Title: Create and demonstrate program for the concept of Multithreading

Objective: Understand the concepts of multithreading in Java.

Problem Statement: Create a Java program to illustrate multithreading concepts. The program should consist of two parts, each addressing different aspects of multithreading.

Instructions:

1. Use proper comments for clarity and understanding.
2. Ensure that the interaction between threads is well-organized and demonstrates the specified aspects of multithreading.

Java Program:

```
// File: MultithreadingBasicsDemo.java

class NumberPrinter extends Thread {
    private static final Object lock = new Object(); // Object for synchronization

    private void printNumbers() {
        for (int i = 1; i <= 5; i++) {
            System.out.println(Thread.currentThread().getName() + " - Count: " + i);
        }
    }


    private void printEvenOddNumbers() {
        for (int i = 2; i <= 10; i += 2) {
            System.out.println(Thread.currentThread().getName() + " - Even Number: " + i);
            try {
                // Introducing a delay of 500 milliseconds
                Thread.sleep(500);
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
                e.printStackTrace();
            }
        }

        for (int i = 1; i <= 9; i += 2) {
            System.out.println(Thread.currentThread().getName() + " - Odd Number: " + i);
            try {
                // Introducing a delay of 500 milliseconds
                Thread.sleep(500);
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
                e.printStackTrace();
            }
        }
    }

    // Entry point for each thread
    public void run() {
        printNumbers();
        printEvenOddNumbers();
    }
}

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

```
        // Creating two threads
        Thread thread1 = new NumberPrinter();
        Thread thread2 = new NumberPrinter();

        // Naming the threads for clarity
        thread1.setName("Thread 1");
        thread2.setName("Thread 2");

        // Starting the threads
        thread1.start();
        thread2.start();
    }
}
```

Explanation:

- The NumberPrinter class extends Thread and defines methods for printing numbers in two different ways.
- The run method serves as the entry point for each thread, executing the specified sub-questions.
- The MultithreadingBasicsDemo class creates two threads, names them for clarity, and starts their execution.

The provided code focuses on understanding multithreading concepts, with well-commented sections for clarity and comprehension.

Title: Create and demonstrate I/O programs.

Objective: Understand the concepts of Input/Output in Java.

Problem Statement: Create a Java program to illustrate I/O concepts. The program should consist of two parts, each addressing different aspects of file input and output.

Instructions:

1. **Write Operation:**
    - Create a text file named "output.txt."
    - Use FileWriter to write data to the file.
    - Write a sequence of lines with information.
2. **Read Operation:**
    - Read data from the "output.txt" file using FileReader.
    - Print the read data to the console.

Java Program:

```
// File: FileIOExample.java

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

public class FileIOExample {

    public static void main(String[] args) {
        // Perform file I/O operations
        writeAndReadFile();
    }

    private static void writeAndReadFile() {
        // File path
        String filePath = "output.txt";
```

```
        // Part 1: Write Operation
        writeToFile(filePath);

        // Part 2: Read Operation
        readFromFile(filePath);
    }

    private static void writeToFile(String filePath) {
        try (FileWriter fw = new FileWriter(filePath)) {
            // Write data to the file
            fw.write("Line 1: This is some information.\n");
            fw.write("Line 2: More details here.\n");

            System.out.println("Data written to file successfully.");

        } catch (IOException e) {
            // Handle exceptions
            System.err.println("Error during file writing: " + e.getMessage());
            e.printStackTrace();
        }
    }

    private static void readFromFile(String filePath) {
        try (FileReader fr = new FileReader(filePath);
             BufferedReader br = new BufferedReader(fr)) {

            String line;
            // Read and print each line
            while ((line = br.readLine()) != null) {
                System.out.println("Read from file: " + line);
            }

            System.out.println("File reading completed successfully.");

        } catch (IOException e) {
            // Handle exceptions
            System.err.println("Error during file reading: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Explanation:

• FileIOExample is a Java program demonstrating file I/O operations.
• The main method orchestrates writing and reading operations through writeAndReadFile.
• writeToFile method uses FileWriter to write data to "output.txt," handling IOExceptions.
• readFromFile method uses FileReader and BufferedReader to read and print lines, handling IOExceptions.