1. Write a Java program that illustrates the difference between using string literals and the new keyword for creating String objects. Your program should demonstrate the memory usage implications and how string comparison behaves differently in each case.

2. Write a Java program that demonstrates the immutability of the String class and how it implements the CharSequence interface. Your program should illustrate the behaviours that highlight String immutability and its usage as a CharSequence.

3. Write a Java program that uses StringBuffer to construct a simple text editor which can perform the following operations:
    a. Append a given string to the existing text.
    b. Insert a given string at a specified index within the existing text.
    c. Delete a portion of text between two specified indices.
    d. Reverse the entire text.
    e. Replace a portion of the text between two specified indices with a given string.

Your program should display a menu with options to perform each of the above operations. After each operation, print the current state of the text. Also, display the current capacity and length of the StringBuffer after each operation to showcase its dynamic nature.

4. Create a Java program that uses StringBuilder to perform a series of text manipulations on a user-provided string. The program should allow users to:
    a. Add a substring at a specified position.
    b. Remove a range of characters from the string.
    c. Modify a character at a specified index.
    d. Concatenate another string at the end.
    e. Display the current string after each operation.

t.insert(position,substring)
t.delete(startIndex,endIndex)
t.setCharAt(index,newChar)
text.append(concatString) concatstr="abc"
sysout(text)

The program should repeatedly prompt the user to choose an operation until they decide to exit. After each operation, it should display the modified string, demonstrating the mutable nature of StringBuilder.

5. Create a Java program that compares the performance of StringBuilder and StringBuffer when performing repeated string concatenations. The program should:
    a. Prompt the user to enter a base string and the number of times it should be concatenated to itself.
    b. Use StringBuilder to concatenate the string the specified number of times, tracking the time taken to complete the operation.
    c. Repeat the process using StringBuffer, again tracking the time taken.
    d. Output the time taken for each operation and the final length of the resulting strings to demonstrate both the time efficiency and the result of using StringBuilder and StringBuffer.

    Example output of the program could look like this:
    Enter the base string:
    > Hello
    Enter the number of concatenations:
    > 10000

Using StringBuilder...
Time taken: 5 milliseconds
Final string length: 50000

Using StringBuffer...
Time taken: 6 milliseconds
Final string length: 50000

For time calculation:
startTime=System.currentTimeMillis();
performoperation
endTime=System.currentTimeMillis();

duration =endTime-StartTime

Comparison: StringBuilder was faster than StringBuffer by 1 millisecond.

6. Case Conversion and Comparison: Prompt the user to input two strings. Convert both strings to lowercase and uppercase. Compare the converted strings to check case-insensitive equality. Display the converted strings and the result of the comparison.
   a.toUpperCase
   a.toLowerCase then for checking e.equals(f)
7. Character Array and Search: Ask for a string from the user. Convert the string to a character array. Prompt the user to enter a character to search in the string. Find the first and last occurrences of the character. Display the character array and the positions found (if any).String a=sc.next(); char []arr = a.toCharArray();
   For searching- char b = sc.next().charAt(0);
8. String Concatenation and Character Retrieval: Take two strings from the user. Concatenate them using the string method and the + operator, then display both results. Ask the user for an index number, then display the character at that index.
   String a, int index, char res= a.charAt(index); afterconcat and +
9. Word Replacement in Sentences: Request a sentence and two words from the user: one to search for and one to replace it with. Find the first occurrence of the search word in the sentence. Replace the word using substring operations and concatenation. Display the original and the modified sentences.
10. Interactive String Explorer: Prompt the user for a string. Display a menu with options to perform various operations: convert to lowercase/uppercase, search for a character/index, or concatenate with another string. Based on user selection, perform the appropriate string operation and show the result.