

PART-VI File Handling & Streams

No.	Aim of the Practical
27.	<p><u>AIM :</u> Write a program that will count the number of lines in each file that is specified on the command line. Assume that the files are text files. Note that multiple files can be specified, as in "java Line Counts file1.txt file2.txt file3.txt". Write each file name, along with the number of lines in that file, to standard output. If an error occurs while trying to read from one of the files, you should print an error message for that file, but you should still process all the remaining files.</p> <p><u>PROGRAM CODE :</u></p> <pre> import java.io.BufferedReader; import java.io.FileReader; import java.io.IOException; public class Prac_27 { public static void main(String[] args) { if (args.length == 0) { args = new String[]{"hello.txt"}; } for (String fileName : args) { try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) { int lineCount = 0; while (reader.readLine() != null) { lineCount++; } System.out.println(fileName + ": " + lineCount + " lines"); } catch (IOException e) { System.err.println("Error reading file " + fileName + ": " + e.getMessage()); } } } } </pre> <p><u>OUTPUT:</u></p> <div style="background-color: black; color: white; padding: 10px; border: 1px solid black;"> <pre> PS C:\Users\Om> cd "d:\0M\23DCS009\JAVA\sem3_23dcs009_java\src\Part6\" hello.txt: 2 lines PS D:\0M\23DCS009\JAVA\sem3_23dcs009_java\src\Part6> </pre> </div> <p style="text-align: center;"><u>OUTPUT: PRACTICAL-27</u></p>

	<p><u>CONCLUSION:</u></p> <p>This program counts the number of lines in a file using Java. It reads each file specified in the command-line arguments or defaults to hello.txt if no arguments are provided. The program uses BufferedReader to read each line and increments a counter for each line read. It handles file reading errors gracefully using a try-with-resources block. The program prints the number of lines for each file processed. This showcases efficient file handling and error management in Java.</p>
28.	<p><u>AIM :</u> Write an example that counts the number of times a particular character, such as e, appears in a file. The character can be specified at the command line. You can use xanadu.txt as the input file.</p> <p><u>PROGRAM CODE :</u></p> <pre>import java.io.BufferedReader; import java.io.FileReader; import java.io.IOException; public class Prac_28 { public static void main(String[] args) { if (args.length != 2) { System.out.println("Usage: java CharCount <file> <character>"); return; } String fileName = args[0]; char targetChar = args[1].charAt(0); try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) { int charCount = 0; int c; while ((c = reader.read()) != -1) { if (c == targetChar) { charCount++; } } System.out.println("The character '" + targetChar + "' appears " + charCount + " times in the file " + fileName); } catch (IOException e) { System.err.println("Error reading file " + fileName + ": " + e.getMessage()); } } }</pre> <p><u>OUTPUT:</u></p>

	<pre>D:\04\23DCS009\JAVA\sem3_23dcs009_java\src\Part6>java Prac_28 xanadu.txt e The character 'e' appears 47 times in the file xanadu.txt</pre> <p style="text-align: center;"><u>OUTPUT: PRACTICAL-28</u></p> <p><u>CONCLUSION:</u></p> <p>This program counts the occurrences of a specific character in a file using Java. It reads the file character by character with <code>BufferedReader</code> and compares each character to the target character. If they match, it increments a counter. The program handles file reading errors using a try-with-resources block to ensure the reader is closed properly. It also provides usage instructions if the required command-line arguments are not provided. This showcases efficient character processing and error management in Java.</p>
<p>29.</p>	<p><u>AIM :</u> Write a Java Program to Search for a given word in a File. Also show use of Wrapper Class with an example.</p> <p><u>PROGRAM CODE :</u></p> <pre>import java.io.BufferedReader; import java.io.FileReader; import java.io.IOException; public class Prac_29 { public static void main(String[] args) { if (args.length != 2) { System.out.println("Usage: java Prac_29 <file> <word>"); return; } String fileName = args[0]; String targetWord = args[1]; try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) { int wordCount = 0; String line; while ((line = reader.readLine()) != null) { String[] words = line.split("\\s+"); for (String word : words) { if (word.equals(targetWord)) { wordCount++; } } } System.out.println("The word '" + targetWord + "' appears " + wordCount + " times in the file " + fileName); } } }</pre>

	<pre> } catch (IOException e) { System.err.println("Error reading file " + fileName + ": " + e.getMessage()); } // Wrapper Class Example Integer wrapperInt = Integer.valueOf(10); // Using Integer wrapper class int primitiveInt = wrapperInt.intValue(); // Converting back to primitive int System.out.println("Wrapper Class Example: Integer value is " + wrapperInt + " and primitive int value is " + primitiveInt); } } </pre> <p><u>OUTPUT:</u></p> <div style="background-color: black; color: white; padding: 5px; border: 1px solid black;"> D:\0M\23DCS009\JAVA\sem3_23dcs009_java\src\Part6>java Prac_29 xanadu.txt is The word 'is' appears 3 times in the file xanadu.txt Wrapper Class Example: Integer value is 10 and primitive int value is 10 </div> <p style="text-align: center;"><u>OUTPUT: PRACTICAL-29</u></p> <p><u>CONCLUSION:</u></p> <p>This program demonstrates how to count the occurrences of a specific word in a file using Java. It reads the file line by line with <code>BufferedReader</code> and splits each line into words. It then compares each word to the target word and increments a counter if they match. The program handles file reading errors gracefully using a try-with-resources block. It also provides usage instructions if the required command-line arguments are not provided. This showcases efficient text processing and error management in Java.</p>
<p>30.</p>	<p><u>AIM :</u> Write a program to copy data from one file to another file.If the destination file does not exist, it is created automatically.</p> <p><u>PROGRAM CODE :</u></p> <pre> import java.io.FileInputStream; import java.io.FileOutputStream; import java.io.IOException; public class Prac_30 { public static void main(String[] args) { if (args.length != 2) { System.out.println("Usage: java Prac_30 <source file> <destination file>"); return; } String sourceFile = args[0]; String destinationFile = args[1]; try (FileInputStream fis = new FileInputStream(sourceFile); FileOutputStream fos = new FileOutputStream(destinationFile)) { </pre>

	<pre> byte[] buffer = new byte[1024]; int bytesRead; while ((bytesRead = fis.read(buffer)) != -1) { fos.write(buffer, 0, bytesRead); } System.out.println("File copied successfully from " + sourceFile + " to " + destinationFile); } catch (IOException e) { System.err.println("Error copying file: " + e.getMessage()); } } } </pre> <p><u>OUTPUT:</u></p> <div style="background-color: black; color: white; padding: 5px;"> D:\0M\23DCS009\JAVA\sem3_23dcs009_java\src\Part6>java Prac_30 source.txt destination.txt File copied successfully from source.txt to destination.txt </div> <p style="text-align: center;"><u>OUTPUT: PRACTICAL-30</u></p> <p><u>CONCLUSION:</u></p> <p>This program demonstrates how to copy data from one file to another using byte streams in Java. It reads from a source file and writes to a destination file, creating the destination file if it does not exist. The program uses FileInputStream to read bytes and FileOutputStream to write bytes. It handles errors using a try-with-resources block to ensure streams are closed properly. The program also provides usage instructions if the required command-line arguments are not provided. This showcases efficient file handling and error management in Java.</p>
<p>31.</p>	<p><u>AIM :</u> Write a program to show use of character and byte stream. Also show use of BufferedReader/BufferedWriter to read console input and write them into a file.</p> <p><u>PROGRAM CODE :</u></p> <pre> import java.io.*; class Prac_31 { public static void main(String[] args) { // Demonstrate character stream try (FileReader fr = new FileReader("input.txt"); FileWriter fw = new FileWriter("output_char.txt")) { int c; while ((c = fr.read()) != -1) { fw.write(c); } System.out.println("Character stream copy completed."); } } } </pre>

```

    } catch (IOException e) {
        System.err.println("Error with character stream: " + e.getMessage());
    }

    // Demonstrate byte stream
    try (FileInputStream fis = new FileInputStream("input.txt");
        FileOutputStream fos = new FileOutputStream("output_byte.txt")) {
        byte[] buffer = new byte[1024];
        int bytesRead;
        while ((bytesRead = fis.read(buffer)) != -1) {
            fos.write(buffer, 0, bytesRead);
        }
        System.out.println("Byte stream copy completed.");
    } catch (IOException e) {
        System.err.println("Error with byte stream: " + e.getMessage());
    }

    // Use BufferedReader and BufferedWriter to read from console and write to a
    file
    try (BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
        BufferedWriter bw = new BufferedWriter(new
FileWriter("console_output.txt"))) {
        System.out.println("Enter text (type 'exit' to finish):");
        String line;
        while (!(line = br.readLine()).equalsIgnoreCase("exit")) {
            bw.write(line);
            bw.newLine();
        }
        System.out.println("Console input written to file.");
    } catch (IOException e) {
        System.err.println("Error with BufferedReader/BufferedWriter: " +
e.getMessage());
    }
}

```

OUTPUT:

```

PS C:\Users\Om> cd "d:\0M\23DCS009\JAVA\sem3\_23dcs009_java\src\Part6\" ;
Character stream copy completed.
Byte stream copy completed.
Enter text (type 'exit' to finish):

```

[OUTPUT: PRACTICAL-31](#)

CONCLUSION:

This program demonstrates the use of character and byte streams in Java. It reads from input.txt and writes to output_char.txt using character streams, and to

	<p>output_byte.txt using byte streams. Additionally, it uses <code>BufferedReader</code> to read console input and <code>BufferedWriter</code> to write the input to <code>console_output.txt</code>. The program continues to read from the console until the user types "exit". This showcases efficient file handling and console interaction in Java.</p>
--	---