

| | | | |
|---------------------------------|------------------------------------|---------------------|-----------------|
| Course Name: | Object Oriented Programming | Semester: | III |
| Date of Performance: | 30 / 09 / 2025 | Batch No: | Batch A1 |
| Faculty Name: | Prof. Amrita Naiksatam | Roll No: | 20 |
| Faculty Sign & Date: | | Grade/Marks: | ___/15 |

Experiment No: 7
Title: I/O Function in Java

| |
|---|
| Aim and Objective of the Experiment: |
| Learn the different types of I/O function |

| |
|---|
| COs to be achieved: |
| CO3: Define exceptions and use I/O streams. |

| |
|----------------------------------|
| Tools used: |
| JetBrains IntelliJ Idea Ultimate |

| |
|---|
| Theory: |
| (Byte stream class , char stream class and different Reading/writing methods) |

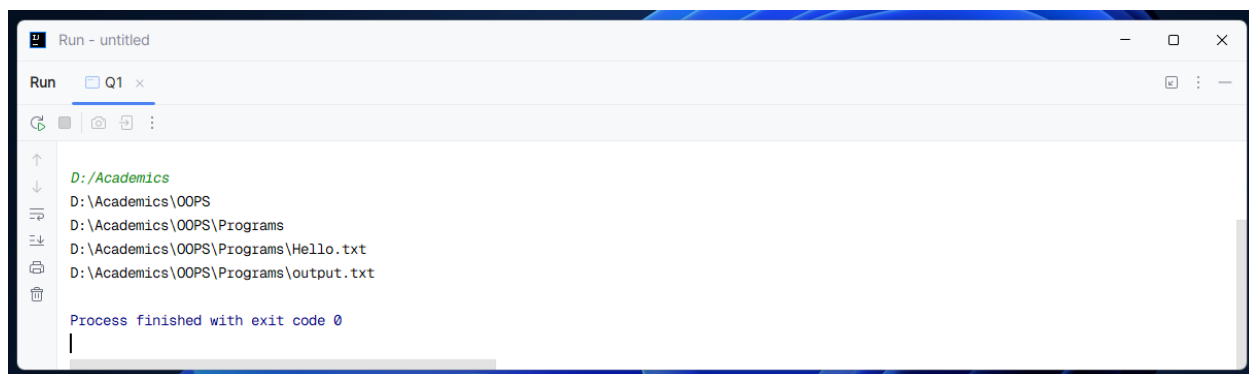
| |
|---|
| Code: |
| <ol style="list-style-type: none"> 1. Write a program that lists all the files in a directory including the files present in all its subdirectories as well. Get name/path of the directory from the user through standard input. [Hint: Use recursion] 2. Write a program to read the contents of a file byte by byte and copy it into another file. Get names of the files from the user through standard input. 3. Write a program that appends data to the file using FileWriter class.[Hint: Use a different constructor of FileWriter class] |

1.**Program:**

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

public class Q1
{
    static void list(File dir)
    {
        File[] files = dir.listFiles();
        for(int i = 0; i < files.length; i++)
        {
            System.out.println(files[i].getPath());
            if(files[i].isDirectory())
            {
                list(files[i]);
            }
        }
    }

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

Output:

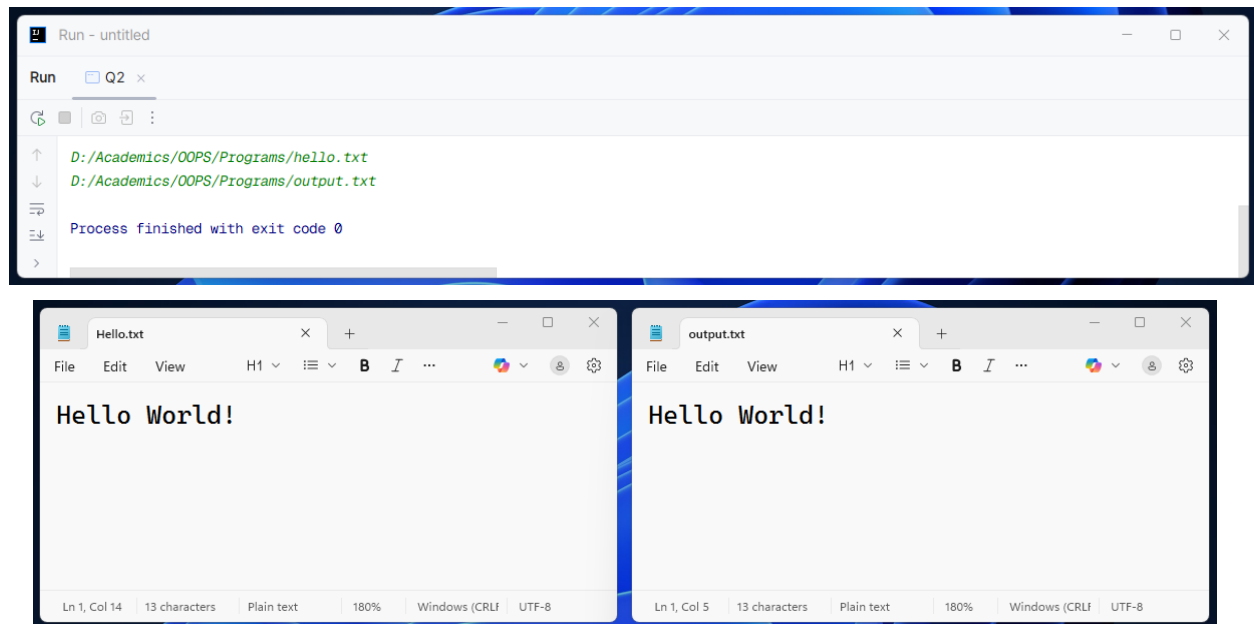
```
Run - untitled
Run  Q1 x
D:/Academics
D:\Academics\OOPS
D:\Academics\OOPS\Programs
D:\Academics\OOPS\Programs\Hello.txt
D:\Academics\OOPS\Programs\output.txt

Process finished with exit code 0
```

2.**Program:**

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

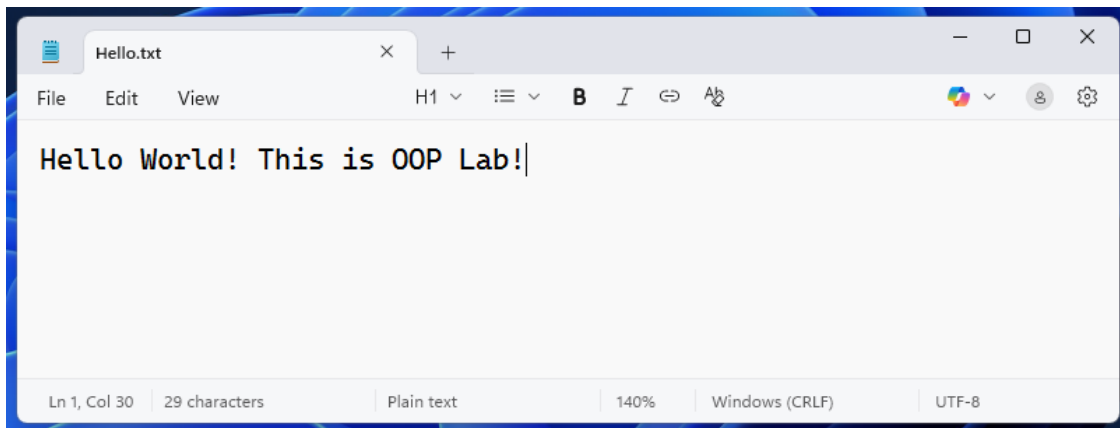
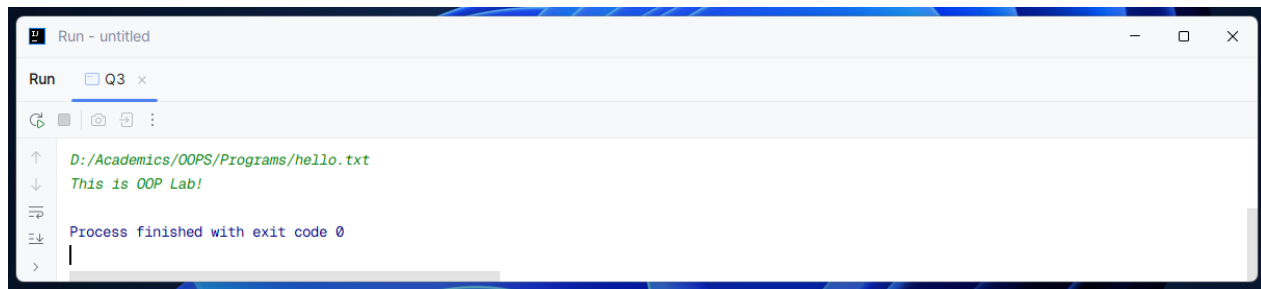
public class Q2
{
    public static void main(String[] args) throws Exception
    {
        Scanner sc = new Scanner(System.in);
        String source = sc.nextLine();
        String destination = sc.nextLine();
        FileInputStream fis = new FileInputStream(source);
        FileOutputStream fos = new FileOutputStream(destination);
        int b;
        while((b = fis.read()) != -1)
        {
            fos.write(b);
        }
        fis.close();
        fos.close();
    }
}
```

Output:

3.**Program:**

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

public class Q3
{
    public static void main(String[] args) throws Exception
    {
        Scanner sc = new Scanner(System.in);
        String filename = sc.nextLine();
        String data = sc.nextLine();
        FileWriter fw = new FileWriter(filename, true);
        fw.write(data);
        fw.close();
    }
}
```

Output:

Post Lab Subjective/Objective type Questions:

1. Explain the difference between FileInputStream and BufferedInputStream. Show an example in support of your answer.

Ans:

1. **FileInputStream**

- Reads bytes directly from a file.
- Each read() call goes to the operating system → relatively **slower**.
- Good for reading small amounts of raw binary data.

2. **BufferedInputStream**

- Wraps around another InputStream (e.g., FileInputStream).
- Uses an internal **buffer (default 8 KB)** to reduce disk I/O calls.
- Each read() first checks the buffer → only when empty, fetches a new chunk from the file.
- Much **faster** for large files.

Example:

```
import java.io.*;

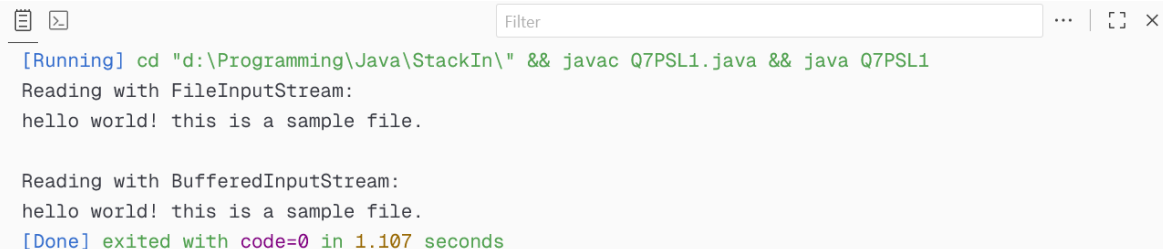
public class SimpleExample {
    public static void main(String[] args) {
        try {
            FileInputStream fis = new FileInputStream("example.txt");
            BufferedInputStream bis = new BufferedInputStream(new FileInputStream("example.txt"));

            System.out.println("Reading with FileInputStream:");
            int ch;
            while ((ch = fis.read()) != -1) {
                System.out.print((char) ch);
            }
            fis.close();

            System.out.println("\n\nReading with BufferedInputStream:");
            while ((ch = bis.read()) != -1) {
                System.out.print((char) ch);
            }
            bis.close();

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Output:



```
[Running] cd "d:\Programming\Java\StackIn\" && javac Q7PSL1.java && java Q7PSL1
Reading with FileInputStream:
hello world! this is a sample file.

Reading with BufferedInputStream:
hello world! this is a sample file.
[Done] exited with code=0 in 1.107 seconds
```

2. How many lines, words, and characters does a file have? Write a program for the same.
Explain in detail all the possible ways of taking inputs from the user.

Program:

```
import java.io.*;

public class FileStats {
    public static void main(String[] args) {
        int lines = 0, words = 0, chars = 0;

        try {
            BufferedReader br = new BufferedReader(new FileReader("example.txt"));
            String line;

            while ((line = br.readLine()) != null) {
                lines++;
                chars += line.length();
                String[] wordArray = line.trim().split("\\s+");
                if (!line.trim().isEmpty()) {
                    words += wordArray.length;
                }
            }
            br.close();

            System.out.println("Lines: " + lines);
            System.out.println("Words: " + words);
            System.out.println("Characters: " + chars);

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Output:



Ways to take input from user in Java

1. **Command-line arguments (String[] args)**
 - Passed at program start.
 - Example: java App Hello → args[0] = "Hello".
2. **Scanner (java.util.Scanner)**
 - Easiest way for beginners.
 - Methods: nextInt(), next(), nextLine().
3. **BufferedReader + InputStreamReader**
 - Reads input line by line (readLine()).
 - Faster for large input.
4. **Console class**
 - readLine(), readPassword().
 - Best for secure inputs (passwords).
5. **System.in.read()**
 - Reads raw bytes/characters.
 - Very low-level, rarely used directly.
6. **DataInputStream**
 - Reads primitive data types (readInt(), readDouble()).
 - Used for binary input, not common for text.
7. **GUI input (JOptionPane / JavaFX)**
 - Input via dialog boxes.
 - Example: JOptionPane.showInputDialog("Enter name:").
8. **Environment variables / System properties**
 - System.getenv("USER"), System.getProperty("java.version").
 - Useful for configuration, not interactive.

Conclusion:

In this experiment, we explored the use of Java I/O functions through practical programs. We learned how to list files recursively using directory traversal, how to copy file contents using byte streams, and how to append data with character streams. We also understood the differences between FileInputStream and BufferedInputStream, along with various ways of taking input from the user (command-line, Scanner, BufferedReader, Console, etc.). Overall, the experiment helped us gain hands-on experience with Java I/O streams, their efficiency, and real-world applications.

Signature of faculty in-charge with Date: