**Exercise 1: Count Characters in a Text File**

**Write a program that reads a text file and counts the total number of characters (including whitespace).**

import java.io.FileReader;

import java.io.IOException;

public class CharacterCount {

public static void main(String[] args) {

try (FileReader reader = new FileReader("input.txt")) {

int charCount = 0;

int character;

while ((character = reader.read()) != -1) {

charCount++;

}

System.out.println("Total characters: " + charCount);

} catch (IOException e) {

e.printStackTrace();

}

}

}

**Exercise 2: Merge Two Files**

**Write a program that reads two text files and merges their contents into a new file.**

import java.io.\*;

public class MergeFiles {

public static void main(String[] args) {

try (BufferedReader reader1 = new BufferedReader(new FileReader("file1.txt"));

BufferedReader reader2 = new BufferedReader(new FileReader("file2.txt"));

BufferedWriter writer = new BufferedWriter(new FileWriter("merged.txt"))) {

String line;

while ((line = reader1.readLine()) != null) {

writer.write(line);

writer.newLine();

}

while ((line = reader2.readLine()) != null) {

writer.write(line);

writer.newLine();

}

System.out.println("Files merged successfully.");

} catch (IOException e) {

e.printStackTrace();

}

}

}

**Exercise 3: Find Longest Word**

**Write a program that reads a text file and finds the longest word in it.**

import java.io.\*;

import java.util.Scanner;

public class LongestWord {

public static void main(String[] args) {

try (Scanner scanner = new Scanner(new File("text.txt"))) {

String longestWord = "";

while (scanner.hasNext()) {

String word = scanner.next();

if (word.length() > longestWord.length()) {

longestWord = word;

}

}

System.out.println("Longest word: " + longestWord);

} catch (IOException e) {

e.printStackTrace();

}

}

}

**Exercise 4: Copy Binary File**

**Write a program to copy a binary file (e.g., image) to a new file.**

import java.io.\*;

public class CopyBinaryFile {

public static void main(String[] args) {

try (InputStream inputStream = new FileInputStream("input.jpg");

OutputStream outputStream = new FileOutputStream("output.jpg")) {

byte[] buffer = new byte[1024];

int bytesRead;

while ((bytesRead = inputStream.read(buffer)) != -1) {

outputStream.write(buffer, 0, bytesRead);

}

System.out.println("File copied successfully.");

} catch (IOException e) {

e.printStackTrace();

}

}

}

**Exercise 5: Read Line Numbers**

**Write a program that reads a text file and prefixes each line with its line number.**

import java.io.\*;

public class LineNumbers {

public static void main(String[] args) {

try (BufferedReader reader = new BufferedReader(new FileReader("input.txt"));

BufferedWriter writer = new BufferedWriter(new FileWriter("output.txt"))) {

String line;

int lineNumber = 1;

while ((line = reader.readLine()) != null) {

writer.write(lineNumber + ": " + line);

writer.newLine();

lineNumber++;

}

System.out.println("Line numbers added.");

} catch (IOException e) {

e.printStackTrace();

}

}

}

**Exercise 6: Reverse File Contents**

**Write a program that reads a text file and writes its contents in reverse order to another file.**

import java.io.\*;

public class ReverseFile {

public static void main(String[] args) {

try (BufferedReader reader = new BufferedReader(new FileReader("input.txt"));

BufferedWriter writer = new BufferedWriter(new FileWriter("reversed.txt"))) {

String line;

StringBuilder reversedContent = new StringBuilder();

while ((line = reader.readLine()) != null) {

reversedContent.insert(0, line + "\n");

}

writer.write(reversedContent.toString());

System.out.println("File contents reversed.");

} catch (IOException e) {

e.printStackTrace();

}

}

}

**Exercise 7: Word Frequency Counter**

**Write a program that reads a text file and counts the frequency of each word.**

import java.io.\*;

import java.util.HashMap;

import java.util.Map;

import java.util.Scanner;

public class WordFrequency {

public static void main(String[] args) {

try (Scanner scanner = new Scanner(new File("input.txt"))) {

Map<String, Integer> wordFrequency = new HashMap<>();

while (scanner.hasNext()) {

String word = scanner.next();

word = word.toLowerCase();

wordFrequency.put(word, wordFrequency.getOrDefault(word, 0) + 1);

}

for (Map.Entry<String, Integer> entry : wordFrequency.entrySet()) {

System.out.println(entry.getKey() + ": " + entry.getValue());

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

**Exercise 8: File Encryption**

**Write a program that reads a text file and encrypts its contents using a simple cipher.**

import java.io.\*;

import java.util.Scanner;

public class FileEncryption {

public static void main(String[] args) {

try (Scanner scanner = new Scanner(new File("input.txt"));

BufferedWriter writer = new BufferedWriter(new FileWriter("encrypted.txt"))) {

int key = 3; // Shift key for the Caesar cipher

while (scanner.hasNextLine()) {

String line = scanner.nextLine();

String encryptedLine = encrypt(line, key);

writer.write(encryptedLine);

writer.newLine();

}

System.out.println("File encrypted successfully.");

} catch (IOException e) {

e.printStackTrace();

}

}

private static String encrypt(String input, int key) {

StringBuilder encrypted = new StringBuilder();

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

if (Character.isLetter(c)) {

char shifted = (char) (c + key);

if ((Character.isLowerCase(c) && shifted > 'z') || (Character.isUpperCase(c) && shifted > 'Z')) {

shifted -= 26;

}

encrypted.append(shifted);

} else {

encrypted.append(c);

}

}

return encrypted.toString();

}

}

**Exercise 9: Search for a Keyword**

**Write a program that searches for a specific keyword in a text file and displays the lines containing the keyword.**

import java.io.\*;

import java.util.Scanner;

public class KeywordSearch {

public static void main(String[] args) {

try (Scanner scanner = new Scanner(new File("input.txt"))) {

String keyword = "Java";

while (scanner.hasNextLine()) {

String line = scanner.nextLine();

if (line.contains(keyword)) {

System.out.println(line);

}

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

**Exercise 10: File Statistics**

**Write a program that reads a text file and displays statistics such as the number of lines, words, and characters.**

import java.io.\*;

import java.util.Scanner;

public class FileStatistics {

public static void main(String[] args) {

try (Scanner scanner = new Scanner(new File("input.txt"))) {

int lineCount = 0;

int wordCount = 0;

int charCount = 0;

while (scanner.hasNextLine()) {

String line = scanner.nextLine();

lineCount++;

charCount += line.length();

wordCount += line.split("\\s+").length;

}

System.out.println("Line count: " + lineCount);

System.out.println("Word count: " + wordCount);

System.out.println("Character count: " + charCount);

} catch (IOException e) {

e.printStackTrace();

}

}

}

**Exercise 11: Word Reversal**

**Write a program that reads a text file, reverses the order of words in each line, and saves the modified content to a new file.**

import java.io.\*;

import java.util.Scanner;

public class WordReversal {

public static void main(String[] args) {

try (Scanner scanner = new Scanner(new File("input.txt"));

BufferedWriter writer = new BufferedWriter(new FileWriter("output.txt"))) {

while (scanner.hasNextLine()) {

String line = scanner.nextLine();

String[] words = line.split("\\s+");

for (int i = words.length - 1; i >= 0; i--) {

writer.write(words[i] + " ");

}

writer.newLine();

}

System.out.println("Words reversed and written to output file.");

} catch (IOException e) {

e.printStackTrace();

}

}

}

**Exercise 12: File Comparison**

**Write a program that reads two text files and compares their contents, reporting whether they are identical or not.**

import java.io.\*;

import java.util.Scanner;

public class FileComparison {

public static void main(String[] args) {

try (Scanner scanner1 = new Scanner(new File("file1.txt"));

Scanner scanner2 = new Scanner(new File("file2.txt"))) {

boolean identical = true;

while (scanner1.hasNextLine() && scanner2.hasNextLine()) {

String line1 = scanner1.nextLine();

String line2 = scanner2.nextLine();

if (!line1.equals(line2)) {

identical = false;

break;

}

}

if (identical) {

System.out.println("Files are identical.");

} else {

System.out.println("Files are not identical.");

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

**Exercise 13: File Permissions**

**Write a program that checks the read and write permissions of a file and displays the result.**

import java.io.File;

public class FilePermissions {

public static void main(String[] args) {

File file = new File("example.txt");

System.out.println("Read permission: " + file.canRead());

System.out.println("Write permission: " + file.canWrite());

}

}

**Exercise 14: Replace Text**

**Write a program that reads a text file, replaces occurrences of a specific word with another word, and saves the modified content to a new file.**

import java.io.\*;

import java.util.Scanner;

public class ReplaceText {

public static void main(String[] args) {

try (Scanner scanner = new Scanner(new File("input.txt"));

BufferedWriter writer = new BufferedWriter(new FileWriter("output.txt"))) {

while (scanner.hasNextLine()) {

String line = scanner.nextLine();

line = line.replace("oldWord", "newWord");

writer.write(line);

writer.newLine();

}

System.out.println("Text replaced and written to output file.");

} catch (IOException e) {

e.printStackTrace();

}

}

}

**Exercise 15: Directory Size**

**Write a program that calculates and displays the total size of a directory and its contents.**

import java.io.File;

public class DirectorySize {

public static void main(String[] args) {

File directory = new File("myDirectory");

long totalSize = calculateSize(directory);

System.out.println("Total directory size: " + totalSize + " bytes");

}

private static long calculateSize(File file) {

if (file.isFile()) {

return file.length();

}

long size = 0;

File[] files = file.listFiles();

if (files != null) {

for (File f : files) {

size += calculateSize(f);

}

}

return size;

}

}

**Exercise 16: File Deletion with Confirmation**

**Write a program that prompts the user to confirm before deleting a file.**

import java.io.File;

import java.util.Scanner;

public class FileDeletion {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the file name to delete: ");

String fileName = scanner.nextLine();

File file = new File(fileName);

if (file.exists()) {

System.out.print("Are you sure you want to delete " + fileName + "? (y/n): ");

String confirmation = scanner.nextLine();

if (confirmation.equalsIgnoreCase("y")) {

if (file.delete()) {

System.out.println("File deleted successfully.");

} else {

System.out.println("Unable to delete the file.");

}

} else {

System.out.println("Deletion canceled.");

}

} else {

System.out.println("File not found.");

}

scanner.close();

}

}

**Exercise 17: Directory Listing**

**Write a program that lists all files and directories in a specified directory.**

import java.io.File;

public class DirectoryListing {

public static void main(String[] args) {

File directory = new File("myDirectory");

if (directory.exists() && directory.isDirectory()) {

File[] files = directory.listFiles();

if (files != null) {

for (File file : files) {

System.out.println(file.getName());

}

} else {

System.out.println("Directory is empty.");

}

} else {

System.out.println("Directory not found.");

}

}

}

**Exercise 18: Backup Files**

**Write a program that creates a backup of a file by copying its content to a new file with a timestamp in the filename.**

import java.io.\*;

import java.text.SimpleDateFormat;

import java.util.Date;

public class FileBackup {

public static void main(String[] args) {

try (BufferedReader reader = new BufferedReader(new FileReader("data.txt"))) {

Date now = new Date();

SimpleDateFormat dateFormat = new SimpleDateFormat("yyyyMMddHHmmss");

String timestamp = dateFormat.format(now);

String backupFileName = "backup\_" + timestamp + ".txt";

BufferedWriter writer = new BufferedWriter(new FileWriter(backupFileName));

String line;

while ((line = reader.readLine()) != null) {

writer.write(line);

writer.newLine();

}

writer.close();

System.out.println("Backup created: " + backupFileName);

} catch (IOException e) {

e.printStackTrace();

}

}

}

**Exercise 19: File Splitter**

**Write a program that reads a large file and splits it into smaller files of a specified size.**

import java.io.\*;

public class FileSplitter {

public static void main(String[] args) {

int chunkSize = 1000; // Size of each chunk in bytes

int chunkNumber = 1;

try (InputStream inputStream = new FileInputStream("largeFile.txt")) {

byte[] buffer = new byte[chunkSize];

int bytesRead;

while ((bytesRead = inputStream.read(buffer)) != -1) {

String chunkFileName = "chunk" + chunkNumber + ".dat";

try (OutputStream outputStream = new FileOutputStream(chunkFileName)) {

outputStream.write(buffer, 0, bytesRead);

}

System.out.println("Chunk " + chunkNumber + " created: " + chunkFileName);

chunkNumber++;

}

System.out.println("File split into chunks.");

} catch (IOException e) {

e.printStackTrace();

}

}

}

**Exercise 20: Word Search**

**Write a program that prompts the user to enter a word and searches for all occurrences of that word in a text file.**

import java.io.\*;

import java.util.Scanner;

public class WordSearch {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter a word to search: ");

String searchWord = scanner.nextLine();

try (BufferedReader reader = new BufferedReader(new FileReader("text.txt"))) {

String line;

int lineNumber = 1;

while ((line = reader.readLine()) != null) {

if (line.contains(searchWord)) {

System.out.println("Line " + lineNumber + ": " + line);

}

lineNumber++;

}

} catch (IOException e) {

e.printStackTrace();

}

scanner.close();

}

}

**Exercise 21: File Renaming**

**Write a program that renames a file by changing its name and extension.**

import java.io.File;

public class FileRenaming {

public static void main(String[] args) {

File oldFile = new File("old.txt");

File newFile = new File("new.txt");

if (oldFile.exists()) {

if (oldFile.renameTo(newFile)) {

System.out.println("File renamed successfully.");

} else {

System.out.println("Failed to rename the file.");

}

} else {

System.out.println("File not found.");

}

}

}

**Exercise 22: File Concatenation**

**Write a program that concatenates the contents of multiple text files into a single file.**

import java.io.\*;

public class FileConcatenation {

public static void main(String[] args) {

try (BufferedWriter writer = new BufferedWriter(new FileWriter("concatenated.txt"))) {

for (int i = 1; i <= 3; i++) {

try (BufferedReader reader = new BufferedReader(new FileReader("file" + i + ".txt"))) {

String line;

while ((line = reader.readLine()) != null) {

writer.write(line);

writer.newLine();

}

}

}

System.out.println("Files concatenated successfully.");

} catch (IOException e) {

e.printStackTrace();

}

}

}

**Exercise 23: Binary Data Statistics**

**Write a program that reads a binary file and calculates the total number of bytes and the average byte value.**

import java.io.\*;

public class BinaryDataStatistics {

public static void main(String[] args) {

try (FileInputStream inputStream = new FileInputStream("data.bin")) {

int totalBytes = 0;

int totalValues = 0;

int byteValue;

while ((byteValue = inputStream.read()) != -1) {

totalBytes++;

totalValues += byteValue;

}

double average = (double) totalValues / totalBytes;

System.out.println("Total bytes: " + totalBytes);

System.out.println("Average byte value: " + average);

} catch (IOException e) {

e.printStackTrace();

}

}

}

**Exercise 24: Directory Structure**

**Write a program that generates a tree-like representation of a directory structure.**

import java.io.File;

public class DirectoryStructure {

public static void main(String[] args) {

File rootDirectory = new File("root");

displayDirectoryTree(rootDirectory, 0);

}

private static void displayDirectoryTree(File directory, int level) {

for (File file : directory.listFiles()) {

for (int i = 0; i < level; i++) {

System.out.print(" ");

}

System.out.println(file.getName());

if (file.isDirectory()) {

displayDirectoryTree(file, level + 1);

}

}

}

}

**Exercise 25: File Date Comparison**

**Write a program that reads two files and compares their last modified dates, reporting which file is more recent.**

import java.io.File;

import java.text.SimpleDateFormat;

import java.util.Date;

public class FileDateComparison {

public static void main(String[] args) {

File file1 = new File("file1.txt");

File file2 = new File("file2.txt");

if (file1.exists() && file2.exists()) {

long lastModified1 = file1.lastModified();

long lastModified2 = file2.lastModified();

SimpleDateFormat dateFormat = new SimpleDateFormat("dd-MM-yyyy HH:mm:ss");

String formattedDate1 = dateFormat.format(new Date(lastModified1));

String formattedDate2 = dateFormat.format(new Date(lastModified2));

if (lastModified1 > lastModified2) {

System.out.println("File 1 was modified more recently: " + formattedDate1);

} else if (lastModified2 > lastModified1) {

System.out.println("File 2 was modified more recently: " + formattedDate2);

} else {

System.out.println("Both files have the same last modified date.");

}

} else {

System.out.println("One or both files not found.");

}

}

}

**Exercise 26: Directory Copy**

**Write a program that copies the contents of one directory to another directory.**

import java.io.\*;

import java.util.Objects;

public class DirectoryCopy {

public static void main(String[] args) {

File sourceDirectory = new File("source");

File targetDirectory = new File("target");

copyDirectory(sourceDirectory, targetDirectory);

System.out.println("Directory copied successfully.");

}

private static void copyDirectory(File source, File target) {

if (source.isDirectory()) {

if (!target.exists()) {

target.mkdir();

}

for (File file : Objects.requireNonNull(source.listFiles())) {

copyDirectory(file, new File(target, file.getName()));

}

} else {

try (InputStream inputStream = new FileInputStream(source);

OutputStream outputStream = new FileOutputStream(target)) {

byte[] buffer = new byte[1024];

int bytesRead;

while ((bytesRead = inputStream.read(buffer)) != -1) {

outputStream.write(buffer, 0, bytesRead);

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

**Exercise 27: File Permissions Check**

**Write a program that checks if a file is readable and writable.**

import java.io.File;

public class FilePermissionsCheck {

public static void main(String[] args) {

File file = new File("sample.txt");

if (file.exists()) {

if (file.canRead()) {

System.out.println("File is readable.");

} else {

System.out.println("File is not readable.");

}

if (file.canWrite()) {

System.out.println("File is writable.");

} else {

System.out.println("File is not writable.");

}

} else {

System.out.println("File not found.");

}

}

}

**Exercise 28: File Timestamp Update**

**Write a program that updates the last modified timestamp of a file to the current time.**

import java.io.File;

import java.util.Date;

public class FileTimestampUpdate {

public static void main(String[] args) {

File file = new File("data.txt");

if (file.exists()) {

long currentTime = System.currentTimeMillis();

file.setLastModified(currentTime);

Date newTimestamp = new Date(file.lastModified());

System.out.println("File timestamp updated to: " + newTimestamp);

} else {

System.out.println("File not found.");

}

}

}

**Exercise 29: File Line Remover**

**Write a program that reads a text file and creates a new file without the lines that contain a specific keyword.**

import java.io.\*;

import java.util.Scanner;

public class LineRemover {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the keyword to remove lines: ");

String keyword = scanner.nextLine();

try (BufferedReader reader = new BufferedReader(new FileReader("input.txt"));

BufferedWriter writer = new BufferedWriter(new FileWriter("output.txt"))) {

String line;

while ((line = reader.readLine()) != null) {

if (!line.contains(keyword)) {

writer.write(line);

writer.newLine();

}

}

System.out.println("Lines containing '" + keyword + "' removed from the file.");

} catch (IOException e) {

e.printStackTrace();

}

scanner.close();

}

}

**Exercise 30: Word Frequency Analysis**

**Write a program that reads a text file, analyzes the frequency of each word, and displays the most frequent words along with their counts.**

import java.io.\*;

import java.util.\*;

public class WordFrequencyAnalysis {

public static void main(String[] args) {

try (Scanner scanner = new Scanner(new File("input.txt"))) {

Map<String, Integer> wordFrequency = new HashMap<>();

while (scanner.hasNext()) {

String word = scanner.next().toLowerCase();

word = word.replaceAll("[^a-zA-Z]", ""); // Remove non-alphabetic characters

if (!word.isEmpty()) {

wordFrequency.put(word, wordFrequency.getOrDefault(word, 0) + 1);

}

}

List<Map.Entry<String, Integer>> sortedWords = new ArrayList<>(wordFrequency.entrySet());

sortedWords.sort((entry1, entry2) -> entry2.getValue().compareTo(entry1.getValue()));

int displayLimit = 10; // Display the top N words

for (int i = 0; i < Math.min(sortedWords.size(), displayLimit); i++) {

Map.Entry<String, Integer> entry = sortedWords.get(i);

System.out.println(entry.getKey() + ": " + entry.getValue());

}

} catch (IOException e) {

e.printStackTrace();

}

}

}