Lab Series 08

Exercise 1: Inner Classes, Enumerations, Serialization and Streams

Define a class `Department` that contains an inner class `Employee` and an enumeration `DepartmentType`. Implement serialization and deserialization of an instance of the `Department` class to a file called `department.ser`.

Exercise 2: Anonymous Classes and Character Streams

Write a program that uses an anonymous class to handle reading from a file called `input.txt` and writing to a file called `output.txt` using character streams (`FileReader` and `FileWriter`).

Exercise 3: Byte Streams with Files and Java.nio

Write a program that uses byte streams (`FileInputStream` and `FileOutputStream`) and Java.nio (`FileChannel` and `ByteBuffer`) to copy the contents of a file called `source.dat` to another file called `destination.dat`.

Série de TP 08

Exercice 1 : Classes internes, énumérations, sérialisation et flux

Définir une classe `Department` qui contient une classe interne `Employee` et une énumération `DepartmentType`. Mettre en œuvre la sérialisation et la désérialisation d'une instance de la classe `Department` dans un fichier appelé `department.ser`.

Exercice 2 : Classes anonymes et flux de caractères

Écrire un programme qui utilise une classe anonyme pour gérer la lecture d'un fichier appelé `input.txt` et l'écriture dans un fichier appelé `output.txt` en utilisant des flux de caractères (`FileReader` et `FileWriter`).

Exercice 3 : Flux d'octets avec des fichiers et Java.nio

Écrire un programme qui utilise des flux d'octets (`FileInputStream` et `FileOutputStream`) et Java.nio (`FileChannel` et `ByteBuffer`) pour copier le contenu d'un fichier appelé `source.dat` dans un autre fichier appelé `destination.dat`.

Série de TP 8

Exercise 1: Inner Classes, Enumerations, Serialization and Streams

Define a class `Department` that contains an inner class `Employee` and an enumeration `DepartmentType`. Implement serialization and deserialization of an instance of the `Department` class to a file called `department.ser`.

Solution:

```
import java.io.*;

public class Department implements Serializable {
    private static final long serialVersionUID = 1L; // For compatibility

    public enum DepartmentType {
        HR, IT, SALES, MARKETING
    }

    private DepartmentType departmentType;
    private Employee manager;

    public Department(DepartmentType type, String managerName, int managerAge) {
        this.departmentType = type;
        this.manager = new Employee(managerName, managerAge);
    }

    public void display() {
        System.out.println("Department Type: " + departmentType);
        manager.display();
    }

    public class Employee implements Serializable {
        private static final long serialVersionUID = 1L;

        private String name;
        private int age;

        public Employee(String name, int age) {
            this.name = name;
            this.age = age;
        }

        public void display() {
            System.out.println("Manager Name: " + name + ", Age: " + age);
        }
    }

    public static void main(String[] args) {
        // Create a Department object
        Department department = new Department(DepartmentType.IT, "John Doe", 35);

        // Serialize the Department object
        try (ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream("department.ser"))) {
            out.writeObject(department);
            System.out.println("Department object serialized.");
        } catch (IOException e) {
```

```
            e.printStackTrace();
        }


        // Deserialize the Department object
                try (ObjectInputStream in = new ObjectInputStream(new FileInputStream("department.ser"))) {
                    Department deserializedDepartment = (Department) in.readObject();
                    System.out.println("Department object deserialized.");
            deserializedDepartment.display();
                            } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

Exercise 2: Anonymous Classes and Character Streams

Write a program that uses an anonymous class to handle reading from a file called `input.txt` and writing to a file called `output.txt` using character streams (`FileReader` and `FileWriter`).

Solution:

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


public class CharacterStreamExample {
    public static void main(String[] args) {


        try {
                final FileReader inputReader = new FileReader("input.txt");
                final FileWriter outputWriter = new FileWriter("output.txt");


            // Use an anonymous class to read and write
            Runnable fileOperations = new Runnable() {
                @Override
                public void run() {
                    try {
                        int c;
                        while ((c = inputReader.read()) != -1) {
                            outputWriter.write(c);
                        }
                        System.out.println("Character stream copy completed.");
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }
            };


            // Execute the file operations
            fileOperations.run();
                            if (inputReader != null) {
                                inputReader.close();
                            }
                            if (outputWriter != null) {
                                outputWriter.close();
                            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Exercise 3: Byte Streams with Files and Java.nio

Write a program that uses byte streams (`FileInputStream` and `FileOutputStream`) and Java.nio (`FileChannel` and `ByteBuffer`) to copy the contents of a file called `source.dat` to another file called `destination.dat`.

Solution:

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.channels.FileChannel;


public class FileCopyWithNio {
    public static void main(String[] args) {
        FileInputStream in = null;
        FileOutputStream out = null;
        FileChannel inputChannel = null;
        FileChannel outputChannel = null;


        try {
            // Open input and output streams
            in = new FileInputStream("source.dat");
                                        out = new FileOutputStream("destination.dat");


            // Get file channels from streams
            inputChannel = in.getChannel();
            outputChannel = out.getChannel();


              // Create a buffer with a capacity of 1024 bytes
                                    ByteBuffer buffer = ByteBuffer.allocate(1024);

             // Copy the contents of the source file to the destination file
            while (inputChannel.read(buffer) > 0) {
                    buffer.flip(); // Prepare the buffer for writing
                outputChannel.write(buffer);
                    buffer.clear(); // Clear the buffer for the next read
            }
            System.out.println("File copy using Java.nio completed successfully.");
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            // Close the channels and streams
            try {
                if (inputChannel != null) {
                    inputChannel.close();
                }
                if (outputChannel != null) {
                    outputChannel.close();
                }
                if (in != null) {
                    in.close();
                }
                if (out != null) {
                    out.close();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```