Advanced Java : **Day-19(Java IO Basics, Serialization and Deserialization, New IO (NIO), Java Networking, Java 8 Date and Time API)**:

Task 1: Java IO Basics Write a program that reads a text file and counts the frequency of each word using FileReader and FileWriter.

```
package wipro.com.assignment15;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

public class WordFrequencyCounter {

    public static void main(String[] args) {
        String inputFile = "input.txt"; // Replace with your input file path
        String outputFile = "output.txt"; // Replace with your output file path

        try {
            // FileReader and BufferedReader for input
            FileReader fileReader = new FileReader(inputFile);
            BufferedReader bufferedReader = new BufferedReader(fileReader);

            // FileWriter and BufferedWriter for output
            FileWriter fileWriter = new FileWriter(outputFile);
            BufferedWriter bufferedWriter = new BufferedWriter(fileWriter);

            // Map to store word frequencies
            Map<String, Integer> wordFrequencyMap = new HashMap<>();

            // Read each line from input file
            String line;
            while ((line = bufferedReader.readLine()) != null) {
                // Split line into words by whitespace
                String[] words = line.split("\\s+");

                // Count frequencies of each word
                for (String word : words) {
                    // Convert to lowercase for case insensitivity
                    String cleanedWord = word.toLowerCase().replaceAll("[^a-zA-Z0-
9]", "");
                    if (!cleanedWord.isEmpty()) {
                        wordFrequencyMap.put(cleanedWord,
wordFrequencyMap.getOrDefault(cleanedWord, 0) + 1);
                    }
                }
            }
```

```
            // Close input resources
            bufferedReader.close();
            fileReader.close();

            // Write word frequencies to output file
            for (Map.Entry<String, Integer> entry : wordFrequencyMap.entrySet()) {
                bufferedWriter.write(entry.getKey() + ": " + entry.getValue());
                bufferedWriter.newLine();
            }

            // Close output resources
            bufferedWriter.close();
            fileWriter.close();

            System.out.println("Word frequencies written to " + outputFile);

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

Task 2: Serialization and Deserialization Serialize a custom object to a file and then deserialize it back to recover the object state.

```
package wipro.com.assignment15;
import java.io.Serializable;

public class Person implements Serializable {
    private String name;
    private int age;

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

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    @Override
    public String toString() {
        return "Person{" +
                "name='" + name + '\'' +
                ", age=" + age +
                '}';
```

```
    }
}
```

Task 3: New IO (NIO) Use NIO Channels and Buffers to read content from a file and write to another file.

```
package wipro.com.assignment15;
import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.channels.FileChannel;
import java.nio.file.*;

public class NIOExample {

    public static void main(String[] args) {
        String sourceFile = "source.txt"; // Replace with your source file path
        String targetFile = "target.txt"; // Replace with your target file path

        try {
            // Open a FileChannel for reading from source file
            FileChannel sourceChannel = FileChannel.open(Paths.get(sourceFile),
StandardOpenOption.READ);

            // Create a ByteBuffer for reading data
            ByteBuffer buffer = ByteBuffer.allocate(1024); // Buffer size is 1KB

            // Open a FileChannel for writing to target file
            FileChannel targetChannel = FileChannel.open(Paths.get(targetFile),
StandardOpenOption.CREATE,
                    StandardOpenOption.WRITE, StandardOpenOption.TRUNCATE_EXISTING);

            // Read data from source file and write it to target file
            while (sourceChannel.read(buffer) > 0 || buffer.position() > 0) {
                buffer.flip(); // Prepare buffer for writing
                targetChannel.write(buffer); // Write buffer data to target file
                buffer.compact(); // Compact buffer for next read
            }

            // Close channels
            sourceChannel.close();
            targetChannel.close();

            System.out.println("Content copied from " + sourceFile + " to " +
targetFile);

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

Task 4: Java Networking Write a simple HTTP client that connects to a URL, sends a request, and displays the response headers and body.

```
package wipro.com.assignment15;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;

public class SimpleHttpClient {

    public static void main(String[] args) {
        String urlString = "https://jsonplaceholder.typicode.com/posts/1"; // Example
URL

        try {
            // Create a URL object
            URL url = new URL(urlString);

            // Open a connection to the URL
            HttpURLConnection connection = (HttpURLConnection) url.openConnection();

            // Set request method to GET
            connection.setRequestMethod("GET");

            // Get response code
            int responseCode = connection.getResponseCode();
            System.out.println("Response Code: " + responseCode);

            // Read and display response headers
            System.out.println("\nResponse Headers:");
            connection.getHeaderFields().forEach((key, value) -> {
                if (key != null) {
                    System.out.println(key + ": " + value);
                }
            });

            // Read and display response body
            System.out.println("\nResponse Body:");
            try (BufferedReader reader = new BufferedReader(new
InputStreamReader(connection.getInputStream()))) {
                String line;
                while ((line = reader.readLine()) != null) {
                    System.out.println(line);
                }
            }

            // Close the connection
            connection.disconnect();

        } catch (IOException e) {
```

```
            e.printStackTrace();
        }
    }
}
```

Output:

```
Response Code: 200

Response Headers:
Server: [cloudflare]
X-Ratelimit-Reset: [1712074275]
Etag: [W/"124-yiKdLzqO5gfBrJFrcdJ8Yq0LGnU"]
Access-Control-Allow-Credentials: [true]
Report-To: [{"group":"heroku-
nel","max_age":3600,"endpoints":[{"url":"https://nel.heroku.com/reports?ts=1712074258
&sid=e11707d5-02a7-43ef-b45e-
2cf4d2036f7d&s=KAwk8PW4ZcZT7%2FT8ZoAYTonLKD58gkxHUD5GEkKRX8k%3D"}]}]
Content-Length: [292]
Age: [10711]
X-Powered-By: [Express]
Content-Type: [application/json; charset=utf-8]
CF-RAY: [898412a58e5da8ec-SIN]
X-Ratelimit-Remaining: [999]
X-Content-Type-Options: [nosniff]
Connection: [keep-alive]
Pragma: [no-cache]
Reporting-Endpoints: [heroku-
nel=https://nel.heroku.com/reports?ts=1712074258&sid=e11707d5-02a7-43ef-b45e-
2cf4d2036f7d&s=KAwk8PW4ZcZT7%2FT8ZoAYTonLKD58gkxHUD5GEkKRX8k%3D]
Date: [Sun, 23 Jun 2024 11:17:38 GMT]
Via: [1.1 vegur]
Accept-Ranges: [bytes]
X-Ratelimit-Limit: [1000]
CF-Cache-Status: [HIT]
Cache-Control: [max-age=43200]
Nel: [{"report_to":"heroku-
nel","max_age":3600,"success_fraction":0.005,"failure_fraction":0.05,"response_header
s":["Via"]}]
Vary: [Origin, Accept-Encoding]
Expires: [-1]
alt-svc: [h3=":443"; ma=86400]

Response Body:
{
  "userId": 1,
  "id": 1,
  "title": "sunt aut facere repellat provident occaecati excepturi optio
reprehenderit",
  "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et
cum\nreprehenderit molestiae ut ut quas totam\nnostrum rerum est autem sunt rem
eveniet architecto"
}
```

Task 5: Java Networking and Serialization Develop a basic TCP client and server application where the client sends a serialized object with 2 numbers and operation to be performed on them to the server, and the server computes the result and sends it back to the client. for eg, we could send 2, 2, "+" which would mean 2 + 2

1. Serializable Object (`CalculationRequest`):

```
package wipro.com.assignment15;

import java.io.Serializable;

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

    private double number1;
    private double number2;
    private String operation;

    public CalculationRequest(double number1, double number2, String operation) {
        this.number1 = number1;
        this.number2 = number2;
        this.operation = operation;
    }

    public double getNumber1() {
        return number1;
    }

    public double getNumber2() {
        return number2;
    }

    public String getOperation() {
        return operation;
    }
}
```

2. TCP Client (`TcpClient`):

```
package wipro.com.assignment15;
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
```

```
public class TcpServer {

    public static void main(String[] args) {
        int serverPort = 12345; // Server port

        try {
            // Create a server socket
            ServerSocket serverSocket = new ServerSocket(serverPort);
            System.out.println("Server is listening on port " + serverPort);

            while (true) {
                // Wait for a client connection
                Socket clientSocket = serverSocket.accept();
                System.out.println("Client connected: " +
clientSocket.getInetAddress().getHostName());

                // Create ObjectInputStream and ObjectOutputStream for receiving and
sending objects
                ObjectInputStream in = new
ObjectInputStream(clientSocket.getInputStream());
                ObjectOutputStream out = new
ObjectOutputStream(clientSocket.getOutputStream());

                // Read CalculationRequest object from client
                CalculationRequest request = (CalculationRequest) in.readObject();
                System.out.println("Request received from client: " +
request.getNumber1() + " " +
                        request.getOperation() + " " + request.getNumber2());

                // Perform calculation based on operation
                double result = 0;
                switch (request.getOperation()) {
                    case "+":
                        result = request.getNumber1() + request.getNumber2();
                        break;
                    case "-":
                        result = request.getNumber1() - request.getNumber2();
                        break;
                    case "*":
                        result = request.getNumber1() * request.getNumber2();
                        break;
                    case "/":
                        result = request.getNumber1() / request.getNumber2();
                        break;
                    default:
                        System.out.println("Unsupported operation: " +
request.getOperation());
                }

                // Send the result back to the client
                out.writeObject(result);

                // Close resources for this client
                in.close();
```

```
                out.close();
                clientSocket.close();
            }

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

To accomplish Task 5, we'll develop a basic TCP client-server application in Java where the client sends a serialized object containing two numbers and an operation to the server. The server will compute the result based on the operation and send it back to the client. We'll use Java's `Socket` for TCP communication and serialization for sending the object over the network.

Steps to Implement:

1. **Create a Serializable Object**:
   - Define a class `CalculationRequest` that implements `Serializable` to hold two numbers and an operation.
2. **TCP Client Implementation**:
   - Client will serialize an instance of `CalculationRequest`, send it to the server, and then deserialize the response to get the result.
3. **TCP Server Implementation**:
   - Server will receive the serialized object, perform the requested operation, serialize the result, and send it back to the client.

Example Implementation:

1. Serializable Object (`CalculationRequest`):

```
java
Copy code
import java.io.Serializable;

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

    private double number1;
    private double number2;
    private String operation;

    public CalculationRequest(double number1, double number2, String
operation) {
        this.number1 = number1;
        this.number2 = number2;
        this.operation = operation;
    }
```

```
    public double getNumber1() {
        return number1;
    }

    public double getNumber2() {
        return number2;
    }

    public String getOperation() {
        return operation;
    }
}
```

2. TCP Client (`TcpClient`):

```
java
Copy code
import java.io.*;
import java.net.Socket;

public class TcpClient {

    public static void main(String[] args) {
        String serverHost = "localhost"; // Server host
        int serverPort = 12345; // Server port

        try {
            // Create a socket to connect to the server
            Socket socket = new Socket(serverHost, serverPort);

            // Create ObjectOutputStream and ObjectInputStream for sending
and receiving objects
            ObjectOutputStream out = new
ObjectOutputStream(socket.getOutputStream());
            ObjectInputStream in = new
ObjectInputStream(socket.getInputStream());

            // Create a CalculationRequest object and send it to the server
            CalculationRequest request = new CalculationRequest(2, 2, "+");
            out.writeObject(request);

            // Read and print the result from the server
            double result = (Double) in.readObject();
            System.out.println("Result received from server: " + result);

            // Close resources
            out.close();
            in.close();
            socket.close();

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

4. TCP Server (`TcpServer`):

```
package wipro.com.assignment15;
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;

public class TcpServer {

    public static void main(String[] args) {
        int serverPort = 12345; // Server port

        try {
            // Create a server socket
            ServerSocket serverSocket = new ServerSocket(serverPort);
            System.out.println("Server is listening on port " + serverPort);

            while (true) {
                // Wait for a client connection
                Socket clientSocket = serverSocket.accept();
                System.out.println("Client connected: " +
clientSocket.getInetAddress().getHostName());

                // Create ObjectInputStream and ObjectOutputStream for receiving and
sending objects
                ObjectInputStream in = new
ObjectInputStream(clientSocket.getInputStream());
                ObjectOutputStream out = new
ObjectOutputStream(clientSocket.getOutputStream());

                // Read CalculationRequest object from client
                CalculationRequest request = (CalculationRequest) in.readObject();
                System.out.println("Request received from client: " +
request.getNumber1() + " " +
                        request.getOperation() + " " + request.getNumber2());

                // Perform calculation based on operation
                double result = 0;
                switch (request.getOperation()) {
                    case "+":
                        result = request.getNumber1() + request.getNumber2();
                        break;
                    case "-":
                        result = request.getNumber1() - request.getNumber2();
                        break;
                    case "*":
                        result = request.getNumber1() * request.getNumber2();
                        break;
                    case "/":
                        result = request.getNumber1() / request.getNumber2();
                        break;
                    default:
```

```
                        System.out.println("Unsupported operation: " +
request.getOperation());
                }

                // Send the result back to the client
                out.writeObject(result);

                // Close resources for this client
                in.close();
                out.close();
                clientSocket.close();
            }

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

Output:

```
Result received from server: 4.0

Server is listening on port 12345
```

Task 6: Java 8 Date and Time API Write a program that calculates the number of days between two dates input by the user.

```
package wipro.com.assignment15;

import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.time.temporal.ChronoUnit;
import java.util.Scanner;

public class DaysBetweenDates {

    public static void main(String[] args) {
        // Create a Scanner object for user input
        Scanner scanner = new Scanner(System.in);

        // Prompt the user to enter the first date
        System.out.print("Enter the first date (yyyy-MM-dd): ");
        String date1Str = scanner.nextLine();

        // Prompt the user to enter the second date
        System.out.print("Enter the second date (yyyy-MM-dd): ");
        String date2Str = scanner.nextLine();
```

```
        // Close the Scanner after input
        scanner.close();

        try {
            // Parse user input strings into LocalDate objects
            LocalDate date1 = LocalDate.parse(date1Str,
DateTimeFormatter.ISO_LOCAL_DATE);
            LocalDate date2 = LocalDate.parse(date2Str,
DateTimeFormatter.ISO_LOCAL_DATE);

            // Calculate the number of days between the two dates
            long daysBetween = ChronoUnit.DAYS.between(date1, date2);

            // Print the result
            System.out.println("Number of days between " + date1 + " and " + date2 +
": " + daysBetween);

        } catch (Exception e) {
            System.out.println("Invalid date format. Please enter dates in yyyy-MM-dd
format.");
        }
    }
}
```

Output:

```
Enter the first date (yyyy-MM-dd): 2020-12-12
```