EXPERIMENT NO. 07: JAVA FILE I/O

Objective

To create a Java program that demonstrates the use of the Java I/O package for reading and writing data to files.

Theory

The java.io package in Java provides classes and interfaces for reading and writing data from files, input streams, and output streams. Some commonly used classes are File, FileWriter, FileReader, BufferedWriter, BufferedReader, Scanner, DataInputStream, and Console.

Advantages of Java I/O

• Handles file reading and writing efficiently
• Supports character and byte stream operations
• Provides buffering for performance improvement
• Offers stream chaining (decorators) for flexible processing

Program Code – Writing and Reading a File

```
import java.io.*;

public class FileDemo {
    public static void main(String[] args) {
        String filename = "output.txt";
        String message = "Java I/O File Handling Example.";

        try (BufferedWriter writer = new BufferedWriter(new FileWriter(filename))) {
            writer.write(message);
            writer.newLine();
            writer.write("This is a second line written to file.");
        } catch (IOException e) {
            System.err.println("Error writing to file: " + e.getMessage());
        }

        try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {
            String line;
            while ((line = reader.readLine()) != null) {
                System.out.println("Read: " + line);
            }
        } catch (IOException e) {
            System.err.println("Error reading from file: " + e.getMessage());
        }
    }
}
```

Sample Output

Read: Java I/O File Handling Example.
Read: This is a second line written to file.

Explanation

• BufferedWriter is used to write data to the file `output.txt`.
• BufferedReader reads content line by line from the file.
• The use of try-with-resources ensures proper stream closure.

Coding Task: Take User Input using Different Methods

```
import java.io.*;

import java.util.Scanner;

public class InputDemo {
    public static void main(String[] args) throws IOException {

        if (args.length > 0) {
            System.out.println("Command Line Argument: " + args[0]);
        } else {
            System.out.println("No Command Line Argument provided.");
        }

        DataInputStream dis = new DataInputStream(System.in);
        System.out.print("Enter using DataInputStream: ");
        String disInput = dis.readLine();
        System.out.println("Received: " + disInput);

        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        System.out.print("Enter using BufferedReader: ");
        String brInput = br.readLine();
        System.out.println("Received: " + brInput);

        Scanner sc = new Scanner(System.in);
        System.out.print("Enter using Scanner: ");
        String scInput = sc.nextLine();
        System.out.println("Received: " + scInput);
        Console console = System.console();
        if (console != null) {
            String consoleInput = console.readLine("Enter using Console: ");
            System.out.println("Received: " + consoleInput);
        } else {
            System.out.println("Console not available.");
        }
    }
}
```

Sample Output

```
Command Line Argument: HelloWorld
Enter using DataInputStream: DataStreamInput
Received: DataStreamInput
Enter using BufferedReader: BufferedReaderInput
Received: BufferedReaderInput
Enter using Scanner: ScannerInput
Received: ScannerInput
Console not available.
```

Conclusion

The Java I/O system offers multiple ways to interact with the user and the file system. In this experiment, we explored file handling using BufferedWriter and BufferedReader, and learned different methods to take user input effectively.