1.Write a program to get the output:

Input: a1b10

Output: abbbbbbbbb

Input: b3c6d15

Output: bbbccccccddddddddddddddd

CODE :

```
import java.util.Scanner;
public class PrintASCount{
    public static void main(String []args){
      int count = 0;
      String result = "";
      Scanner scan = new Scanner(System.in);
      System.out.println("Input : ");
      String s1 = scan.nextLine();
      char ch1 = s1.charAt(0);
      for (int i = 0; i < s1.length(); i++) {
         if (Character.isDigit(s1.charAt(i))) {
            count = count * 10 + Character.digit(s1.charAt(i), 10);
         } else {
            for (int j = 0; j < count; j++) {
               result += ch1;
            }
            ch1 = s1.charAt(i);
            count = 0;
         }
      }
      for (int j = 0; j < count; j++) {
         result += ch1;
      }
      System.out.println(result);
    }
}
```

2.Compression of String

Input: AAABBC

Output: A3B2C (or) A3BBC

Input: AAABBCCCDE

Output: A3B2C3DE (or) A3BBC3DE

CODE :

```
import java.util.Scanner;
public class CompressionOfString{
    public static void main(String []args){
      Scanner scan = new Scanner(System.in);
      System.out.println("Enter the string: ");
      String s = scan.nextLine();
      StringBuilder result = new StringBuilder();
      char currentChar = s.charAt(0);
      int count = 1;
      for (int i = 1; i < s.length(); i++) {
         char nextChar = s.charAt(i);
         if (currentChar == nextChar) {
            count++;
         } else {
            result.append(currentChar);
            result.append(count);
            count = 1;
            currentChar = nextChar;
         }
      }
      result.append(currentChar);
      result.append(count);
      System.out.println(result.toString());
    }
}
```

3. Write a program for the following.

Input: 1213

Output: One Thousand Two Hundred and Thirteen

Input range: 0-99999

CODE :

```
import java.util.Scanner;

public class HelloWorld{

    private static final String[] units = {"", "one", "two", "three", "four", "five", "six", "seven", "eight",
"nine"};

   private static final String[] teens = {"ten", "eleven", "twelve", "thirteen", "fourteen", "fifteen",
"sixteen", "seventeen", "eighteen", "nineteen"};

   private static final String[] tens = {"", "", "twenty", "thirty", "forty", "fifty", "sixty", "seventy",
"eighty", "ninety"};

    public static void main(String []args){

        Scanner scanner = new Scanner(System.in);

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

      int number = scanner.nextInt();

      System.out.println(convertToWords(number));

    }

     public static String convertToWords(int number) {

      if (number == 0) {

         return "zero";

      }

      return convertToWordsHelper(number).trim();

   }

   private static String convertToWordsHelper(int number) {

      String words = "";

      if (number < 10) {

         words += units[number];

      } else if (number < 20) {

         words += teens[number - 10];

      } else if (number < 100) {

         words += tens[number / 10] + " " + units[number % 10];
```

```
        } else if (number < 1000) {
            words += units[number / 100] + " hundred " + convertToWordsHelper(number % 100);
        } else if (number < 10000) {
            words += units[number / 1000] + " thousand " + convertToWordsHelper(number % 1000);
        } else {
            words += "Number is too large to convert.";
        }
        return words;
    }
}
```

4. Compare 2 equal length strings and find the mismatched pair of strings

Input:

str * 1 = "antonyandcleopatra"

str * 2 = "antaniandcleapadra"

compare(str1,str2)

Output:

o, a

y.i

o, a

t,d

Input:

str1 = "abcddefgikom"

str * 2 ="abdcdeffgklm^ prime prime

compare(str1,str2)

Output:

cd, dc

gi,fg

0.1

CODE :

```
import java.util.Scanner;
public class StringComparisons{
    public static void main(String []args){
     String s1="antonyandcleopatra";
```

```
 String s2="antaniandcleapadra";
    if(s1.length()!=s2.length()){
        System.out.println("the length of strings are not equal");
    }
    else{
        for(int i=0;i<s1.length();i++){
            if(s1.charAt(i)!=s2.charAt(i)){
                System.out.println(s1.charAt(i)+","+s2.charAt(i));
            }
        }
    }
   }
}
```

5.Justify the Text:

Input:

Text = Zoho_Corp_Madurai //(length is 17) Peuuing= 25

-Space between the strings should be evenly distributed.

Output: Zoho_Corp_Madurai //(lenth is 25)

CODE :

```
public class HelloWorld {
    public static void main(String[] args) {
        String input = "zoho_corp_madurai";
        int desiredLength = 25;
        char paddingChar = '_';

        String paddedString = padString(input, desiredLength, paddingChar);

        System.out.println("Input: " + input + " // Length: " + input.length());
        System.out.println("Output: " + paddedString + " // Length: " + paddedString.length());
    }

    public static String padString(String input, int desiredLength, char paddingChar) {
        // Split the input string into segments
```

```
        String[] segments = input.split("_");
        int numSegments = segments.length;

        // Calculate total number of padding spaces needed
        int totalPaddingSpaces = desiredLength - input.length();

        // Calculate number of padding spaces between each segment
        int paddingSpacesPerSegment = numSegments > 1 ? totalPaddingSpaces / (numSegments - 1) :
0;

        // Calculate remaining padding spaces
        int remainingPaddingSpaces = numSegments > 1 ? totalPaddingSpaces % (numSegments - 1) :
totalPaddingSpaces;

        // Construct padded string with evenly distributed padding spaces
        StringBuilder paddedString = new StringBuilder();
        for (int i = 0; i < numSegments; i++) {
            paddedString.append(segments[i]);
            // Add padding spaces between segments (except after the last segment)
            if (i < numSegments - 1) {
                for (int j = 0; j < paddingSpacesPerSegment; j++) {
                    paddedString.append(paddingChar);
                }
                // Add remaining padding spaces evenly until exhausted
                if (remainingPaddingSpaces > 0) {
                    paddedString.append(paddingChar);
                    remainingPaddingSpaces--;
                }
            }
        }
        return paddedString.toString();
    }
}
```

6.Palindrome

Rule: Skip the special characters (!@#$%^&*)

Input: malayalam

Output: True

Input: m@ala$$y*a &lam (malayalam)

Output: True

Input: Something

Output: false

CODE :

```
/* Online Java Compiler and Editor */
public class StringPalindrome{
    public static void main(String []args){
       String s = "m@ala$$y*a&lam";
       String reverse = "";
       int len = s.length();
       for(int i=(len-1);i>=0;--i){
          if(s.charAt(i)=='@'||s.charAt(i)=='$'||s.charAt(i)=='*'||s.charAt(i)=='&'){
             continue;
          }
           reverse = reverse + s.charAt(i);
       }
       String string_without_special=s.replaceAll("['@$*&']","");
       String reverse_without_special=reverse.replaceAll("['@$*&']","");
       System.out.println(reverse);
          if(string_without_special.equals(reverse_without_special)){
             System.out.println("true");
          }
          else{
             System.out.println("false");
          }
    }
}
```

7.Permutation:

For a given string, find all it's permutations without repetition

Input: Good

Output: Good, Godo, Gdoo, dooG, doGo, dGoo, oGdo, oGod,...

CODE :

```
import java.util.ArrayList;
import java.util.List;
public class Permutations {
   public static void main(String[] args) {
      String input = "good";
      List<String> permutations = findPermutations(input);
      for (String permutation : permutations) {
         System.out.println(permutation);
      }
   }
   public static List<String> findPermutations(String input) {
      List<String> permutations = new ArrayList<>();
      backtrack("", input, permutations);
      return permutations;
   }
   private static void backtrack(String currentPermutation, String remainingChars, List<String>
permutations) {
      if (remainingChars.length() == 0) {
         permutations.add(currentPermutation);
         return;
      }
      for (int i = 0; i < remainingChars.length(); i++) {
         char currentChar = remainingChars.charAt(i);
         String newPermutation = currentPermutation + currentChar;
         String newRemainingChars = remainingChars.substring(0, i) + remainingChars.substring(i + 1);
         backtrack(newPermutation, newRemainingChars, permutations);
      }
   }
}
```

9.Print the vowels count in the given String.

Input: India

Output:

a:1

e / 0

1:2

0:0

u:0

CODE :

```
import java.util.Scanner;

public class HelloWorld{

    public static void main(String []args){

      String input = "india";

     int[] vowelCount = new int[5]; // 'a', 'e', 'i', 'o', 'u'

       input = input.toLowerCase(); // Convert to lowercase to handle both upper and lower case
vowels


       for (char ch : input.toCharArray()) {

          switch (ch) {

             case 'a':

                vowelCount[0]++;

                break;

             case 'e':

                vowelCount[1]++;

                break;

             case 'i':

                vowelCount[2]++;

                break;

             case 'o':

                vowelCount[3]++;

                break;

             case 'u':

                vowelCount[4]++;
```

```
                break;
            }
        }

        System.out.println("a-" + vowelCount[0] + "\ne-" + vowelCount[1] + "\ni-" + vowelCount[2] +
"\no-" + vowelCount[3] + "\nu-" + vowelCount[4]);
    }
}
```

10.Print the next Palindrome number for the given input.

Input: 123

Output: 131

Input: 12345

Output: 12421

Note: Don't use Brute Force Method

CODE :

```
public class HelloWorld {
    public static void main(String[] args) {
        int input = 12321;
        int nextPalindrome = findNextPalindrome(input);
        System.out.println("Next palindrome number after " + input + " is: " + nextPalindrome);
    }

    public static int findNextPalindrome(int input) {
        char[] digits = String.valueOf(input).toCharArray();
        int n = digits.length;

        // Find the middle index for odd-length numbers, or the left half for even-length numbers
        int left = n / 2 - 1;
        int right = (n % 2 == 0) ? n / 2 : n / 2 + 1;

        // Increment the middle and adjust digits if necessary to ensure a palindrome
        while (left >= 0 && right < n && digits[left] == digits[right]) {
```

```
            left--;
            right++;
        }

        // If the number is already a palindrome, increase the central digit(s)
        if (left < 0 || digits[left] < digits[right]) {
            while (left >= 0) {
                digits[right] = digits[left];
                left--;
                right++;
            }
        } else {
            // Mirror the left half to the right half to construct the next palindrome
            while (left >= 0) {
                digits[right] = digits[left];
                left--;
                right++;
            }
        }

        // Convert the modified digits back to an integer
        return Integer.parseInt(new String(digits));
    }
}
```