24MCA10066

LAB Assignment-5

MCA (1001)

Programs on Files and I/O Streams

Objective:

The purpose of this lab assignment is to enhance your understanding of file handling and I/O streams in Java. You will work on various problems that involve reading from, writing to, and manipulating files using different I/O stream classes. By the end of this assignment, you should be comfortable with performing file operations and handling exceptions effectively.

1.File Reading and Writing

Code:

```
package com.company.LabAssignment;
import java.io.*;

public class FileReadExample {

    public static void appendStrToFile(String fileName,String str)
    {
        try {

            BufferedWriter out = new BufferedWriter(
                    new FileWriter(fileName, true));

            out.write(str);
            out.close();
        }
        catch (IOException e) {

            System.out.println("exception occurred" + e);
        }
    }

    public static void main(String[] args) throws Exception
    {
        String fileName = "hello.pages";

        try {
            BufferedWriter out = new BufferedWriter(
                    new FileWriter(fileName));
            out.write("Hello World:\n");
            out.close();
        }

        catch (IOException e) {
```

```
            System.out.println("Exception Occurred" + e);
        }

        String str = "Hello World";
        appendStrToFile(fileName, str);

        try {
            BufferedReader in = new BufferedReader(
                    new FileReader("hello.txt"));

            String mystring;
            while ((mystring = in.readLine()) != null) {
                System.out.println(mystring);
            }
        }
        catch (IOException e) {
            System.out.println("Exception Occurred" + e);
        }
    }
}
```

2. Character Streams:

Code:

```
package com.company.LabAssignment;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.*;

public class fileCharacterStreams {
        public static void main(String[] args) {
//Using FileReader and FileWriter
            try (FileReader reader = new FileReader("test.txt");
                 FileWriter writer = new FileWriter("output2.txt")) {
                int character;
                while ((character = reader.read()) != -1) {
                    writer.write(character);
                }
            } catch (IOException e) {
                e.printStackTrace();
            }

            try (BufferedReader reader = new BufferedReader(new
FileReader("test.txt"));
                 BufferedWriter writer = new BufferedWriter(new
FileWriter("output2.txt"))) {
                String line;
                while ((line = reader.readLine()) != null) {
                    writer.write(line);
                    writer.newLine();
                }
            } catch (IOException e) {
                e.printStackTrace();
```

```
            }
        }
}
```

3. Binary File Operations:

Code:

```
package com.company.LabAssignment;
import java.io.*;
//import java.util.*;

public class BinaryFileOperation {
    public static void main(String[] args) {
// a. Copying a binary file
            try (FileInputStream inStream = new
FileInputStream("input.bin");
                 FileOutputStream outStream = new
FileOutputStream("output.bin")) {
                int byteData;
                while ((byteData = inStream.read()) != -1) {
                    outStream.write(byteData);
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
// b. Copying a binary file with buffer support for larger files
            try (FileInputStream inStream = new
FileInputStream("input.bin");
                 FileOutputStream outStream = new
FileOutputStream("output.bin")) {
                byte[] buffer = new byte[1024];
                int bytesRead;
                while ((bytesRead = inStream.read(buffer)) != -1) {
                    outStream.write(buffer, 0, bytesRead);
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
```

4. Handling Data Streams:

Code:

```
import java.io.*;
public class File_ioStreams {
    public static void main(String[] args) {
// Writing primitive data types
        try (DataOutputStream dos = new DataOutputStream(new
                FileOutputStream("data.bin"))) {
            dos.writeInt(42);
            dos.writeFloat(3.14f);
            dos.writeDouble(2.71828);
        } catch (IOException e) {
```

```
            e.printStackTrace();
        }
// Reading primitive data types
        try (DataInputStream dis = new DataInputStream(new
                FileInputStream("data.bin"))) {
            System.out.println("Int: " + dis.readInt());
            System.out.println("Float: " + dis.readFloat());
            System.out.println("Double: " + dis.readDouble());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

5. Serialization and Deserialization:

Code:

```
import java.io.*;

class Student implements Serializable {
    private String name;
    private int id;
    private double grade;
    public Student(String name, int id, double grade) {
        this.name = name;
        this.id = id;
        this.grade = grade;
    }
    @Override
    public String toString() {
        return "Student{" +
                "name='" + name + '\'' +
                ", id=" + id +
                ", grade=" + grade +
                '}';
    }
}
public class File_ioStreams {
    public static void main(String[] args) {
// Serializing an object
        Student = new Student("Alice", 123, 3.8);
        try (ObjectOutputStream oos = new ObjectOutputStream(new
                FileOutputStream("student.ser"))) {
            oos.writeObject(student);
        } catch (IOException e) {
            e.printStackTrace();
        }
// Deserializing the object
        try (ObjectInputStream ois = new ObjectInputStream(new
                FileInputStream("student.ser"))) {
            Student deserializedStudent = (Student) ois.readObject();
            System.out.println(deserializedStudent);
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

6. Exception Handling with Files:

Code:

```
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

public class File_ioStreams {
    public static void main(String[] args) {
// Exception handling example with file operations
        FileReader reader = null;
        try {
            reader = new FileReader("test.txt");
            int character;
            while ((character = reader.read()) != -1) {
                System.out.print((char) character);
            }
        } catch (FileNotFoundException e) {
            System.out.println("The specified file was not found.");
        } catch (IOException e) {
            System.out.println("An I/O error occurred.");
        } finally {
            try {
                if (reader != null) reader.close();
            } catch (IOException e) {
                System.out.println("Error closing the file.");
            }
        }
    }
}
```