Day 20

Task 1: Java IO Basics

Write a program that reads a text file and counts the frequency of each word using FileReader and FileWriter.

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

public class WordFrequencyCounter {

    public static void main(String[] args) {
        String inputFileName = "input.txt";  // Replace with your input file name
        String outputFileName = "output.txt"; // Replace with your output file name

        countWordFrequency(inputFileName, outputFileName);
    }

    public static void countWordFrequency(String inputFileName, String outputFileName) {
        // Using HashMap to store word frequencies
        Map<String, Integer> wordCountMap = new HashMap<>();

        try (BufferedReader reader = new BufferedReader(new FileReader(inputFileName));
             PrintWriter writer = new PrintWriter(new FileWriter(outputFileName))) {
```

```
            String line;
            while ((line = reader.readLine()) != null) {
                String[] words = line.split("\\s+");
                for (String word : words) {
                    // Remove punctuation (replace non-word characters with empty string)
                    word = word.replaceAll("[^\\w]", "");

                    if (!word.isEmpty()) {
                        // Convert to lowercase to count words case-insensitively
                        String lowercaseWord = word.toLowerCase();
                        wordCountMap.put(lowercaseWord,
wordCountMap.getOrDefault(lowercaseWord, 0) + 1);
                    }
                }
            }

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

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

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

Main Method:

- In the main() method, specify your input file name (input.txt) and output file name (output.txt), then call the **countWordFrequency()** method.

countWordFrequency Method:

- HashMap: Use a HashMap to store word frequencies, with the word as the key and the frequency as the value.

Reading the Input File and Writing to Output File:

- Use try-with-resources to automatically manage resources (BufferedReader and PrintWriter). This ensures that these resources are properly closed regardless of whether an exception occurs.
- Inside the try block:
- Read the input file line by line using BufferedReader.
- Split each line into words using split("\\s+"), which splits the line on any whitespace.
- Iterate through each word, remove punctuation using replaceAll("[^\\w]", ""), and convert the word to lowercase to count words case-insensitively.
- Update the HashMap with each word and its frequency.
- Write each word and its frequency to the output file using PrintWriter.
- If an IOException occurs, it is caught and printed using e.printStackTrace().

Suppose you have a file input.txt with the following content:

Hello world

Hello again world

Hello World

After running the program, the output file output.txt would contain:

world: 3

again: 1

hello: 3

Task 2: Serialization and Deserialization

Serialize a custom object to a file and then deserialize it back to recover the object state.

```
import java.io.Serializable;
public class Employee implements Serializable {
		private transient int eid;
		private  String ename;  // declare it static and try
		public Employee(int eid, String ename) {
			super();
			this.eid = eid;
			this.ename = ename;
		}
		@Override
		public String toString() {
			return "Employee [eid=" + eid + ", ename=" + ename + "]";
		}
}
```

Student Class:

- Implements Serializable interface, which is required for serialization.
- Includes a serialVersionUID field to provide version control (recommended).
- Contains fields for id, ename, which will not be serialized.

Step 2: Serialize the Object to a File:

```
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;

public class SerializationExample {
```

```
        public static void main(String[] args) throws IOException, FileNotFoundException {
                Employee emp = new Employee(101,"Sanjay");
                FileOutputStream fos = new FileOutputStream("employee.ser");

                ObjectOutputStream oos = new ObjectOutputStream(fos);
                oos.writeObject(emp);

                System.out.println("Employee obj Serialized");
}
}
```

SerializationExample Class:

Creates a SerializationExample object.

Serializes the Student object using ObjectOutputStream and writes it to a file

step 3: Deserialize the Object from the File:

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.ObjectInputStream;

public class DeserializationExample {

        public static void main(String[] args) throws FileNotFoundException, IOException ,
ClassNotFoundException {
                FileInputStream fis = new FileInputStream("employee.ser");
                ObjectInputStream ois = new ObjectInputStream(fis);
                        Object obj = ois.readObject();

                        Employee e1 = (Employee) obj;
```

```
                    System.out.println(e1);
        }
}
```

DeserializationExample Class:

- Deserializes the Student object from the file (SerializationExample.ser) using ObjectInputStream.
- Casts the deserialized object to Student class.

Output:

Deserialized Employee object:

Employee{id=1, name=”Sanjay”

Task 3: New IO (NIO)

Use NIO Channels and Buffers to read content from a file and write to another file.

```
import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.channels.FileChannel;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;

public class NIOFileCopyExample {

   public static void main(String[] args) {
      String inputFile = "input.txt";
      String outputFile = "output.txt";
```

```
    copyFile(inputFile, outputFile);
}

public static void copyFile(String sourceFile, String destFile) {
    // Define the paths for the source and destination files
    Path sourcePath = Paths.get(sourceFile);
    Path destPath = Paths.get(destFile);

    // Create channels for reading and writing
            try (FileChannel sourceChannel = FileChannel.open(sourcePath,
                                    StandardOpenOption.READ);
 FileChannel destChannel = FileChannel.open(destPath, StandardOpenOption.CREATE,
                                    StandardOpenOption.WRITE)) {

        // Allocate a buffer
        ByteBuffer buffer = ByteBuffer.allocate(1024);

        // Read data from the source file into the buffer
        while (sourceChannel.read(buffer) > 0) {
            // Prepare the buffer for writing by flipping it
            buffer.flip();

            // Write data from the buffer to the destination file
            destChannel.write(buffer);

            // Clear the buffer for the next read operation
            buffer.clear();
        }

        System.out.println("File copied from " + sourceFile + " to " + destFile);
```

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

Explanation:

Paths:

Paths.get(sourceFile) and Paths.get(destFile) create Path objects for the source and destination files.

Channels:

- FileChannel.open(sourcePath, StandardOpenOption.READ) opens the source file for reading.
- FileChannel.open(destPath,StandardOpenOption.CREATE, StandardOpenOption.WRITE) opens or creates the destination file for writing.

Buffer:

ByteBuffer.allocate(1024) creates a buffer with a capacity of 1024 bytes. You can adjust the buffer size as needed.

Read and Write Loop:

- **sourceChannel.read(buffer)** > 0 reads data from the source file into the buffer. The read method returns the number of bytes read, or -1 if the end of the file is reached.
- **buffer.flip()** prepares the buffer for writing by setting the limit to the current position and then setting the position to zero.
- **destChannel.write**(buffer) writes the buffer's content to the destination file.
- **buffer.clear()** resets the buffer for the next read operation.

try-with-resources:

The try-with-resources statement ensures that the FileChannel objects are closed automatically after the operations are completed, preventing resource leaks.

input.txt Content:

Hello, this is a test file.

This file will be copied using NIO Channels and Buffers.

output.txt Content:

Hello, this is a test file.

This file will be copied using NIO Channels and Buffers.

Task 4: Java Networking

Write a simple HTTP client that connects to a URL, sends a request, and displays the response headers and body.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.List;
import java.util.Map;

public class SimpleHttpClient {

    public static void main(String[] args) {
        String urlString = "http://www.example.com";

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

            // Open a connection to the URL
```

```
HttpURLConnection connection = (HttpURLConnection) url.openConnection();

// Set the request method (GET, POST, etc.)
connection.setRequestMethod("GET");

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

// Get response headers
Map<String, List<String>> headers = connection.getHeaderFields();
System.out.println("Response Headers:");
for (Map.Entry<String, List<String>> entry : headers.entrySet()) {
   System.out.println(entry.getKey() + ": " + entry.getValue());
}

// Get response body
                    BufferedReader in = new BufferedReader(new
                InputStreamReader(connection.getInputStream()));
String inputLine;
StringBuilder responseBody = new StringBuilder();
while ((inputLine = in.readLine()) != null) {
   responseBody.append(inputLine).append("\n");
}
in.close();

// Print response body
System.out.println("Response Body:");
System.out.println(responseBody.toString());
```

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

Explanation:

Create URL Object:

- Create a URL object using the specified URL string.

Open a Connection:

- Open a connection to the URL using HttpURLConnection.

Set Request Method:

- Set the request method (e.g., "GET", "POST", etc.). In this example, we use "GET".

Get Response Code:

- Retrieve and print the HTTP response code

Get Response Headers:

- Retrieve and print the response headers using getHeaderFields() which returns a map of header fields.

Get Response Body:

- Read the response body using a BufferedReader wrapped around the InputStream from the connection.
- Append each line of the response body to a StringBuilder and print it out**.**

Exception Handling:

- Catch and print any IOException that might occur during the process.

Output:

Response Code**: 200**

Response Headers:

null: **[HTTP/1.1 200 OK]**

Content-Encoding: **[gzip]**

Accept-Ranges**: [bytes]**

Cache-Control**: [max-age=604800]**

Content-Type**: [text/html; charset=UTF-8]**

Date: [M**on, 09 Jun 2024 00:00:00 GMT]**

Etag**: ["3147526947+gzip"]**

Expires**: [Mon, 16 Jun 2024 00:00:00 GMT]**

Last-Modified: **[Thu, 17 Oct 2019 07:18:26 GMT]**

Server: **[ECS (nyb/1D13)]**

Vary**: [Accept-Encoding]**

X-Cache**: [HIT]**

Content-Length: **[648]**

Response Body:

```
<!doctype html>
<html>
<head>
   <title>Example Domain</title>
   ...
</head>
<body>
   <div>
      <h1>Example Domain</h1>
      ...
```

```
    </div>
</body>
</html>
```

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

Step-by-Step Implementation

Define a Serializable Object for the Operation Request

Develop the TCP Server

Develop the TCP Client

Define a Serializable Object:

```
import java.io.Serializable;

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

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

    public OperationRequest(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;
    }
}
```

Develop the TCP Server:

```
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;

public class TCPServer {
    public static void main(String[] args) {
        int port = 12345;


        try (ServerSocket serverSocket = new ServerSocket(port)) {
            System.out.println("Server is listening on port " + port);

            while (true) {
                try (Socket socket = serverSocket.accept();
```

```
            ObjectInputStream ois = new ObjectInputStream(socket.getInputStream());
      ObjectOutputStream oos = new ObjectOutputStream(socket.getOutputStream())) {


            OperationRequest request = (OperationRequest) ois.readObject();


            double result = computeResult(request);


            oos.writeDouble(result);
            oos.flush();
         } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
         }
      }
   } catch (IOException e) {
      e.printStackTrace();
   }
}

private static double computeResult(OperationRequest request) {
   double number1 = request.getNumber1();
   double number2 = request.getNumber2();
   String operation = request.getOperation();

   switch (operation) {
      case "+":
         return number1 + number2;
```

```
        case "-":
            return number1 - number2;
        case "*":
            return number1 * number2;
        case "/":
            if (number2 != 0) {
                return number1 / number2;
            } else {
                throw new IllegalArgumentException("Division by zero is not allowed.");
            }
        default:
            throw new UnsupportedOperationException("Unsupported operation: " + operation);
    }
  }
}
```

Develop the TCP Client:

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

public class TCPClient {
  public static void main(String[] args) {
    String hostname = "localhost";
    int port = 12345;

    // Create an operation request (example: 2 + 2)
    OperationRequest request = new OperationRequest(2, 2, "+");

    try (Socket socket = new Socket(hostname, port);
```

```
            ObjectOutputStream oos = new ObjectOutputStream(socket.getOutputStream());
            ObjectInputStream ois = new ObjectInputStream(socket.getInputStream())) {

            // Send the operation request to the server
            oos.writeObject(request);
            oos.flush();

            // Receive the result from the server
            double result = ois.readDouble();

            // Display the result
            System.out.println("Result: " + result);

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

Explanation:

OperationRequest Class:

- A simple serializable class to hold two numbers and an operation.

TCPServer Class:

- Listens for incoming connections on a specified port.
- Reads the OperationRequest object from the client.
- Computes the result based on the operation.
- Sends the result back to the client

TCPClient Class:

Connects to the server.

Sends an OperationRequest object to the server.

Receives the result from the server.

Prints the result.

Running the Example:

Start the Server:

Run the TCPServer class. It will start listening on port 12345.

Run the Client:

Run the TCPClient class. It will send a request to the server and print the result.

Output:

Result: 4.0

Task 6: Java 8 Date and Time API

Write a program that calculates the number of days between two dates input by the user.

Step-by-Step Implementation

- Import necessary classes from java.time package.
- Get the two dates from user input.
- Parse the dates and calculate the difference in days.
- Display the result

```
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) {
```

```
        Scanner scanner = new Scanner(System.in);

        DateTimeFormatter dateFormatter = DateTimeFormatter.ofPattern("yyyy-MM-
    dd");

        System.out.print("Enter the first date (yyyy-MM-dd): ");
        String firstDateString = scanner.nextLine();
        LocalDate firstDate = LocalDate.parse(firstDateString, dateFormatter);

        System.out.print("Enter the second date (yyyy-MM-dd): ");
        String secondDateString = scanner.nextLine();
        LocalDate secondDate = LocalDate.parse(secondDateString, dateFormatter);

        long daysBetween = ChronoUnit.DAYS.between(firstDate, secondDate);
        System.out.println("Number of days between " + firstDate + " and " +
    secondDate + ": " + daysBetween);
      }
    }
```

Output:

```
Enter the first date (yyyy-MM-dd): 2022-01-01
Enter the second date (yyyy-MM-dd): 2022-06-01
Number of days between 2022-01-01 and 2022-06-01: 151
```

Task 7: Timezone

Create a timezone converter that takes a time in one timezone and converts it to another timezone.

```
import java.time.LocalDateTime;

import java.time.ZoneId;

import java.time.ZonedDateTime;

import java.time.format.DateTimeFormatter;

import java.util.Scanner;


public class TimezoneConverter {


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

```
        Scanner scanner = new Scanner(System.in);


        DateTimeFormatter dateTimeFormatter = DateTimeFormatter.ofPattern("yyyy-MM-dd
HH:mm");


        System.out.print("Enter the time (yyyy-MM-dd HH:mm): ");
        String timeString = scanner.nextLine();
        LocalDateTime localDateTime = LocalDateTime.parse(timeString, dateTimeFormatter);



        System.out.print("Enter the source timezone (e.g., America/New_York): ");
        String sourceTimezone = scanner.nextLine();
        ZoneId sourceZoneId = ZoneId.of(sourceTimezone);

        System.out.print("Enter the destination timezone (e.g., Europe/London): ");
        String destinationTimezone = scanner.nextLine();
        ZoneId destinationZoneId = ZoneId.of(destinationTimezone);

        ZonedDateTime sourceZonedDateTime = localDateTime.atZone(sourceZoneId);


                ZonedDateTime destinationZonedDateTime =
            sourceZonedDateTime.withZoneSameInstant(destinationZoneId);


        String convertedTimeString = destinationZonedDateTime.format(dateTimeFormatter);
System.out.println("Converted time in " + destinationTimezone + ": " +
convertedTimeString);
}
}
```

Explanation:

Imports:

- LocalDateTime, ZoneId, ZonedDateTime, and DateTimeFormatter from java.time package for handling dates, times, and time zones.
- Scanner from java.util package for reading user input.

DateTimeFormatter:

- Defines the format of the input and output date-time strings (yyyy-MM-dd HH:mm).

User Input:

- Reads the input time, source timezone, and destination timezone from the user.

LocalDateTime and ZonedDateTime:

- Parses the input time into a LocalDateTime object.
- Converts this local date-time to a ZonedDateTime object in the source timezone.
- Converts the ZonedDateTime object to the destination timezone using withZoneSameInstant method.

Input and Output:

Enter the time (yyyy-MM-dd HH:mm): 2024-06-10 15:00

Enter the source timezone (e.g., America/New_York): Asia/Tokyo

Enter the destination timezone (e.g., Europe/London): America/Los_Angeles

Converted time in America/Los_Angeles: 2024-06-10 23:00