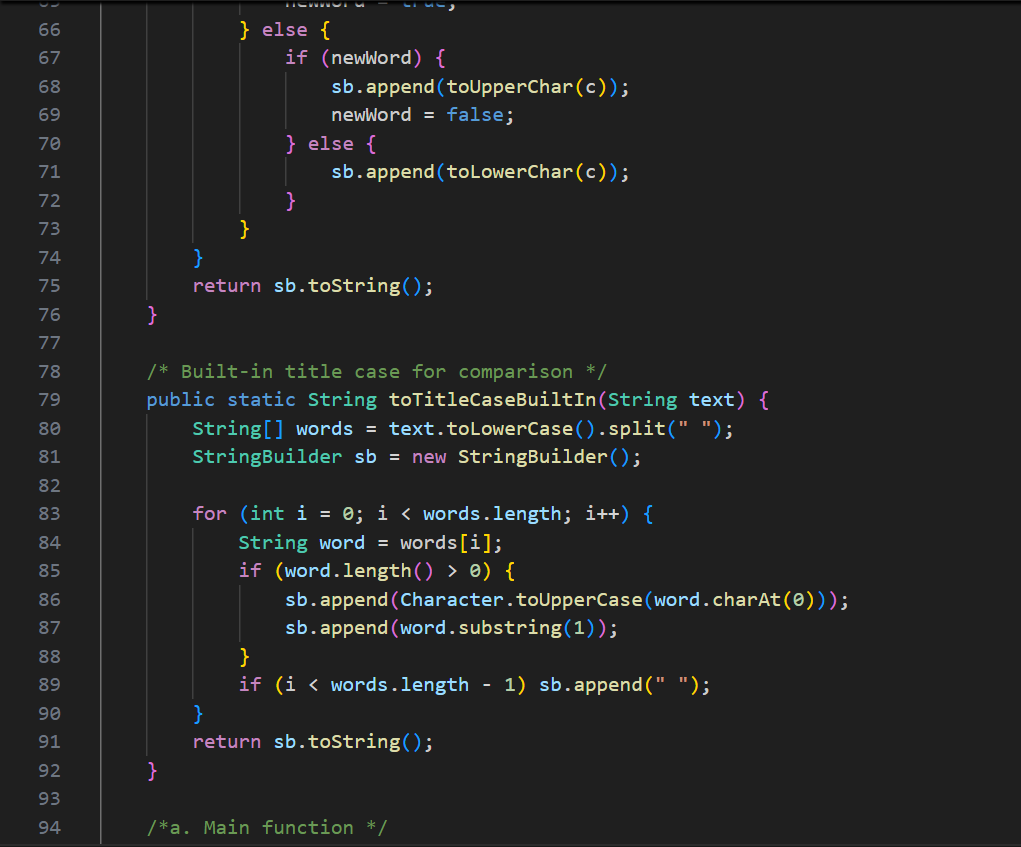
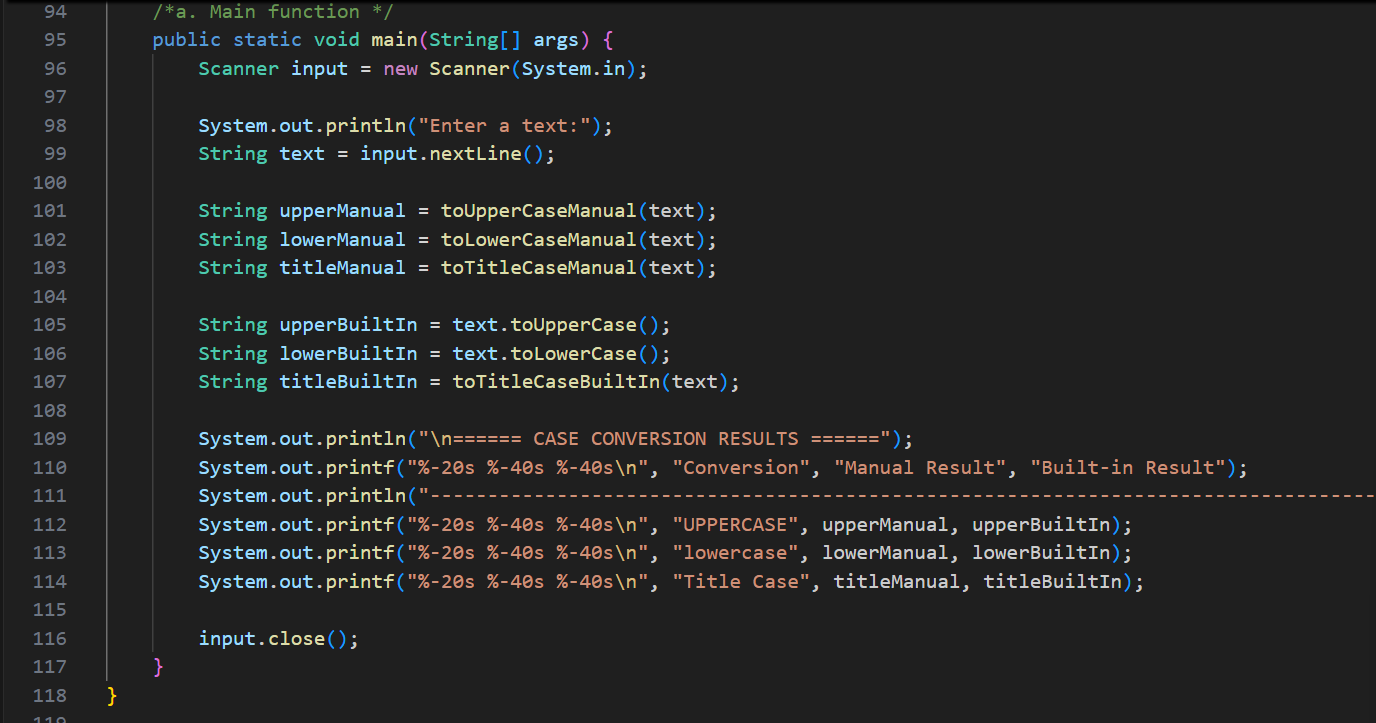
**● i. Convert the first character of each word to uppercase**

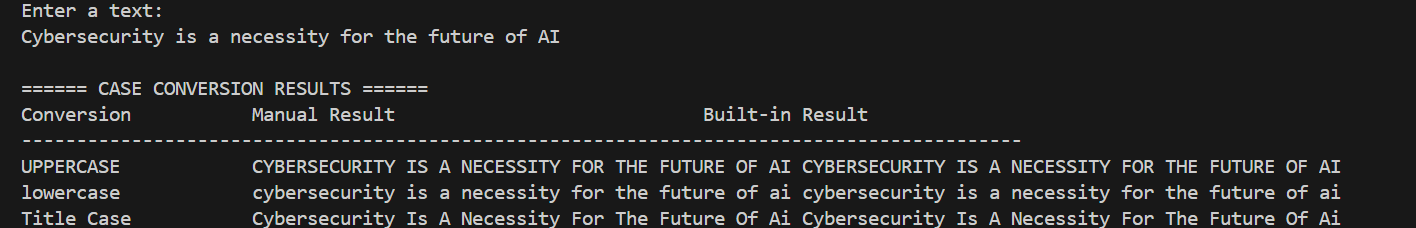
**● ii. Convert all other characters to lowercase**

**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**

**** ****  

**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**

**● 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:**

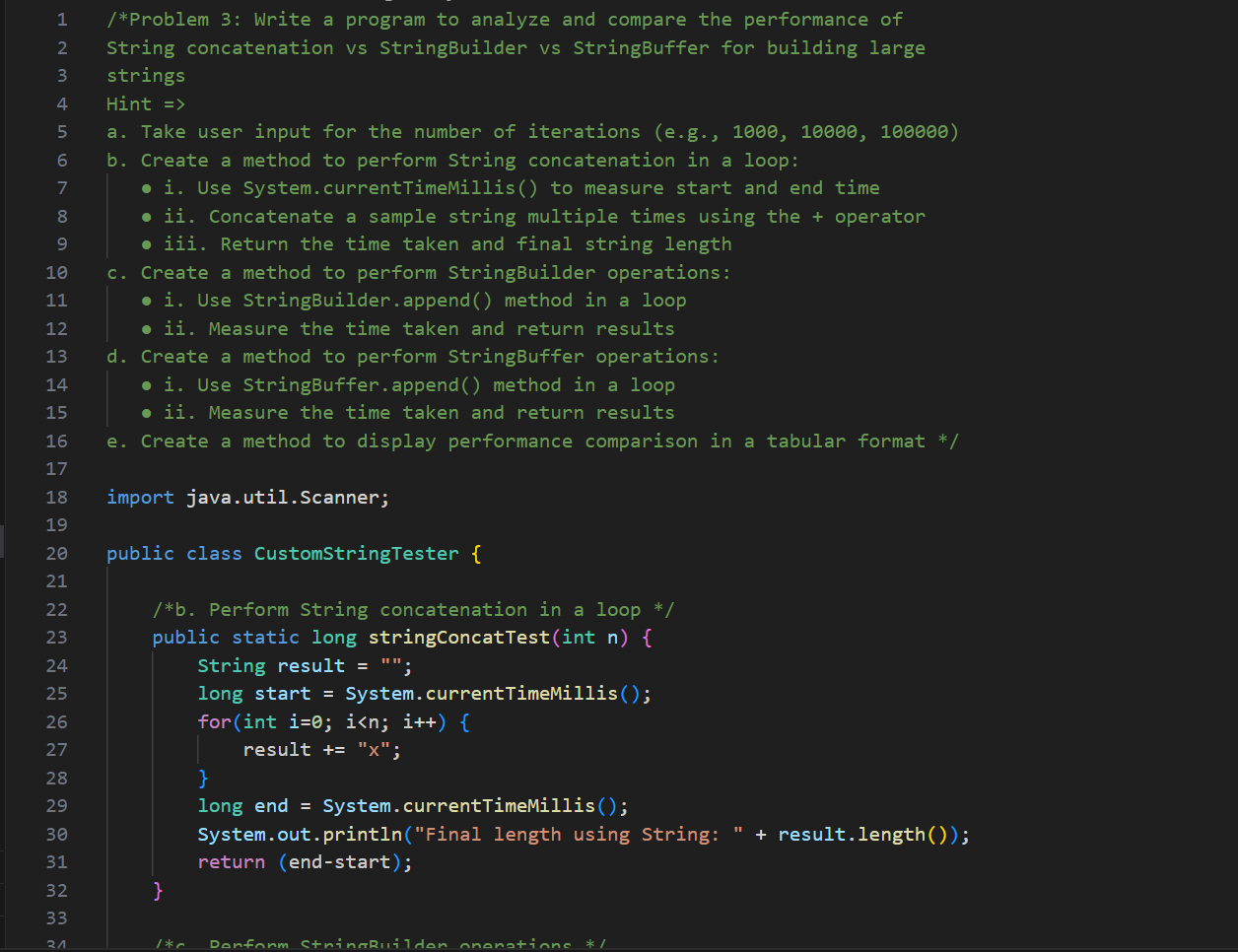
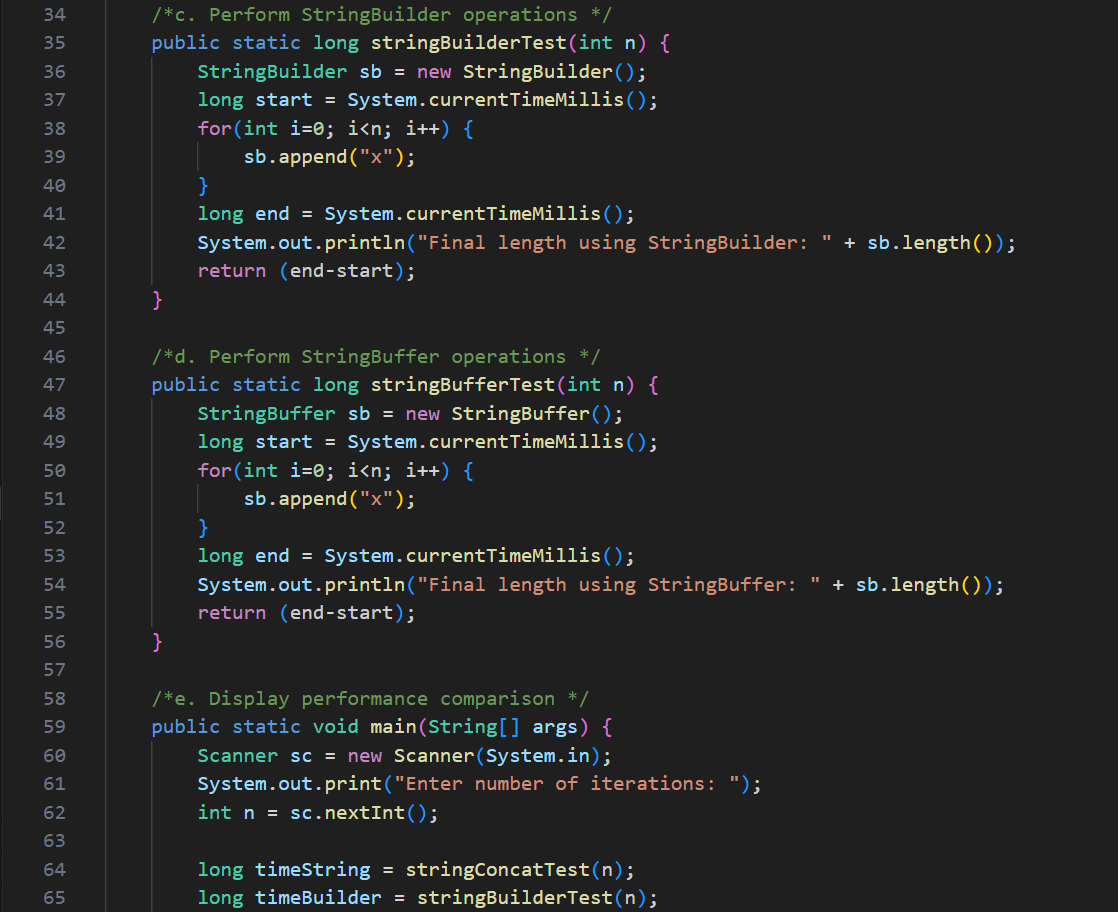
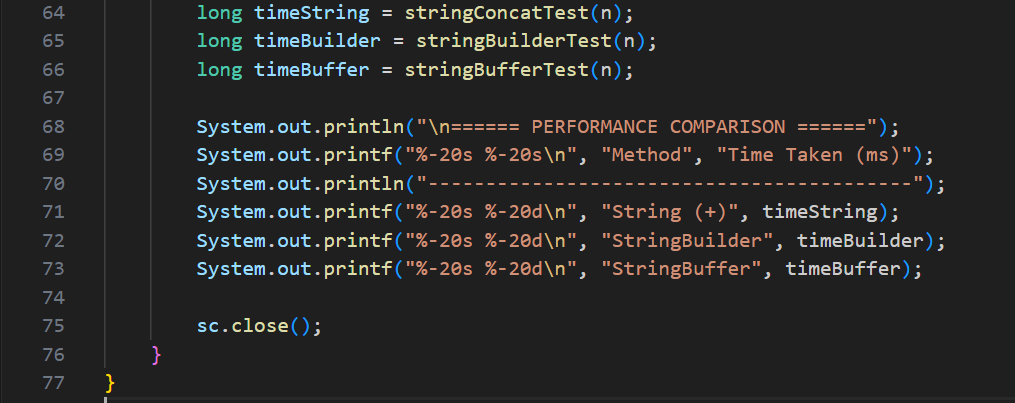
**● i. Use StringBuffer.append() method in a loop**

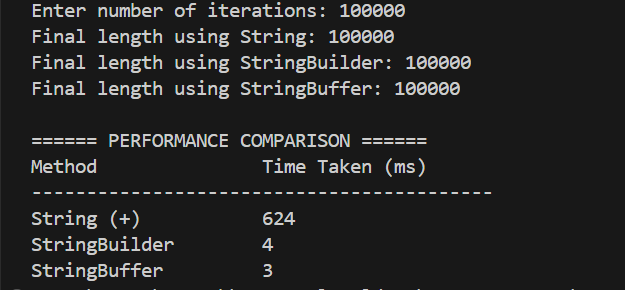
**● ii. Measure the time taken and return results**

**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**

**** **** 

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**

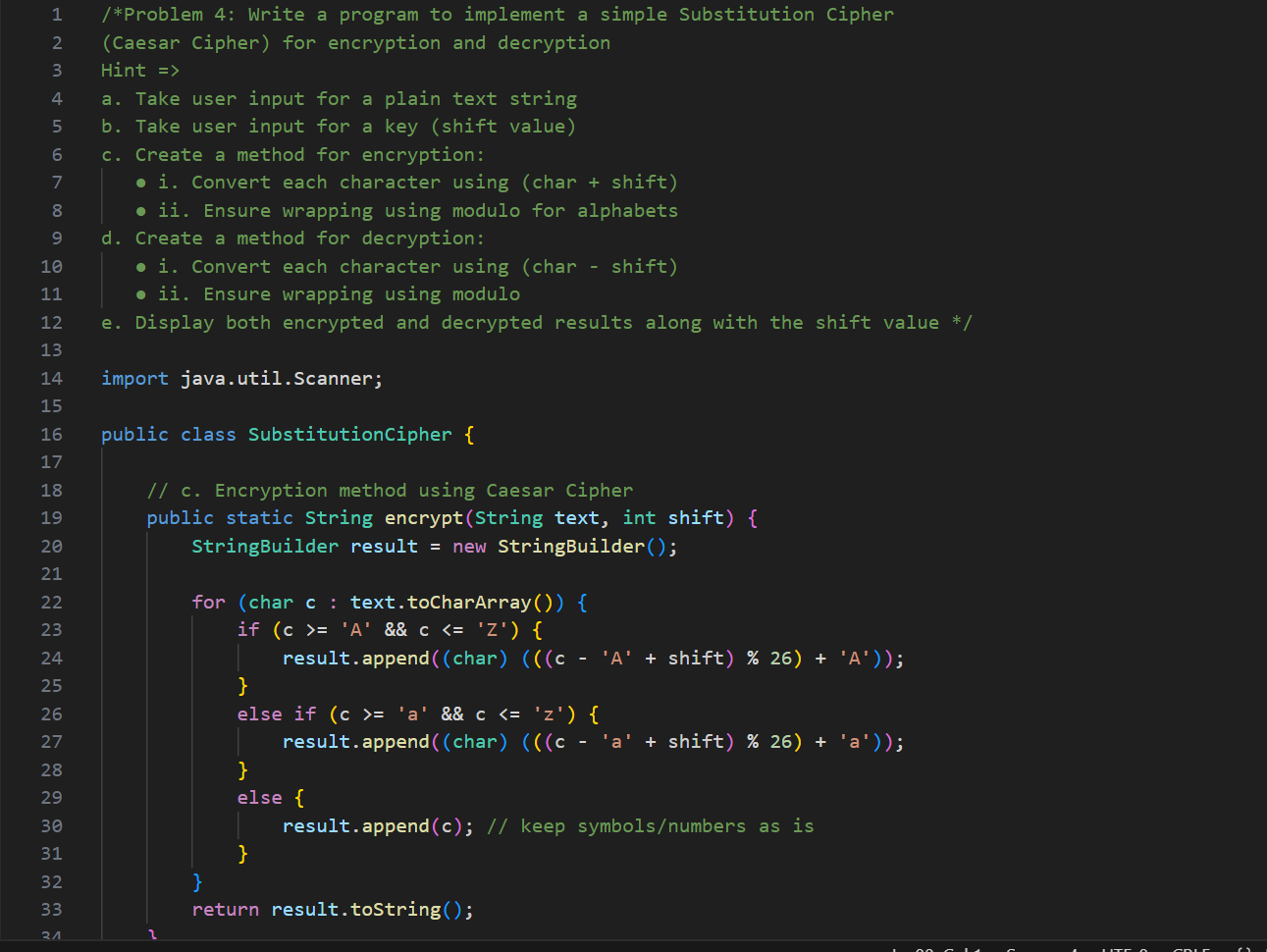
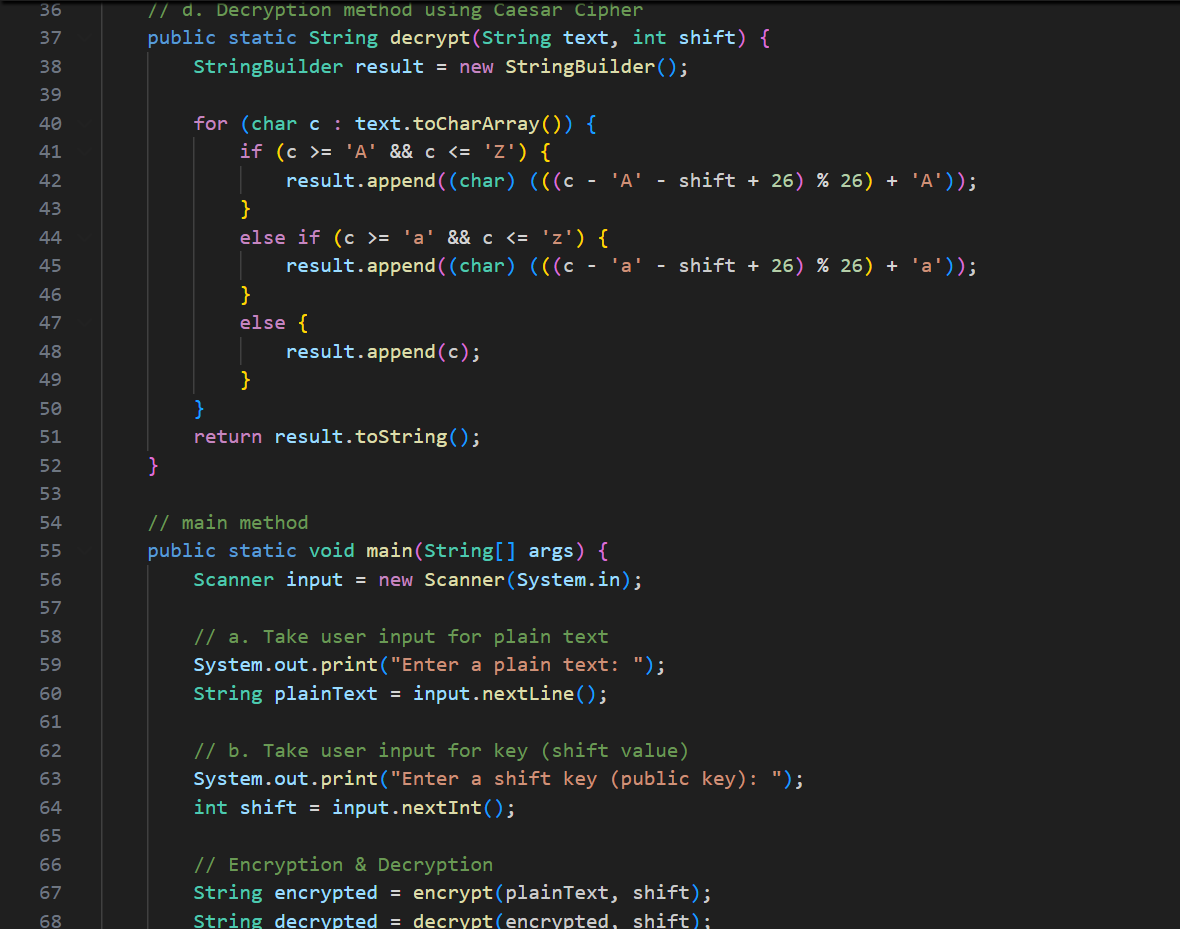
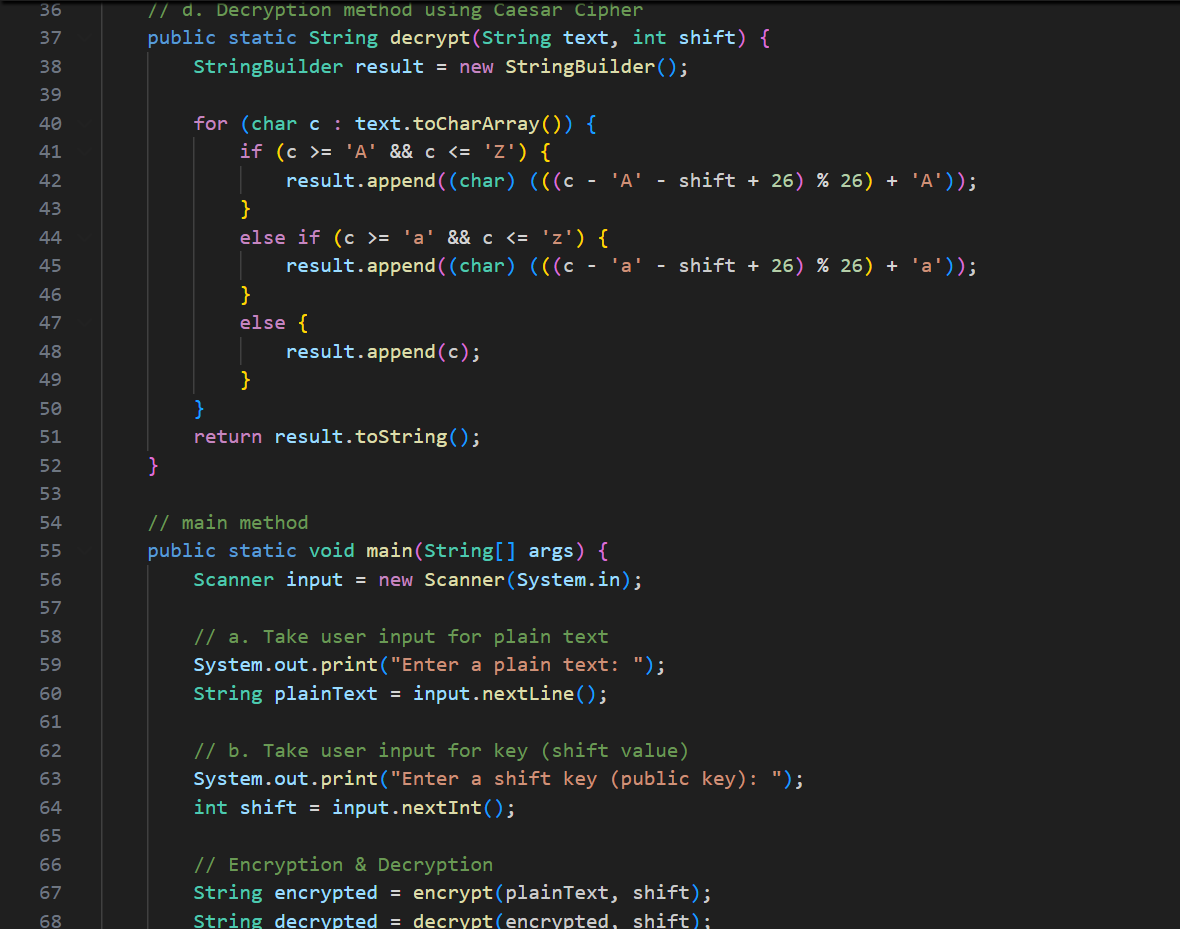
**e. Create a method to validate that decryption returns the original text**

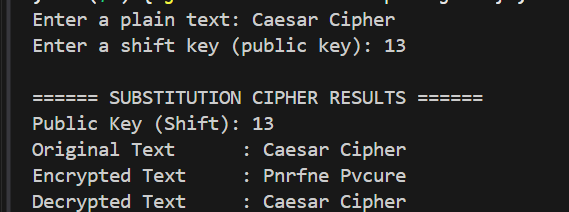
**f. The main function takes inputs, calls encryption/decryption methods, and displays:**

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

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

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

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**

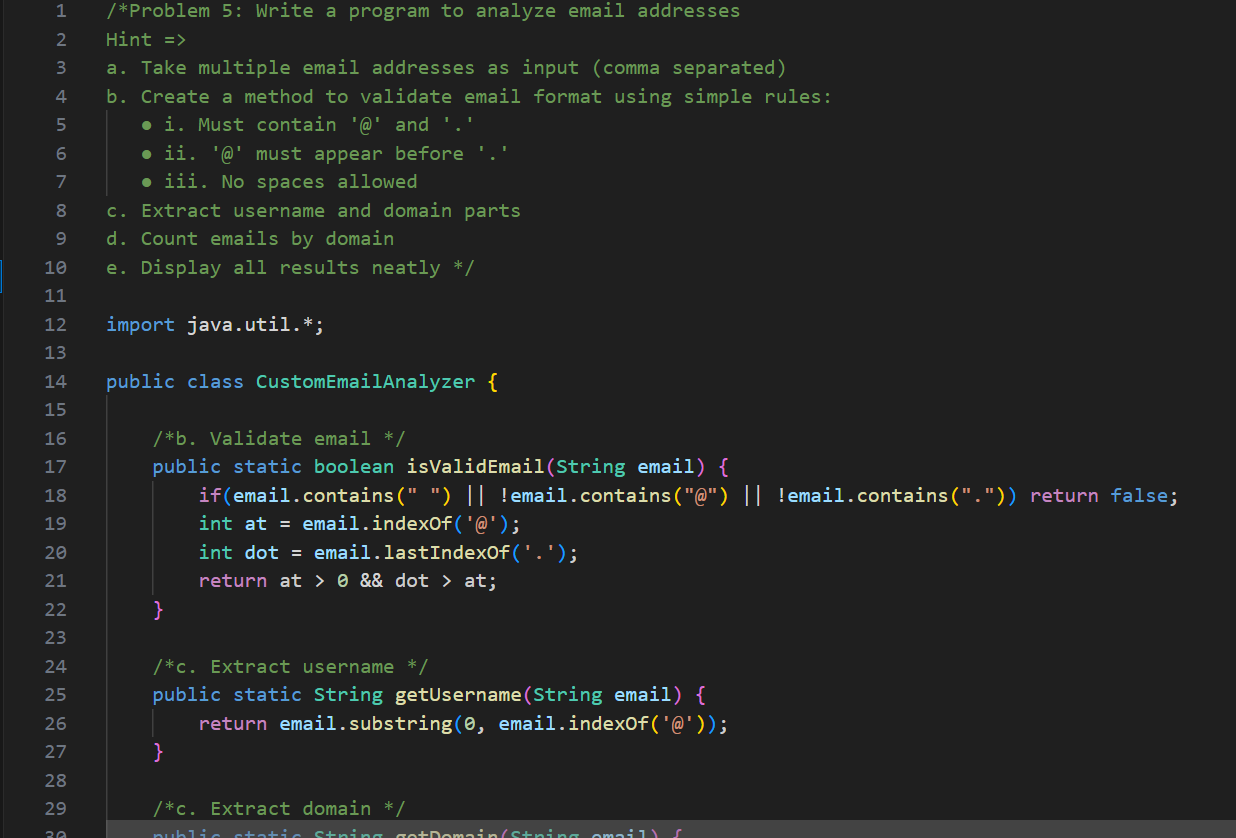
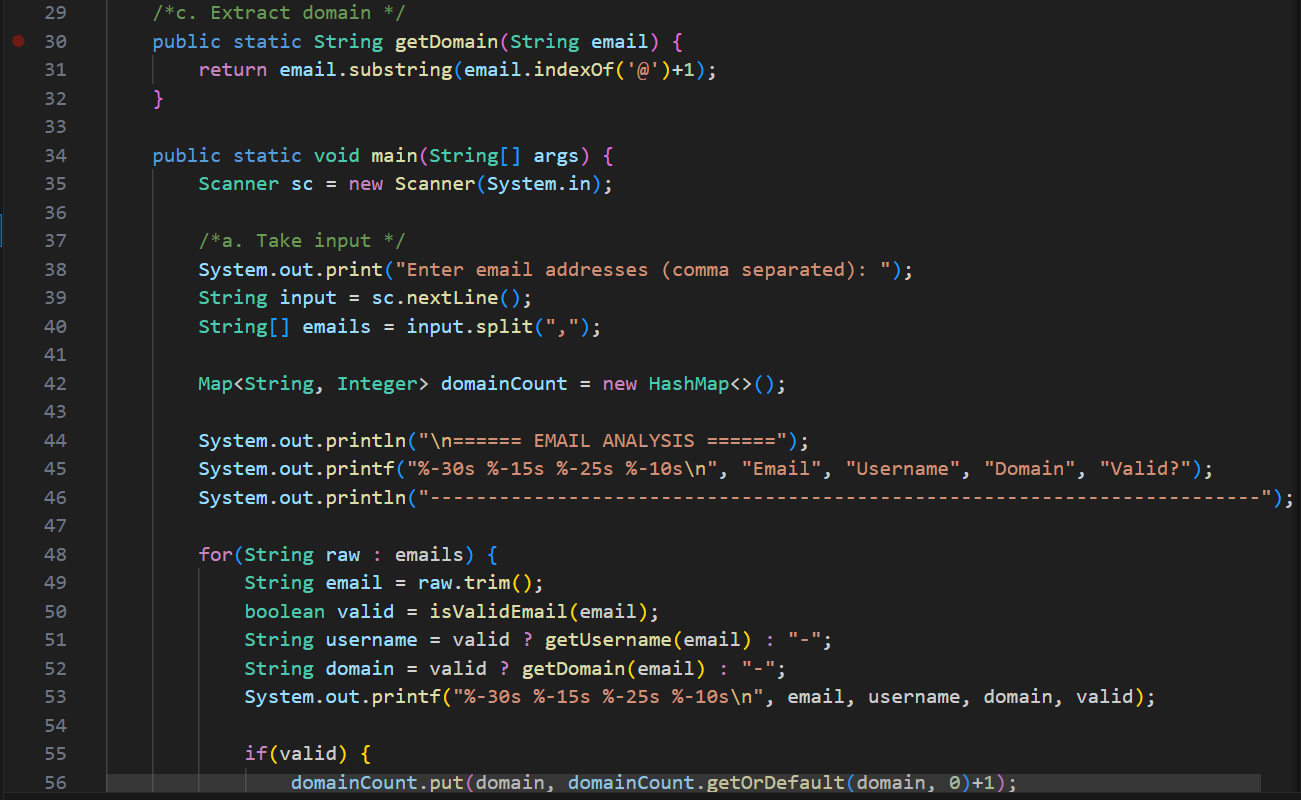
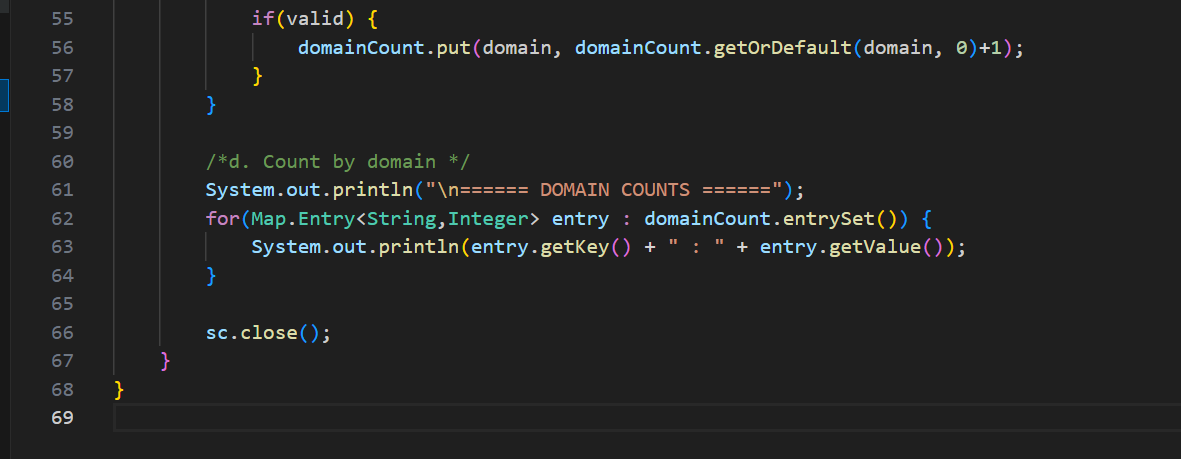
**● ii. Find most common domain**

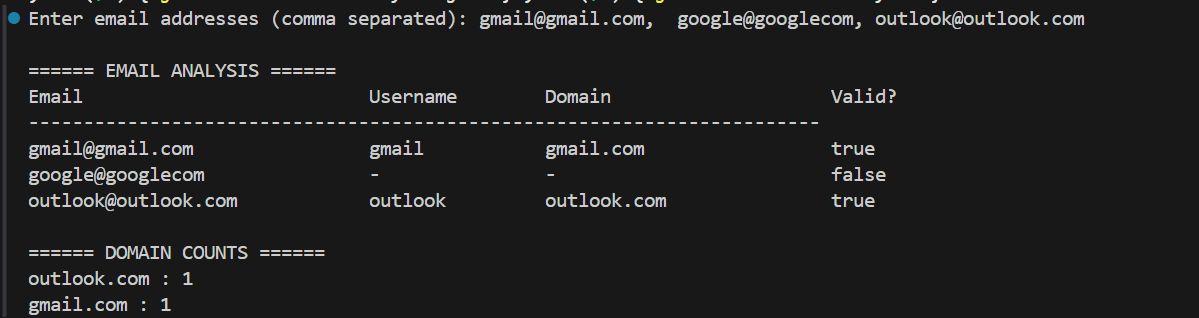
**● iii. Calculate average username length**

**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**

**** **** 

**OUTPUT🡪** 

**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**

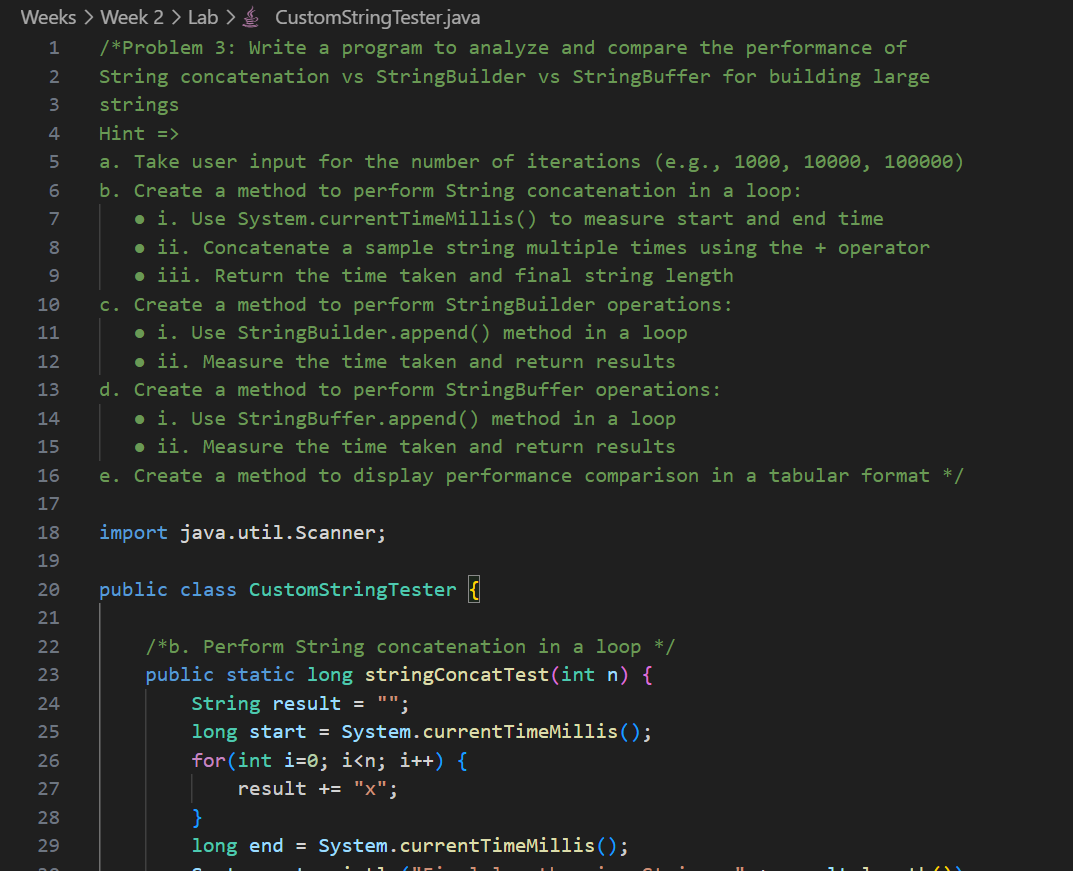
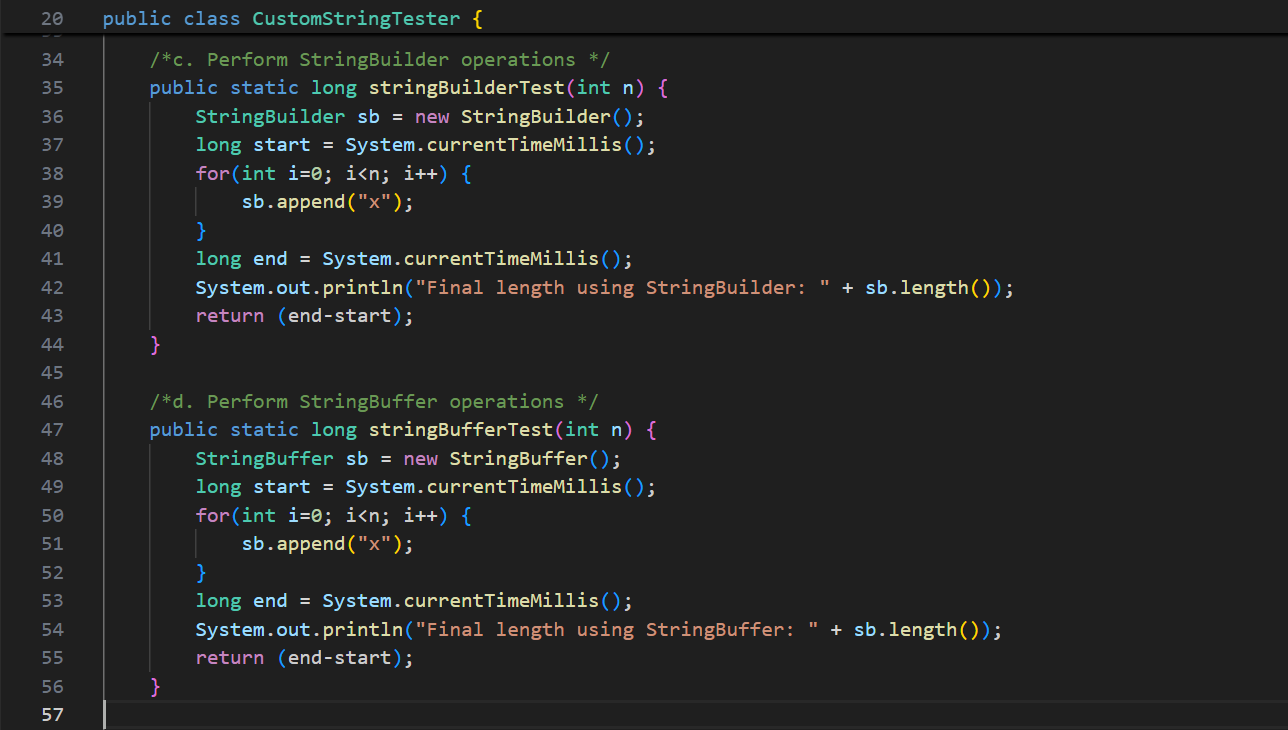
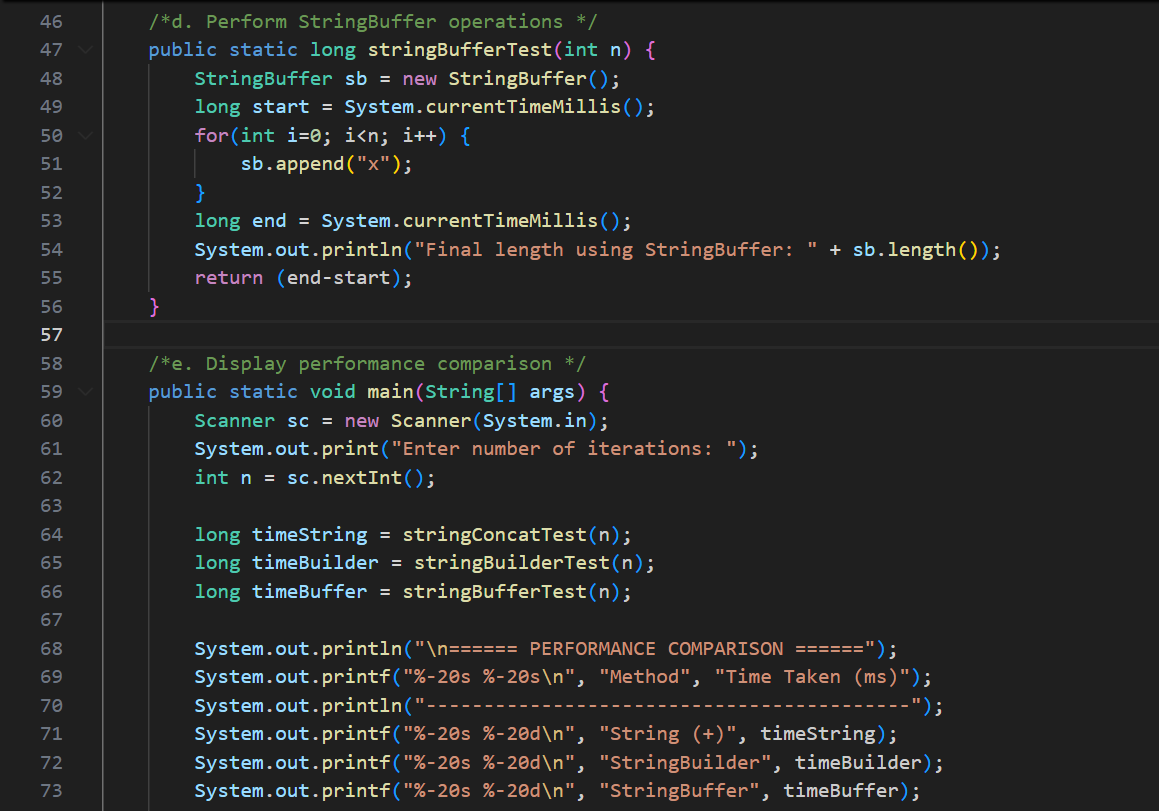
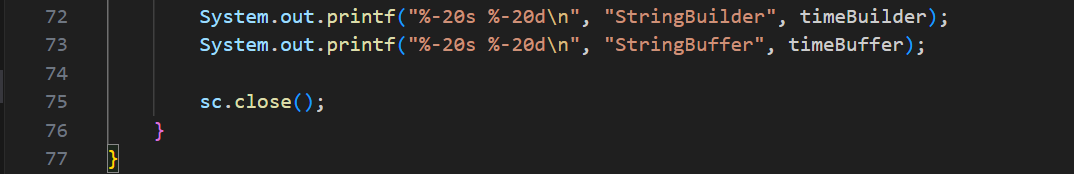
**g. The main function calls all methods and displays:**

**● i. Original text**

**● ii. Left-justified text**

**● iii. Center-aligned text**

**● iv. Performance analysis**

**** ****  

**OUTPUT🡪**

