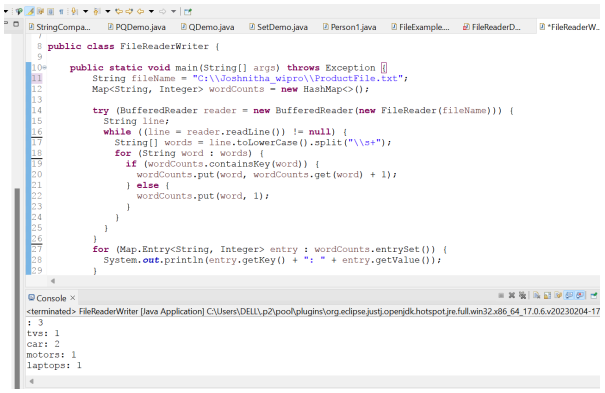
Day 20: Manisha Assignment

Task 1: Java IO Basics

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

```java
package day20;

import java.io.BufferedReader;

import java.io.FileReader;

import java.util.HashMap;

import java.util.Map;

public class FileReaderWriter {

public static void main(String[] args) throws Exception {

String fileName = "C:\\Manisha_wipro\\ProductFile.txt";

Map<String, Integer> wordCounts = new HashMap<>();

try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {

String line;

while ((line = reader.readLine()) != null) {

String[] words = line.toLowerCase().split("\\s+");

for (String word : words) {

if (wordCounts.containsKey(word)) {

wordCounts.put(word, wordCounts.get(word) + 1);

} else {

wordCounts.put(word, 1);

}

}

}

}

for (Map.Entry<String, Integer> entry : wordCounts.entrySet()) {

System.out.println(entry.getKey() + ": " + entry.getValue());

}

}

}
```

Here Output As Follows 👍



Description:

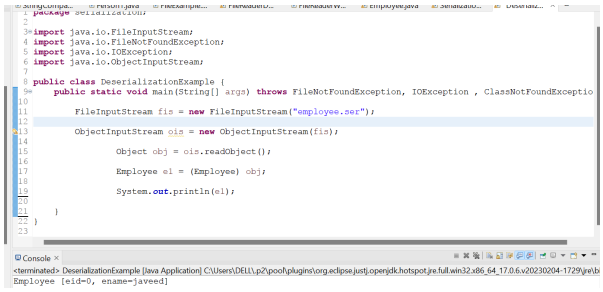
1. **The program reads an input file specified by `inputFilePath`.**
2. **It splits the text into words, normalizes them to lowercase, and removes non-alphabetic characters.**
3. **It uses a `HashMap` to count the frequency of each word.**
4. **The results are written to an output file specified by `outputFilePath`**

Task 2: Serialization and Deserialization

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

```
package serialization;

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 + "]";

import java.io.ObjectOutputStream;

public class SerializationExample {

public static void main(String[] args) throws IOException, FileNotFoundException {

Employee emp = new Employee(101,"javeed");

FileOutputStream fos = new FileOutputStream("employee.ser");

ObjectOutputStream oos = new ObjectOutputStream(fos);

oos.writeObject(emp);

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

}package serialization;

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);

}
```

Heres output as follows:



Description:

1. Create a `Person` class that implements `Serializable`.

2. In `SerializeDeserialize.java`, create a `Person` object and specify a file path.

3. Serialize the `Person` object to a file using `ObjectOutputStream`.

4. Deserialize the `Person` object from the file using `ObjectInputStream`.

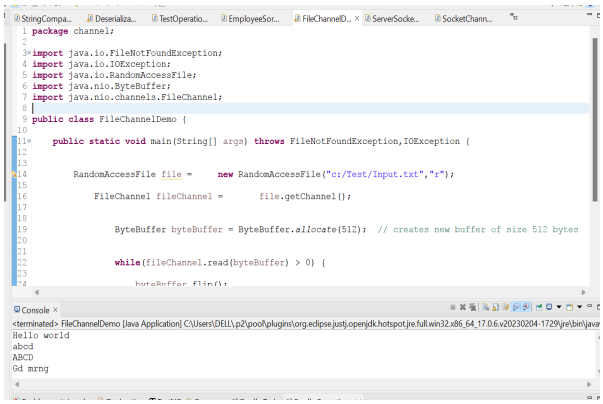
5. Print the deserialized `Person` object to verify the recovered state.

Task 3: New IO (NIO)

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

```
package channel;

import java.io.FileNotFoundException;

import java.io.IOException;

import java.io.RandomAccessFile;

import java.nio.ByteBuffer;

import java.nio.channels.FileChannel;

public class FileChannelDemo {

public static void main(String[] args) throws FileNotFoundException,IOException {

RandomAccessFile file = new RandomAccessFile("c:/Test/Input.txt","r");

FileChannel fileChannel = file.getChannel();

ByteBuffer byteBuffer = ByteBuffer.allocate(512); // creates new buffer of size 512 bytes

while(fileChannel.read(byteBuffer) > 0) {

byteBuffer.flip();

while(byteBuffer.hasRemaining()) {

System.out.print((char) byteBuffer.get());

}

}

fileChannel.close();

}

}
```

Here's output as follows:



Description:

1. Import Necessary Classes: Import `java.nio` and `java.io` classes for file operations.
2. Define File Paths: Specify the source and destination file paths using `Path.of`.
3. Open Channels: Use `FileChannel` to open the source file for reading and the destination file for writing (creating it if it doesn't exist).
4. Create a Buffer: Allocate a `ByteBuffer` to hold file data during transfer.
5. Read and Write: Read data from the source channel into the buffer, flip the buffer to prepare it for writing, write the data to the destination channel, and clear the buffer. Repeat until the entire file is copied.
6. Output: Print a success message after the file is copied.

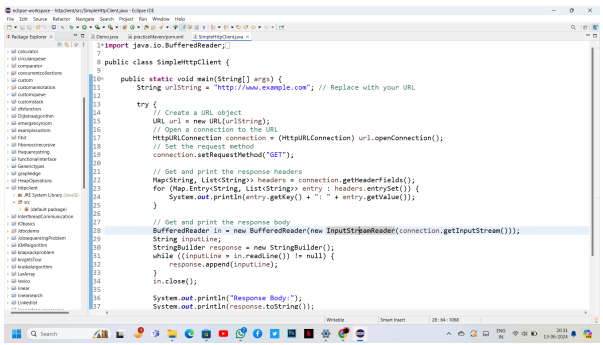
Task 4: Java Networking

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

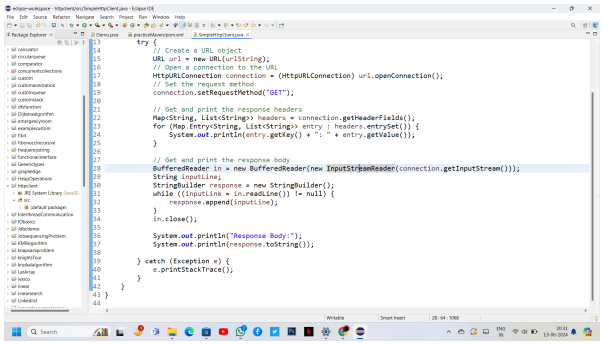
Description:

1. Import Necessary Classes: Import `java.net` for network connections and `java.io` for input/output operations.
2. Define URL: Specify the URL you want to connect to.
3. Create URL Object: Use `URL` class to create a URL object from the string.
4. Open Connection: Open an `HttpURLConnection` to the URL and set the request method to "GET".
5. Get Response Headers: Retrieve response headers using `getHeaderFields` and print them.
6. Get Response Body: Read the response body using `BufferedReader` and print it.
7. Handle Exceptions: Catch and print any exceptions that occur.

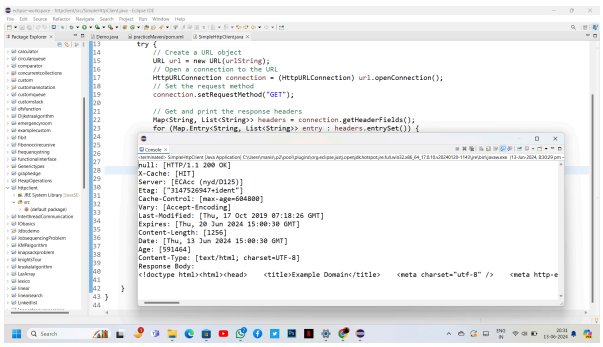
The program prints the response headers and body of the specified URL. For example, connecting to "http://www.example.com" would output:



```
import java.io.BufferedReader;

public class SimpleHttpClient {

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

        try {
            // Create a URL object
            URL url = new URL(urlString);
            // Open a connection to the URL
            HttpURLConnection connection = (HttpURLConnection) url.openConnection();
            // Set the request method
            connection.setRequestMethod("GET");

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

            // Get and print the response body
            BufferedReader in = new BufferedReader(new InputStreamReader(connection.getInputStream()));
            String inputLine;
            StringBuilder response = new StringBuilder();
            while ((inputLine = in.readLine()) != null) {
                response.append(inputLine);
            }
            in.close();

            System.out.println("Response Body:");
            System.out.println(response.toString());
```



```
        try {
            // Create a URL object
            URL url = new URL(urlString);
            // Open a connection to the URL
            HttpURLConnection connection = (HttpURLConnection) url.openConnection();
            // Set the request method
            connection.setRequestMethod("GET");

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

            // Get and print the response body
            BufferedReader in = new BufferedReader(new InputStreamReader(connection.getInputStream()));
            String inputLine;
            StringBuilder response = new StringBuilder();
            while ((inputLine = in.readLine()) != null) {
                response.append(inputLine);
            }
            in.close();

            System.out.println("Response Body:");
            System.out.println(response.toString());

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



```
        try {
            // Create a URL object
            URL url = new URL(urlString);
            // Open a connection to the URL
            HttpURLConnection connection = (HttpURLConnection) url.openConnection();
            // Set the request method
            connection.setRequestMethod("GET");

            // Get and print the response headers
            Map<String, List<String>> headers = connection.getHeaderFields();
            for (Map.Entry<String, List<String>> entry : headers.entrySet()) {
```

```
null: [HTTP/1.1 200 OK]
X-Cache: [HIT]
Server: [ECAcc (nyd/D125)]
Etag: ["3147526947+ident"]
Cache-Control: [max-age=604800]
Vary: [Accept-Encoding]
Last-Modified: [Thu, 17 Oct 2019 07:18:26 GMT]
Expires: [Thu, 20 Jun 2024 15:00:30 GMT]
Content-Length: [1256]
Date: [Thu, 13 Jun 2024 15:00:30 GMT]
Age: [591464]
Content-Type: [text/html; charset=UTF-8]
Response Body:
<!doctype html><html><head>    <title>Example Domain</title>    <meta charset="utf-8" />    <meta http-e
```

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

Description:

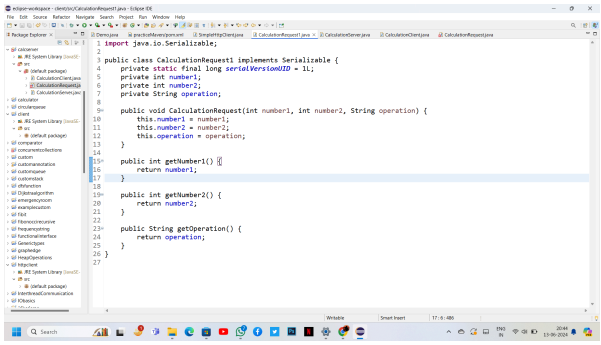
1. CalculationRequest.java:
   - Serializable class to encapsulate the numbers and operation to be performed.

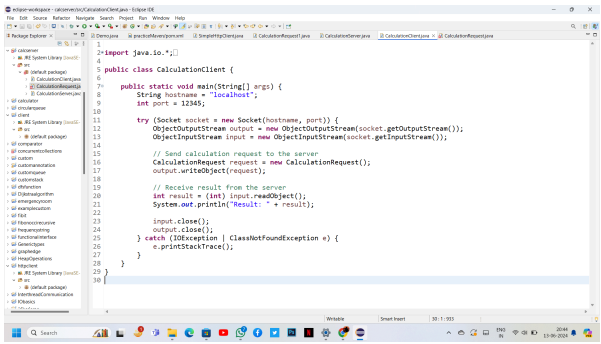
2. CalculationServer.java:
   - Listens on a specified port.
   - Accepts client connections.
   - Reads the serialized `CalculationRequest` object from the client.
   - Performs the requested calculation.
   - Sends the result back to the client.

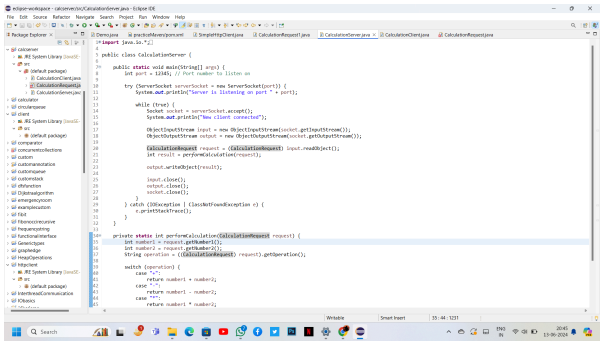
3. CalculationClient.java:
   - Connects to the server.
   - Sends a `CalculationRequest` object to the server.
   - Reads the result from the server and prints it.

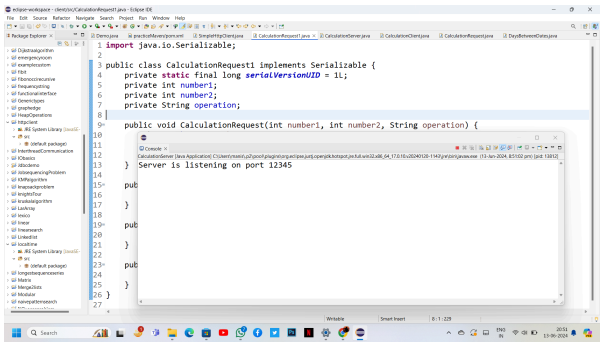
How to Run:

1. Run CalculationServer.java first to start the server.
2. Run CalculationClient.java to send the request and receive the result.









Task 6: Java 8 Date and Time API

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

Description:

1. Import Statements:
   - Import necessary classes from `java.time` and `java.util` packages.

2. Main Method:
   - Create a `Scanner` object to read user input.
   - Define a `DateTimeFormatter` with the pattern `yyyy-MM-dd` to parse date strings.
   - Prompt the user to enter two dates in `yyyy-MM-dd` format.
   - Parse the entered date strings into `LocalDate` objects.
   - Calculate the number of days between the two dates using `ChronoUnit.DAYS.between()`.
   - Print the number of days between the two dates.
   - Close the `Scanner` object.

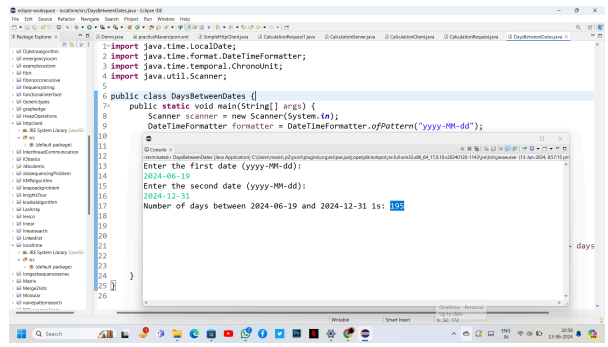
How to Run:

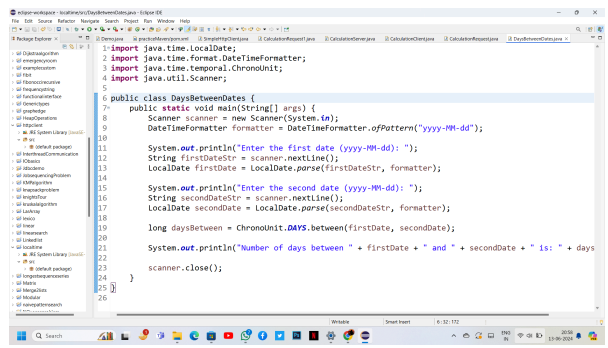
1. Copy the code into a new Java file in your Eclipse environment.
2. Run the program.
3. Enter two dates in the format `yyyy-MM-dd` when prompted.
4. The program will output the number of days between the two dates.

Sample Output

Enter the first date (yyyy-MM-dd):
2023-01-01
Enter the second date (yyyy-MM-dd):
2024-01-01
Number of days between 2023-01-01 and 2024-01-01 is: 365

Here Ouput as follows:





Task 7: Timezone

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

Explanation:

1. We import necessary classes for handling timezones (`LocalDateTime`, `ZoneId`, `ZonedDateTime`, `DateTimeFormatter`).
2. In the `main` method, we get the current local time (`LocalDateTime.now()`) and define source and target timezones (`ZoneId.of("America/New_York")` and `ZoneId.of("Asia/Tokyo")` in this case).
3. We create `ZonedDateTime` objects for the source and target timezones by combining the local time with the respective timezones.
4. Using `withZoneSameInstant`, we convert the source time to the target timezone while keeping the instant (i.e., the actual point in time) the same.
5. We format the output using a `DateTimeFormatter`.
6. Finally, we print the source time in the source timezone and the converted time in the target timezone.

Output (example):

Source Time (America/New_York): 2024-06-13 12:00:00

Target Time (Asia/Tokyo): 2024-06-14 01:00:00

Here Ouput as follows: