**FR. CONCEICAO RODRIGUES COLLEGE OF ENGINEERIG**

**Department of Computer Science Engineering**

|  |  |  |  |
| --- | --- | --- | --- |
| **Academic Year** | **2025-2026** | **Estimated Time** | **Experiment No.1 – 02 Hours** |
| **Course & Semester** | **S.E. CSE** | **Subject Name** | **Object Oriented**  **Programming with Java Lab** |
| **Module No.** | **05** | **Chapter Title** |  |
| **Experiment Type** | **Software Performance** | **Subject Code** | **PCC12CE07** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name of Student** | Atharva Dharmendra Jagtap | **Roll No.** | 10937 |
| **Date of Performance.:** | 05-10-2025 | **Date of Submission.:** | 05-10-2025 |
| **CO Mapping** |  | | |

**Objective of Experiment:** Strings: Introduction to strings and string manipulation

**Pre-Requisite:**Anyprogramminglanguagelike C,C++

**Tools:**JavaIDLE

**Theory:**

**What is a String?**

In Java, **a string is a sequence of characters**. The String class provides methods to manipulate these characters, like concatenating two strings, converting characters to uppercase, and so on.

**Key Features of String**

**Immutability:** Strings are immutable in Java, meaning once created, they cannot be changed. Any modification to a string results in a new object, leaving the original string unaltered.

**String Pool:** Java uses a special memory area known as the [String Pool](https://www.javaguides.net/2018/07/guide-to-java-string-constant-pool.html) to store string literals. This helps in saving memory, as multiple references to the same string literally point to the same object in the pool.

**String Creation:** Strings can be created using either string literals or the *new* keyword. Creating strings with literals promotes reusing existing objects from the String Pool, whereas the *new* keyword forces a new object's creation.

**Concatenation:** Strings can be concatenated using the *+* operator or the *concat()* method. Concatenation creates a new string object since strings are immutable.

**Comparison:** Strings can be compared using the *equals()* method for content comparison or *==* for reference comparison. The *equalsIgnoreCase()* method can be used for case-insensitive content comparisons.

**Case Handling:** Methods like t*oLowerCase()* and *toUpperCase()* allow conversion between different

cases.

**Substring & Character Access:** Methods like *substring()*, *charAt()*, and *indexOf()* help in accessing specific parts or characters of a string.

**Trimming & Replacing:** *trim()* is used to remove leading and trailing whitespace. *replace()* and *replaceAll()* help in replacing specific characters or sequences within a string.

**Converting to Other Types:** Strings can be converted to arrays of characters using *toCharArray()*. Various parsing methods like *Integer.parseInt()* can convert a string to numerical types.

**Performance Consideration:** Since strings are immutable, frequent modifications can lead to performance issues. In scenarios with intense string manipulation, consider using [StringBuilder](https://www.javaguides.net/2018/08/java-stringbuilder-class-api-guide.html) or [StringBuffer](https://www.javaguides.net/2019/09/java-stringbuffer-tutorial-with-examples.html).

**Creating String Objects**

There are two ways to create a String object:

By string literal

By new keyword

|  |  |
| --- | --- |
| **Description** | **Example** |
| **Using String Literal**  Java String literal is created by using double quotes.  **For Example:**  String s="javaguides";  Each time you create a string literal, the JVM checks the string constant pool first. If the string already exists in the pool, a reference to the pooled instance is returned. If a string doesn't exist in the pool, a new string instance is created and placed in the pool. | String s1="javaguides";  String s2="javaguides"; |
| **Using a new Keyword**  Let's create a simple example to demonstrate by creating String objects using the *new*keyword. | public static void main(String[] args) {  String str = new String("Java Guides");  // create String object using new Keyword  int length = str.length();  System.out.println(" length of the string '" + str + "' is :: " + length);  } |
|  |  |

### C Important String Class Methods:

### Java's String class, part of the *java.lang* package provides various methods to perform different operations on strings, such as trimming, replacing, converting, comparing, and more. Let's explore the important methods of the String class in Java and illustrates how they can be used.

|  |  |
| --- | --- |
|  | |
| **length(): Finding the Length**  This method returns the length of the string, i.e., the number of characters in it. | String name = "JavaGuides";  int len = name.length();  System.out.println("Length: " + len); // Output: Length: 10 |
| concat(): Concatenating StringsThe *concat()* method appends one string to the end of another. | String first = "Java";  String second = "Guides";  String full = first.concat(second);  System.out.println(full); // Output: JavaGuides |
| **charAt(): Accessing Specific Characters**  The *charAt()* method returns the character at a specific index in the string. | String word = "JavaGuides";  char letter = word.charAt(4);  System.out.println(letter); // Output: G |
| **substring(): Extracting Substrings**  This method returns a part of the string. | String word = "JavaGuides";  String part = word.substring(4, 9);  System.out.println(part); // Output: Guide |
| **toLowerCase() and toUpperCase(): Changing Case**  These methods convert the string to lowercase and uppercase, respectively. | String mixed = "JavaGuides";  System.out.println(mixed.toLowerCase()); // Output: javaguides  System.out.println(mixed.toUpperCase()); // Output: JAVAGUIDES |
| **trim(): Removing Whitespace**  The *trim()* method eliminates leading and trailing spaces. | String spaced = " JavaGuides ";  System.out.println(spaced.trim()); // Output: JavaGuides |
| **replace(): Replacing Characters**  This method replaces occurrences of a specific character or character sequence. | String original = "JavaGuides";  String replaced = original.replace("Java", "Spring");  System.out.println(replaced); // Output: SpringGuides |
| **equals(): Comparing Strings**  The *equals()* method checks if two strings are the same. | String one = "JavaGuides";  String two = "javaguides";  boolean isEqual = one.equals(two); // false |
| **indexOf() and lastIndexOf(): Finding Occurrences**  These methods return the index of the first and last occurrence of a character or substring. | String example = "JavaGuides";  int firstIndex = example.indexOf('a'); // 1  int lastIndex = example.lastIndexOf('a'); // 3 |
| **contains()**  The contains() method returns true if the string contains the specified sequence of char values. | public class StringExample {  public static void main(String[] args) {  String str = "Hello, World!";  boolean contains = str.contains("World");  System.out.println("Contains 'World': " + contains);  }  }  o/p Contains 'World': true |
| **equals()**  The equals() method compares the specified string to the current string for equality. | public class StringExample {  public static void main(String[] args) {  String str1 = "Hello";  String str2 = "Hello";  boolean isEqual = str1.equals(str2);  System.out.println("Strings are equal: " + isEqual);  }  }  **Output:**  Strings are equal: true |
| **equalsIgnoreCase()**  The equalsIgnoreCase() method compares the specified string to the current string for equality, ignoring case considerations. | public class StringExample {  public static void main(String[] args) {  String str1 = "Hello";  String str2 = "hello";  boolean isEqual = str1.equalsIgnoreCase(str2);  System.out.println("Strings are equal (ignore case): " + isEqual);  }  }  **Output:**  Strings are equal (ignore case): true |
| **compareTo()**  The compareTo() method compares two strings lexicographically. | public class StringExample {  public static void main(String[] args) {  String str1 = "Hello";  String str2 = "World";  int result = str1.compareTo(str2);  System.out.println("Comparison result: " + result);  }  }  **Output:**  Comparison result: -15 |
| **indexOf()**  The indexOf() method returns the index within the string of the first occurrence of the specified character or substring. | public class StringExample {  public static void main(String[] args) {  String str = "Hello, World!";  int index = str.indexOf('o');  System.out.println("Index of 'o': " + index);  }  }  **Output:**  Index of 'o': 4 |
| **split()**  The split() method splits the string around matches of the given regular expression. | import java.util.Arrays;  public class StringExample {  public static void main(String[] args) {  String str = "Hello, World!";  String[] parts = str.split(", ");  System.out.println(Arrays.toString(parts));  }  }  **Output:**  [Hello, World!] |
|  |  |
|  |  |
|  |  |

**Why String is Immutable in Java**

In Java, strings are immutable, meaning that once a string object is created, its content cannot be altered. Instead, any operation that seems to change the content of a string actually results in a new string object. The original string remains unchanged.

Here's why string immutability is a design feature in Java:

**1. Security:** Immutable objects are inherently thread-safe since they cannot be modified after creation. This property eliminates synchronization issues in multithreaded applications, making string handling more secure.

**2. Performance Optimization through String Pooling:** Since strings are immutable, Java can cache them in a special memory region called the String Pool. If the same string literal is used elsewhere in the program, both references will point to the same object in the pool. This saves memory and boosts performance.

**3. Hashcode Caching:** Strings in Java often act as keys in collections like HashMap and HashSet. The hashcode of an object, once calculated, can be cached as long as the object is not modified. Since strings are immutable, their hashcode remains constant, allowing for efficient retrieval from collections.

**4. Integrity and Reliability:** Immutability ensures that once a string object is created, it will not be changed by any part of the code, intentionally or unintentionally. This behavior makes the code more predictable and maintains the integrity of the data.

**5. Simplifies Complex Systems:** In a large and complex application, tracking and controlling object modification can be cumbersome. Immutability alleviates this problem, simplifying code understanding and maintenance.

**6. Class Loading and Security Concerns:** Strings are used in various parts of Java's Class Loading mechanism, including specifying the classpath. Mutable strings could pose security risks and create unpredictable behaviors within the JVM's security model.

# Java String Best Practices

|  |  |
| --- | --- |
| **Use String Literals Where Possible:**  String literals are managed in the String Pool, allowing for efficient memory usage. Compare using *equals* instead of  = =: | String str1 = "JavaGuides";  String str2 = "JavaGuides";  // Use equals for content comparison  if (str1.equals(str2)) { /\*...\*/ } |
| **Avoid Concatenation in Loops:**  String concatenation inside a loop can lead to performance issues due to the creation of multiple objects. Use [StringBuilder](https://www.javaguides.net/2018/08/java-stringbuilder-class-api-guide.html) instead: | StringBuilder builder = new StringBuilder();  for (int i = 0; i < 100; i++) {  builder.append(i);  }  String result = builder.toString(); |
| **Use String Formatting:**  For complex string creation, the *String.format* provides a more readable approach: | String formatted = String.format("Order %d: %s", orderId, orderName); |
| **Avoid Using = = for String Comparison:**  The *==* operator compares object references, not content. Use *equals()*: | if (str1.equals(str2)) { /\*...\*/ } |
| **Use equalsIgnoreCase for Case-Insensitive Comparison:** | if (str1.equalsIgnoreCase(str2)) { /\*...\*/ } |
| **Prefer isEmpty Over length() Check for Emptiness:** | if (str.isEmpty()) { /\*...\*/ } |
| **Use StringBuilder Over StringBuffer for Single-Threaded Operations:**  [StringBuilder](https://www.javaguides.net/2018/08/java-stringbuilder-class-api-guide.html) provides better performance as it doesn't synchronize every operation like StringBuffer. | |
| **Handle null Strings Carefully:**  Consider using Objects.equals(str1, str2) to avoid *NullPointerException*. | |
| **Utilize String Methods for Cleaning and Parsing:**  Make use of methods like trim(), split(), toLowerCase(), toUpperCase(), etc., to manipulate strings without reinventing the wheel. | |
| **Be Careful with Character Encoding:**  If working with different character encodings, be conscious of the encoding used, especially when reading or writing strings to files or over a network. | |

**Use Cases**

Text processing: Manipulating and processing text data.

Data formatting: Formatting strings for display or output.

Input validation: Checking and validating string input.

File handling: Reading and writing text files.

**Problem Description:**

|  |  |
| --- | --- |
| A university is developing a tool to check whether given sentences are **palindromes** (the same when read forwards and backwards, ignoring spaces and cases).  The tool should:   1. Accept a sentence from the user. 2. Remove all spaces from the sentence. 3. Convert the sentence to lowercase. 4. Reverse the sentence. 5. Compare the original and reversed versions to check if it is a **palindrome** using:    * equals()    * equalsIgnoreCase()    * compareTo() 6. Implement this using both:    * **Immutable String**    * **Mutable StringBuilder / StringBuffer** | |
| Input: | Expected Output: |

**Code:**

import java.util.Scanner;

public class PalindromeChecker {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

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

        String input = sc.nextLine();

        // Step 1: Immutable String operations

        String processed = input.replaceAll("\\s+", "").toLowerCase(); // remove spaces and convert to lowercase

        String reversed = new StringBuilder(processed).reverse().toString(); // reverse using StringBuilder

        System.out.println("\n--- Using Immutable String ---");

        System.out.println("Processed String: " + processed);

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

        // equals()

        if (processed.equals(reversed))

            System.out.println("Palindrome check (equals): TRUE");

        else

            System.out.println("Palindrome check (equals): FALSE");

        // equalsIgnoreCase()

        if (processed.equalsIgnoreCase(reversed))

            System.out.println("Palindrome check (equalsIgnoreCase): TRUE");

        else

            System.out.println("Palindrome check (equalsIgnoreCase): FALSE");

        // compareTo()

        if (processed.compareTo(reversed) == 0)

            System.out.println("Palindrome check (compareTo): TRUE");

        else

            System.out.println("Palindrome check (compareTo): FALSE");

        // Step 2: Mutable StringBuilder

        StringBuilder sb = new StringBuilder(input);

        StringBuilder sbProcessed = new StringBuilder();

        // remove spaces and convert to lowercase manually

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

            char c = sb.charAt(i);

            if (c != ' ')

                sbProcessed.append(Character.toLowerCase(c));

        }

        StringBuilder sbReversed = new StringBuilder(sbProcessed).reverse();

        System.out.println("\n--- Using Mutable StringBuilder ---");

        System.out.println("Processed StringBuilder: " + sbProcessed);

        System.out.println("Reversed StringBuilder: " + sbReversed);

        // equals() and equalsIgnoreCase() don’t work directly for StringBuilder, so convert to String

        String str1 = sbProcessed.toString();

        String str2 = sbReversed.toString();

        System.out.println("Palindrome check (equals): " + str1.equals(str2));

        System.out.println("Palindrome check (equalsIgnoreCase): " + str1.equalsIgnoreCase(str2));

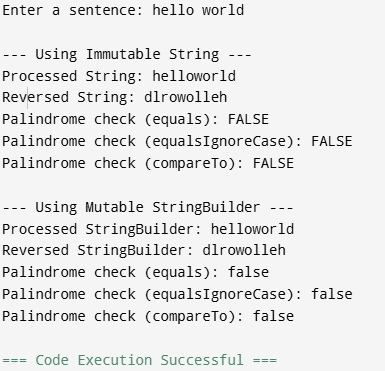
        System.out.println("Palindrome check (compareTo): " + (str1.compareTo(str2) == 0));

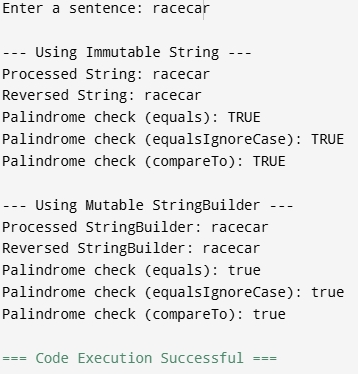
        sc.close();

    }

}

**Output:**

****

****

**PostLabQuestions:**

|  |
| --- |
|  |
| 1. **WAC to implement all string functions for mutable string and immutable string.** |
|  |

**Code:**

import java.util.\*;

public class StringFunctionsDemo {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

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

        String input = sc.nextLine();

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

        System.out.println("   IMMUTABLE STRING OPERATIONS  ");

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

        // Original String

        String str = input;

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

        // String operations

        System.out.println("Length: " + str.length());

        System.out.println("Character at index 0: " + str.charAt(0));

        System.out.println("Substring (0,3): " + str.substring(0, Math.min(3, str.length())));

        System.out.println("To Uppercase: " + str.toUpperCase());

        System.out.println("To Lowercase: " + str.toLowerCase());

        System.out.println("Concatenation: " + str.concat(" Java"));

        System.out.println("Replace 'a' with '@': " + str.replace('a', '@'));

        System.out.println("Trimmed String: " + str.trim());

        System.out.println("Index of 'a': " + str.indexOf('a'));

        System.out.println("Last Index of 'a': " + str.lastIndexOf('a'));

        System.out.println("Starts with 'H': " + str.startsWith("H"));

        System.out.println("Ends with 'd': " + str.endsWith("d"));

        System.out.println("Equals to 'java': " + str.equals("java"));

        System.out.println("Equals Ignore Case to 'java': " + str.equalsIgnoreCase("java"));

        System.out.println("Compare To 'Java': " + str.compareTo("Java"));

        System.out.println("Contains 'a': " + str.contains("a"));

        System.out.println("Split words:");

        for (String word : str.split(" ")) {

            System.out.println(" - " + word);

        }

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

        System.out.println("   MUTABLE STRING OPERATIONS   ");

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

        // Using StringBuilder

        StringBuilder sb = new StringBuilder(input);

        System.out.println("Original StringBuilder: " + sb);

        sb.append(" Language");

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

        sb.insert(0, "Welcome ");

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

        sb.replace(0, 7, "Hi");

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

        sb.delete(0, 3);

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

        sb.reverse();

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

        sb.reverse(); // revert back to normal

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

        System.out.println("Length: " + sb.length());

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

        System.out.println("Character at index 2: " + sb.charAt(2));

        sb.setCharAt(2, '\*');

        System.out.println("After setCharAt(2, '\*'): " + sb);

        // Convert to String for comparison

        String sbString = sb.toString();

        System.out.println("Converted to String: " + sbString);

        System.out.println("Compare To (String version): " + sbString.compareTo(str));

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

        System.out.println("   USING STRINGBUFFER (Thread-Safe) ");

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

        StringBuffer sbf = new StringBuffer(input);

        System.out.println("Original StringBuffer: " + sbf);

        sbf.append(" Example");

        System.out.println("After append(): " + sbf);

        sbf.insert(0, "Safe ");

        System.out.println("After insert(): " + sbf);

        sbf.replace(0, 4, "Thread");

        System.out.println("After replace(): " + sbf);

        sbf.delete(0, 6);

        System.out.println("After delete(): " + sbf);

        sbf.reverse();

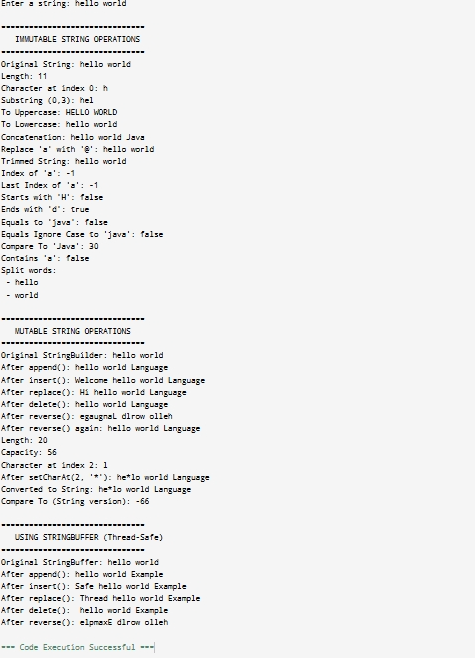
        System.out.println("After reverse(): " + sbf);

        sc.close();

    }

}

**Output:**

****

|  |  |  |  |
| --- | --- | --- | --- |
| **On time Completion and Submission (2)** | **Knowledge of the topic (4)** | **Implementation and Output (4)** | **Total (10)** |
|  |  |  |  |

**References:**

|  |  |
| --- | --- |
| **Study Materials**  [**https://www.w3schools.com/java/**](https://www.w3schools.com/java/)  [**https://www.geeksforgeeks.org/java/**](https://www.geeksforgeeks.org/java/)  https://www.codecademy.com/learn/learn-java | **Video Channels**:  [**https://www.youtube.com/user/programmingwithmosh**](https://www.youtube.com/user/programmingwithmosh)  [**https://www.youtube.com/c/TheNetNinja**](https://www.youtube.com/c/TheNetNinja)  [**https://www.youtube.com/c/Freecodecamp**](https://www.youtube.com/c/Freecodecamp)  [**https://www.youtube.com/user/Simplilearn**](https://www.youtube.com/user/Simplilearn) |
| **Study Materials used for Demo**  <Add links here> | |

**Note:-students are expected to paste screen shot of the program with output**