



UNIVERSITY OF DHAKA

Department of Computer Science and Engineering

CSE-3111 : Computer Networking Lab

Lab Report 2: Introduction to Socket Programming —
Exercises on Simple Client-Server Communication

Submitted By:

Name: Mahmudul Hasan

Roll No : 20

Name : Arif Billah

Roll No : 32

Submitted On :

Januray 24, 2023

Submitted To :

Dr. Md. Abdur Razzaque

Md Mahmudur Rahman

Md. Ashraful Islam

Md. Fahim Arefin

1 Introduction

This lab report describes the exercises performed on simple client-server communication using socket programming. Socket programming is a method of creating a communication link between two devices over a network using sockets. The exercises in this lab report focused on creating a simple client-server application using the Java programming language.

1.1 Objectives

1. To understand the principles of network communication and how data is transmitted over a network.
2. To learn how to create a client-server application using socket programming and the basics of programming with sockets.
3. To learn how to use socket programming to create applications that can connect to different types of networks and protocols, such as TCP, UDP, IPv4, and IPv6.
4. To learn how to use socket programming as a way to implement network security by implementing secure communication protocols such as SSL or TLS.
5. To understand the use of socket programming in various real-world applications such as chat applications, file transfer, remote access, and more.
6. To develop the ability to troubleshoot and debug network communication issues and to have knowledge on how to optimize network communication.
7. To have a good understanding of networking concepts and to be able to apply that knowledge to create different types of networked applications.
8. To have a fundamental understanding of the low-level details of network communication and use that knowledge to design and implement efficient and reliable networked systems.

2 Theory

Socket programming is a method of creating a communication link between two devices over a network using sockets. Sockets are the endpoints of a bidirectional communication link between two programs running on the network. A socket has an IP address and a port number.

The lab report covers exercises that demonstrate the basic principles of socket programming and simple client-server communication using the Java programming language. The exercises involve creating a simple server that listens for incoming connections on a specified port and creating a simple client that connects to the server. In exercise A(i), the server listens to the client and in response, the server returns the incoming message converting it into uppercase letters. In exercise A(ii), the client requests the server to tell whether the sent number is "Prime" or "Not Prime" and the server returns the response by calculating the numbers type as "Prime" or "Not Prime". In exercise B(i), we've implemented a protocol between the ATM and the Bank's centralized server and extended version of exercise B(i), in exercise B(ii), we handled errors in client-server communication.

The lab report also covers the concepts of IP addresses and ports, which are used to uniquely identify a specific process running on a device on a network. IP addresses are used to identify a device on the network, while ports are used to identify a specific process running on that device.

Overall, the lab report provides a hands-on introduction to the basics of socket programming and simple client-server communication, providing a foundation for further exploration of socket programming and its applications in network communication.

3 Methodology

The exercises were performed on a computer running the Linux operating system. JDK version 17.0.2 was used for the programming. The `java.net.ServerSocket` and `java.net.Socket` libraries are imported to create the socket and perform the communication.

3.1 Converting Client Message to Upper Case

In the exercise, the client sends a message to the server and the server converts the message characters into uppercase and sends it in response. Here, we've made a server and made a terminal-based interface for clients to send requests to the server. The Client is connected to the server through socket. When the server gets the request from the client, it converts the message and sends it to the client.

Server.java [1]

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

public class Server {
    //initialize socket and input stream
    private Socket socket = null;
    private ServerSocket server = null;
    private DataInputStream in = null;
    private DataOutputStream out = null;

    // constructor with port
    public Server(int port) {
        // starts server and waits for a connection
        try {
            server = new ServerSocket(port);
            System.out.println("Server started");

            System.out.println("Waiting for a client ...");

            socket = server.accept();
            System.out.println("Client accepted");

            // takes input from the client socket
            in = new DataInputStream(new BufferedInputStream(socket.getInputStream()));
            out = new DataOutputStream(socket.getOutputStream());

            String line = "";
```

```

        // reads message from client until "Over" is sent
        while (!line.equals("Over")) {
            try {
                line = in.readUTF();
                System.out.println(line);
                out.writeUTF(line.toUpperCase());
                out.flush();
            } catch (IOException i) {
                System.out.println(i);
            }
        }
        System.out.println("Closing connection");

        // close connection
        socket.close();
        in.close();
    } catch (IOException i) {
        System.out.println(i);
    }
}

public static void main(String args[]) {
    Server server = new Server(1234);
}
}

```

Client.java

```

import java.io.*;
import java.net.*;
import java.util.Enumeration;
import java.util.Scanner;

public class Client {
    // Socket for the client
    private Socket socket;
    // BufferedReader and BufferedWriter for reading and writing data
    private BufferedReader bufferedReader;
    private BufferedWriter bufferedWriter;
    // String to store the client's username
    private String username;
}

```

```

// Constructor to initialize the client
public Client(Socket socket, String username){
    try{
        // Initialize the socket, BufferedReader, BufferedWriter, and
        this.socket = socket;
        this.bufferedWriter= new BufferedWriter(new OutputStreamWriter
        this.bufferedReader = new BufferedReader((new InputStreamReader
        this.username = username;
    } catch (IOException e){
        e.printStackTrace();
        // Close all resources if an exception is thrown
        closeEverything(socket, bufferedReader, bufferedWriter);
    }
}

// Method to send messages
public void sendMessage(){
    try{
        // Send the client's username to the server
        bufferedWriter.write(username);
        bufferedWriter.newLine();
        bufferedWriter.flush();

        Scanner scanner = new Scanner((System.in));
        // Loop until the socket is connected
        while(socket.isConnected()){
            // Read a message from the user
            String messageToSend = scanner.nextLine();
            // Send the message to the server
            bufferedWriter.write(username+": "+messageToSend);
            bufferedWriter.newLine();
            bufferedWriter.flush();
        }
    } catch (IOException e) {
        // Close all resources if an exception is thrown
        closeEverything(socket, bufferedReader, bufferedWriter);
    }
}

```

```

// Method to listen for messages
public void listenForMessage(){
    new Thread(new Runnable() {
        @Override
        public void run() {
            String messageFromGroupChat;
            // Loop until the socket is connected
            while(socket.isConnected()){
                try {
                    // Read a message from the server
                    messageFromGroupChat = bufferedReader.readLine();
                    // Print the message to the console
                    System.out.println(messageFromGroupChat);
                } catch (IOException e) {
                    // Close all resources if an exception is thrown
                    closeEverything(socket, bufferedReader, bufferedWriter);
                }
            }
        }
    }).start();
}

// Method to close all resources
private void closeEverything(Socket socket, BufferedReader bufferedReader,
    BufferedWriter bufferedWriter) {
    try {
        if(bufferedReader != null){
            bufferedReader.close();
        }
        if (bufferedWriter != null){
            bufferedWriter.close();
        }
        if(socket != null){
            socket.close();
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}

// Main method

```

```

        public static void main (String[] args) throws IOException {
            Scanner scanner = new Scanner(System.in);
            System.out.print("Enter username: ");
            String username = scanner.nextLine();
            String serverip = "10.33.3.23";
            int serverport = 1234;
            Socket socket = new Socket(serverip, serverport);
            Client client = new Client(socket, username);
            client.listenForMessage();
            client.sendMessage();
        }
    }
}

```

3.2 Checking whether a number is prime or not

In the exercise, the client sends a number to the server, and the server checks whether the number is prime or not and returns the result in response. On the server side, a method is called to check whether a sent number is prime or not, after checking the server sends the response to the client.

Server.java

```

import java.net.*;
import java.io.*;

public class Server {
    //initialize socket and input stream
    private Socket socket = null;
    private ServerSocket server = null;
    private DataInputStream in = null;
    private DataOutputStream out = null;

    String isPrime(int n){
        if(n==1) return "Not Prime.";
        for(int i=2; i<n; ++i){
            if(n%i==0) return "Not Prime.";
        }
        return "Prime";
    }
}

```



```

    }
    // constructor with port
    public Server(int port) {
        // starts server and waits for a connection
        try {
            server = new ServerSocket(port);
            System.out.println("Server started");

            System.out.println("Waiting for a client ...");

            socket = server.accept();
            System.out.println("Client accepted");

            // takes input from the client socket
            in = new DataInputStream(new BufferedInputStream(socket.getInputStream()));
            out = new DataOutputStream(socket.getOutputStream());

            String line = "";

            // reads message from client until "Over" is sent
            while (!line.equals("Over")) {
                try {
                    line = in.readUTF();
                    System.out.println(line);
                    out.writeUTF(isPrime(Integer.valueOf(line)));
                } catch (IOException i) {
                    System.out.println(i);
                }
            }
            System.out.println("Closing connection");

            // close connection
            socket.close();
            in.close();
        } catch (IOException i) {
            System.out.println(i);
        }
    }
}

```

```

        public static void main(String args[]) {
            Server server = new Server(1234);
        }
    }

```

Client.java

```

import java.io.*;
import java.net.*;

public class Client {
    // initialize socket and input-output streams
    private Socket socket = null;
    private DataInputStream input = null;
    private DataInputStream in = null;
    private DataOutputStream out = null;

    // constructor to put ip address and port
    public Client(String address, int port)
    {
        // establish a connection
        try {
            socket = new Socket(address, port);
            System.out.println("Connected");

            // takes input from terminal
            input = new DataInputStream(System.in);
            in = new DataInputStream(socket.getInputStream());

            // sends output to the socket
            out = new DataOutputStream(
                socket.getOutputStream());
        }
        catch (UnknownHostException u) {
            System.out.println(u);
            return;
        }
        catch (IOException i) {
            System.out.println(i);
            return;
        }
    }
}

```

```

    }

    // string to read message from input
    String line = "";

    // keep reading until "Over" is input
    while (!line.equals("over")) {
        try {
            line = input.readLine();
            out.writeUTF(line);
            String msg = in.readUTF();
            if (!line.equals("over"))
                System.out.println(msg);

        }
        catch (IOException i) {
            i.printStackTrace();
        }
    }

    // close the connection
    try {
        input.close();
        out.close();
        socket.close();
    }
    catch (IOException i) {
        System.out.println(i);
    }
}

public static void main(String args[])
{
    Client client = new Client("10.33.3.24", 1234);
}
}

```

3.3 A protocol between an ATM and a bank's centralized server

In the exercise, the client sends various types of requests to the bank's central server and the server handles all the requests depending on validity. If a request is valid, the server runs the requested operation and sends the necessary data to the client. If an invalid or unauthorized request is found on the server side, the server rejects the request and sends the necessary warnings to the client side.

Server.java

```
import javax.swing.plaf.basic.BasicInternalFrameTitlePane;
import java.awt.*;
import java.awt.desktop.UserSessionEvent;
import java.io.*;
import java.net.*;
import java.util.ArrayList;
import java.util.List;
import java.io.File;

import User.User;

public class Server{
    private static Socket socket;
    private static ServerSocket server;

    private static DataInputStream input ;
    private static DataOutputStream out;
    static int balance = 0;
    public Server() {
    }
    private static List<User> list = new ArrayList<>();
    static User user2 = new User("arif", "1234", 0);

    public static void main(String[] args) throws Exception{
        init();
        list.add(new User("arif", "1234", 0));
        list.add(new User("mahmud", "124", 0));
        list.add(new User("ab", "124", 0));
    }
}
```

```

server = new ServerSocket(1225);
System.out.println("Waiting for client");
socket = server.accept();
System.out.println("connected!");
input = new DataInputStream(new BufferedInputStream(socket.getInputStream()));
out = new DataOutputStream(socket.getOutputStream());

String line = "";
String username = "";
String password = "";
String amount = null;
while(!line.equals("0")) {
    line = input.readUTF();
    if(line.equals("0")) break;
    System.out.println(line);
    String[] op = line.split(" ");
    String type = op[0];
    username = op[1];
    password = op[2];
    int id = find_id(username, password);

    if(type.equals("00")){
        if(id != -1)
            out.writeUTF("1");
        else {
            out.writeUTF("0");
            continue;
        }
    }
    User user = list.get(id);
    if(type.equals("01")){
        int money = Integer.parseInt(op[3]);
        money += money;
        user.setBalance(user.getBalance() + money);
        list.set(id, user);
        out.writeUTF("credited");
    }
    else if(type.equals("02")){

        int money = Integer.parseInt(op[3]);

```

```

        if (money > balance)
            out.writeUTF("low");
        else {
            user.setBalance(user.getBalance() - money);
            user.setBalance(money);
            list.set(id, user);
            out.writeUTF("debited");
        }

    }
    else if (type.equals("03")){
        System.out.println("checking balance");
        String current = String.valueOf(user.getBalance());
        System.out.println(current);
        out.writeUTF(current);
    }
    System.out.flush();

}

socket.close();
server.close();
input.close();
out.close();
finish();

}

static int find_id(String usr, String pass){
    int id = 0;
    for (User u : list){
        if (u.getUserName().equals(usr) && u.getPassword().equals(pass))
            return id;
    }
    return -1;
}

static void init(){

```

```

        File file = new File("File");
        while (file.hasNextLine()){
            String data = file.nextLine();
            String[] op = data.split(" ");
            User user = new User(op[0], op[1], Integer.parseInt(op[2]));
            list.add(user);
        }
    }
    static void finish(){
        File f = new File("File");
        for(User u : list){
            f.write(u.getUserName() + " " + u.getPassword() + " " + u.getb
        }
    }
}

```

Client.java

```

import javax.swing.plaf.basic.BasicInternalFrameTitlePane;
import java.awt.*;
import java.awt.desktop.UserSessionEvent;
import java.io.*;
import java.net.*;
import java.util.ArrayList;
import java.util.List;
import java.io.File;

import User.User;

public class Server{
    private static Socket socket;
    private static ServerSocket server;

    private static DataInputStream input ;
    private static DataOutputStream out;
    static int balance = 0;
    public Server() {
    }
    private static List<User> list = new ArrayList<>();
    static User user2 = new User("arif", "1232", 0);
}

```

```

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

    server = new ServerSocket(1240);
    System.out.println("Waiting for client");
    socket = server.accept();
    System.out.println("connected!");
    input = new DataInputStream(new BufferedInputStream(socket.getInputStream()));
    out = new DataOutputStream(socket.getOutputStream());

    String line = "";
    String username = "";
    String password = "";
    String amount = null;
    while(!line.equals("0")) {
        line = input.readUTF();
        if(line.equals("0")) break;
        System.out.println(line);
        String[] op = line.split(" ");
        String type = op[0];
        username = op[1];
        password = op[2];
        int id = find_id(username, password);

        if(type.equals("00")){
            if(id != -1)
                out.writeUTF("1");
            else {
                out.writeUTF("0");
                continue;
            }
        }
        User user = list.get(id);
        if(type.equals("01")){
            int money = Integer.parseInt(op[3]);
            money += money;
            user.setBalance(user.getBalance() + money);
            list.set(id, user);
            out.writeUTF("credited");
        }
    }
}

```



```

        else if (type.equals("02")){

            int money = Integer.parseInt(op[3]);
            if (money > user.getBalance())
                out.writeUTF("low");
            else {
                user.setBalance(user.getBalance() - money);
                list.set(id, user);
                out.writeUTF("debited");
            }

        }
        else if (type.equals("03")){
            System.out.println("checking balance");
            String current = String.valueOf(user.getBalance());
            System.out.println(current);
            out.writeUTF(current);
        }
        System.out.flush();

    }

    socket.close();
    server.close();
    input.close();
    out.close();
    finish();

}

static int find_id(String usr, String pass){
    int id = 0;
    for (User u : list){
        if (u.getUserName().equals(usr) && u.getPassword().equals(pass))
            return id;
    }
    return -1;
}

```

```

    }
    static void init() throws IOException {
        File file = new File("File");
        BufferedReader br
            = new BufferedReader(new FileReader(file));
        String data = "";
        while ((data = br.readLine()) != null){

            String[] op = data.split(" ");
            //      System.out.println(op[0]);
            //      System.out.println(op[1]);
            //      System.out.println(op[2]);
            User user = new User(op[0], op[1], Integer.parseInt(op[2]));
            list.add(user);
        }
    }
    static void finish() throws IOException {
        File file = new File("File");
        BufferedWriter br
            = new BufferedWriter(new FileWriter(file));
        for(User u : list){
            String st = u.getUserName() + " " + u.getPassword() + " " + u.;
            br.write(st);
        }
        br.close();
    }
}

```

3.4 Error Handling in ATM and Bank server communication

In this section, the server handles probable errors that can occur during server-client communication.

Server.java

```

import javax.swing.plaf.basic.BasicInternalFrameTitlePane;
import java.awt.*;
import java.awt.desktop.UserSessionEvent;
import java.io.*;

```

```

import java.net.*;
import java.util.ArrayList;
import java.util.List;
import java.io.File;
import java.lang.Math;
import java.time.LocalDateTime;
import User.User;

public class Server{
    private static Socket socket;
    private static ServerSocket server;

    private static DataInputStream input ;
    private static DataOutputStream out;
    static int balance = 0;
    public Server() {
    }
    private static List<User> list = new ArrayList<>();
    static User user2 = new User(" arif", "1232", 0);

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

        server = new ServerSocket(1253);
        System.out.println("Waiting for client");
        socket = server.accept();
        System.out.println("connected!");
        input = new DataInputStream(new BufferedInputStream(socket.getInputStream()));
        out = new DataOutputStream(socket.getOutputStream());

        String line = "";
        String username = "";
        String password = "";
        String amount = null;
        while(!line.equals("0")) {
            line = input.readUTF();
            if(line.equals("0")) break;
            System.out.println(line);
            String[] op = line.split(" ");
            String type = op[0];

```

```

username = op[1];
password = op[2];
int id = find_id(username, password);

if (type.equals("00")){
    if (id != -1)
        out.writeUTF("1");
    else {
        out.writeUTF("0");
        continue;
    }
}
User user = list.get(id);
if (type.equals("01")){
    if (!permit()){
        System.out.println("not");
        continue;
    }
    int money = Integer.parseInt(op[3]);
    money += money;
    user.setBalance(user.getBalance() + money);
    list.set(id, user);
    out.writeUTF("credited");
}
else if (type.equals("02")){
    if (!permit()){
        continue;
    }
    int money = Integer.parseInt(op[3]);
    if (money > user.getBalance())
        out.writeUTF("low");
    else {
        user.setBalance(user.getBalance() - money);
        list.set(id, user);
        out.writeUTF("debited");
    }
}

}
else if (type.equals("03")){

```

```

        if (!permit()) {
            continue;
        }
        System.out.println("checking balance");
        String current = String.valueOf(user.getBalance());
        System.out.println(current);
        out.writeUTF(current);
    }
    System.out.flush();
}

socket.close();
server.close();
input.close();
out.close();
finish();
}

static int find_id(String usr, String pass){
    int id = 0;
    for(User u : list){
        if(u.getUserName().equals(usr) && u.getPassword().equals(pass))
            return id;
    }
    return -1;
}

static void init() throws IOException {
    File file = new File("File");
    BufferedReader br
        = new BufferedReader(new FileReader(file));
    String data = "";
    while ((data = br.readLine()) != null){

        String[] op = data.split(" ");
//        System.out.println(op[0]);
//        System.out.println(op[1]);
    }
}

```

```

//          System.out.println(op[2]);
          User user = new User(op[0], op[1], Integer.parseInt(op[2]));
          list.add(user);
      }
  }
  static void finish() throws IOException {
      File file = new File("File");
      BufferedWriter br
          = new BufferedWriter(new FileWriter(file));
      for(User u : list){
          String st = u.getUserName() + " " + u.getPassword() + " " + u.;
          br.write(st);
      }
      br.close();
  }

  static boolean permit(){
      double num = Math.random();

      return !(num < 0.25);
  }
}

```

Client.java

```

import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
import java.time.LocalDate;
import java.time.temporal.ChronoUnit;
import java.util.Scanner;
import java.time.LocalDateTime;
import static java.lang.Thread.sleep;

public class Client {

    private static Socket socket;
    private static ServerSocket server;
    private static DataInputStream input = null ;
    private static DataOutputStream out ;

```

```

private static Scanner scanner = null;
public static void main(String[] args) throws IOException, InterruptedException

    // Create client socket
    socket = new Socket("localhost", 1253);
    input = new DataInputStream(socket.getInputStream());
    out = new DataOutputStream(socket.getOutputStream());
    scanner = new Scanner(System.in);


    String op = "";
    String line = "";

    boolean auth = false;
    String username = null;
    String password = null;
    boolean mainmenu = true;
    while(mainmenu){
        if(!auth) {
            System.out.println("Enter Username");
            username = scanner.nextLine();
            System.out.println("Enter Password");
            password = scanner.nextLine();
            op = "00 " + username + " " + password;
            if(password == "" || username == "") {
                System.out.println("Authentication Failed!");
            }
        }
        else{
            out.writeUTF(op);
            String response = input.readUTF();
            if(response.equals("0")){
                System.out.println("Authentication Failed!");
            }
            else auth = true;
            System.out.flush();
        }
    }
}

```

```

else{

    System.out.println("_____");
    System.out.println("1.Credit \n2.Debit\n3.Check balance\nE
        line = scanner.nextLine();
    if(line.equals("1")){
        System.out.println("Enter amount");
        String amount = scanner.nextLine();
        if(valid(amount) != -1){
            op = "01 " + username + " " + password + " " + amo
            out.writeUTF(op);
            LocalDateTime time = LocalDateTime.now().plus(1, C
            String response = null;
            while(LocalDateTime.now().isBefore(time)){
                if(input.available() > 0)
                    response = input.readUTF();
                if(response != null)
                    break;

            }
            if(response == null){
                System.out.println("Time out!\nTry Again!");
            }
            else{
                System.out.println("Account is Credited!");
                System.out.flush();
            }
        }
    }
    else{
        System.out.println("Incorrect Number");
    }
}

else if(line.equals("2")){
    System.out.println("Enter amount");
    String amount = scanner.nextLine();
    System.out.println(valid(amount));

    if(valid(amount) != -1){

```



```

        op = "02 " + username + " " + password + " " + amount;
        out.writeUTF(op);
        LocalDateTime time = LocalDateTime.now().plus(1, ChronoUnit.HOURS);
        String response = null;
        while(LocalDateTime.now().isBefore(time)){
            if(input.available() > 0)
                response = input.readUTF();
            if(response != null)
                break;
        }
        if(response == null){
            System.out.println("Time out!\nTry Again!");
        }
        else if(response.equals("low")){
            System.out.println("Low balance\nTransaction Failed!");
        }
        else{
            System.out.println("Account is Debited!");
        }
    }
    else{
        System.out.println("Incorrect Number");
    }
}
else if(line.equals("3")){
    out.writeUTF("03 " + username + " " + password);
    LocalDateTime time = LocalDateTime.now().plus(1, ChronoUnit.HOURS);
    String response = null;
    while(LocalDateTime.now().isBefore(time)){
        if(input.available() > 0)
            response = input.readUTF();
        if(response != null)
            break;
    }
    if(response == null){
        System.out.println("Time out!\nTry Again");
    }
}

```

```

        else System.out.println("Your current balance is " + r);
    }
    else if (line.equals("0")){
        out.writeUTF("0");
        mainmenu = false;

    }
    else{
        System.out.println("Wrong number");
    }
    sleep(20);
    System.out.flush();
}

socket.close();

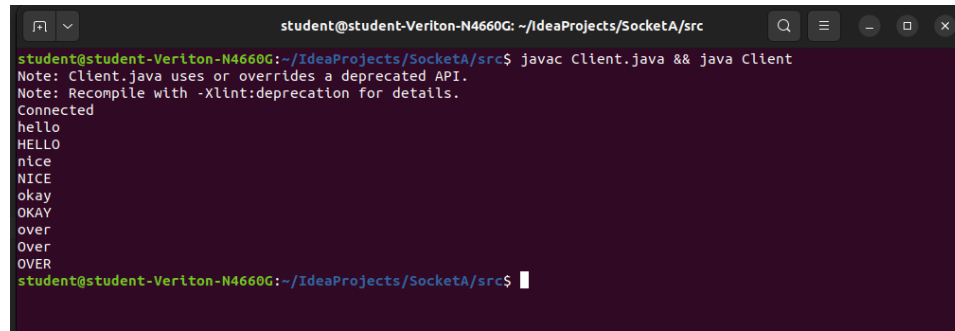
}

static int valid(String str){
    try{
        int amount = Integer.parseInt(str);
        if (amount < 0)
            return -1;
        return amount;
    }
    catch (Exception e){
        return -1;
    }
}
}

```

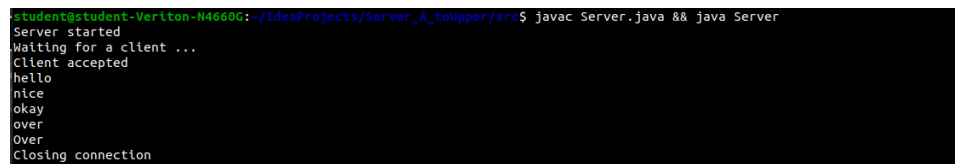
4 Experimental result

4.1 Converting Client Message to Upper Case

A terminal window with a dark background and light green text. The window title is 'student@student-Veriton-N4660G: ~/IdeaProjects/SocketA/src'. The command 'javac Client.java && java Client' has been executed. The output shows a series of messages: 'Note: Client.java uses or overrides a deprecated API.', 'Note: Recompile with -Xlint:deprecation for details.', 'Connected', 'hello', 'HELLO', 'nice', 'NICE', 'okay', 'OKAY', 'over', 'Over', 'OVER'. The prompt 'student@student-Veriton-N4660G:~/IdeaProjects/SocketA/src\$' is visible at the bottom.

```
student@student-Veriton-N4660G: ~/IdeaProjects/SocketA/src
student@student-Veriton-N4660G:~/IdeaProjects/SocketA/src$ javac Client.java && java Client
Note: Client.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
Connected
hello
HELLO
nice
NICE
okay
OKAY
over
Over
OVER
student@student-Veriton-N4660G:~/IdeaProjects/SocketA/src$
```

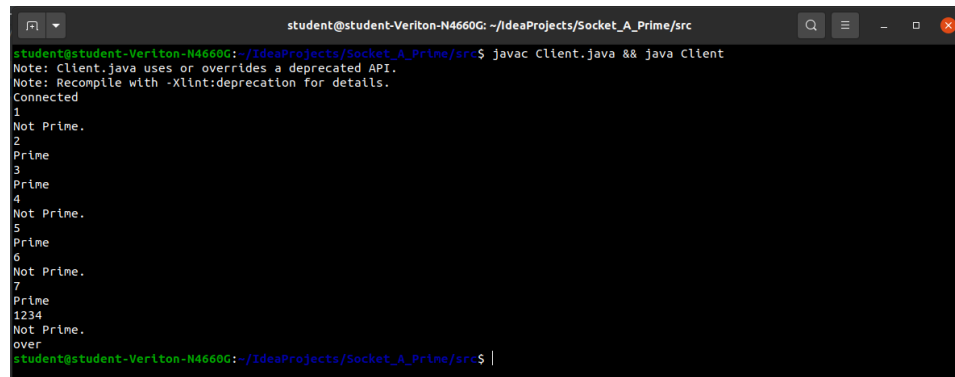
Figure 1: Client Side Requests and Responses

A terminal window with a dark background and light green text. The window title is 'student@student-Veriton-N4660G: ~/IdeaProjects/Server_A_toUpper/src'. The command 'javac Server.java && java Server' has been executed. The output shows: 'Server started', 'Waiting for a client ...', 'Client accepted', 'hello', 'nice', 'okay', 'over', 'Over', 'Closing connection'. The prompt 'student@student-Veriton-N4660G:~/IdeaProjects/Server_A_toUpper/src\$' is visible at the bottom.

```
student@student-Veriton-N4660G:~/IdeaProjects/Server_A_toUpper/src$ javac Server.java && java Server
Server started
Waiting for a client ...
Client accepted
hello
nice
okay
over
Over
Closing connection
student@student-Veriton-N4660G:~/IdeaProjects/Server_A_toUpper/src$
```

Figure 2: Client's sent requests to server

4.2 Checking whether a number is prime or not



```
student@student-Veriton-N4660G: ~/IdeaProjects/Socket_A_Prime/src
student@student-Veriton-N4660G:~/IdeaProjects/Socket_A_Prime/src$ javac Client.java && java Client
Note: Client.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
Connected
1
Not Prime.
2
Prime
3
Prime
4
Not Prime.
5
Prime
6
Not Prime.
7
Prime
1234
Not Prime.
over
student@student-Veriton-N4660G:~/IdeaProjects/Socket_A_Prime/src$
```

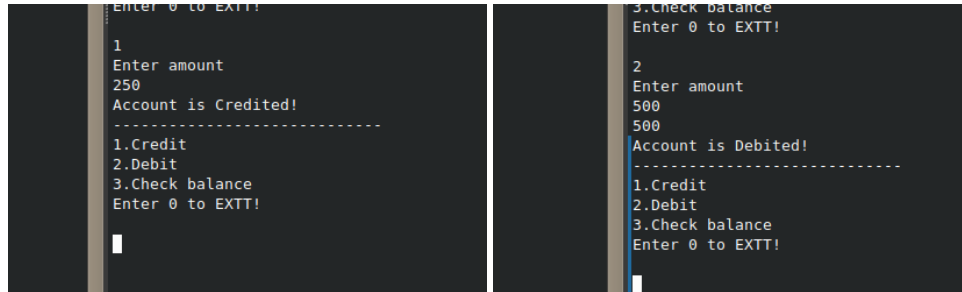
Figure 3: Client Side Requests and Responses



```
student@student-Veriton-N4660G: ~/IdeaProjects/SocketA/src
student@student-Veriton-N4660G:~/IdeaProjects/SocketA/src$ javac Server.java && java Server
Server started
Waiting for a client ...
Client accepted
1
2
UTF3
nt14
s('5
6
TF(7
1234
TF(over
Closing connection
student@student-Veriton-N4660G:~/IdeaProjects/SocketA/src$
```

Figure 4: Client's sent requests to server

4.3 A protocol between an ATM and a bank's centralized server



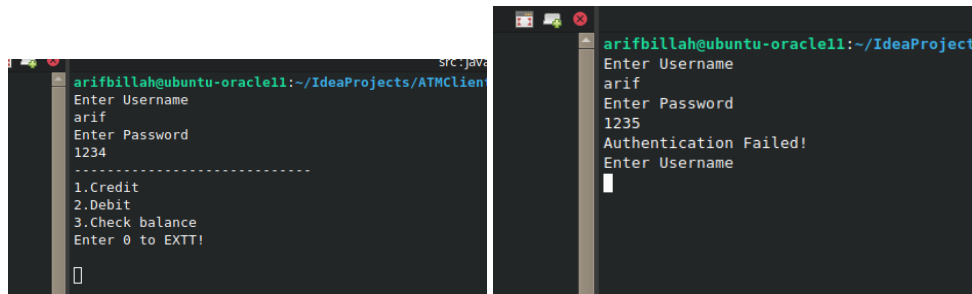
The image contains two side-by-side terminal windows. The left window shows a menu with options 1 (Credit), 2 (Debit), and 3 (Check balance). Option 1 is selected, and the user enters the amount 250. The terminal displays 'Account is Credited!' followed by a separator line and the menu again. The right window shows the same menu, but option 2 is selected, and the user enters the amount 500. The terminal displays 'Account is Debited!' followed by a separator line and the menu again.

```
Enter 0 to EXIT!  
1  
Enter amount  
250  
Account is Credited!  
-----  
1.Credit  
2.Debit  
3.Check balance  
Enter 0 to EXTT!  
  
3.Check balance  
Enter 0 to EXTT!  
2  
Enter amount  
500  
500  
Account is Debited!  
-----  
1.Credit  
2.Debit  
3.Check balance  
Enter 0 to EXTT!
```

(a) Account Credited

(b) Account Debited

Figure 5: Credit and Debit Operation on client side



The image contains two side-by-side terminal windows. The left window shows a login prompt where the user enters 'arif' as the username and '1234' as the password. The terminal displays the menu from Figure 5. The right window shows the same login prompt, but the user enters 'arif' as the username and '1235' as the password. The terminal displays 'Authentication Failed!' and prompts the user to enter the username again.

```
arifbillah@ubuntu-oracle11:~/IdeaProjects/ATMClient  
Enter Username  
arif  
Enter Password  
1234  
-----  
1.Credit  
2.Debit  
3.Check balance  
Enter 0 to EXTT!  
  
arifbillah@ubuntu-oracle11:~/IdeaProject  
Enter Username  
arif  
Enter Password  
1235  
Authentication Failed!  
Enter Username
```

(a) Successful Login

(b) Invalid Login Response

Figure 6: Login Operation

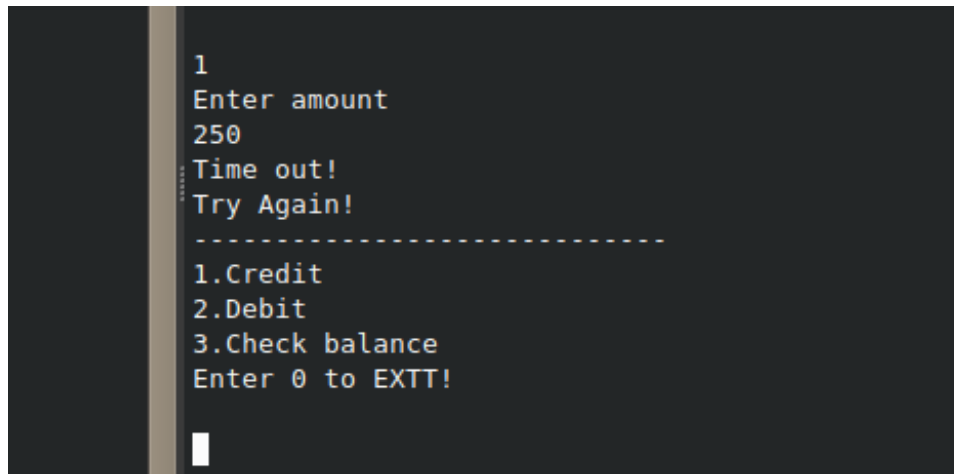
```
2.Debit
3.Check balance
Enter 0 to EXTT!

3
Your current balance is 4500
-----
1.Credit
2.Debit
3.Check balance
Enter 0 to EXTT!


```

Figure 7: Balance Checking

4.4 Error Handling in ATM and Bank server communication

A terminal window with a dark background and light-colored text. The text shows an ATM menu with options 1, 2, and 3. A user has entered '1' and '250'. The system responds with 'Time out!' and 'Try Again!'. A dashed line separates this from the main menu. The menu options are '1.Credit', '2.Debit', and '3.Check balance', followed by 'Enter 0 to EXT!'. A cursor is visible at the bottom left.

```
1
Enter amount
250
Time out!
Try Again!
-----
1.Credit
2.Debit
3.Check balance
Enter 0 to EXT!
```

Figure 8: Request denied as request validity time is over

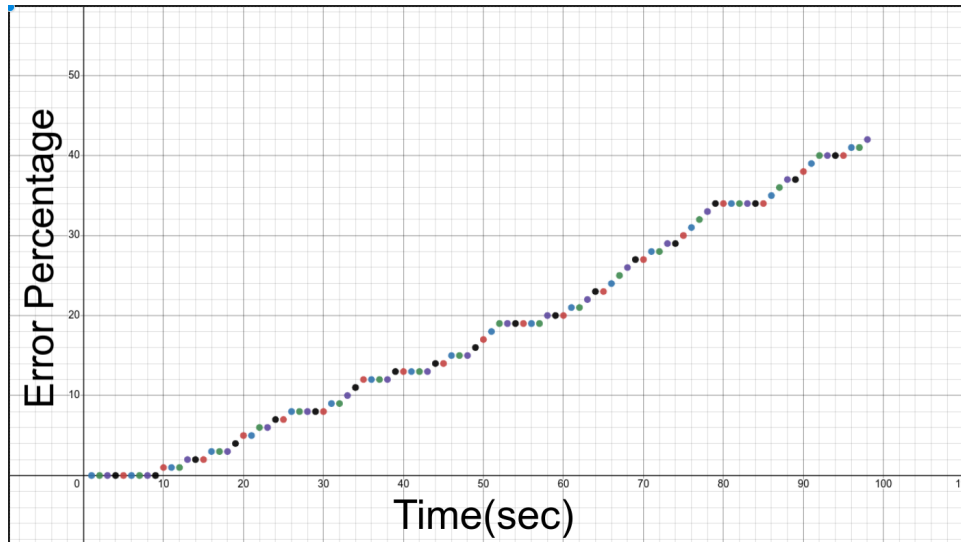


Figure 9: Time vs Error Percentage graph

5 Experiences

1. Hands-on experience in writing Java code for creating a simple server and client using the Java socket API.
2. Understanding of the basics of network communication and how data is transmitted over a network using sockets.
3. Knowledge on how to use socket programming as a tool for creating distributed systems, where multiple devices work together to perform a task.
4. Understanding of the use of socket programming in various real-world applications such as chat applications, file transfer, remote access, and more.
5. Experience in troubleshooting and debugging network communication issues and knowledge of how to optimize network communication.
6. Understanding of the low-level details of network communication and the ability to design and implement efficient and reliable networked systems.

7. Familiarity with the concepts of IP addresses and ports, and how they are used to uniquely identify a specific process running on a device on a network.

References

- [1] GeeksForGeeks:<https://www.geeksforgeeks.org/establishing-the-two-way-communication-between-server-and-client-in-java/>