# CHAROTAR UNIVERSITY OF SCIENCE & TECHNOLOGY

**DEVANG PATEL INSTITUTE OF ADVANCE TECHNOLOGY & RESEARCH**

Department of Computer Science Engineering

# Subject Name: Java Programming Semester: III

**Subject Code: CSE201 Academic year: 2024-25**

Part - 6

|  |  |
| --- | --- |
| **No.** | **Aim of the Practical** |
| **27.** | 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.  **PROGRAM CODE:**  import java.io.BufferedReader;  import java.io.FileReader;  import java.io.IOException;  public class prc27 {  public static void main(String[] args) {  // Check if there are any file arguments  if (args.length == 0) {  System.out.println("Please specify at least one file.");  return;  }  // Process each file specified in the command line arguments  for (String fileName : args) {  int lineCount = countLinesInFile(fileName);  if (lineCount >= 0) {  System.out.println(fileName + ": " + lineCount + " lines");  }  }  }  private static int countLinesInFile(String fileName) {  int lineCount = 0;  try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {  while (reader.readLine() != null) {  lineCount++;  }  } catch (IOException e) {  System.err.println("Error reading file " + fileName + ": " + e.getMessage());  return -1; // Indicate an error occurred  }  return lineCount;  }  }  **OUTPUT:** |
|  | **CONCLUSION:**  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. |

|  |  |
| --- | --- |
| **28.** | 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.  **PROGRAM CODE :**  import java.io.BufferedReader;  import java.io.FileReader;  import java.io.IOException;  import java.util.Scanner;  public class prc28 {  public static void main(String[] args) {  Scanner scanner = new Scanner(System.in);  System.out.print("Enter the character to search for: ");  char searchChar = scanner.next().charAt(0);  System.out.print("Enter the file name (e.g., xanadu.txt): ");  String filename = scanner.next();  int count = 0;  System.out.println("Searching for character: '" + searchChar + "' in file: " + filename);  try (BufferedReader br = new BufferedReader(new FileReader(filename))) {  int currentChar;  while ((currentChar = br.read()) != -1) {  if ((char) currentChar == searchChar) {  count++;  }  }  } catch (IOException e) {  System.out.println("Error reading file: " + e.getMessage());  }  System.out.println("The character '" + searchChar + "' appears " + count + " times in " + filename);  }  } |

|  |  |
| --- | --- |
|  | **OUTPUT:**    **CONCLUSION:**  This program counts the occurrences of a specific character in a file using Java. It reads the file character by character with BufferedReader 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. |

|  |  |
| --- | --- |
| **30** | Write a program to copy data from one file to another file.If the destination file does not exist, it is created automatically.  **PROGRAM CODE:**  import java.io.\*;  import java.util.Scanner;  public class prc30 {  public static void main(String[] args) {  try {  File file = new File("myfile1.txt");  if (file.createNewFile()) {  System.out.println("File " + file.getName()+ " is created\n" );  } else {  System.out.println("File already exist");  }  FileWriter fwrite = new FileWriter("myfile1.txt");  fwrite.write("Hello, How are you?");  fwrite.close();  File file1 = new File("myfile2.txt");  if (file.createNewFile()) {  System.out.println("File " + file1.getName()+ " is created\n" );  } else {  System.out.println("File already exist");  }  FileReader fread = new FileReader("myfile1.txt");  BufferedReader bufferedReader = new BufferedReader(fread);  String line;  FileWriter fwrite1 = new FileWriter("myfile2.txt");  while((line = bufferedReader.readLine())!=null)  {  fwrite1.write(line + "\n");  }  bufferedReader.close();  fwrite1.close();  FileReader fread1 = new FileReader("myfile2.txt");  BufferedReader bufferedReader1 = new BufferedReader(fread1);  System.out.println("Content of myfile2: ");  while((line = bufferedReader1.readLine())!=null)  {  System.out.println(line);  }  bufferedReader1.close();  }  catch(IOException e){  e.printStackTrace();  }  }  }  **OUTPUT:**    **CONCLUSION:**  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. |

|  |  |
| --- | --- |
| **31** | 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.  **PROGRAM CODE :**  import java.io.\*;  public class prc31 {  public static void main(String[] args) {  try {  System.out.println("Character Stream - Reading from one file and writing to another.");  FileWriter charWriter = new FileWriter("charOutput.txt");  charWriter.write("This is written using a character stream.");  charWriter.close();  FileReader charReader = new FileReader("charOutput.txt");  BufferedReader bufferedReader = new BufferedReader(charReader);  String c;  System.out.println("Contents of charOutput.txt (Character Stream):");  while ((c = bufferedReader.readLine()) != null) {  System.out.println(c);  }  bufferedReader.close();  charReader.close();  System.out.println("\n\nByte Stream - Reading from one file and writing to another.");  FileOutputStream byteWriter = new FileOutputStream("byteOutput.txt");  byteWriter.write("This is written using a byte stream.".getBytes());  byteWriter.close();  FileInputStream byteReader = new FileInputStream("byteOutput.txt");  int b;  System.out.println("Contents of byteOutput.txt (Byte Stream):");  while ((b = byteReader.read()) != -1) {  System.out.print((char) b);  }  byteReader.close();  System.out.println("\n\nNow, using BufferedReader to take input from the console and BufferedWriter to write it to a file.");  BufferedReader consoleReader = new BufferedReader(new InputStreamReader(System.in));  System.out.println("Enter some text (BufferedReader will read this):");  String inputText = consoleReader.readLine();  BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter("bufferedOutput.txt"));  bufferedWriter.write(inputText);  bufferedWriter.close();  consoleReader.close();  System.out.println("Your input has been written to bufferedOutput.txt.");  } catch (IOException e) {  e.printStackTrace();  }  }  }  **OUTPUT:**    **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 output\_byte.txt using byte streams. Additionally, it uses BufferedReader to read console input and BufferedWriter to write the input to console\_output.txt. The program continues to read from the console until the user types "exit". This showcases efficient file handling and console interaction in Java. |