Java Assignment 6

Java Strings Programs

Java Program to Convert char to String and String to Char

To convert a `char` to a `String` in Java, you can use the `String.valueOf()` method or simply concatenate the `char` with an empty string. To convert a `String` to a `char`, you can use the `charAt()` method. Here's a Java program that demonstrates these conversions with sample output:

```java
public class CharStringConversion {
    public static void main(String[] args) {
        // Convert char to String
        char charValue = 'A';
        String stringValue1 = String.valueOf(charValue);
        String stringValue2 = charValue + "";

        System.out.println("Converting char to String:");
        System.out.println("charValue: " + charValue);
        System.out.println("stringValue1: " + stringValue1);
        System.out.println("stringValue2: " + stringValue2);

        // Convert String to char
        String stringValue = "Hello";
        char charValue1 = stringValue.charAt(0); // Get the first character
        char charValue2 = stringValue.charAt(3); // Get the fourth character

        System.out.println("\nConverting String to char:");
        System.out.println("stringValue: " + stringValue);
        System.out.println("charValue1: " + charValue1);
        System.out.println("charValue2: " + charValue2);
    }
}
```

Output:

```
Converting char to String:
charValue: A
stringValue1: A
stringValue2: A

Converting String to char:
stringValue: Hello
charValue1: H
charValue2: l
```

In the program, we first convert a `char` to a `String` using two different methods and then convert a `String` to a `char` by using the `charAt()` method to access specific characters in the `String`.

Java Program to find duplicate characters in a String

You can find duplicate characters in a Java string by iterating through the characters in the string and using a `HashMap` to keep track of the character frequency. Here's a Java program to find duplicate characters in a string with sample output:

```java
import java.util.HashMap;
import java.util.Map;

public class DuplicateCharacters {
    public static void main(String[] args) {
        String inputString = "programming";
        Map<Character, Integer> charFrequency = new HashMap<>();

        // Count the frequency of each character in the string
        for (char c : inputString.toCharArray()) {
            charFrequency.put(c, charFrequency.getOrDefault(c, 0) + 1);
        }

        // Display duplicate characters
        System.out.println("Duplicate characters in the string \"" + inputString + "\":");
        for (Map.Entry<Character, Integer> entry : charFrequency.entrySet()) {
            if (entry.getValue() > 1) {
                System.out.println(entry.getKey() + " appears " + entry.getValue() + " times");
            }
        }
    }
}
```

Output for the input string "programming":

```
Duplicate characters in the string "programming":
g appears 2 times
r appears 2 times
m appears 2 times
```

In this program, we create a `HashMap` to store the frequency of each character in the input string. We iterate through the characters in the string and update the frequency in the map. Afterward, we iterate through the map entries and print the characters that have a frequency greater than 1, indicating they are duplicate characters in the string.

Java Program to check Palindrome String using Stack, Queue, For and While loop

You can check if a string is a palindrome (reads the same forwards and backward) using different methods such as a stack, a queue, for loop, and while loop. Here's a Java program that demonstrates each approach with sample output:

```java
import java.util.Stack;
import java.util.LinkedList;
import java.util.Queue;

public class PalindromeCheck {
    public static void main(String[] args) {
        String inputString = "racecar";

        // Using a Stack
        boolean isPalindromeUsingStack = isPalindromeUsingStack(inputString);
        System.out.println("Using Stack: " + isPalindromeUsingStack);

        // Using a Queue
        boolean isPalindromeUsingQueue = isPalindromeUsingQueue(inputString);
        System.out.println("Using Queue: " + isPalindromeUsingQueue);

        // Using a For Loop
        boolean isPalindromeUsingForLoop = isPalindromeUsingForLoop(inputString);
        System.out.println("Using For Loop: " + isPalindromeUsingForLoop);

        // Using a While Loop
        boolean isPalindromeUsingWhileLoop = isPalindromeUsingWhileLoop(inputString);
        System.out.println("Using While Loop: " + isPalindromeUsingWhileLoop);
    }

    // Function to check palindrome using a stack
    private static boolean isPalindromeUsingStack(String str) {
        Stack<Character> stack = new Stack<>();
        for (char c : str.toCharArray()) {
            stack.push(c);
        }

        String reversedString = "";
        while (!stack.isEmpty()) {
            reversedString += stack.pop();
        }

        return str.equals(reversedString);
    }
```

```
// Function to check palindrome using a queue
 private static boolean isPalindromeUsingQueue(String str) {
 Queue<Character> queue = new LinkedList<>();
 for (char c : str.toCharArray()) {
 queue.add(c);
 }

 String reversedString = "";
 while (!queue.isEmpty()) {
 reversedString += queue.poll();
 }
 return str.equals(reversedString);
 }

 // Function to check palindrome using a for loop
 private static boolean isPalindromeUsingForLoop(String str) {
 int length = str.length();
 for (int i = 0; i < length / 2; i++) {
 if (str.charAt(i) != str.charAt(length - i - 1)) {
 return false;
 }
 }
 return true;
 }

 // Function to check palindrome using a while loop
 private static boolean isPalindromeUsingWhileLoop(String str) {
 int start = 0;
 int end = str.length() - 1;
 while (start < end) {
 if (str.charAt(start) != str.charAt(end)) {
 return false;
 }
 start++;
 end--;
 }
 return true;
 }
}
```

Sample Output:

```
Using Stack: true
Using Queue: true
Using For Loop: true
Using While Loop: true
```

This program checks if the input string "racecar" is a palindrome using a stack, a queue, a for loop, and a while loop. All four methods correctly identify the string as a palindrome.

Java Program to sort strings in alphabetical order

You can sort an array of strings in alphabetical order in Java using the `Arrays.sort()` method or by implementing your own sorting algorithm. Here's a Java program that demonstrates both approaches with sample output:

```
import java.util.Arrays;
public class SortStringsAlphabetically {
    public static void main(String[] args) {
        // Array of strings
        String[] strings = {"apple", "banana", "cherry", "date", "fig"};
        // Sort using Arrays.sort()
        Arrays.sort(strings);
        System.out.println("Sorted strings using Arrays.sort():");
        for (String str : strings) {
            System.out.println(str);
        }
        // Sort using custom sorting algorithm
        sortStringsCustom(strings);
        System.out.println("\nSorted strings using custom sorting algorithm:");
        for (String str : strings) {
            System.out.println(str);
        }
    }
    // Custom sorting algorithm (Bubble Sort) to sort strings
    private static void sortStringsCustom(String[] strings) {
        int n = strings.length;
        boolean swapped;
        do {
            swapped = false;
            for (int i = 1; i < n; i++) {
                if (strings[i - 1].compareTo(strings[i]) > 0) {
                    // Swap the strings
                    String temp = strings[i - 1];
                    strings[i - 1] = strings[i];
                    strings[i] = temp;
                    swapped = true;
                }
            }
        } while (swapped);
    }
}
```

Sample Output:

```
Sorted strings using Arrays.sort():
apple
banana
cherry
date
fig
```

Java Program to reverse words in a String

You can reverse the words in a string in Java by splitting the string into words, reversing each word, and then joining them back together. Here's a Java program to reverse words in a string with sample output:

```java
public class ReverseWordsInString {
    public static void main(String[] args) {
        String inputString = "Hello World Java Program";

        // Split the string into words
        String[] words = inputString.split(" ");

        // Reverse each word and build the reversed string
        StringBuilder reversedString = new StringBuilder();
        for (int i = 0; i < words.length; i++) {
            String word = words[i];
            String reversedWord = reverseWord(word);
            reversedString.append(reversedWord);

            // Add a space between words except for the last word
            if (i < words.length - 1) {
                reversedString.append(" ");
            }
        }

        System.out.println("Original string: " + inputString);
        System.out.println("Reversed string: " + reversedString.toString());
    }

    // Function to reverse a word
    private static String reverseWord(String word) {
        StringBuilder reversedWord = new StringBuilder();
        for (int i = word.length() - 1; i >= 0; i--) {
            reversedWord.append(word.charAt(i));
        }
        return reversedWord.toString();
    }
}
```

Sample Output:

```
Original string: Hello World Java Program
Reversed string: olleH dlroW avaJ margorP
```

In this program, we split the input string into words using the space character as the delimiter. Then, we reverse each word using the `reverseWord` function, and finally, we build the reversed string by joining the reversed words with spaces in between.

Java Program to perform bubble sort on Strings

Bubble sort is a simple sorting algorithm that can be applied to sort strings as well. Here's a Java program that demonstrates the bubble sort algorithm for sorting an array of strings in alphabetical order, along with sample output:

```java
public class BubbleSortStrings {
    public static void main(String[] args) {
        // Array of strings
        String[] strings = {"apple", "banana", "cherry", "date", "fig"};

        // Sort the strings using bubble sort
        bubbleSort(strings);

        System.out.println("Sorted strings using Bubble Sort:");
        for (String str : strings) {
            System.out.println(str);
        }
    }

    // Bubble Sort for strings
    private static void bubbleSort(String[] arr) {
        int n = arr.length;
        boolean swapped;
        do {
            swapped = false;
            for (int i = 1; i < n; i++) {
                if (arr[i - 1].compareTo(arr[i]) > 0) {
                    // Swap the strings
                    String temp = arr[i - 1];
                    arr[i - 1] = arr[i];
                    arr[i] = temp;
                    swapped = true;
                }
            }
        } while (swapped);
    }
}
```

Sample Output:

```
Sorted strings using Bubble Sort:
apple
banana
cherry
date
fig
```

Java program to find occurrence of a character in a String

To find the occurrence of a character in a Java string, you can iterate through the string and count how many times the character appears. Here's a Java program that does that:

```java
public class CharacterOccurrenceInString {
    public static void main(String[] args) {
        String input = "Hello, World!";
        char target = 'o';

        int occurrenceCount = countCharacterOccurrences(input, target);

        System.out.println("The character '" + target + "' appears " + occurrenceCount + " times in the string.");
    }

    public static int countCharacterOccurrences(String str, char target) {
        int count = 0;

        for (int i = 0; i < str.length(); i++) {
            if (str.charAt(i) == target) {
                count++;
            }
        }

        return count;
    }
}
```

In this program, we have a method called `countCharacterOccurrences` that takes the input string and the target character as parameters. It iterates through the string and counts how many times the target character appears. The result is then printed to the console.

When you run this program with the input string "Hello, World!" and the target character 'o', it will output:

```
The character 'o' appears 2 times in the string.
```

You can use this program to find the occurrence of any character in a given string.

Java program to count vowels and consonants in a String

You can create a Java program to count the number of vowels and consonants in a given string by iterating through the characters in the string and checking whether each character is a vowel or a consonant. Here's a Java program to do just that, along with sample output:

```java
public class CountVowelsConsonants {
    public static void main(String[] args) {
        String inputString = "Hello, World!";

        // Remove non-alphabetic characters and convert to lowercase
        String cleanString = inputString.replaceAll("[^a-zA-Z]", "").toLowerCase();

        int vowelCount = 0;
        int consonantCount = 0;

        for (int i = 0; i < cleanString.length(); i++) {
            char ch = cleanString.charAt(i);

            if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') {
                vowelCount++;
            } else {
                consonantCount++;
            }
        }

        System.out.println("Original string: " + inputString);
        System.out.println("Cleaned and lowercased string: " + cleanString);
        System.out.println("Vowels: " + vowelCount);
        System.out.println("Consonants: " + consonantCount);
    }
}
```

Sample Output:

```
Original string: Hello, World!
Cleaned and lowercased string: hello
Vowels: 2
Consonants: 3
```

In this program, we first remove non-alphabetic characters and convert the string to lowercase to simplify the counting process. Then, we iterate through the cleaned and lowercased string, checking each character to determine if it's a vowel (a, e, i, o, u) or a consonant. We maintain separate counters for vowels and consonants and print the counts at the end.

Java Program to check two strings are anagram or not

An anagram is a word or phrase formed by rearranging the letters of another. To check if two strings are anagrams, you can compare whether they contain the same set of characters with the same frequency. Here's a Java program to check if two strings are anagrams or not, along with sample output:

```
import java.util.Arrays;
public class AnagramCheck {
    public static void main(String[] args) {
        String str1 = "listen";
        String str2 = "silent";
        boolean areAnagrams = areAnagrams(str1, str2);
        System.out.println("String 1: " + str1);
        System.out.println("String 2: " + str2);
        if (areAnagrams) {
            System.out.println("The strings are anagrams.");
        } else {
            System.out.println("The strings are not anagrams.");
        }
    }
    // Function to check if two strings are anagrams
    private static boolean areAnagrams(String str1, String str2) {
        // Remove spaces and convert to lowercase
        str1 = str1.replaceAll("\\s", "").toLowerCase();
        str2 = str2.replaceAll("\\s", "").toLowerCase();
        // Check if the lengths of the strings are the same
        if (str1.length() != str2.length()) {
            return false;
        }
        // Convert strings to character arrays and sort them
        char[] charArray1 = str1.toCharArray();
        char[] charArray2 = str2.toCharArray();
        Arrays.sort(charArray1);
        Arrays.sort(charArray2);
        // Compare the sorted character arrays
        return Arrays.equals(charArray1, charArray2);
    }
}
```

Sample Output:

```
String 1: listen
String 2: silent
The strings are anagrams.
```

In this program, we remove spaces and convert both strings to lowercase to ensure case-insensitive comparison. Then, we check if the lengths of the two strings are equal. If they have the same length, we convert the strings to character arrays, sort the arrays, and compare them. If the sorted character arrays are equal, the strings are considered anagrams.

Java Program to divide a string in ‘n’ equal parts

You can create a Java program to divide a string into 'n' equal parts. Here's a Java program that demonstrates how to do this with sample output:

```java
public class DivideString {
    public static void main(String[] args) {
        String inputString = "This is a sample string to divide into equal parts.";
        int n = 5;

        String[] parts = divideString(inputString, n);

        System.out.println("Original string: " + inputString);
        System.out.println("Divided into " + n + " equal parts:");
        for (String part : parts) {
            System.out.println(part);
        }
    }

    // Function to divide a string into 'n' equal parts
    private static String[] divideString(String str, int n) {
        int length = str.length();
        int partSize = length / n;
        String[] parts = new String[n];

        for (int i = 0; i < n; i++) {
            int startIndex = i * partSize;
            int endIndex = startIndex + partSize;
            parts[i] = str.substring(startIndex, endIndex);
        }
        return parts;
    }
}
```

Sample Output:

```
Original string: This is a sample string to divide into equal parts.
Divided into 5 equal parts:
This
is a
 sample
string
 to divide into equal parts.
```

In this program, we divide the input string into 'n' equal parts by calculating the length of each part. We then use the `substring` method to extract the parts from the original string, and the resulting parts are stored in an array. Finally, we print the divided parts of the string.

Java Program to find all subsets of a string

To find all subsets of a string, you can use a recursive approach. Each character in the string has two choices: either be part of a subset or not. By exploring all possible combinations, you can generate all subsets of the string. Here's a Java program to find all subsets of a string with sample output:

```java
public class AllSubsetsOfString {
    public static void main(String[] args) {
        String inputString = "abc";
        System.out.println("Subsets of \"" + inputString + "\":");
        findSubsets(inputString, "");
    }

    // Recursive function to find all subsets of a string
    private static void findSubsets(String remaining, String currentSubset) {
        if (remaining.length() == 0) {
            System.out.println(currentSubset);
        } else {
            // Include the current character in the subset
            findSubsets(remaining.substring(1), currentSubset + remaining.charAt(0));

            // Exclude the current character from the subset
            findSubsets(remaining.substring(1), currentSubset);
        }
    }
}
```

Sample Output:

```
Subsets of "abc":
abc
ab
ac
a
bc
b
c
```

In this program, we use a recursive function `findSubsets` to generate all subsets of the input string. The function takes two arguments: `remaining`, which is the part of the string that has not been considered yet, and `currentSubset`, which is the subset currently being constructed.

The base case occurs when `remaining` is an empty string, at which point we print the `currentSubset`. In the recursive step, we consider two possibilities for the next character: include it in the current subset or exclude it. This leads to the generation of all possible subsets of the string.

Java Program to find longest substring without repeating characters

To find the longest substring in a string without repeating characters, you can use the sliding window approach. Here's a Java program to accomplish this, along with sample output:

```
import java.util.HashMap;
import java.util.Map;
public class LongestSubstringWithoutRepeatingChars {
    public static void main(String[] args) {
        String inputString = "abcabcbb";
        String longestSubstring = findLongestSubstringWithoutRepeatingChars(inputString);
        System.out.println("Input string: " + inputString);
        System.out.println("Longest substring without repeating characters: " + longestSubstring);
    }
    public static String findLongestSubstringWithoutRepeatingChars(String s) {
        int n = s.length();
        int maxLength = 0;
        int start = 0;
        int end = 0;
        Map<Character, Integer> charIndexMap = new HashMap<>();
        int left = 0;
        int right = 0;
        while (right < n) {
            char currentChar = s.charAt(right);
            if (charIndexMap.containsKey(currentChar) && charIndexMap.get(currentChar) >= left) {
                left = charIndexMap.get(currentChar) + 1;
            }
            charIndexMap.put(currentChar, right);
            if (right - left > maxLength) {
                maxLength = right - left;
                start = left;
                end = right;
            }
            right++;
        }
        return s.substring(start, end + 1);
    }
}
```

Sample Output:

```
Input string: abcabcbb
Longest substring without repeating characters: abc
```

In this program, we use a sliding window to keep track of the longest substring without repeating characters. We maintain two pointers, `left` and `right`, and use a `charIndexMap` to store the most recent index of each character in the string.

We iterate through the input string and check if the current character has already been encountered within the current window (i.e., its index is greater than or equal to `left`). If it has, we update the `left` pointer to the next position after the last occurrence of that character.

The `maxLength` variable keeps track of the length of the longest substring without repeating characters, and `start` and `end` store the indices of the longest substring. Finally, we extract the longest substring using the `substring` method and return it.

Java Program to find longest repeating sequence in a string

To find the longest repeating sequence in a string, you can use a variation of the Longest Common Substring algorithm. Here's a Java program to accomplish this with sample output:

```
public class LongestRepeatingSequence {
    public static void main(String[] args) {
        String inputString = "banana";
        String longestRepeatingSequence = findLongestRepeatingSequence(inputString);
        System.out.println("Input string: " + inputString);
        System.out.println("Longest repeating sequence: " + longestRepeatingSequence);
    }
    public static String findLongestRepeatingSequence(String str) {
        int n = str.length();
        int[][] dp = new int[n + 1][n + 1];
        int maxLength = 0;
        int endIndex = 0;
        for (int i = 1; i <= n; i++) {
            for (int j = i + 1; j <= n; j++) {
                if (str.charAt(i - 1) == str.charAt(j - 1)) {
                    dp[i][j] = dp[i - 1][j - 1] + 1;
                    if (dp[i][j] > maxLength) {
                        maxLength = dp[i][j];
                        endIndex = i - 1;
                    }
                } else {
                    dp[i][j] = 0;
                }
            }
        }
        if (maxLength > 0) {
            return str.substring(endIndex - maxLength + 1, endIndex + 1);
        } else {
            return "No repeating sequence found";
        }
    }
}
```

Sample Output:

```
Input string: banana
Longest repeating sequence: ana
```

In this program, we use dynamic programming to find the longest repeating sequence in the input string. The `dp` array stores the length of the common substring at the given indices. We iterate through the string, and when we find two characters that match, we increment the length of the common substring. If the length exceeds the previous maximum, we update `maxLength` and `endIndex`.

Finally, we extract the longest repeating sequence using the `substring` method and return it. If no repeating sequence is found, we return a message indicating that.

Java Program to remove all the white spaces from a string

You can remove all the white spaces from a string in Java using the `replaceAll()` method. Here's a Java program to demonstrate this, along with sample output:

```java
public class RemoveWhiteSpaces {
    public static void main(String[] args) {
        String inputString = "This is a sample string with white spaces.";

        String stringWithoutSpaces = removeWhiteSpaces(inputString);

        System.out.println("Original string: " + inputString);
        System.out.println("String without white spaces: " + stringWithoutSpaces);
    }

    public static String removeWhiteSpaces(String str) {
        return str.replaceAll("\\s", "");
    }
}
```

Sample Output:

```
Original string: This is a sample string with white spaces.
String without white spaces: Thisisasamplestringwithwhitespaces.
```

In this program, we use the `replaceAll()` method to replace all white spaces (including spaces, tabs, and newlines) with an empty string, effectively removing them from the input string. The resulting string is one where all white spaces have been removed.