### **1. File Handling - Read and Write a Text File**

📌 **Problem Statement:**Write a Java program that reads the contents of a text file and writes it into a new file. If the source file does not exist, display an appropriate message.

#### **Requirements:**

* Use FileInputStream and FileOutputStream.
* Handle IOException properly.
* Ensure that the destination file is created if it does not exist.

|  |
| --- |
| package io;  import java.io.\*;  public class FileCopyBasic {  public static void main(String[] args) {  File source = new File("io/q1.txt");  File destination = new File("io/q1.txt");   if (!source.exists()) {  System.out.println("Source file does not exist.");  return;  }   try (FileInputStream fis = new FileInputStream(source);  FileOutputStream fos = new FileOutputStream(destination)) {   int data;  while ((data = fis.read()) != -1) {  fos.write(data);  }  System.out.println("File copied successfully.");  } catch (IOException e) {  e.printStackTrace();  }  } } |

### **2. Buffered Streams - Efficient File Copy**

📌 **Problem Statement:**Create a Java program that copies a large file (e.g., 100MB) from one location to another using **Buffered Streams** (BufferedInputStream and BufferedOutputStream). Compare the performance with normal file streams.

#### **Requirements:**

* Read and write in chunks of **4 KB (4096 bytes)**.
* Use System.nanoTime() to measure execution time.
* Compare execution time with **unbuffered streams**.

|  |
| --- |
| import java.io.\*;  public class BufferedFileCopy {  public static void main(String[] args) throws IOException {  File source = new File("largefile.txt");  File dest = new File("copy\_largefile.txt");   long start = System.nanoTime();  try (BufferedInputStream bis = new BufferedInputStream(new FileInputStream(source));  BufferedOutputStream bos = new BufferedOutputStream(new FileOutputStream(dest))) {   byte[] buffer = new byte[4096];  int length;  while ((length = bis.read(buffer)) != -1) {  bos.write(buffer, 0, length);  }  }  long end = System.nanoTime();  System.out.println("Buffered copy time: " + (end - start) / 1\_000\_000 + " ms");  } } |

### **3. Read User Input from Console**

📌 **Problem Statement:**Write a program that asks the user for their **name, age, and favorite programming language**, then saves this information into a file.

#### **Requirements:**

* Use BufferedReader for console input.
* Use FileWriter to write the data into a file.
* Handle exceptions properly.

|  |
| --- |
| import java.io.\*;  public class ConsoleToFile {  public static void main(String[] args) {  try (BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));  FileWriter writer = new FileWriter("userinfo.txt")) {   System.out.print("Enter your name: ");  String name = reader.readLine();  System.out.print("Enter your age: ");  String age = reader.readLine();  System.out.print("Favorite programming language: ");  String lang = reader.readLine();   writer.write("Name: " + name + "\nAge: " + age + "\nLanguage: " + lang);  System.out.println("Data saved to userinfo.txt");  } catch (IOException e) {  e.printStackTrace();  }  } } |

### **4. Serialization - Save and Retrieve an Object**

📌 **Problem Statement:**Design a Java program that allows a user to **store a list of employees in a file** using **Object Serialization** and later retrieve the data from the file.

#### **Requirements:**

* Create an Employee class with fields: id, name, department, salary.
* Serialize the list of employees into a file (ObjectOutputStream).
* Deserialize and display the employees from the file (ObjectInputStream).
* Handle ClassNotFoundException and IOException.

|  |
| --- |
| import java.io.\*; import java.util.\*;  class Employee implements Serializable {  int id;  String name, department;  double salary;   Employee(int id, String name, String dept, double salary) {  this.id = id; this.name = name; this.department = dept; this.salary = salary;  }   public String toString() {  return id + ", " + name + ", " + department + ", " + salary;  } }  public class EmployeeSerialization {  public static void main(String[] args) {  List<Employee> list = List.of(  new Employee(1, "Alice", "HR", 50000),  new Employee(2, "Bob", "IT", 60000)  );   try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("employees.ser"))) {  oos.writeObject(list);  } catch (IOException e) {  e.printStackTrace();  }   try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream("employees.ser"))) {  List<Employee> deserialized = (List<Employee>) ois.readObject();  deserialized.forEach(System.out::println);  } catch (IOException | ClassNotFoundException e) {  e.printStackTrace();  }  } } |

### **5. ByteArray Stream - Convert Image to ByteArray**

📌 **Problem Statement:**Write a Java program that **converts an image file into a byte array** and then writes it back to another image file.

#### **Requirements:**

* Use ByteArrayInputStream and ByteArrayOutputStream.
* Verify that the new file is identical to the original image.
* Handle IOException.

|  |
| --- |
| import java.io.\*;  public class ImageByteArray {  public static void main(String[] args) throws IOException {  File img = new File("image.jpg");  ByteArrayOutputStream baos = new ByteArrayOutputStream();   try (FileInputStream fis = new FileInputStream(img)) {  byte[] buffer = new byte[1024];  int length;  while ((length = fis.read(buffer)) != -1) {  baos.write(buffer, 0, length);  }  }   byte[] imageBytes = baos.toByteArray();  try (FileOutputStream fos = new FileOutputStream("image\_copy.jpg")) {  fos.write(imageBytes);  }   System.out.println("Image copied via byte array.");  } } |

### **6. Filter Streams - Convert Uppercase to Lowercase**

📌 **Problem Statement:**Create a program that reads a text file and writes its contents into another file, converting all uppercase letters to lowercase.

#### **Requirements:**

* Use FileReader and FileWriter.
* Use BufferedReader and BufferedWriter for efficiency.
* Handle character encoding issues.

|  |
| --- |
| import java.io.\*;  public class UpperToLower {  public static void main(String[] args) throws IOException {  try (BufferedReader reader = new BufferedReader(new FileReader("input.txt"));  BufferedWriter writer = new BufferedWriter(new FileWriter("output.txt"))) {   String line;  while ((line = reader.readLine()) != null) {  writer.write(line.toLowerCase());  writer.newLine();  }  System.out.println("Content converted to lowercase.");  }  } } |

### 

### 

### **7. Data Streams - Store and Retrieve Primitive Data**

📌 **Problem Statement:**Write a Java program that stores **student details** (roll number, name, GPA) in a binary file and retrieves it later.

#### **Requirements:**

* Use DataOutputStream to write primitive data.
* Use DataInputStream to read data.
* Ensure proper closing of resources.

|  |
| --- |
| import java.io.\*;  public class StudentData {  public static void main(String[] args) {  try (DataOutputStream dos = new DataOutputStream(new FileOutputStream("students.dat"))) {  dos.writeInt(101);  dos.writeUTF("John");  dos.writeDouble(8.5);  } catch (IOException e) {  e.printStackTrace();  }   try (DataInputStream dis = new DataInputStream(new FileInputStream("students.dat"))) {  int roll = dis.readInt();  String name = dis.readUTF();  double gpa = dis.readDouble();  System.out.println(roll + " " + name + " " + gpa);  } catch (IOException e) {  e.printStackTrace();  }  } } |

### **8. Piped Streams - Inter-Thread Communication**

📌 **Problem Statement:**Implement a Java program where one thread **writes data** into a PipedOutputStream and another thread **reads data** from a PipedInputStream.

#### **Requirements:**

* Use **two threads** for reading and writing.
* Synchronize properly to prevent data loss.
* Handle IOException.

|  |
| --- |
| import java.io.\*;  public class PipeExample {  public static void main(String[] args) throws IOException {  PipedOutputStream pos = new PipedOutputStream();  PipedInputStream pis = new PipedInputStream(pos);   Thread writer = new Thread(() -> {  try {  pos.write("Hello from writer thread!".getBytes());  pos.close();  } catch (IOException e) {  e.printStackTrace();  }  });   Thread reader = new Thread(() -> {  try {  int data;  while ((data = pis.read()) != -1) {  System.out.print((char) data);  }  } catch (IOException e) {  e.printStackTrace();  }  });   writer.start();  reader.start();  } } |

### 

### **9. Read a Large File Line by Line**

📌 **Problem Statement:**Develop a Java program that efficiently reads a **large text file** (500MB+) **line by line** and prints only lines containing the word **"error"**.

#### **Requirements:**

* Use BufferedReader for efficient reading.
* Read line-by-line instead of loading the entire file.
* Display only lines containing "error" (case insensitive).

|  |
| --- |
| import java.io.\*;  public class LargeFileReader {  public static void main(String[] args) throws IOException {  try (BufferedReader reader = new BufferedReader(new FileReader("largefile.txt"))) {  String line;  while ((line = reader.readLine()) != null) {  if (line.toLowerCase().contains("error")) {  System.out.println(line);  }  }  }  } } |

### **10. Count Words in a File**

📌 **Problem Statement:**Write a Java program that **counts the number of words in a given text file** and displays the **top 5 most frequently occurring words**.

#### **Requirements:**

* Use FileReader and BufferedReader to read the file.
* Use a HashMap<String, Integer> to count word occurrences.
* Sort the words based on frequency and display the top 5.

|  |
| --- |
| import java.io.\*; import java.util.\*;  public class WordCounter {  public static void main(String[] args) throws IOException {  Map<String, Integer> freq = new HashMap<>();   try (BufferedReader reader = new BufferedReader(new FileReader("words.txt"))) {  String line;  while ((line = reader.readLine()) != null) {  for (String word : line.toLowerCase().split("\\W+")) {  if (!word.isEmpty()) {  freq.put(word, freq.getOrDefault(word, 0) + 1);  }  }  }  }   freq.entrySet().stream()  .sorted((e1, e2) -> e2.getValue().compareTo(e1.getValue()))  .limit(5)  .forEach(e -> System.out.println(e.getKey() + ": " + e.getValue()));  } } |