CodInClub

Powered By BridgeLabz

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.

```java
import java.io.*;
import java.util.Scanner;

public class FileReadWrite {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter source file path: ");
        String sourcePath = sc.nextLine();
        System.out.print("Enter destination file path: ");
        String destPath = sc.nextLine();

        try (FileInputStream fis = new
FileInputStream(sourcePath);
             FileOutputStream fos = new
FileOutputStream(destPath)) {

            int b;
            while ((b = fis.read()) != -1) {
                fos.write(b);
            }
            System.out.println("File copied successfully.");
        } catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

---

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 BufferedCopyPerformance {
    public static void main(String[] args) throws IOException {
        String source = "largefile.dat"; // Create a 100MB file
manually
        String dest1 = "copy_unbuffered.dat";
        String dest2 = "copy_buffered.dat";

        // Unbuffered
        long start1 = System.nanoTime();
        try (FileInputStream fis = new FileInputStream(source);
             FileOutputStream fos = new FileOutputStream(dest1)) {
            byte[] buffer = new byte[4096];
            int len;
            while ((len = fis.read(buffer)) != -1) {
                fos.write(buffer, 0, len);
            }
        }
        long end1 = System.nanoTime();

        // Buffered
        long start2 = System.nanoTime();
        try (BufferedInputStream bis = new BufferedInputStream(new
FileInputStream(source));
             BufferedOutputStream bos = new BufferedOutputStream(new
FileOutputStream(dest2))) {
            byte[] buffer = new byte[4096];
            int len;
            while ((len = bis.read(buffer)) != -1) {
                bos.write(buffer, 0, len);
            }
        }
        long end2 = System.nanoTime();

        System.out.println("Unbuffered Time: " + (end1 - start1) /
1_000_000 + " ms");
        System.out.println("Buffered Time: " + (end2 - start2) /
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.

```java
import java.io.*;

public class ConsoleInputToFile {
    public static void main(String[] args) {
        try (BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
             FileWriter fw = new
FileWriter("userinfo.txt")) {

            System.out.print("Enter your name: ");
            String name = br.readLine();
            System.out.print("Enter your age: ");
            String age = br.readLine();
            System.out.print("Enter favorite programming
language: ");
            String lang = br.readLine();
            fw.write("Name: " + name + "\nAge: " + age +
"\nLanguage: " + lang);
            System.out.println("Information saved to
file.");
        } catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

---

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`.

```java
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 EmployeeSerialize {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<Employee> list = new ArrayList<>();
        System.out.print("Enter number of employees: ");
        int n = sc.nextInt();
        sc.nextLine();

        for (int i = 0; i < n; i++) {
            System.out.print("Enter id, name, department, salary: ");
            int id = sc.nextInt();
            String name = sc.next();
            String dept = sc.next();
            double salary = sc.nextDouble();
            list.add(new Employee(id, name, dept, salary));
        }

        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream("employees.ser"))) {
            oos.writeObject(list);
        } catch (IOException e) {
            System.out.println("Write Error: " + e.getMessage());
        }

        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream("employees.ser"))) {
            List<Employee> empList = (List<Employee>)
ois.readObject();
            for (Employee e : empList) {
                System.out.println(e);
            }
        } catch (IOException | ClassNotFoundException e) {
            System.out.println("Read Error: " + e.getMessage());
        }
    }
}
```

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 ImageToByteArray {
    public static void main(String[] args) {
        String inputImage = "input.jpg";
        String outputImage = "output.jpg";

        try (FileInputStream fis = new
FileInputStream(inputImage);
             ByteArrayOutputStream baos = new
ByteArrayOutputStream()) {

            int b;
            while ((b = fis.read()) != -1) {
                baos.write(b);
            }

            byte[] imageBytes = baos.toByteArray();

            try (ByteArrayInputStream bais = new
ByteArrayInputStream(imageBytes);
                 FileOutputStream fos = new
FileOutputStream(outputImage)) {
                while ((b = bais.read()) != -1) {
                    fos.write(b);
                }
                System.out.println("Image copied
successfully.");
            }

        } catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

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) {
        try (BufferedReader br = new BufferedReader(new
FileReader("input.txt"));
             BufferedWriter bw = new BufferedWriter(new
FileWriter("output.txt"))) {

            int ch;
            while ((ch = br.read()) != -1) {
                bw.write(Character.toLowerCase(ch));
            }
            System.out.println("Converted to lowercase.");
        } catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

---

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.*;
import java.util.Scanner;

public class StudentDataStream {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String file = "students.dat";
        try (DataOutputStream dos = new
DataOutputStream(new FileOutputStream(file))) {
            System.out.print("Enter number of students: ");
            int n = sc.nextInt();
            sc.nextLine();

            for (int i = 0; i < n; i++) {
                System.out.print("Enter roll, name, GPA:
");
                int roll = sc.nextInt();
                String name = sc.next();
                float gpa = sc.nextFloat();
                dos.writeInt(roll);
                dos.writeUTF(name);
                dos.writeFloat(gpa);
            }
        } catch (IOException e) {
            System.out.println("Write Error: " +
e.getMessage());
        }

        try (DataInputStream dis = new DataInputStream(new
FileInputStream(file))) {
            while (true) {
                int roll = dis.readInt();
                String name = dis.readUTF();
                float gpa = dis.readFloat();
                System.out.println(roll + " " + name + " "
+ gpa);
            }
        } catch (EOFException e) {
            // End of file
        } catch (IOException e) {
            System.out.println("Read Error: " +
e.getMessage());
        }
    }
}
```

---

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 PipedCommunication {
    public static void main(String[] args) throws IOException {
        PipedOutputStream pos = new PipedOutputStream();
        PipedInputStream pis = new PipedInputStream(pos);

        Thread writer = new Thread(() -> {
            try {
                String msg = "Hello from writer thread!";
                pos.write(msg.getBytes());
                pos.close();
            } catch (IOException e) {
                System.out.println("Writer Error: " +
e.getMessage());
            }
        });

        Thread reader = new Thread(() -> {
            try {
                int ch;
                while ((ch = pis.read()) != -1) {
                    System.out.print((char) ch);
                }
                pis.close();
            } catch (IOException e) {
                System.out.println("Reader Error: " +
e.getMessage());
            }
        });

        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 ReadLargeFile {
    public static void main(String[] args) {
        try (BufferedReader br = new BufferedReader(new
FileReader("log.txt"))) {
            String line;
            while ((line = br.readLine()) != null) {
                if (line.toLowerCase().contains("error")) {
                    System.out.println(line);
                }
            }
        } catch (IOException e) {
            System.out.println("Read Error: " + e.getMessage());
        }
    }
}
```

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 WordFrequencyTop5 {
    public static void main(String[] args) {
        Map<String, Integer> map = new HashMap<>();

        try (BufferedReader br = new BufferedReader(new
FileReader("input.txt"))) {
            String line;
            while ((line = br.readLine()) != null) {
                line = line.replaceAll("[^a-zA-Z ]",
"").toLowerCase();
                String[] words = line.split("\\s+");
                for (int i = 0; i < words.length; i++) {
                    if (!words[i].isEmpty()) {
                        map.put(words[i], map.getOrDefault(words[i],
0) + 1);
                    }
                }
            }
        } catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
        }

        List<Map.Entry<String, Integer>> list = new
ArrayList<>(map.entrySet());
        Collections.sort(list, (a, b) -> b.getValue() -
a.getValue());

        System.out.println("Top 5 words:");
        for (int i = 0; i < Math.min(5, list.size()); i++) {
            System.out.println(list.get(i).getKey() + " = " +
list.get(i).getValue());
        }
    }
}
```