**String:**

**===========**

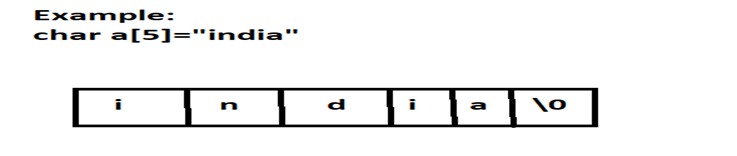
**A string represents a group of characters, space is also considered as character in strings. Strings are represented within double quotes(“.............”).**

**interview question:**

**what is the difference between C,C++ strings and java string?**

**in C,C++ any string is a character type array and terminated with null character(\0).**

**in java any string is by default string class object.**

****

**creating a strings:**

**Type-1:**

**we can declare string type variable and directly initialize it with characters.**

**Example:**

**String s1=”hello world”;**

**Type-2:**

**we can create string class object using new operator and store a group of characters.**

**Example:**

**String s2=new String(“hello world”);**

**Type-3:**

**we can convert a character type array into string object by passing the array to the object.**

**Example:**

**char arr[]={‘c’,’o’,’u’,’n’,’t’,’r’,’y’};**

**String s3=new String(arr);**

**Note:**

**string methods are present in java.lang package**

### **Case 2:**

| **String s=new String("bhaskar");**  **s.concat("software");**  **System.out.println(s);//bhaskar**  **Once we create a String object we can't perform any changes in the existing object. If we are try to perform any changes with those changes a new object will be created. This behavior is called immutability of the String object.  Diagram:** | **StringBuffer sb=new StringBuffer("bhaskar");**  **sb.append("software");**  **System.out.println(sb); //bhaskarsoftware**  **Once we created a StringBuffer object we can perform any changes in the existing object. This behavior is called mutability of the StringBuffer object.  Diagram:** |
| --- | --- |

### **Case 3:**

| **String s1=new String("ashok");**  **String s2=new String("ashok");**  **System.out.println(s1==s2);//false**  **System.out.println(s1.equals(s2));//true**  **In String class .equals() method is overridden for content comparison hence if the content is same .equals() method returns true even though objects are different.** | **StringBuffer sb1=new StringBuffer("ashok");**  **StringBuffer sb2=new StringBuffer("ashok");**  **System.out.println(sb1==sb2);//false**  **System.out.println(sb1.equals(sb2));//false**  **In StringBuffer class .equals() method is not overridden for content comparison hence Object class .equals() method got executed which is always meant for reference comparison. Hence if objects are different .equals() method returns false even though content is same.** |
| --- | --- |

### **Case 4 :**

| **String s=new String("bhaskar");**  **In this case two objects will be created one is on the heap the other one is SCP(String constant pool) and s is always pointing to heap object.  Diagram:** | **String s="bhaskar";**  **In this case only one object will be created in SCP and s is always referring that object.  Diagram :** |
| --- | --- |

**1.** **Object creation in SCP is always optional 1st JVM will check is any object already created with required content or not. If it is already available then it will reuse existing object instead of creating new object. If it is not already there then only a new object will be created. Hence there is no chance of existing 2 objects with same content on SCP that is duplicate objects are not allowed in SCP.**

**2.** **Garbage collector can't access SCP area hence even though object doesn't have any reference still that object is not eligible for GC if it is present in SCP.**

**3.** **All SCP objects will be destroyed at the time of JVM shutdown automatically.**

**Example 1:**

**String s1=new String("bhaskar");**

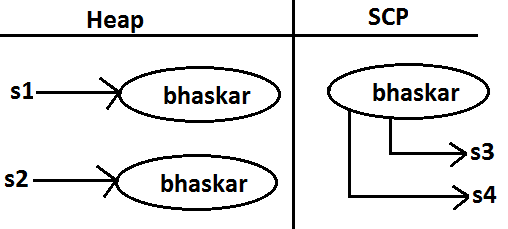
**String s2=new String("bhaskar");**

**String s3="bhaskar";**

**String s4="bhaskar";**

**Note :**

**When ever we are using new operator compulsory a new object will be created on the Heap . There may be a chance of existing two objects with same content on the heap but there is no chance of existing two objects with same content on SCP . i.e., duplicate objects possible in the heap but not in SCP .**

**Diagram :  
  
**

**case 5:**

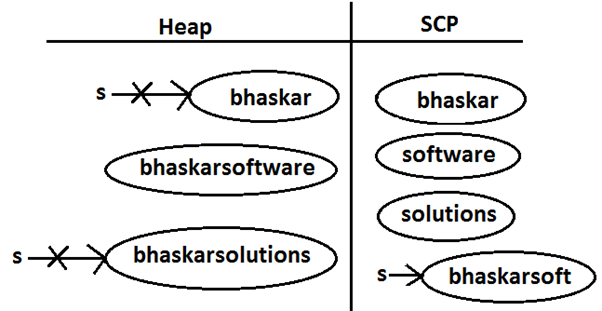
**Example 2:**

**String s=new String("bhaskar");**

**s.concat("software");**

**s=s.concat("solutions");**

**s="bhaskarsoft";**

**Diagram :  
  
  
  
 For every String Constant one object will be created in SCP. Because of runtime operation if an object is required to create compulsory that object should be placed on the heap but not SCP**

**Example 3:**

**String s1=new String("spring");**

**s1.concat("fall");**

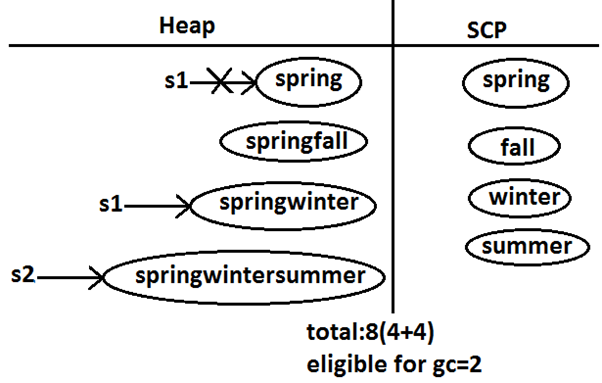
**s1=s1+"winter";**

**String s2=s1.concat("summer");**

**System.out.println(s1);**

**System.out.println(s2);**

**Diagram :**

****

**Example:**

**class StringDemo**

**{**

**public static void main(String[] args)**

**{**

**String s1=new String("you cannot change me!");**

**String s2=new String("you cannot change me!");**

**System.out.println(s1==s2);//false**

**String s3="you cannot change me!";**

**System.out.println(s1==s3);//false**

**String s4="you cannot change me!";**

**System.out.println(s3==s4);//true**

**String s5="you cannot "+"change me!";**

**System.out.println(s3==s5);//true**

**String s6="you cannot ";**

**String s7=s6+"change me!";**

**System.out.println(s3==s7);//false**

**final String s8="you cannot ";**

**String s9=s8+"change me!";**

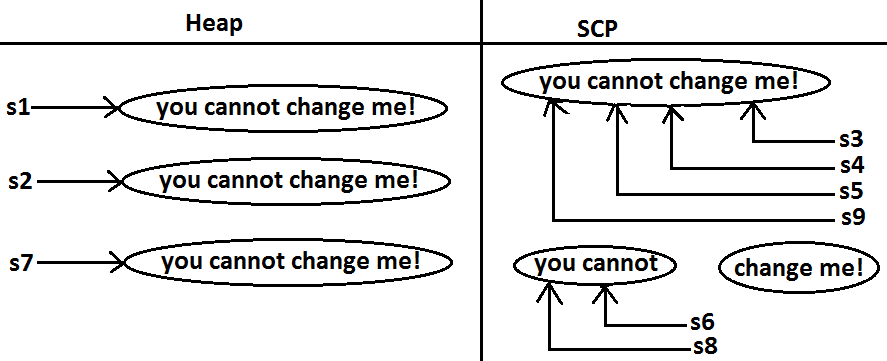
**System.out.println(s3==s9);//true**

**System.out.println(s6==s8);//true**

**}**

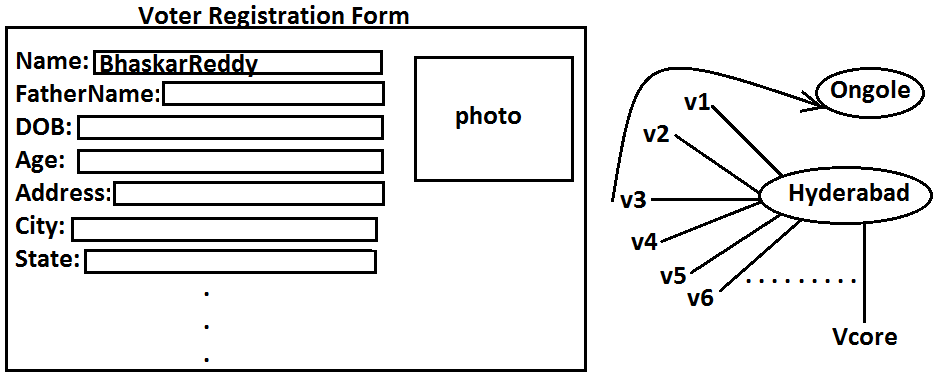
**}**

**Diagram:**

****

## **Importance of String constant pool (SCP) :**

**Diagram:**

****

1. **In our program if any String object is required to use repeatedly then it is not recommended to create multiple object with same content it reduces performance of the system and effects memory utilization.**
2. **We can create only one copy and we can reuse the same object for every requirement. This approach improves performance and memory utilization we can achieve this by using "scp".**
3. **In SCP several references pointing to same object the main disadvantage in this approach is by using one reference if we are performing any change the remaining references will be impacted. To overcome this problem sun people implemented immutability concept for String objects.**
4. **According to this once we creates a String object we can't perform any changes in the existing object if we are trying to perform any changes with those changes a new String object will be created hence immutability is the main disadvantage of scp.**

## **FAQS :**

1. **What is the main difference between String and StringBuilder?**
2. **What is the main difference between String and StringBuffer ?**
3. **Other than immutability and mutability is there any other difference between String and StringBuffer ?  
   In String .equals( ) method meant for content comparison where as in StringBuffer meant for reference comparision .**
4. **What is the meaning of immutability and mutability?**
5. **Explain immutability and mutability with an example?**
6. **What is SCP?  
   A specially designed memory area for the String literals/objects .**
7. **What is the advantage of SCP?  
   Instead of creating a separate object for every requirement we can create only one object and we can reuse same object for every requirement. This approach improves performance and memory utilization.**
8. **What is the disadvantage of SCP?  
   In SCP as several references pointing to the same object by using one reference if we are performing any changes the remaining references will be inflected. To prevent this compulsory String objects should be immutable. That is immutability is the disadvantage of SCP.**
9. **Why SCP like concept available only for the String but not for the StringBuffer?  
   As String object is the most commonly used object sun people provided a specially designed memory area like SCP to improve memory utilization and performance.  
   But StringBuffer object is not commonly used object hence specially designed memory area is not at all required.**
10. **Why String objects are immutable where as StringBuffer objects are mutable?  
    In the case of String as several references pointing to the same object, by using one reference if we are allowed perform the change the remaining references will be impacted. To prevent this once we created a String object we can't perform any change in the existing object that is immutability is only due to SCP.  
    But in the case of StringBuffer for every requirement we are creating a separate object will be created by using one reference if we are performing any change in the object the remaining references won't be impacted hence immutability concept is not require for the StringBuffer.**
11. **Similar to String objects any other objects are immutable in java?  
    In addition to String objects , all wrapper objects are immutable in java.**
12. **Is it possible to create our own mutable class?  
    Yes.**

## **Interning of String objects :**

—---------------------------------------------------------

**By using heap object reference, if we want to get corresponding SCP object , then we should go for intern() method.**

**Example 1:**

**class StringDemo {**

**public static void main(String[] args) {**

**String s1=new String("bhaskar");**

**String s2=s1.intern();**

**System.out.println(s1==s2); //false**

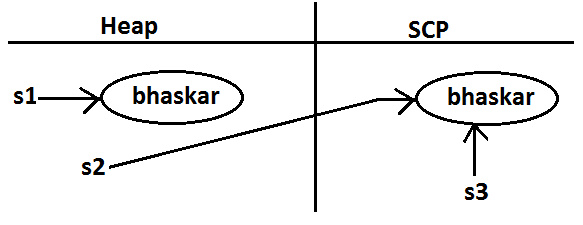
**String s3="bhaskar";**

**System.out.println(s2==s3);//true**

**}**

**}**

**Diagram:**

****

**If the corresponding object is not there in SCP then intern() method itself will create that object and returns it.**

**Example 2:**

**class StringDemo {**

**public static void main(String[] args) {**

**String s1=new String("bhaskar");**

**String s2=s1.concat("software");**

**String s3=s2.intern();**

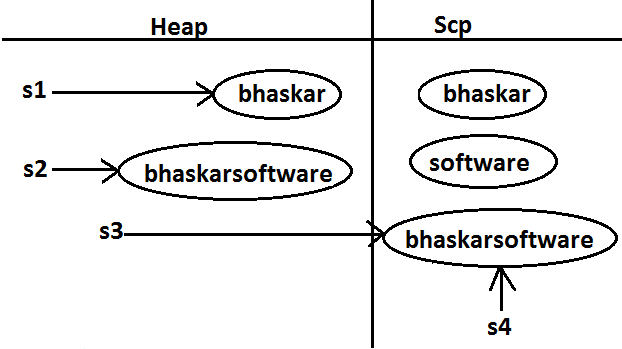
**String s4="bhaskarsoftware";**

**System.out.println(s3==s4);//true**

**}**

**}**

**Diagram 2:**

****

## **String class constructors :**

1. **String s=new String();  
   Creates an empty String Object.**
2. **String s=new String(String literals);  
   To create an equivalent String object for the given String literal on the heap.**
3. **String s=new String(StringBuffer sb);  
   Creates an equivalent String object for the given StringBuffer.**

**4. String s=new String(char[] ch);  
creates an equivalent String object for the given char[ ] array.  
Example:**

**class StringDemo {**

**public static void main(String[] args) {**

**char[] ch={'a','b','c'} ;**

**String s=new String(ch);**

**System.out.println(ch);//abc**

**}**

**}**

**5. String s=new String(byte[] b);  
Create an equivalent String object for the given byte[] array.  
Example:**

**class StringDemo {**

**public static void main(String[] args) {**

**byte[] b={100,101,102};**

**String s=new String(b);**

**System.out.println(s);//def**

**}**

**}**

**Methods:**

**1.String concat(String str):concatenates the calling string with str.**

**2.int length():It is used to find out the length of the given string.**

**3.char charAt(int index):Returns the character at the specified location( from 0).**

**4.int compareTo(String str):**

**Example:**

**Int n=s1.compareTo(s2);**

**Returns zero if s1 and s2 are equal.Returns a positive number if s1 comes after s2.Returns a negative value if s1 comes before s2 in dictionary order.**

**Example:**

**String s1=”Boy”,String s2=”Box”;**

**int n=s1.comapareTo(s2);**

**If s1==s2,then n==0**

**If s1>s2,then n>0**

**If s1<s2,then n<0**

**5.boolean equals(String str):Returns true if the calling string equals str,otherwise it returns false.**

**6.boolean equalsIgnoreCase(String str):same as above,this is case insensitive comparison.**

**7.boolean startsWith(String suffix):Returns true if the invoking string ends with the substring “suffix”.**

**8.boolean startsWith(String prefix):Returns true if the calling string starts with the substring “prefix”.**

**9.int indexOf(String str):Returns the first occurrences of str in the main string(i.e., calling string).**

**10.int lastIndexOf(String str):Returns the last occurrence of str in the main string.**

**Note:Both the above methods return negative value,if str is not found in the calling string .Counting starts from zero.**

**11.string replace(char old char,char new char):Replaces all occurrences of ‘oldchar’ with ‘newchar’ in the string.**

**12.String substring(int beginIndex):Returns a new string consisting of all characters from beginIndex until the end of the string.**

**13.String substring(int beginIndex,int endIndex):Returns a new string consisting of all characters from beginIndex until endIndex.**

**14.String toLowerCase():converts all characters into lower case.**

**15.String toUpperCase():converts all characters into upper case.**

**16.String trim():Eliminates all leading and trailing spaces.**

**Program-1:**

**class StringEx**

**{**

**public static void main(String args[])**

**{**

**String t="Welcome";**

**System.out.println("Length is:"+t.length());**

**System.out.println("Charecter Position:"+t.charAt(3));**

**System.out.println("Index of:"+t.indexOf('o'));**

**System.out.println("Trimming:"+t.trim());**

**t=t.concat(" India");**

**System.out.println("Concat:"+t);**

**String p[]=t.split(" ");**

**System.out.println("After Split:"+p[0]+ ":"+p[1]);**

**System.out.println("Sub string:"+t.substring(0,5));**

**System.out.println("converting upper case:"+t.toUpperCase());**

**}**

**}**

**Equality of strings:**

**It can be done in 3 ways:**

**1.equals():**

**Program-1:**

**import java.lang.\*;**

**class StringEx2**

**{**

**public static void main(String args[])**

**{**

**String s1="sushma";**

**String s2="sushma";**

**String s3=new String("sushma");**

**String s4="sowmya";**

**System.out.println(s1.equals(s2));**

**System.out.println(s1.equals(s3));**

**System.out.println(s1.equals(s4));**

**}**

**}**

**Program-2 :**

**import java.lang.\*;**

**class StringEx3{**

**public static void main(String args[]) {**

**String s1="sushma";**

**String s2="SUSHMA";**

**System.out.println(s1.equals(s2));**

**System.out.println(s1.equalsIgnoreCase(s2));**

**}}**

**2.compareTo():**

**Program:**

**import java.lang.\*;**

**class StringEx5{**

**public static void main(String args[]){**

**String s1="sushma";**

**String s2="sushma";**

**String s3="sowmya";**

**System.out.println(s1.compareTo(s2));**

**System.out.println(s1.compareTo(s3));**

**System.out.println(s3.compareTo(s1));**

**}}**

**3.== operator**

**Program:**

**import java.lang.\*;**

**class StringEx4{**

**public static void main(String args[]) {**

**String s1="hello";**

**String s2= "hello";**

**if(s1==s2)**

**{**

**System.out.println("both are same");**

**}**

**else**

**{**

**System.out.println("not same");**

**}**

**}**

**}**

## **Final vs immutability :**

**1.** **final modifier applicable for variables where as immutability concept applicable for objects**

**2.** **If reference variable declared as final then we can't perform reassignment for the reference variable it doesn't mean we can't perform any change in that object.**

**3.** **That is by declaring a reference variable as final we won't get any immutability nature .**

**4.** **final and immutability both are different concepts .**

**Example:**

**class Test {**

**public static void main(String[] args) {**

**final StringBuffer sb=new StringBuffer("ashok");**

**sb.append("software");**

**System.out.println(sb);//ashoksoftware**

**sb=new StringBuffer("solutions"); //C.E: cannot assign a value to final variable sb**

**}**

**}**

**In the above example even though "sb" is final we can perform any type of change in the corresponding object. That is through final keyword we are not getting any immutability nature.  
 StringBuffer class:**

**StringBuffer is mutable(modify).StringBuffer is synchronized.**

**Creating StringBuffer class:**

**Type-1:**

**StringBuffer sb=new StringBuffer(“hello”);**

**Type-2:**

**StringBuffer sb=new StringBuffer(50);//initial capacity of a string**

**Type-3:**

**StringBuffer sb=new StringBuffer();//It stores ‘n’ no.of characters**

**Methods:**

**1.StringBuffer append(x):x may be int,float,double,char,string or StringBuffer.It will be append to the calling string buffer.**

**2.StringBuffer insert(int offset,x): x may be int,float,double,char,string or StringBuffer.It will be inserted into the string buffer,at offset,counting starts from zero.**

**3.StringBuffer delete(int start,int end):Removes the characters from start to end-1,counting starts from zero.**

**4.StringBuffer replace(int i,int j,String str):Replaces characters from i to j-1 by the string ‘str’ to the string buffer object.**

**5.StringBuffer reverse():Reverses the character sequence in the string buffer.**

**6.String toString():converts string buffer into a string.**

**7.int length():Returns the length of the string.**

**Program:**

**import java.lang.\*;**

**import java.util.\*;**

**public class stringBuffer**

**{**

**public static void main(String[] args)**

**{**

**Scanner sc=new Scanner(System.in);**

**String str;**

**System.out.print("Enter your name: ");**

**str = sc.nextLine();**

**StringBuffer strbuf = new StringBuffer();**

**strbuf.append(str);**

**System.out.println(strbuf);**

**//append()**

**strbuf.append("Hello");**

**strbuf.append("World");**

**//print HelloWorld**

**System.out.println(strbuf);**

**//insert()**

**strbuf.insert(8,"\_Java ");**

**//print Hello\_Java World**

**System.out.println(strbuf);**

**//reverse()**

**strbuf.reverse();**

**System.out.print("Reversed string : ");**

**System.out.println(strbuf);**

**}**

**}**

**StringBuilder class:**

**String builder is not synchronized.**

**Creating StringBuilder class:**

**Type-1:**

**StringBuilder sb=new StringBuilder(“hello”);**

**Type-2:**

**StringBuilder sb=new StringBuilder(50);//initial capacity of a string**

**Type-3:**

**StringBuilder sb=new StringBuilder();//It stores ‘n’ no.of characters**

**Methods:**

**1.StringBuilder append(x)**

**2.StringBuilder insert(int offset,x)**

**3.StringBuilder delete(int start,int end):**

**4.StringBuilder replace(int I,int j,String str)**

**5.StringBuilder reverse()**

**6.String toString()**

**7.int length()**

**Program:**

**import java.lang.\*;**

**import java.util.\*;**

**public class stringBuffer**

**{**

**public static void main(String[] args)**

**{**

**Scanner sc=new Scanner(System.in);**

**String str;**

**System.out.print("Enter your name: ");**

**str = sc.nextLine();**

**StringBuffer strbuf = new StringBuffer();**

**strbuf.append(str);**

**System.out.println(strbuf);**

**//append()**

**strbuf.append("Hello");**

**strbuf.append("World");**

**//print HelloWorld**

**System.out.println(strbuf);**

**//insert()**

**strbuf.insert(8,"\_Java ");**

**//print Hello\_Java World**

**System.out.println(strbuf);**

**//reverse()**

**strbuf.reverse();**

**System.out.print("Reversed string : ");**

**System.out.println(strbuf);**

**}**

**}**

**Interview Question:**

**What is the difference between String and StringBuffer?**

| **String** | **StringBuffer** |
| --- | --- |
| **1.It is immutable.** | **1.It is mutable.** |
| **2.String is slow and consumes more memory when you concat too many strings because every time it creates new instances.** | **2.It is fast and consumes less memory when you concat strings.** |
| **3.It overrides the equals() method of Object class.So you can compare the contents of two strings by equals() method** | **3.It doesn’t override the equals method of Object class.** |

**Interview Question:**

**What is the difference between StringBuffer and StringBuilder?**

| **StringBuffer** | **StringBuilder** |
| --- | --- |
| **1.It is thread safe or synchronized i.e multiple threads can’t access simultaneously.** | **1.It is not thread safe i.e multiple threads can access at a time.** |
| **2.Its methods are synchronized.** | **2.Its methods are not synchronized.** |
| **3.Introduce in 1.0 version.** | **3.Introduce in 1.5 version.** |
| **4.Performance is slow.** | **4.Performance is high.** |

## **String vs StringBuffer** vs **StringBuilder :**

1. **If the content is fixed and won't change frequently then we should go for String.**
2. **If the content will change frequently but Thread safety is required then we should go for StringBuffer.**
3. **If the content will change frequently and Thread safety is not required then we should go for StringBuilder.**

## **Method chaining:**

**For most of the methods in String, StringBuffer and StringBuilder the return type is same type only. Hence after applying method on the result we can call another method which forms method chaining.  
Example:**

**sb.m1().m2().m3()...............**

1. **In method chaining all methods will be evaluated from left to right.  
   Example:**

**public class StringBufferDemo {**

**public static void main(String[] args) {**

**// Create an instance of StringBuffer**

**StringBuffer sb = new StringBuffer();**

**// Chain method calls to manipulate the string**

**sb.append("ashok") // Append "ashok"**

**.insert(5, "arunkumar") // Insert "arunkumar" at position 5**

**.delete(11, 13) // Delete characters from position 11 to 13**

**.reverse() // Reverse the StringBuffer**

**.append("solutions") // Append "solutions"**

**.insert(18, "abcdf") // Insert "abcdf" at position 18**

**.reverse(); // Reverse again**

**// Print the final result**

**System.out.println(sb); // Output: snofdcbaitulosashokarunkumar**

**}**

**}**

**Important programs**

**=============================**

### **1. Reverse a String**

**This program reverses a given string without using any built-in reverse methods.**

**package com.codegnan.stringexamples;**

**public class StringReversal {**

**public static String reverseString(String str) {**

**// Create an empty string to store the reversed result**

**String reversed = "";**

**// Iterate through the original string from the end**

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

**// Extract the character at the current index**

**char currentChar = str.charAt(i);**

**// Append the character to the reversed string**

**reversed += currentChar;**

**}**

**return reversed;**

**}**

**public static void main(String[] args) {**

**String input = "hello";**

**System.*out*.println("Reversed String: " + *reverseString*(input));**

**}**

**}**

**2. Check if a String is a Palindrome**

**package com.codegnan.stringexamples;**

**public class PalindromeCheck {**

**public static boolean isPalindrome(String str) {**

**// Initialize two pointers: one at the beginning and one at the end**

**int left = 0;**

**int right = str.length() - 1;**

**// Iterate until the pointers meet**

**while (left < right) {**

**// Compare characters at the pointers**

**if (str.charAt(left) != str.charAt(right)) {**

**return false; // If characters don't match, it's not a palindrome**

**}**

**// Move pointers towards the center**

**left++;**

**right--;**

**}**

**// If the loop completes without finding mismatches, it's a palindrome**

**return true;**

**}**

**public static void main(String[] args) {**

**String input1 = "racecar";**

**String input2 = "hello";**

**System.*out*.println(input1 + " is palindrome: " + *isPalindrome*(input1));**

**System.*out*.println(input2 + " is palindrome: " + *isPalindrome*(input2));**

**}**

**}**

**3. Check if Two Strings are Anagrams**

**—------------------------------------------------**

**Definition: An anagram is a word or phrase formed by rearranging the letters of another word or phrase, typically using all the original letters exactly once.**

**"elbow" and "below"**

**"heart" and "earth"**

**"funeral" and "real fun"**

**package com.codegnan.stringexamples;**

**public class AnagramCheck {**

**public static boolean areAnagrams(String str1, String str2) {**

**// If lengths are different, they can't be anagrams**

**if (str1.length() != str2.length()) {**

**return false;**

**}**

**// Create an array to count character frequencies**

**int[] charCounts = new int[26]; // Assuming only lowercase letters**

**// Count character frequencies in str1**

**for (int i = 0; i < str1.length(); i++) {**

**charCounts[str1.charAt(i) - 'a']++;**

**}**

**// Decrement frequencies for characters in str2**

**for (int i = 0; i < str2.length(); i++) {**

**charCounts[str2.charAt(i) - 'a']--;**

**}**

**// Check if all frequencies are zero**

**for (int count : charCounts) {**

**if (count != 0) {**

**return false; // If any frequency is not zero, they are not anagrams**

**}**

**}**

**// If all frequencies are zero, they are anagrams**

**return true;**

**}**

**public static void main(String[] args) {**

**String str1 = "listen";**

**String str2 = "silent";**

**System.*out*.println(str1 + " and " + str2 + " are anagrams: " + *areAnagrams*(str1, str2));**

**}**

**}**

**Or**

**—------------------------------------------**

**import java.util.Arrays;**

**public class AnagramCheck {**

**public static boolean areAnagrams(String str1, String str2) {**

**// Convert both strings to character arrays**

**char[] charArray1 = str1.toCharArray();**

**char[] charArray2 = str2.toCharArray();**

**// Sort both character arrays alphabetically**

**Arrays.sort(charArray1);**

**Arrays.sort(charArray2);**

**// Compare the sorted arrays**

**return Arrays.equals(charArray1, charArray2);**

**}**

**public static void main(String[] args) {**

**String str1 = "listen";**

**String str2 = "silent";**

**// Check if the strings are anagrams and print the result**

**System.out.println(str1 + " and " + str2 + " are anagrams: " + areAnagrams(str1, str2));**

**}**

**}**

**4. Remove Duplicates from a String**

**—-----------------------------------------------**

**package com.codegnan.stringexamples;**

**public class RemoveDuplicates {**

**public static String removeDuplicates(String str) {**

**String result = "";**

**// Iterate through the original string**

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

**// Check if the current character already exists in the result string**

**if (!result.contains(String.*valueOf*(str.charAt(i)))) {**

**// If not, append it to the result**

**result += str.charAt(i);**

**}**

**}**

**return result;**

**}**

**public static void main(String[] args) {**

**String input = "hello";**

**System.*out*.println("String after removing duplicates: " + *removeDuplicates*(input));**

**}**

**}**

**5. Print duplicates in string**

**—---------------------------------------**

**package com.codegnan.stringexamples;**

**public class FindDuplicates {**

**public static String findDuplicates(String str) {**

**String duplicates = "";**

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

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

**if (str.charAt(i) == str.charAt(j) && !duplicates.contains(String.*valueOf*(str.charAt(i)))) {**

**duplicates += str.charAt(i);**

**}**

**}**

**}**

**return duplicates;**

**}**

**public static void main(String[] args) {**

**String string = "hellaao";**

**String duplicates = *findDuplicates*(string);**

**System.*out*.println("Duplicates in the string: " + duplicates);**

**}**

**}**

**6. Program to count the duplicate characters in a string**

**—---------------------------------------------------------------------------**

**package com.codegnan.stringexamples;**

**public class DuplicateCharacterCount {**

**public static int countDuplicates(String str) {**

**int count = 0;**

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

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

**if (str.charAt(i) == str.charAt(j)) {**

**count++;**

**break; // Avoid counting duplicates of the same character multiple times**

**}**

**}**

**}**

**return count;**

**}**

**public static void main(String[] args) {**

**String str = "hellaappmmzzrro";**

**int duplicateCount = *countDuplicates*(str);**

**System.*out*.println("Number of duplicates: " + duplicateCount); // Output: 4**

**}**

**}**

**7. Vowels and consonants in a given string**

**===========================================**

**package com.codegnan.stringexamples;**

**public class VowelConsonantSeparator {**

**public static void main(String[] args) {**

**// 1. Declare and initialize the input string**

**String str = "hello world";**

**// 2. Declare empty strings to store vowels and consonants**

**String vowels = "";**

**String consonants = "";**

**// 3. Iterate through each character in the string**

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

**// 4. Get the current character and convert it to lowercase**

**char ch = Character.*toLowerCase*(str.charAt(i));**

**// 5. Check if the character is a vowel**

**if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') {**

**// 6. Append the vowel to the vowels string**

**vowels += ch;**

**}**

**// 7. Check if the character is a letter (consonant)**

**else if (Character.*isLetter*(ch)) {**

**// 8. Append the consonant to the consonants string**

**consonants += ch;**

**}**

**}**

**// 9. Print the extracted vowels and consonants**

**System.*out*.println("Vowels: " + vowels);**

**System.*out*.println("Consonants: " + consonants);**

**}**

**}**

**8. Vowels and consonant counts in a given string**

**—---------------------------------------------------------------**

**package com.codegnan.stringexamples;**

**public class VowelConsonantCounter {**

**public static void countVowelsConsonants(String str) {**

**// 1. Initialize counters for vowels and consonants**

**int vowelCount = 0;**

**int consonantCount = 0;**

**// 2. Iterate through each character in the string**

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

**// 3. Get the current character and convert it to lowercase**

**char ch = Character.*toLowerCase*(str.charAt(i));**

**// 4. Check if the character is a vowel**

**if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') {**

**// 5. Increment the vowel count**

**vowelCount++;**

**}**

**// 6. Check if the character is a letter (consonant)**

**else if (Character.*isLetter*(ch)) {**

**// 7. Increment the consonant count**

**consonantCount++;**

**}**

**}**

**// 8. Print the counts of vowels and consonants**

**System.*out*.println("Vowel count: " + vowelCount);**

**System.*out*.println("Consonant count: " + consonantCount);**

**}**

**public static void main(String[] args) {**

**// 9. Declare and initialize the input string**

**String str = "Hello World!";**

**// 10. Call the method to count vowels and consonants**

***countVowelsConsonants*(str);**

**}**

**}**

**10.. Count Occurrences of a Character in a String**

**=----------------------------------------------------------------------**

**package com.codegnan.stringexamples;**

**public class CountCharacterOccurrences {**

**public static int countOccurrences(String str, char c) {**

**// 1. Initialize a counter to keep track of occurrences**

**int count = 0;**

**// 2. Iterate through each character in the string**

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

**// 3. Check if the current character matches the target character**

**if (str.charAt(i) == c) {**

**// 4. If a match is found, increment the count**

**count++;**

**}**

**}**

**// 5. Return the total count of occurrences**

**return count;**

**}**

**public static void main(String[] args) {**

**// 6. Define the input string and the target character**

**String inputString = "hello world";**

**char targetChar = 'l';**

**// 7. Call the countOccurrences method to get the result**

**int occurrences = *countOccurrences*(inputString, targetChar);**

**// 8. Print the result to the console**

**System.*out*.println("Occurrences of '" + targetChar + "' in '" + inputString + "': " + occurrences);**

**}**

**}**

**11. Count Occurrence Of each character in a given String**

**—-----------------------------------------------------------------------------------**

**package com.codegnan.stringexamples;**

**public class CharacterCount {**

**public static void countCharacterOccurrences(String str) {**

**// 1. Initialize an empty string to store unique characters encountered**

**String uniqueChars = "";**

**// 2. Outer loop to iterate through each character in the string**

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

**char currentChar = str.charAt(i);**

**// 3. Check if the current character has already been processed**

**if (!uniqueChars.contains(String.*valueOf*(currentChar))) {**

**// 4. If not processed, add the character to the list of unique characters**

**uniqueChars += currentChar;**

**// 5. Initialize a counter for the current character**

**int count = 0;**

**// 6. Inner loop to count occurrences of the current character**

**for (int j = 0; j < str.length(); j++) {**

**// 7. Compare the current character with each character in the string**

**if (str.charAt(j) == currentChar) {**

**// 8. Increment the count if a match is found**

**count++;**

**}**

**}**

**// 9. Print the character and its count**

**System.*out*.println(currentChar + ": " + count);**

**}**

**}**

**}**

**public static void main(String[] args) {**

**String str = "hello world";**

***countCharacterOccurrences*(str);**

**}**

**}**

**Or**

**package com.codegnan.stringexamples;**

**import java.util.HashMap;**

**import java.util.Map;**

**public class CharacterCount {**

**public static void countCharacterOccurrences(String str) {**

**// 1. Create a HashMap to store character frequencies**

**Map<Character, Integer> charCountMap = new HashMap<>();**

**// 2. Iterate through each character in the string**

**for (char c : str.toCharArray()) {**

**// 3. Get the current count of the character from the map**

**int count = charCountMap.getOrDefault(c, 0);**

**// 4. Update the count of the character in the map**

**charCountMap.put(c, count + 1);**

**}**

**// 5. Print the character frequencies**

**for (Map.Entry<Character, Integer> entry : charCountMap.entrySet()) {**

**System.*out*.println(entry.getKey() + ": " + entry.getValue());**

**}**

**}**

**public static void main(String[] args) {**

**String str = "hello world";**

***countCharacterOccurrences*(str);**

**}**

**}**

**12. Remove white space**

**=============================**

**package com.codegnan.stringexamples;**

**public class RemoveWhitespace {**

**public static String removeWhitespace(String str) {**

**// 1. Create an empty string to store the result**

**String result = "";**

**// 2. Iterate through each character in the string**

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

**char ch = str.charAt(i);**

**// 3. Check if the character is not a space, tab, or newline**

**if (ch != ' ' && ch != '\t' && ch != '\n') {**

**// 4. Append the character to the result string**

**result += ch;**

**}**

**}**

**// 5. Return the string without whitespace**

**return result;**

**}**

**public static void main(String[] args) {**

**String str = " This is a string with whitespace ";**

**String result = *removeWhitespace*(str);**

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

**System.*out*.println("String without whitespace: " + result);**

**}**

**}**

**Or**

**package com.codegnan.stringexamples;**

**public class RemoveWhitespace {**

**public static String removeWhitespace(String str) {**

**// 1. Create an empty string to store the result**

**String result = "";**

**// 2. Iterate through each character in the string**

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

**// 3. Get the current character**

**char ch = str.charAt(i);**

**// 4. Check if the character is not a whitespace character**

**if (!Character.*isWhitespace*(ch)) {**

**// 5. Append the character to the result string**

**result += ch;**

**}**

**}**

**// 6. Return the string without whitespace**

**return result;**

**}**

**public static void main(String[] args) {**

**String str = " This is a string with whitespace ";**

**String result = *removeWhitespace*(str);**

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

**System.*out*.println("String without whitespace: " + result);**

**}**

**}**