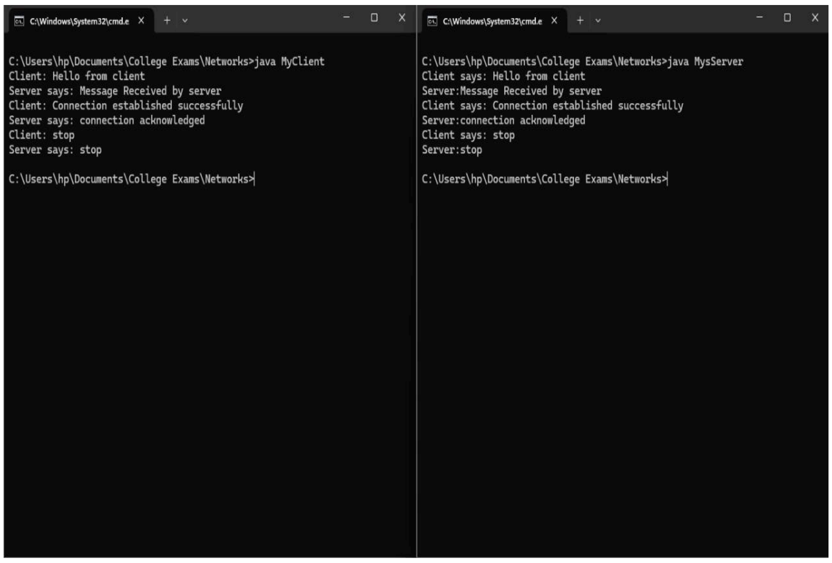
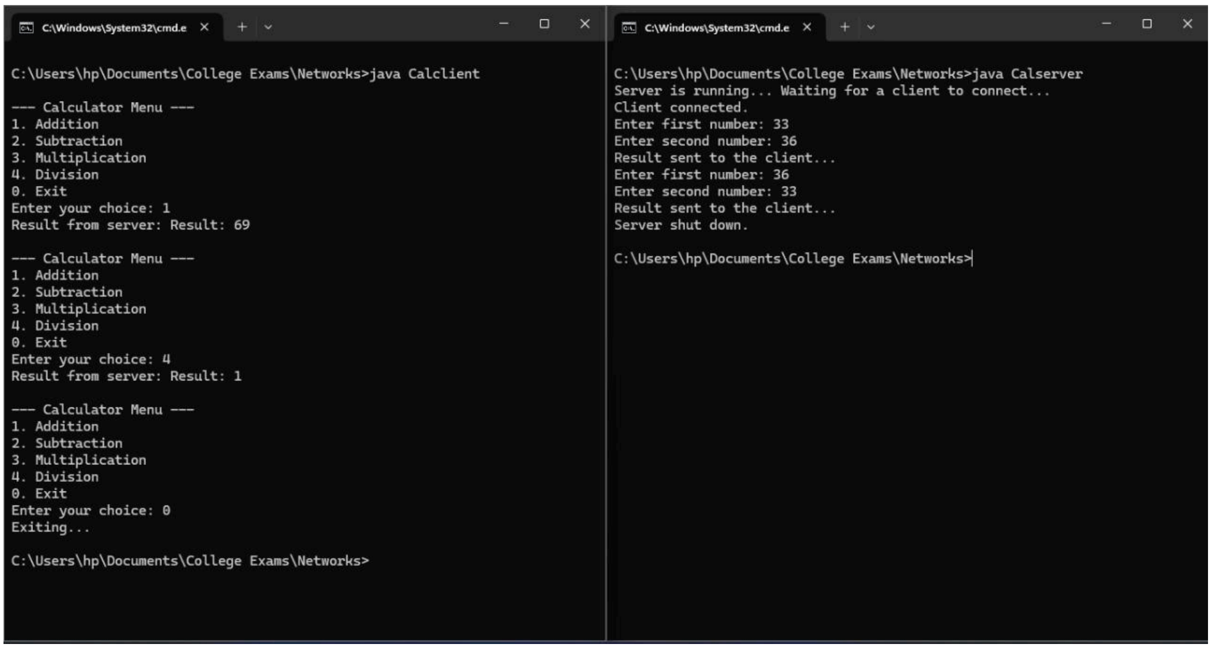


```
C:\Users\hp\Documents\College Exams\Networks>java MyClient
Client: Hello from client
Server says: Message Received by server
Client: Connection established successfully
Server says: connection acknowledged
Client: stop
Server says: stop

C:\Users\hp\Documents\College Exams\Networks>
```

```
C:\Users\hp\Documents\College Exams\Networks>java MysServer
Client says: Hello from client
Server:Message Received by server
Client says: Connection established successfully
Server:connection acknowledged
Client says: stop
Server:stop

C:\Users\hp\Documents\College Exams\Networks>
```

Conclusion

In this program, we developed a simple server-side socket application using Java. The server successfully established a connection with a client, received messages, and responded interactively until a termination command ("stop") was received. This implementation demonstrated the use of key Java networking classes such as ServerSocket, Socket, DataInputStream, and DataOutputStream. Overall, the program highlights the fundamentals of TCP-based communication and provides a solid foundation for building more complex networked applications.

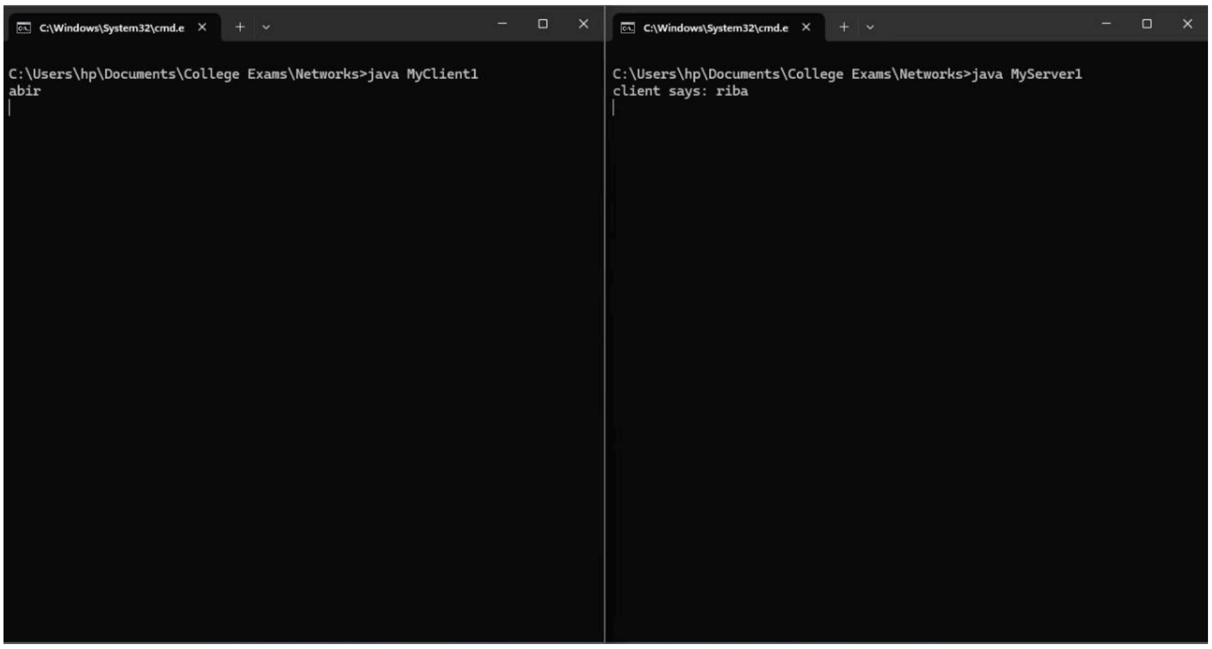


```
C:\Users\hp\Documents\College Exams\Networks>java Calclient

--- Calculator Menu ---
1. Addition
2. Subtraction
3. Multiplication
4. Division
0. Exit
Enter your choice: 1
Result from server: Result: 69

--- Calculator Menu ---
1. Addition
2. Subtraction
3. Multiplication
4. Division
0. Exit
Enter your choice: 4
Result from server: Result: 1

--- Calculator Menu ---
1. Addition
2. Subtraction
3. Multiplication
4. Division
0. Exit
Enter your choice: 0
Exiting...

C:\Users\hp\Documents\College Exams\Networks>
```

```
C:\Users\hp\Documents\College Exams\Networks>java Calserver
Server is running... Waiting for a client to connect...
Client connected.
Enter first number: 33
Enter second number: 36
Result sent to the client...
Enter first number: 36
Enter second number: 33
Result sent to the client...
Server shut down.

C:\Users\hp\Documents\College Exams\Networks>
```

Conclusion :

In this program, we developed a simple client-server calculator application using Java socket programming. The client connected to the server over a TCP socket, sent an operation request, and received the result after the server performed the computation based on server-side user input. This program effectively demonstrated the use of essential Java networking classes such as Socket, ServerSocket, DataInputStream, and DataOutputStream. It also showed how continuous communication can be maintained between a client and server in a loop until termination. Overall, this project reinforces the core concepts of TCP-based client-server interaction and lays a strong groundwork for more advanced networked applications.

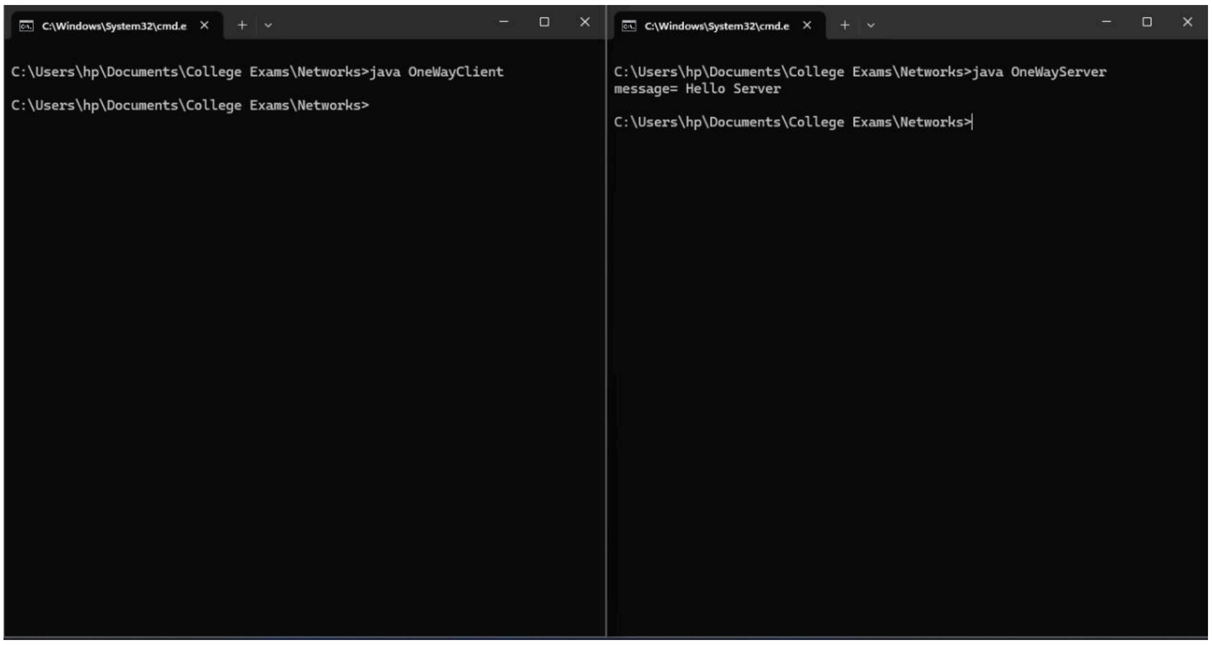


```
C:\Users\hp\Documents\College Exams\Networks>java MyClient1
abir
```

```
C:\Users\hp\Documents\College Exams\Networks>java MyServer1
client says: riba
```

Conclusion

In this program, we developed a simple client-side socket application using Java. The client successfully connected to the server over a TCP socket, took user input from the console, reversed the string locally, and sent it to the server. It then received a response from the server and displayed it to the user. This program demonstrated the use of key Java networking and I/O classes such as Socket, DataInputStream, DataOutputStream, and BufferedReader. Overall, the application showcases the basics of client-server communication and forms a solid foundation for building more interactive and advanced network-based systems.

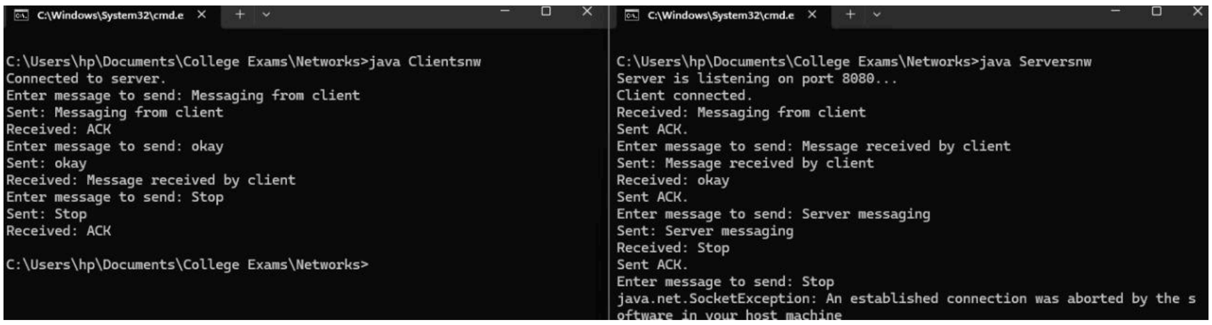


```
C:\Users\hp\Documents\College Exams\Networks>java OneWayClient

C:\Users\hp\Documents\College Exams\Networks>
```

```
C:\Users\hp\Documents\College Exams\Networks>java OneWayServer
message= Hello Server

C:\Users\hp\Documents\College Exams\Networks>
```

Conclusion:

The given Java program successfully demonstrates a basic client-server communication model using TCP sockets. The server listens for incoming connections on a specified port and receives a UTF-encoded message from the client, while the client establishes a connection to the server and sends a message. This program highlights the use of essential networking classes in Java, such as ServerSocket, Socket, DataInputStream, and DataOutputStream. It provides a clear understanding of how reliable, connection-oriented communication can be established between two programs running on the same machine or over a network.

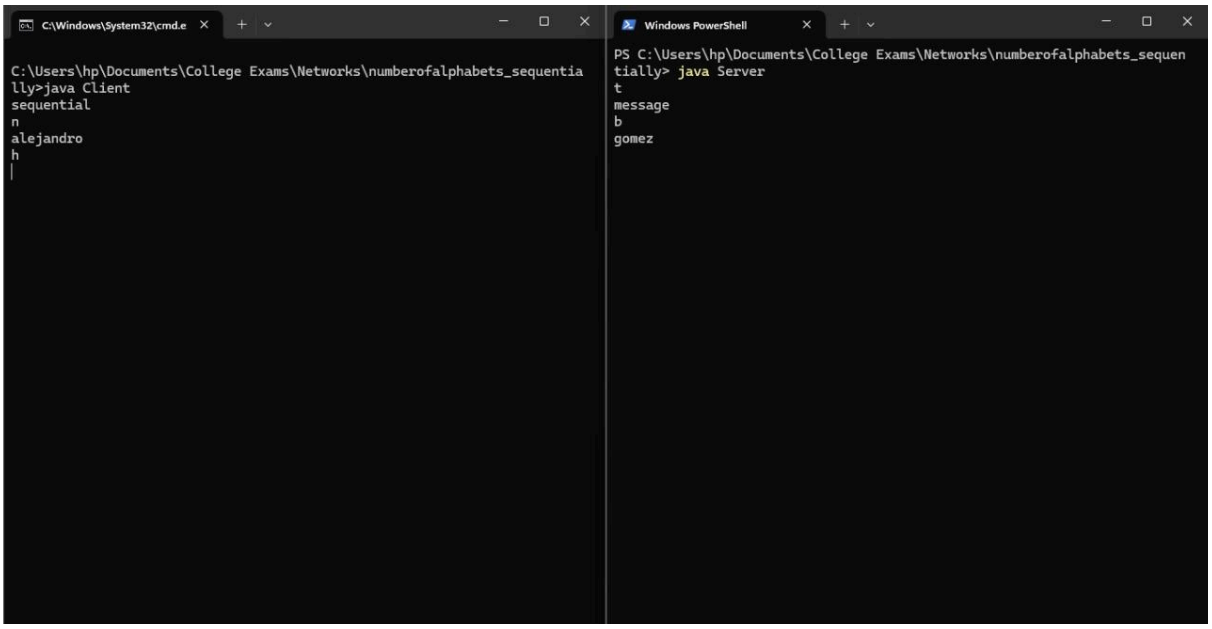


```
C:\Users\hp\Documents\College Exams\Networks>java Clientsnw
Connected to server.
Enter message to send: Messaging from client
Sent: Messaging from client
Received: ACK
Enter message to send: okay
Sent: okay
Received: Message received by client
Enter message to send: Stop
Sent: Stop
Received: ACK

C:\Users\hp\Documents\College Exams\Networks>
```

```
C:\Users\hp\Documents\College Exams\Networks>java Serversnw
Server is listening on port 8080...
Client connected.
Received: Messaging from client
Sent ACK.
Enter message to send: Message received by client
Sent: Message received by client
Received: okay
Sent ACK.
Enter message to send: Server messaging
Sent: Server messaging
Received: Stop
Sent ACK.
Enter message to send: Stop
java.net.SocketException: An established connection was aborted by the s
oftware in your host machine
```

Conclusion:

The given client-server Java program successfully demonstrates a two-way communication system using TCP sockets. The server listens on port 8080 and waits for a client to connect. Once the connection is established, both the server and client can continuously exchange messages until either sends the termination command "stop". The server responds to each client message with an acknowledgment ("ACK") and also sends its own messages interactively. This implementation effectively shows how to manage input and output streams for real-time communication using Socket, ServerSocket, DataInputStream, and DataOutputStream. Overall, the program illustrates the fundamentals of synchronous socket-based communication and is a practical example of how client-server systems operate over a network in Java.



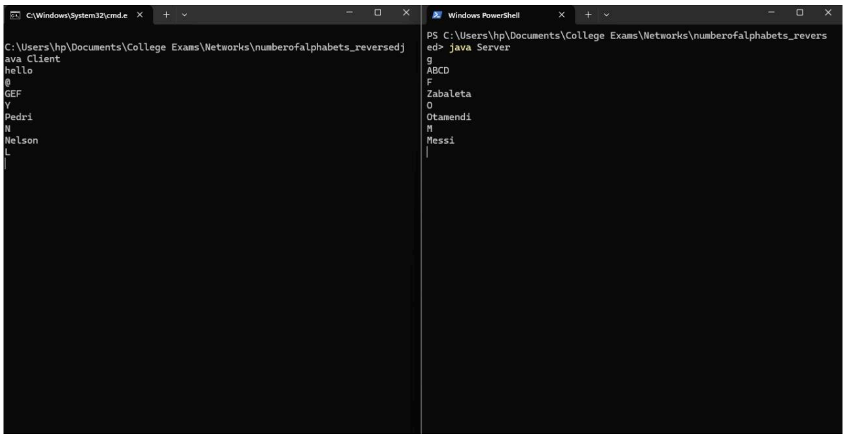
```
C:\Users\hp\Documents\College Exams\Networks\numberofalphabets_sequentia
lly>java Client
sequential
n
alejandro
h
```

```
PS C:\Users\hp\Documents\College Exams\Networks\numberofalphabets_sequen
tially> java Server
t
message
b
gomez
```

Conclusion:

The Serverabcd and Clientabcd programs demonstrate a simple implementation of socket-based communication between a server and a client in Java. The server listens for incoming connections on a specified port, accepts messages from the client, processes them by incrementing the first character, and responds based on user input. Meanwhile, the client sends messages, receives the server's response, and also performs a character manipulation. This bidirectional data exchange continues until either party sends the message "Stop", signalling termination. The program effectively showcases basic concepts of Java networking, such as socket creation, stream handling, message passing, and graceful connection closure.

```
DataInputStream din = new DataInputStream(s.getInputStream());
DataOutputStream dout = new DataOutputStream(s.getOutputStream());
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
String str = "", str2 = "";
while(!str.equals("Stop")){
str = br.readLine();
dout.writeUTF(str);
dout.flush();
str2 = din.readUTF();
if(str2.equals("Stop"))
break;
char ch = str2.charAt(0);
ch -=1;
System.out.println(ch);
}
dout.flush();
dout.close();
s.close();
}
}
```

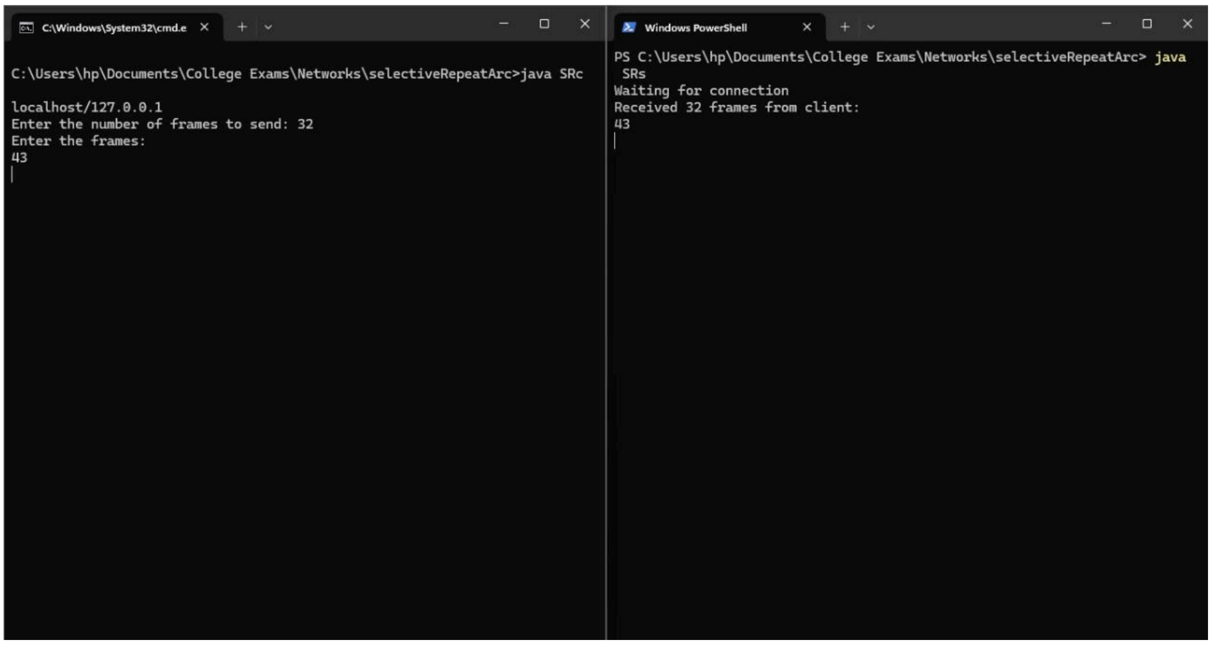


```
C:\Users\hp\Documents\College Exams\Networks\numberofalphabets_reversedj
ava Client
hello
@
GEF
Y
Pedri
N
Nelson
L
```

```
PS C:\Users\hp\Documents\College Exams\Networks\numberofalphabets_revers
ed> java Server
g
ABCD
F
Zabaleta
O
Otamendi
M
Messi
```

Conclusion:

The Serverabcdrev and Clientabcdrev programs demonstrate a basic TCP-based client-server communication system using Java sockets. In this interaction, the client sends a string input to the server, which processes the first character of the received string by decrementing its ASCII value by one and displays it. The server then waits for user input on its side, sends this response back to the client, and the client performs a similar character manipulation and displays the result. This exchange continues until either side sends the termination keyword "Stop", after which both the client and server close their respective connections and streams gracefully. The program effectively illustrates socket creation, input/output stream handling, real-time message exchange, and termination control, making it a useful example for understanding basic network communication in Java.



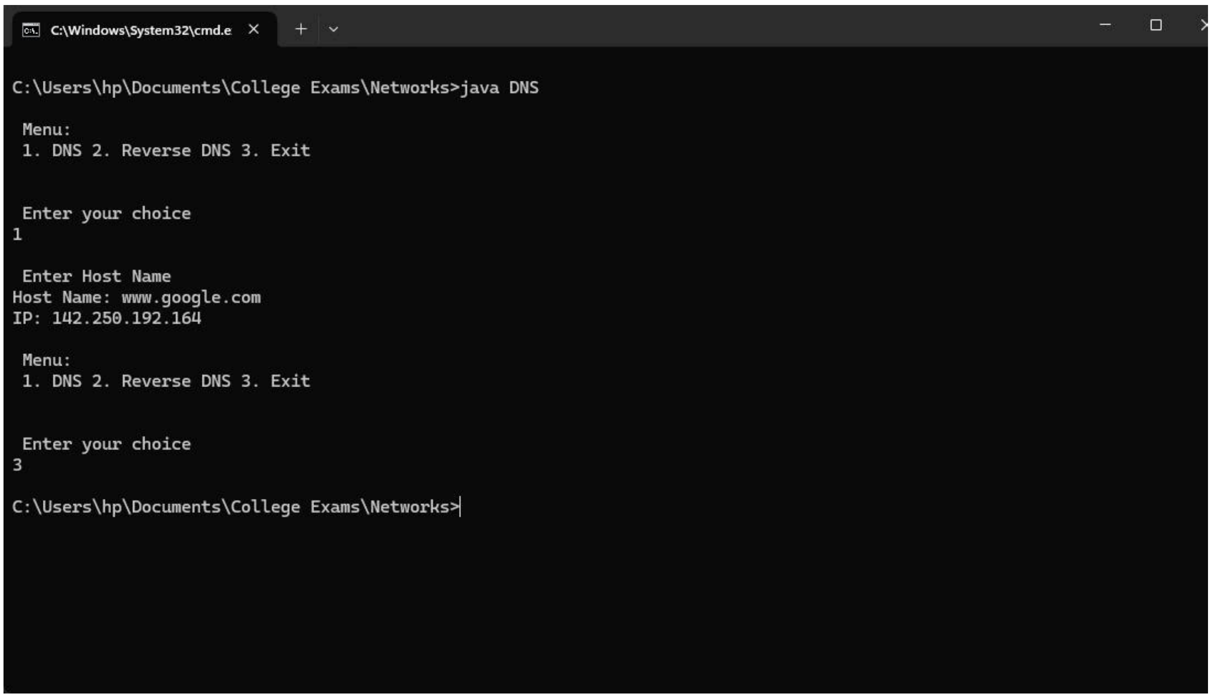
```
C:\Users\hp\Documents\College Exams\Networks\selectiveRepeatArc>java SRc

localhost/127.0.0.1
Enter the number of frames to send: 32
Enter the frames:
43
```

```
PS C:\Users\hp\Documents\College Exams\Networks\selectiveRepeatArc> java
 SRs
Waiting for connection
Received 32 frames from client:
43
```

Conclusion:

The SRs and SRc programs effectively simulate the Selective Repeat ARQ protocol using Java socket programming to ensure reliable data transmission over a network. The client (SRc) sends a series of data frames to the server (SRs), which checks for missing or corrupted frames indicated by a -1. If such a frame is detected, the server requests retransmission of only that specific frame, demonstrating the selective retransmission mechanism that improves efficiency compared to other ARQ methods like Go-Back-N. This project provides a clear understanding of how error detection and correction can be implemented in network communication, highlighting key concepts such as frame handling, socket connection, stream management, and interaction between client and server in a controlled and efficient manner.

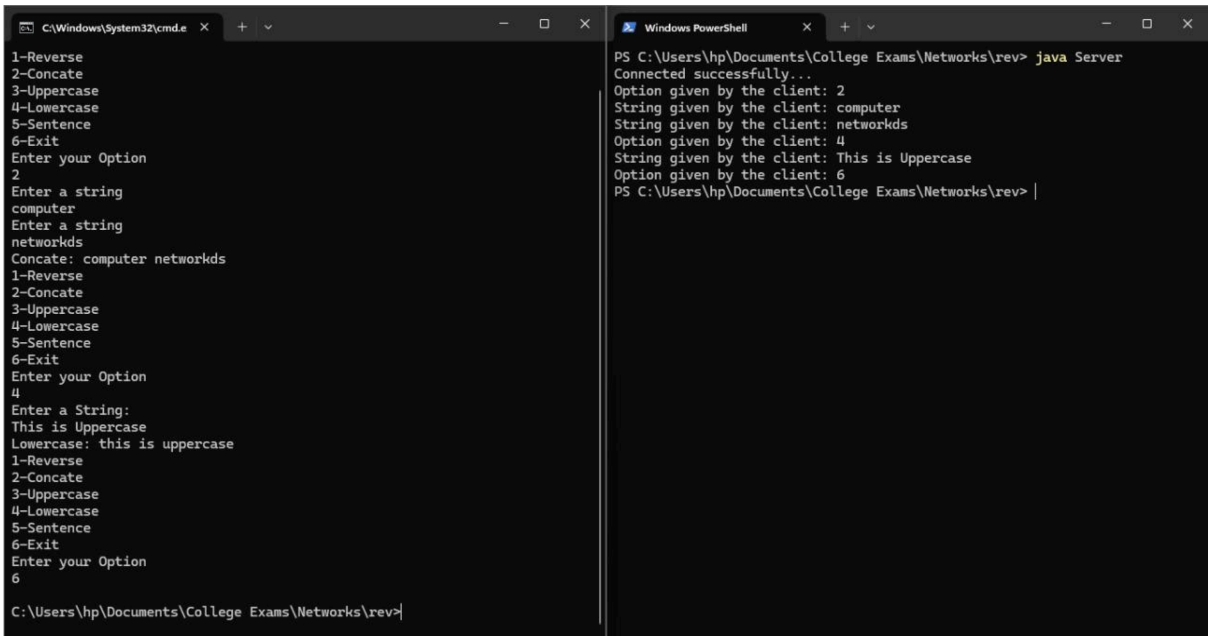
```
{
System.out.println("\n Enter IP address");
String ipstr = in.readLine();
InetAddress ia = InetAddress.getByName(ipstr);
System.out.println("IP: "+ipstr);
System.out.println("Host Name: " +ia.getHostName());
}
catch(IOException ioe)
{
ioe.printStackTrace();
}
}
}while(!(n==3));
}
}
```



```
C:\Users\hp\Documents\College Exams\Networks>java DNS

 Menu:
 1. DNS 2. Reverse DNS 3. Exit


 Enter your choice
1

 Enter Host Name
Host Name: www.google.com
IP: 142.250.192.164

 Menu:
 1. DNS 2. Reverse DNS 3. Exit


 Enter your choice
3

C:\Users\hp\Documents\College Exams\Networks>
```

Conclusion:

The DNS lookup program demonstrates the practical use of Java's networking capabilities to perform both forward and reverse DNS resolution. By utilizing the InetAddress class, the program successfully retrieves IP addresses from hostnames and hostnames from IP addresses, reinforcing core concepts of internet communication and domain name resolution. The menu-driven structure provides an interactive way for users to explore how domain names are translated into machine-understandable addresses and vice versa. Overall, the program serves as an effective educational tool for understanding the fundamentals of DNS functionality in network programming.



```
1-Reverse
2-Concate
3-Uppercase
4-Lowercase
5-Sentence
6-Exit
Enter your Option
2
Enter a string
computer
Enter a string
networkds
Concate: computer networkds
1-Reverse
2-Concate
3-Uppercase
4-Lowercase
5-Sentence
6-Exit
Enter your Option
4
Enter a String:
This is Uppercase
Lowercase: this is uppercase
1-Reverse
2-Concate
3-Uppercase
4-Lowercase
5-Sentence
6-Exit
Enter your Option
6

C:\Users\hp\Documents\College Exams\Networks\rev>
```

```
PS C:\Users\hp\Documents\College Exams\Networks\rev> java Server
Connected successfully...
Option given by the client: 2
String given by the client: computer
String given by the client: networkds
Option given by the client: 4
String given by the client: This is Uppercase
Option given by the client: 6
PS C:\Users\hp\Documents\College Exams\Networks\rev>
```

Conclusion:

The client-server string manipulation program effectively demonstrates basic socket programming in Java by enabling real-time interaction between a client and a server. It allows the user to perform various string operations such as reversing a string, concatenating two strings, converting to uppercase or lowercase, and formatting a string into sentence case. The client sends the user's input and chosen operation to the server using DataOutputStream, and the server processes the request, performs the corresponding string operation, and returns the result using DataInputStream. This program highlights the seamless data exchange and processing between two endpoints over a network, reinforcing key concepts of network communication, input/output stream handling, and string manipulation in Java.