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.

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

public class ReadAndWrite {

    public static void copyFile(String src, String des) {
        try {
            FileInputStream fs = new FileInputStream(src);
            FileOutputStream fos = new FileOutputStream(des);

            int data;
            while ((data = fs.read()) != -1) {
                fos.write(data);
            }

            System.out.println("Copied!");

            fs.close();
            fos.close();

        } catch (Exception e) {
            System.out.println("An error occurred!");
            e.printStackTrace();
        }
    }
```

```
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the path of source file to copy data: ");
        String src = sc.next();

        System.out.print("Enter destination file name if exists otherwise enter
new file name: ");
        String des = sc.next();

        copyFile(src, des);

        sc.close();
    }

}
```

---

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.util.*;
import java.io.*;

public class EfficientFileCopy {


    public static void copyFile(String src, String des, int buffersize) {
        long startTimeBuffered = System.nanoTime();
        long endTimeBuffered = System.nanoTime();

        try {
            BufferedInputStream bis = new BufferedInputStream(new
FileInputStream(src), buffersize);
            BufferedOutputStream bos = new BufferedOutputStream(new
FileOutputStream(des), buffersize);

            byte arr[] = new byte[buffersize];
            int bytesRead;
            while ((bytesRead = bis.read(arr)) != -1) {
                bos.write(arr, 0, bytesRead);
            }

            endTimeBuffered = System.nanoTime();

            System.out.println("Buffered Input Stream: " + (endTimeBuffered -
startTimeBuffered) + " nanoseconds");

            bis.close();
            bos.close();

        } catch (Exception e) {
            System.out.println("Buffered Input Stream: " + e.getMessage());
        }

        long startTimeUnBuffered = System.nanoTime();
        long endTimeUnBuffered = System.nanoTime();

        try {
            FileInputStream fis = new FileInputStream(src);
            FileOutputStream fos = new FileOutputStream(des);

            byte arr[] = new byte[buffersize];
            int bytesRead;
```

```
            while ((bytesRead = fis.read(arr)) != -1) {
                fos.write(arr, 0, bytesRead);
            }

            endTimeUnBuffered = System.nanoTime();

            System.out.println("File Input Stream: " + (endTimeUnBuffered -
startTimeUnBuffered) + " nanoseconds");

            fis.close();
            fos.close();

        } catch (Exception e) {
            e.printStackTrace();
        }

    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the path of source file to copy data: ");
        String src = sc.next();

        System.out.print("Enter destination file name if exists otherwise enter
new file name: ");
        String des = sc.next();

        System.out.println("Enter buffer size: ");
        int buffersize = sc.nextInt();

        copyFile(src, des, buffersize);

        sc.close();
    }

}
```

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 UserInputToFile {
    public static void main(String[] args) throws IOException {
        saveUserInput();
    }

    static void saveUserInput() throws IOException {
        BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
        FileWriter fw = new FileWriter("user_data.txt");
        fw.write(br.readLine() + "\n" + br.readLine() + "\n" + br.readLine());
        fw.close();
    }
}
```

---

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 department, double salary) {
        this.id = id;
        this.name = name;
        this.department = department;
        this.salary = salary;
    }
    public String toString() { return id + " " + name + " " + department + " "
+ salary; }
}
public class EmployeeSerialization {
    public static void main(String[] args) throws IOException,
ClassNotFoundException {
        processEmployees();
    }
```

```
    static void processEmployees() throws IOException, ClassNotFoundException {
        List<Employee> employees = Arrays.asList(new Employee(1, "ABC", "HR",
50000));
        ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream("employees.dat"));
        oos.writeObject(employees);
        oos.close();
        ObjectInputStream ois = new ObjectInputStream(new
FileInputStream("employees.dat"));
        System.out.println(ois.readObject());
        ois.close();
    }
}
```

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`.

```java
import java.io.*;

public class ImageByteArray {
    public static void main(String[] args) throws IOException {
        processImage();
    }

    static void processImage() throws IOException {
        FileInputStream fis = new FileInputStream("input.jpg");
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        byte[] buffer = new byte[4096];
        int bytesRead;
        while ((bytesRead = fis.read(buffer)) != -1)
            baos.write(buffer, 0, bytesRead);
        fis.close();
        ByteArrayInputStream bais = new
ByteArrayInputStream(baos.toByteArray());
        FileOutputStream fos = new FileOutputStream("output.jpg");
        while ((bytesRead = bais.read(buffer)) != -1)
            fos.write(buffer, 0, bytesRead);
        fos.close();
    }
}
```

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 UppercaseToLowercase {
    public static void main(String[] args) throws IOException {
        convertToLowercase();
    }

    static void convertToLowercase() throws IOException {
        BufferedReader br = new BufferedReader(new FileReader("input.txt"));
        BufferedWriter bw = new BufferedWriter(new FileWriter("output.txt"));
        String line;
        while ((line = br.readLine()) != null)
            bw.write(line.toLowerCase() + "\n");
        br.close();
        bw.close();
    }
}
```

---

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) throws IOException {
        processStudentData();
    }

    static void processStudentData() throws IOException {
        DataOutputStream dos = new DataOutputStream(new
FileOutputStream("student.dat"));
        dos.writeInt(1);
        dos.writeUTF("PQR");
        dos.writeDouble(3.8);
        dos.close();
        DataInputStream dis = new DataInputStream(new
FileInputStream("student.dat"));
        System.out.println(dis.readInt() + " " + dis.readUTF() + " " +
dis.readDouble());
        dis.close();
    }
}
```

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 {
        processPipedCommunication();
    }

    static void processPipedCommunication() throws IOException {
        PipedOutputStream pos = new PipedOutputStream();
        PipedInputStream pis = new PipedInputStream(pos);
        new Thread(() -> {
            try {
                pos.write("Hello".getBytes());
                pos.close();
            } catch (IOException ignored) {
            }
        }).start();
        new Thread(() -> {
            try {
                int d;
                while ((d = pis.read()) != -1)
```

```
                    System.out.print((char) d);
                pis.close();
            } catch (IOException ignored) {
            }
        }).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 {
        readLargeFile();
    }
```

```
    static void readLargeFile() throws IOException {
        BufferedReader br = new BufferedReader(new FileReader("large.txt"));
        String line;
        while ((line = br.readLine()) != null)
            if (line.toLowerCase().contains("error"))
                System.out.println(line);
        br.close();
    }
}
```

---

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 WordCount {
    public static void main(String[] args) throws IOException {
        countWords();
    }
```

```
    static void countWords() throws IOException {
        BufferedReader br = new BufferedReader(new FileReader("text.txt"));
        HashMap<String, Integer> map = new HashMap<>();
        String line;
        while ((line = br.readLine()) != null)
            for (String word : line.split("\\s+"))
                map.put(word, map.getOrDefault(word, 0) + 1);
        br.close();
        map.entrySet().stream().sorted((a, b) -> b.getValue() -
a.getValue()).limit(5)
                .forEach(e -> System.out.println(e.getKey() + " " +
e.getValue()));
    }
}
```