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.

```
public class StringTest {
    public static void main(String[] args) {
        // Using String literals
        String s1 = "Java";
        String s2 = "Java";

        // Using new keyword
        String s3 = new String("Java");
        String s4 = new String("Java");

        // Memory and comparison
        System.out.println("s1 == s2 : " + (s1 == s2)); // true, same memory location in
string pool
        System.out.println("s3 == s4 : " + (s3 == s4)); // false, different memory locations
in heap

        // Using equals method for content comparison
        System.out.println("s1.equals(s3) : " + s1.equals(s3)); // true, content is same
    }
}
```

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.

```
public class StringImmutability {
    public static void main(String[] args) {
        String str = "Hello";
        System.out.println("Original String: " + str);

        // Trying to modify using concat
        str.concat(" World");
        System.out.println("After concat: " + str); // Remains "Hello" due to immutability

        // Correct way to concatenate
        str = str.concat(" World");
        System.out.println("After correct concat: " + str);

        // Using as CharSequence
        CharSequence cs = "Immutable";
        System.out.println("Char at 3 in CharSequence: " + cs.charAt(3)); // 'm'
    }
```

}

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.

```
import java.util.Scanner;

public class StringBufferEditor {
    public static void main(String[] args) {
        StringBuffer buffer = new StringBuffer();
        Scanner scanner = new Scanner(System.in);

        while (true) {
            System.out.println("\n1. Append \n2. Insert \n3. Delete \n4. Reverse \n5. Replace \n6.
Exit");
            System.out.print("Choose an operation: ");
            int choice = scanner.nextInt();
            scanner.nextLine(); // consume newline

            switch (choice) {
                case 1:
                    System.out.print("Enter string to append: ");
                    buffer.append(scanner.nextLine());
                    break;
                case 2:
                    System.out.print("Enter index and string to insert: ");
                    int insertIndex = scanner.nextInt();
                    scanner.nextLine();
                    buffer.insert(insertIndex, scanner.nextLine());
                    break;
                case 3:
                    System.out.print("Enter start and end indices to delete: ");
                    int start = scanner.nextInt();
                    int end = scanner.nextInt();
                    buffer.delete(start, end);
                    break;
                case 4:
                    buffer.reverse();
```

```
                break;
            case 5:
                System.out.print("Enter start, end index, and replacement string: ");
                start = scanner.nextInt();
                end = scanner.nextInt();
                scanner.nextLine();
                buffer.replace(start, end, scanner.nextLine());
                break;
            case 6:
                System.exit(0);
            default:
                System.out.println("Invalid choice!");
        }
        System.out.println("Current Buffer: " + buffer);
        System.out.println("Length: " + buffer.length() + ", Capacity: " + buffer.capacity());
    }
  }
}
```

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.

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.

```
import java.util.Scanner;

public class StringBuilderUsage {
   public static void main(String[] args) {
      StringBuilder sb = new StringBuilder();
      Scanner scanner = new Scanner(System.in);

      System.out.print("Enter initial string: ");
      sb.append(scanner.nextLine());

      while (true) {
         System.out.println("\n1. Add substring \n2. Remove characters \n3. Modify character
\n4. Concatenate string \n5. Show string \n6. Exit");
         System.out.print("Choose operation: ");
         int choice = scanner.nextInt();
         scanner.nextLine();

         switch (choice) {
```

```
            case 1:
                System.out.print("Enter substring and position: ");
                String substr = scanner.nextLine();
                int pos = scanner.nextInt();
                sb.insert(pos, substr);
                break;
            case 2:
                System.out.print("Enter s
```

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

    Comparison: StringBuilder was faster than StringBuffer by 1 millisecond.

```
public class PerformanceTest {
    public static void main(String[] args) {
        final int ITERATIONS = 10000;
        String baseString = "Hello";

        // StringBuilder performance test
        long startTime = System.currentTimeMillis();
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < ITERATIONS; i++) {
            sb.append(baseString);
        }
        long endTime = System.currentTimeMillis();
```

```
        System.out.println("StringBuilder time: " + (endTime - startTime) + " ms");

        // StringBuffer performance test
        startTime = System.currentTimeMillis();
        StringBuffer buffer = new StringBuffer();
        for (int i = 0; i < ITERATIONS; i++) {
            buffer.append(baseString);
        }
        endTime = System.currentTimeMillis();
        System.out.println("StringBuffer time: " + (endTime - startTime) + " ms");
    }
}
```

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.

```
import java.util.Scanner;

public class CaseConversionComparison {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter first string: ");
        String str1 = scanner.nextLine();
        System.out.print("Enter second string: ");
        String str2 = scanner.nextLine();

        String str1Lower = str1.toLowerCase();
        String str2Lower = str2.toLowerCase();

        System.out.println("Lowercase versions: " + str1Lower + ", " + str2Lower);
        System.out.println("Uppercase versions: " + str1.toUpperCase() + ", " +
str2.toUpperCase());

        boolean areEqualIgnoringCase = str1Lower.equals(str2Lower);
        System.out.println("Are the two strings equal (case insensitive)? " +
areEqualIgnoringCase);
    }
}
```

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).

```
import java.util.Scanner;
```

```
public class CharacterArraySearch {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a string: ");
        String str = scanner.nextLine();
        char[] charArray = str.toCharArray();

        System.out.print("Enter a character to search for: ");
        char searchChar = scanner.next().charAt(0);

        int firstIndex = str.indexOf(searchChar);
        int lastIndex = str.lastIndexOf(searchChar);

        System.out.println("Character array: ");
        for (char c : charArray) {
            System.out.print(c + " ");
        }
        System.out.println("\nFirst occurrence of '" + searchChar + "': " + firstIndex);
        System.out.println("Last occurrence of '" + searchChar + "': " + lastIndex);
    }
}
```

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.

```
import java.util.Scanner;

public class ConcatAndCharAt {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the first string: ");
        String firstStr = scanner.nextLine();
        System.out.print("Enter the second string: ");
        String secondStr = scanner.nextLine();

        String concatenatedStrMethod = firstStr.concat(secondStr);
        String concatenatedStrPlus = firstStr + secondStr;

        System.out.println("Concatenated using method: " + concatenatedStrMethod);
        System.out.println("Concatenated using + operator: " + concatenatedStrPlus);

        System.out.print("Enter an index to retrieve character from concatenated string:
");
        int index = scanner.nextInt();
```

```
            if (index >= 0 && index < concatenatedStrPlus.length()) {
                System.out.println("Character at index " + index + ": " +
        concatenatedStrPlus.charAt(index));
            } else {
                System.out.println("Index out of range.");
            }
        }
    }
```

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.

```
import java.util.Scanner;

public class WordReplacement {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter a sentence: ");
        String sentence = scanner.nextLine();

        System.out.print("Enter the word to replace: ");
        String wordToReplace = scanner.next();

        System.out.print("Enter the replacement word: ");
        String replacementWord = scanner.next();

        int wordIndex = sentence.indexOf(wordToReplace);
        if (wordIndex != -1) {
            String newSentence = sentence.substring(0, wordIndex) +
                    replacementWord +
                    sentence.substring(wordIndex + wordToReplace.length());
            System.out.println("Original sentence: " + sentence);
            System.out.println("Modified sentence: " + newSentence);
        } else {
            System.out.println("Word not found in the sentence.");
        }
    }
}
```

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.

```
import java.util.Scanner;
```

```
public class InteractiveStringExplorer {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a string: ");
        String str = scanner.nextLine();

        while (true) {
            System.out.println("1. To lowercase\n2. To uppercase\n3. Search character\n4.
Concatenate\n5. Exit");
            System.out.print("Choose an operation: ");
            int choice = scanner.nextInt();
            scanner.nextLine(); // consume newline

            switch (choice) {
                case 1:
                    System.out.println(str.toLowerCase());
                    break;
                case 2:
                    System.out.println(str.toUpperCase());
                    break;
                case 3:
                    System.out.print("Enter character to search for: ");
                    char c = scanner.nextLine().charAt(0);
                    System.out.println("First occurrence: " + str.indexOf(c));
                    break;
                case 4:
                    System.out.print("Enter string to concatenate: ");
                    String toConcat = scanner.nextLine();
                    str += toConcat;
                    System.out.println("New string: " + str);
                    break;
                case 5:
                    return;
                default:
                    System.out.println("Invalid choice");
                    break;
            }
        }
    }
}
```