CHAROTAR UNIVERSITY OF SCIENCE TECHNOLOGY

DEVANG PATEL INSTITUTE OF ADVANCE TECHNOLOGY & RESEARCH

Department of Computer Science & Engineering

Subject Name: Java Programming

Semester: 3rd

Subject Code: CSE201

Academic year: 2024

SET- 6

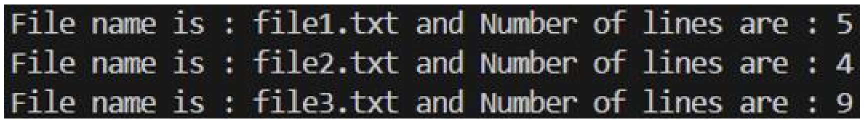
No	Aim of the Practical
27.	Write a program that will count the number of lines in each file that is specified on the command line. Assume that the files are text files. Note that multiple files can be specified, as in "java Line Counts file1.txt file2.txt file3.txt". Write each file name, along with 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.*;

public class prac27 {
    public static void main(String[] args) throws Exception {
        if (args.length == 0) {
            System.out.println("No file Found!");
        } else {
            for (int i = 0; i < args.length; i++) {
                try {
                    BufferedReader f = new BufferedReader(new
FileReader(args[i]));
                    String j;
                    int count = 0;
                    while ((j = f.readLine()) != null) {
                        count++;
                    }
                    System.out.println("File name is : " + args[i] + " and Number of
lines are : " + count);
                } catch (Exception e) {
                    System.out.println(e);
                }
            }
        }
    }
}
```

OUTPUT:



```
File name is : file1.txt and Number of lines are : 5
File name is : file2.txt and Number of lines are : 4
File name is : file3.txt and Number of lines are : 9
```

CONCLUSION:

This Java program reads several files named by the command line arguments and counts the number of lines in each. If no files are provided as command-line arguments, it will print out the appropriate message. Exception handling ensures graceful error management during file reading, thus a stable program.

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.

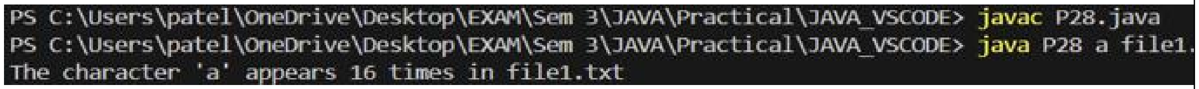
PROGRAM CODE :

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class P28{

   public static void main(String[] args) {
      if (args.length < 2) {
         System.out.println("Usage: java P28 <character> <filename>");
         return;
      }
      char targetChar = args[0].charAt(0);
      String fileName = args[1];
      int count = 0;
      try (BufferedReader reader = new BufferedReader(new
FileReader(fileName))) {
         int ch;
         while ((ch = reader.read()) != -1) {
            if (ch == targetChar) {
               count++;
            }
         }
         System.out.println("The character '" + targetChar + "' appears " +
count + " times in " + fileName);
      } catch (IOException e) {
         System.out.println("Error reading " + fileName + ": " +
e.getMessage());
      }
   }
}
```

OUTPUT:



```
PS C:\Users\patel\OneDrive\Desktop\EXAM\Sem 3\JAVA\Practical\JAVA_VSCODE> javac P28.java
PS C:\Users\patel\OneDrive\Desktop\EXAM\Sem 3\JAVA\Practical\JAVA_VSCODE> java P28 a file1.
The character 'a' appears 16 times in file1.txt
```

CONCLUSION:

The Java program successfully counts the occurrences of a specified character in a given file, providing the result in a clear format. It handles file read errors gracefully, ensuring robust performance even if issues arise during file access.

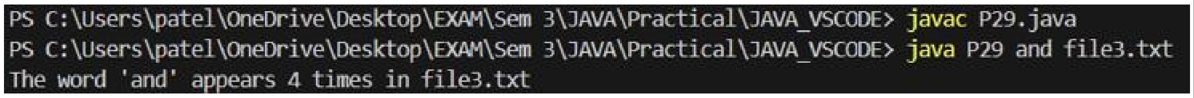
29 Write a Java Program to Search for a given word in a
File. Also show use of Wrapper Class with an example.

PROGRAM CODE :

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class P29 {
    public static void main(String[] args) {
        if (args.length < 2) {
            System.out.println("Usage: java P29 <word> <filename>");
            return;
        }
        String searchWord = args[0];
        String fileName = args[1];
        Integer count = 0;
        try (BufferedReader reader = new BufferedReader(new
FileReader(fileName))) {
            String line;
            while ((line = reader.readLine()) != null) {
                String[] words = line.split("\\W+");
                for (String word : words) {
                    if (word.equalsIgnoreCase(searchWord)) {
                        count++;
                    }
                }
            }
            System.out.println("The word '" + searchWord + "' appears " + count +
" times in " + fileName);
        } catch (IOException e) {
            System.out.println("Error reading " + fileName + ": " +
e.getMessage());
        }
    }
}
```

OUTPUT:



```
PS C:\Users\patel\OneDrive\Desktop\EXAM\Sem 3\JAVA\Practical\JAVA_VSCODE> javac P29.java
PS C:\Users\patel\OneDrive\Desktop\EXAM\Sem 3\JAVA\Practical\JAVA_VSCODE> java P29 and file3.txt
The word 'and' appears 4 times in file3.txt
```

CONCLUSION:

This Java program effectively searches for a specified word in a given file and counts its occurrences. It demonstrates the use of the Integer wrapper class to manage the count, showcasing how wrapper classes can be used for object manipulation in Java.

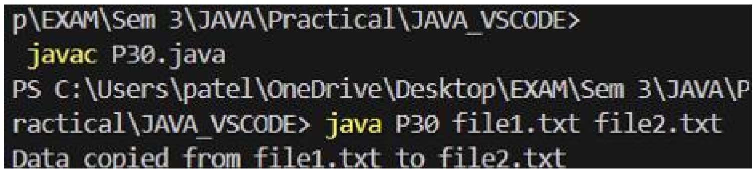
30 Write a program to copy data from one file to another file.
If the destination file does not exist, it is created
automatically.

PROGRAM CODE :

```
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class P30 {
   public static void main(String[] args) {
      if (args.length < 2) {
              System.out.println("Usage: java P30 <source file> <destination
file>");
         return;
      }
      String sourceFile = args[0];
      String destinationFile = args[1];
      try (FileReader fr = new FileReader(sourceFile);
            FileWriter fw = new FileWriter(destinationFile)) {
         int ch;
         while ((ch = fr.read()) != -1) {
            fw.write(ch);
         }
              System.out.println("Data copied from " + sourceFile + " to " +
destinationFile);
      } catch (IOException e) {
         System.out.println("Error: " + e.getMessage());
      }
   }
}
```

OUTPUT:



```
p\EXAM\Sem 3\JAVA\Practical\JAVA_VSCODE>
 javac P30.java
PS C:\Users\patel\OneDrive\Desktop\EXAM\Sem 3\JAVA\P
ractical\JAVA_VSCODE> java P30 file1.txt file2.txt
Data copied from file1.txt to file2.txt
```

CONCLUSION:

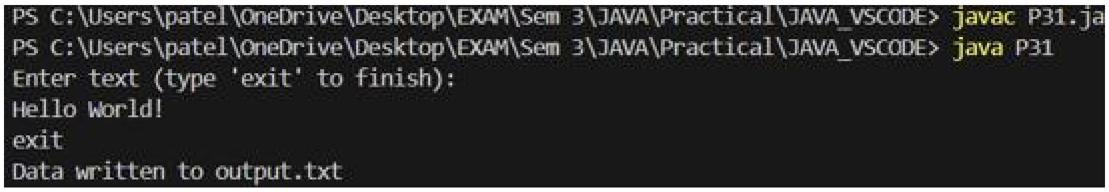
This Java program efficiently copies data from a source file to a destination file, automatically creating the destination file if it does not already exist. It handles any potential I/O exceptions during the process, ensuring robust performance.

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 P31 {
   public static void main(String[] args) {
                  BufferedReader consoleReader = new BufferedReader(new
InputStreamReader(System.in));
      String fileName = "output.txt";
                  try (BufferedWriter fileWriter = new BufferedWriter(new
FileWriter(fileName))) {
         System.out.println("Enter text (type 'exit' to finish):");
         String input;
          while (!(input = consoleReader.readLine()).equalsIgnoreCase("exit"))
{
             fileWriter.write(input);
             fileWriter.newLine();
         }
         System.out.println("Data written to " + fileName);
      } catch (IOException e) {
         System.out.println("Error: " + e.getMessage());
      }
   }
}
```

OUTPUT:



```
PS C:\Users\patel\OneDrive\Desktop\EXAM\Sem 3\JAVA\Practical\JAVA_VSCODE> javac P31.ja
PS C:\Users\patel\OneDrive\Desktop\EXAM\Sem 3\JAVA\Practical\JAVA_VSCODE> java P31
Enter text (type 'exit' to finish):
Hello World!
exit
Data written to output.txt
```

CONCLUSION:

This program effectively demonstrates the use of character streams via BufferedReader and BufferedWriter for reading console input and writing it to a file. It showcases how to handle text data efficiently while managing resources properly with try-with-resources.