Project Title : Implementing DFA using JAVA Socket Programming

Name : Vikrant Sharma

USN : 1NH15CS758

CHAPTER 1

INTRODUCTION

The aim of the project is to build a code that implements a DFA. The idea is to implement it using client and server program. The program takes string as an input , and it should print the states the string goes through and prints whether the string is accepted by the DFA or not accepted by the DFA.

The client-server is implemented using socket programming. The input is accepted at the client end , this input is further sent to the server. The server accepts the input from the socket streams, and it runs the DFA module, here the string is checked for the validity according to the given DFA.

It shows all the states passed by the string and displays whether the string is accepted or not accepted by the given DFA.

CHAPTER 2

DETERMINISTIC FINITE AUTOMATION (DFA)

2.2 INTRODUCTION

In the theory of computation, a branch of theoretical technology, a settled finite automation(DFA) additionally called settled finite accepter(DFA) and settled finite state machine may be a finite machine that accepts/rejects finite strings of symbols and solely produces a singular computation (or run) of the automation for every input string, ‘Deterministic’ refers to the distinctiveness of the computation. “In search of the best models to capture the finite state machines, McCulloch and Pitts” were among the primary investigator to introduce a thought of Finite Automation in 1943. A settled finite automaton (DFA) consists of:

1. a finite set of states (“often denoted by Q”)

2. a finite set ∑ of symbols (“alphabet”)

3. a transition operate that takes argument as a state and a logo and it returns a state (often denoted

4. a begin state typically denoted by q0

5. a group of ultimate or acceptive states (often denoted F)

We have q0 Q and F Q

So a DFA is mathematically delineated as a five tuple

(Q,∑, , q0, F)

The transition operate may be a operate in

Q × ∑Q

Q × ∑ is that the set of 2-tuples (q, α) with Q and α

2.2 DFA WITH TRANSITION TABLE

	0	1
$\rightarrow q_0$	q_2	q_0
$*q_1$	q_1	q_1
q_2	q_2	q_1

Figure 1 Transition table

The → indicates the start state: here *q0*

The * indicates the final state(s) ("here only one final state *q1"*)

This defines the following transition diagram:

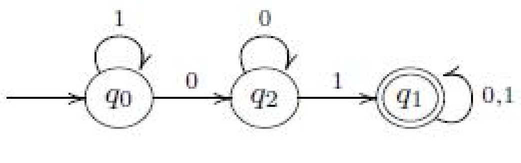


Figure 2 Transition diagram of DFA

"For this example

Q = {q0, q1, q2}

start state *q0*

F = {q1}

∑ = *{0, 1}*

δ is a function from *Q* × ∑ *to Q*

δ: Q × ∑→*Q*

δ(q0, 1) = q0

δ(q0, 0) = q2"

Example: passcode

When does the automaton accepts a word??

It reads the word and accepts it if it stops in an accepting state

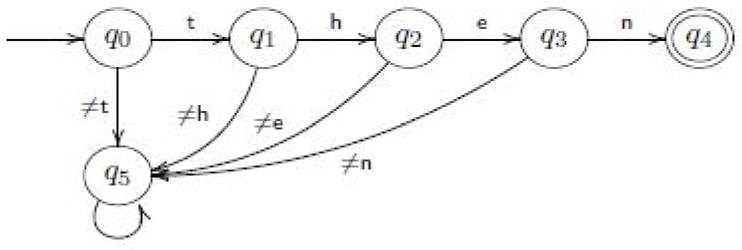


Figure 3 Transition diagram of DFA to read a string

Only the word then is accepted

Here *Q = {q0, q1, q2, q3, q4}*

∑ is the set of all characters

F = {q4}

We have a "stop" or "dead" state *q5*, not accepting

2.3 HOW A DFA PROCESSES A STRING

Let us build an automaton that accepts the words that contain 01 as a subword

∑= {0, 1}

L = {x01y | x, y 2 ∑}*

We use the following states

A: start

B: the most recent input was 1

C: the most recent input was 0

D: we have encountered 01 (final state)

We get the following automaton

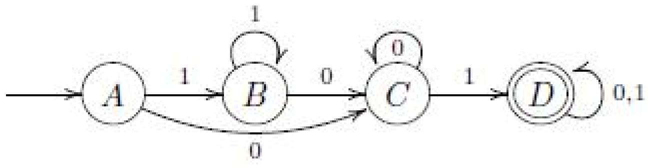
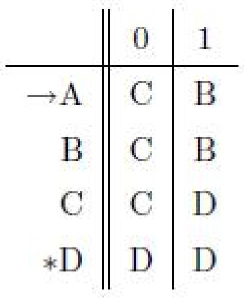


Fig 3: Transition diagram to process a string

Transition table



	0	1
→A	C	B
B	C	B
C	C	D
*D	D	D

Figure 4 Transition table 2

Q = {A, B, C, D}, ∑ = {0,1}, start state A, final state(s) {D}

2.4 APPLICATION OF DFA

Deterministic Finite Automata, or DFAs, have a “chic background in terms of the mathematical theory underlying their development and use”. This theoretical foundation is that the main stress of ECS 120's coverage of DFAs. However, this handout can target examining real-world applications of DFAs to achieve associate degree appreciation of the utility of this theoretical conception. DFA uses embody protocol analysis, “text parsing, computer game character behavior, security analysis, hardware management units, language process, and speech recognition. to boot, several straightforward (and not therefore simple) mechanical devices square measure ofttimes designed and enforced victimization DFAs”,such as elevators, marketing machines, and traffic-sensitive traffic lights.

2.4.1 Vending Machines

Picture displays a DFA that describes the behavior of a coin machine that accepts greenbacks and quarters, and charges 1.25 per soda. Once the machine receives a minimum of 1.25, equivalent to the blue-colored states within the diagram, it'll enable the user to pick a soda. Self-loops represent unnoticed input: the machine won't dispense a soda till a minimum of 1.25 has been deposited, and it'll not settle for extra money once it's already received bigger than or capable 1.25.

To express the DFA as a 5-tuple, the elements are outlined as follows:

1. Q = f0:00; 0:25; 0:50; 0:75; 1:00; 1:25; 1:50; 1:75; 2:00 are the states
2. ∑ = f0:25; 1:00; choose is that the alphabet
3. , the transition operate, is delineated by the state diagram.
4. q0 = 0:00 is that the begin state
5. F =; is that the set of settle for states

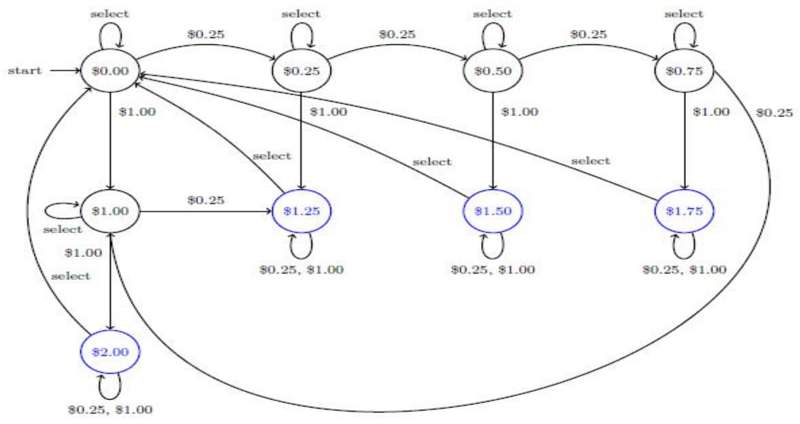


Figure 5 Vending machine state diagram

2.4.2 AI in Video Games: Pac-Man's Ghosts

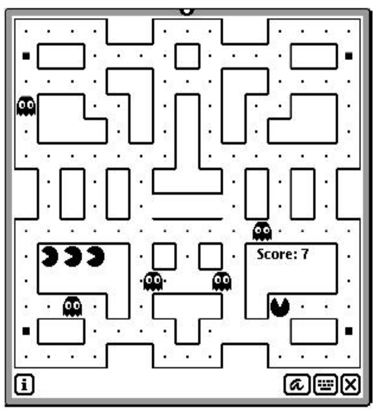


Figure 6 Screenshot of Pacman's game

Finite state machines lend themselves to representing the behavior of pc controller characters in video games. The states of the machine correspond to the character's behaviors, that modification consistent with varied events. These changes area unit sculptured by transitions within the state diagram. State machines area unit never|on no account|under no circumstances|not at all|in no way} the foremost subtle means of implementing by artificial means intelligent agents in games, however several games embody characters with straightforward, state-based behaviors that area unit simply and effectively sculptured exploitation state machines.

Here we have a tendency to think about the classic game, Pac-Man. “For those unfamiliar the game- play, needs the player to navigate through a maze, uptake pellets and avoiding the ghosts who chase him through the maze. sometimes, Pac-Man will flip the tables on his pursuers by uptake an influence pellet, that quickly grants him the facility to eat the ghosts”. once this happens, the ghosts' behavior adapts, and rather than chasing Pac-Man they struggle to avoid him.

The ghosts in Pac-Man have four behaviors:

“1. willy-nilly wander the maze

2. Chase Pac-Man, once he's at intervals line of sight

3. scarper Pac-Man, when Pac-Man has consumed an influence pellet

4. come back to the central base to regenerate”

These four behaviors correspond on to a four-state DFA. Transitions area unit determined by things within the game. as an example, a ghost DFA in state two (Chase Pac-Man) can transition to state three (Flee) once Pac-Man consumes an influence pellet.

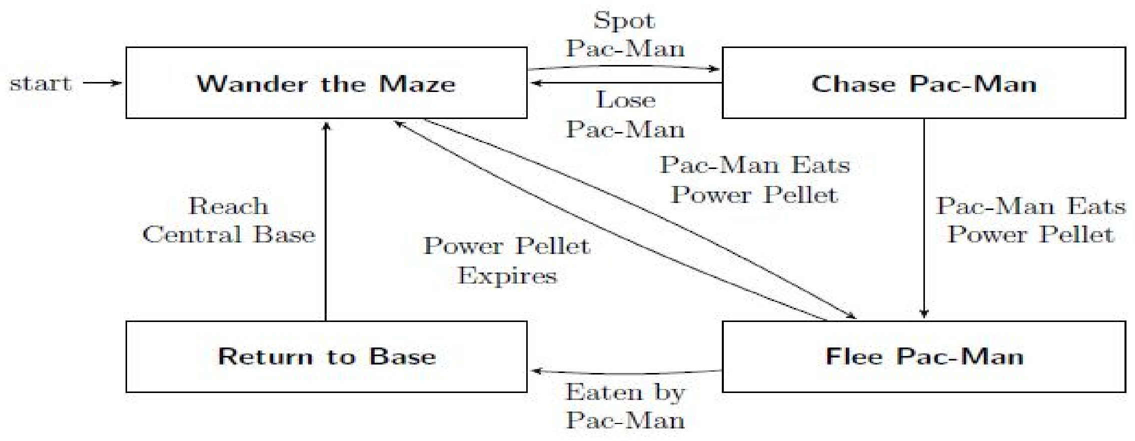


Figure 7 Behavior of a Pac Man ghost

CHAPTER 3

SOCKET PROGRAMMING

3.1 INTRODUCTION

Internet and WWW have been emerged as global universal media for communication and changed the way we conduct science, engineering, and commerce. “They are also altering the way we learn, live, enjoy, communicate, interact, engage, etc. Nowadays, the modern life activities are getting completely centered around or driven by the Internet. To make use of opportunities presented by the Internet, businesses are continuously looking for new and innovative ways and means for offering their services via the Internet. This resulted in an enormous demand for software designers and skilled engineers in creating new Internetenabled applications or modifying existing/legacy applications to the Internet platform”. The important fundamentals for developing Internet-enabled applications are a good understanding of the issues involved in implementing distributed applications and sound knowledge of the fundamental network programming models.

4.2 CLIENT/SERVER COMMUNICATION

In a network, there is a server, client, and a media for communication. “A computer running a program that produces a ask for administrations is called client machine. A computer running a program that provides asked administrations from one or more clients is called server machine. The media for communication can be wired or wireless network”.

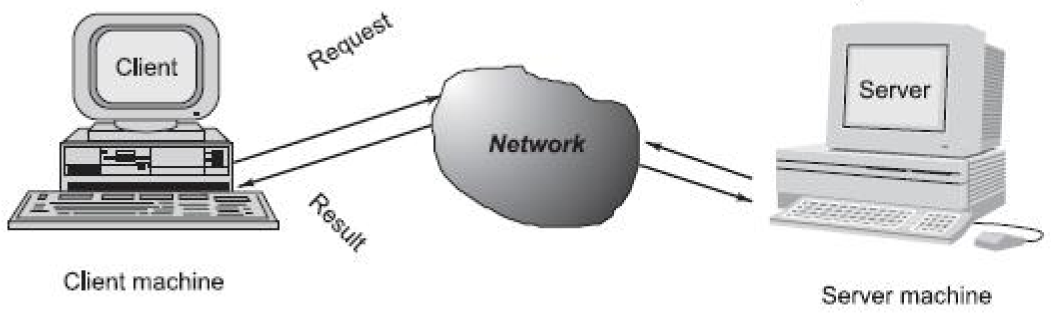


Figure 8 Client-Server communication

Commonly, the programs running on client machines make demands to a program (regularly called as server program) running on a server machine. “They involve networking administration given by the transport layer, which is part of Web computer program stack, frequently called TCP/IP (Transport Control Protocol/Internet Protocol) stack, shown in Fig. 5. TCP is a connection-oriented protocol that solid streams of information between two
computers. Example applications that use such administration are HTTP, FTP, and Telnet. UDP is a convention that sends autonomous parcel of information, called datagrams, from one computer to another with no assurances about coming and sequencing. A Port is denoted by a positive (16-bit) integer value. A few ports have been held in reserve to support common/well known services:

- ftp 21/tcp
- telnet 23/tcp
- smtp 25/tcp
- login 513/tcp
- http 80/tcp,udp
- https 443/tcp,udp”

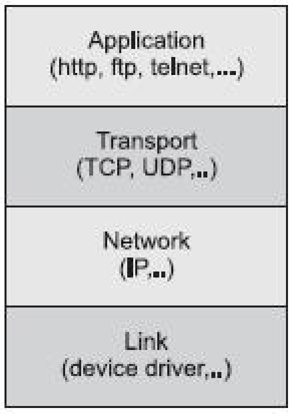


Figure 9 TCP/IP software stack

3.3 OBJECT ORIENTED PROGRAMMING WITH JAVA

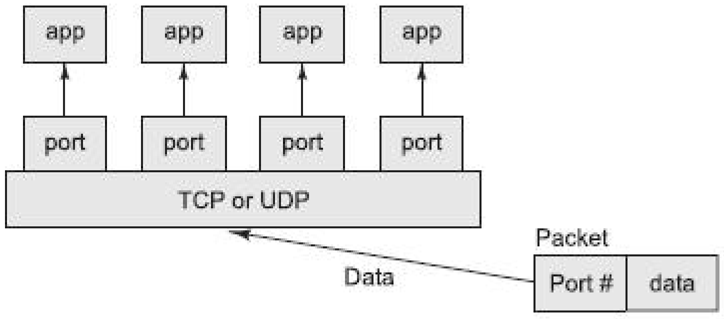


Figure 10 TCP/UDP mapping of incoming packets to appropriate port/process

Object-oriented Java technologies—Sockets, threads, RMI, clustering, Web services—have risen as foremost solutions for making convenient, effective, and sustainable large and complex Web applications.

3.4 SOCKETS AND SOCKET BASED COMMUNICATION

Sockets acts as an interface for programming systems at the transport layer. "Network communication utilizing Sockets is exponentially much comparative to performing file I/O. Actually, socket handle is treated like file handle. The streams utilized in file I/O operation are to appropriate to socket-based I/O. A server (program) runs on a specific computer and contains a socket that is bound to a particular port. The server in the socket looks for a client to make a connection request (see Fig. 7a). If everything goes well, the server accepts the connection (see Fig. 7b)". Upon acknowledgement, the server gets a new socket bound to a different port.

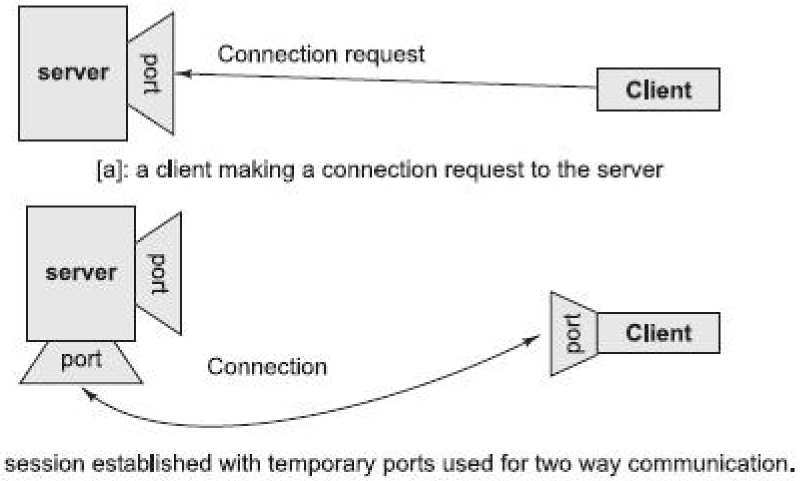

server
port
Connection request
Client
[a]: a client making a connection request to the server
server
port
port
Connection
port
Client
[b]: session established with temporary ports used for two way communication.

Figure 11 Establishment of path for two-way communication between a client and server

3.5 SOCKET PROGRAMMING AND JAVA.NET CLASS

A socket is an endpoint of a two-way communication connects between two programs running on the network. "The socket is bound to a port number so that the TCP layer can distinguish the application that information is predetermined to be sent. Java gives a set of classes, characterized in a package called java.net, to enable the quick advancement of network applications. Key classes, interfaces, and exceptions in java.net package simplifying the complexity included in making client and server programs are:

The Classes

- ContentHandler
- DatagramPacket
- DatagramSocket
- DatagramSocketImpl
- HttpURLConnection
- InetAddress
- MulticastSocket
- ServerSocket
- Socket
- SocketImpl
- URL
- URLConnection
- URLEncoder
- URLStreamHandler

The Interfaces

- ContentHandlerFactory
- FileNameMap
- SocketImplFactory
- URLStreamHandlerFactory

Exceptions

- BindException
- ConnectException
- MalformedURLException
- NoRouteToHostException
- ProtocolException
- SocketException
- UnknownHostException
- UnknownServiceException"

3.7 TCP/IP SOCKET PROGRAMMING

The two key classes from the java.net package utilized in creation of server and client programs are: "ServerSocket Socket A server program creates a specific type of socket that is used to listen for client requests (server socket). Within the case of a connection request, the program creates a new socket through which it will interchange information with the client utilizing input and output streams. The socket abstraction is exceptionally comparable to the file concept: developers have to open a socket, perform I/O, and close it. Figure 8 illustrates key steps involved in creating socket-based server and client programs".

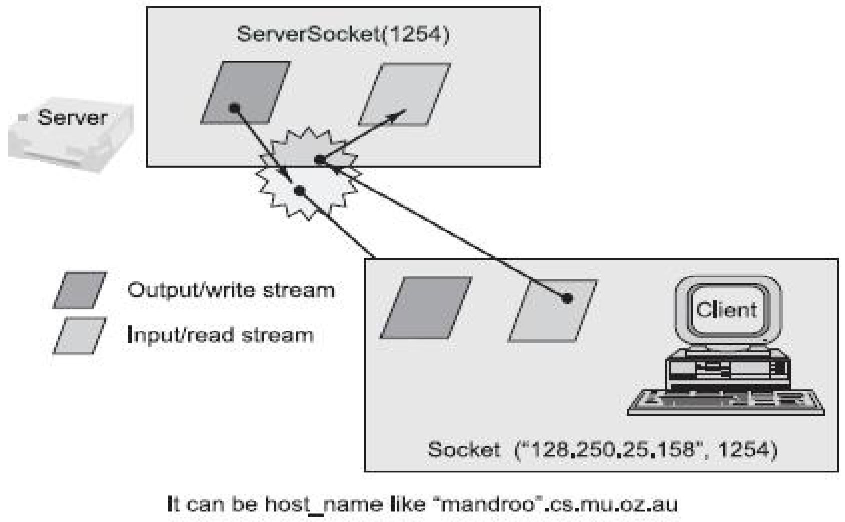

ServerSocket(1254)
Server
Output/write stream
Input/read stream
Client
Socket ("128.250.25.158", 1254)
It can be host_name like "mandroo".cs.mu.oz.au

Figure 12 Socket-based client and server programming

CHAPTER 4

IMPLEMENTATION

// “A Java program for a Client

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

public class Client
{
   // initialize socket and input output streams
   private Socket socket                  = null;
   private DataInputStream  input   = 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);

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

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


         try
```

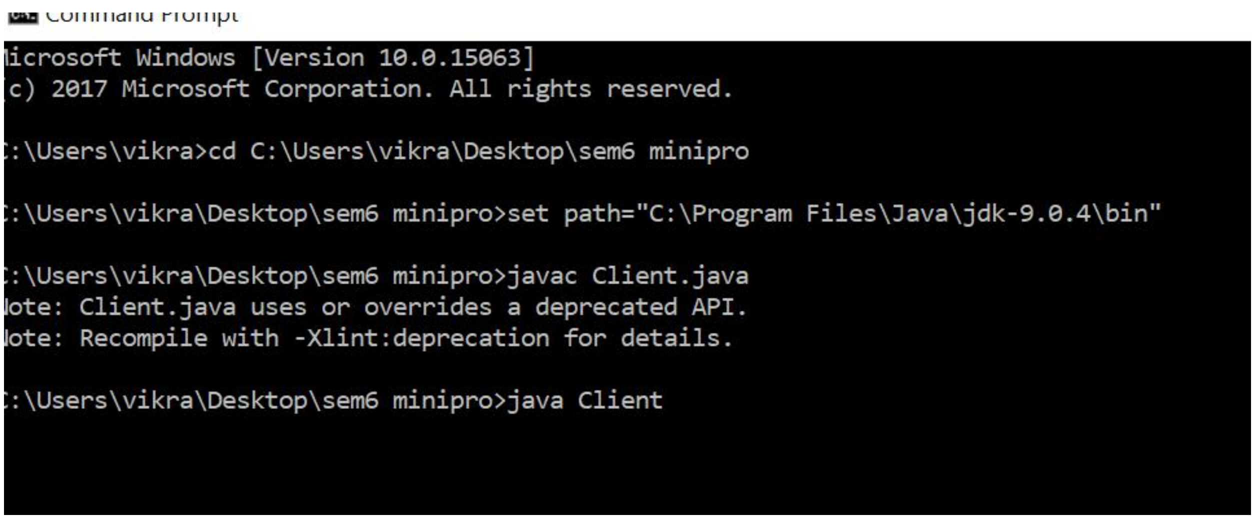
```
            {
                line = input.readLine();
                out.writeUTF(line);
            }
            catch(IOException i)
            {
                System.out.println(i);
            }


        // 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("127.0.0.1", 5000);
    }
}
```

// A Java program for a Server

```
import java.net.*;
import java.io.*;
import java.util.Scanner;
import java.lang.*;
class  dfa{
static int flag=0;
public static int Start(char c)
{       System.out.println("state: start");
            if(c=='g') return 1;
            else return 0;
}
public static int Q0(char c)
{       System.out.println("state: Q0");
            if(c=='g') return 1;
            else if(c=='o') return 2;
            else return 0;
}
public static int Q1(char c)
{       System.out.println("state: Q1");
            if(c=='g') return 1;
            else if(c=='o') return 3;
            else return 0;
}
public static int Q2(char c,int i)
{
            System.out.println("state: Q2");
            if(c=='g') return 1;
            else if(c=='d'){
        i=i-3;
        System.out.println("state: final");
        System.out.println("string found at index "+i);
        return 4;
        }
            else return 0;


}
public static int Qf(char c)
{       flag=1;
            int state=0;
            return state;
}
}
```

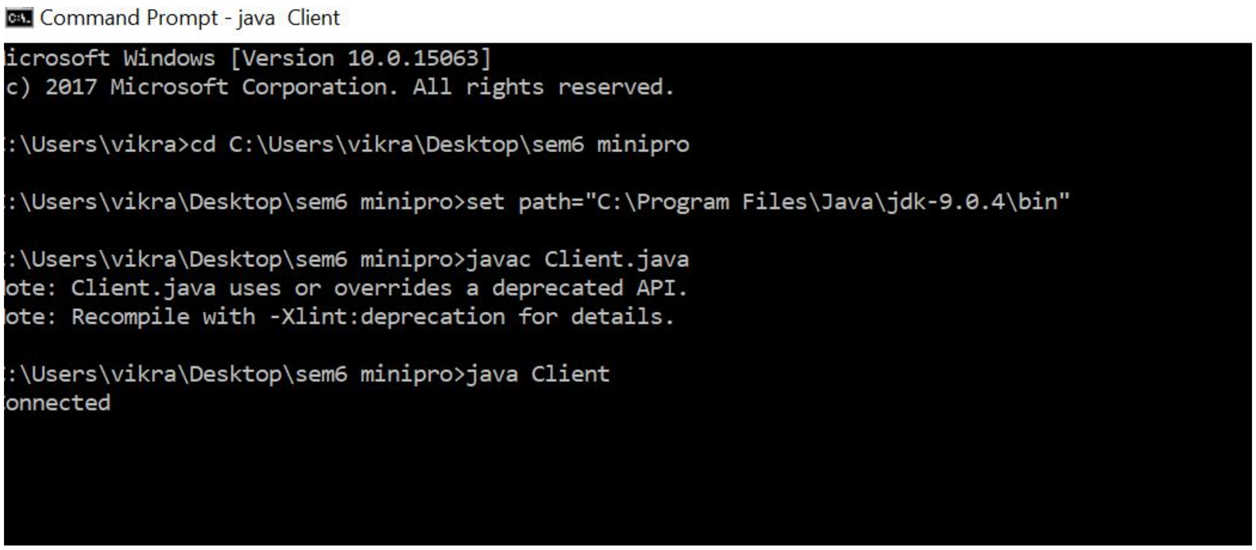
```
class pro extends dfa {
 pro(String a)
 {
          //String a;
      int state=0;
          Scanner sc=new Scanner(System.in) ;
          //System.out.println("enter the string\n");
          //a=sc.nextLine();
      char c;
          for(int i=0;i<a.length();i++)
          {
                  c=a.charAt(i);
                  if(state==0)
                          state=Start(c);
                  else if(state==1)
                          state=Q0(c);
                  else if(state==2)
                          state=Q1(c);
                  else if(state==3)
                          state=Q2(c,i);
                  else if(state==4)
                          {
                          state=Qf(c);
                          }

          }
       if(flag==0)
         System.out.println("string is not accepted");


 }
}
```

```
public class Server
{
    //initialize socket and input stream
    private Socket          socket   = null;
    private ServerSocket    server   = null;
    private DataInputStream in       =  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()));

            String line = "";
                try
                {
                    line = in.readUTF();
                    System.out.println(line);

                }
            catch(IOException i)
            {
                System.out.println(i);
            }

           pro p=new pro(line);
        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(5000);
}
}”
```



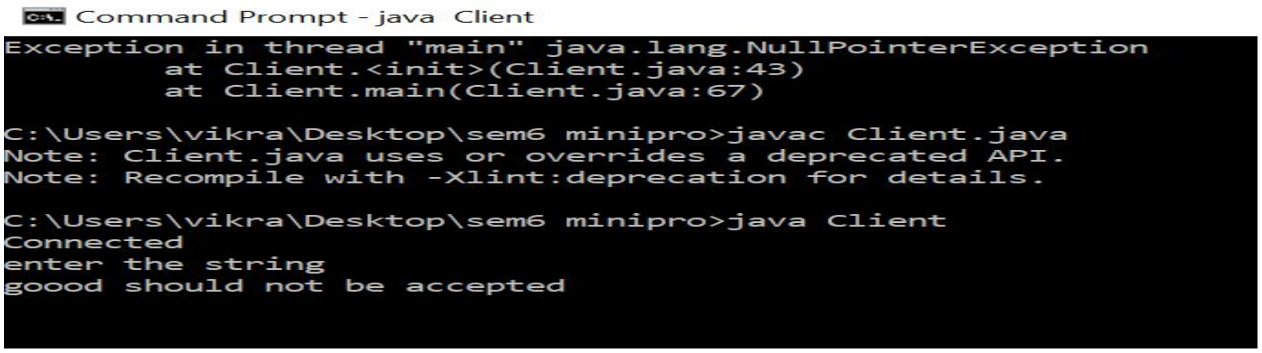
```
licrosoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

:\Users\vikra>cd C:\Users\vikra\Desktop\sem6 minipro

:\Users\vikra\Desktop\sem6 minipro>set path="C:\Program Files\Java\jdk-9.0.4\bin"

:\Users\vikra\Desktop\sem6 minipro>javac Client.java
Note: Client.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

:\Users\vikra\Desktop\sem6 minipro>java Client
```

Figure 13 1st Screenshot of Client Program



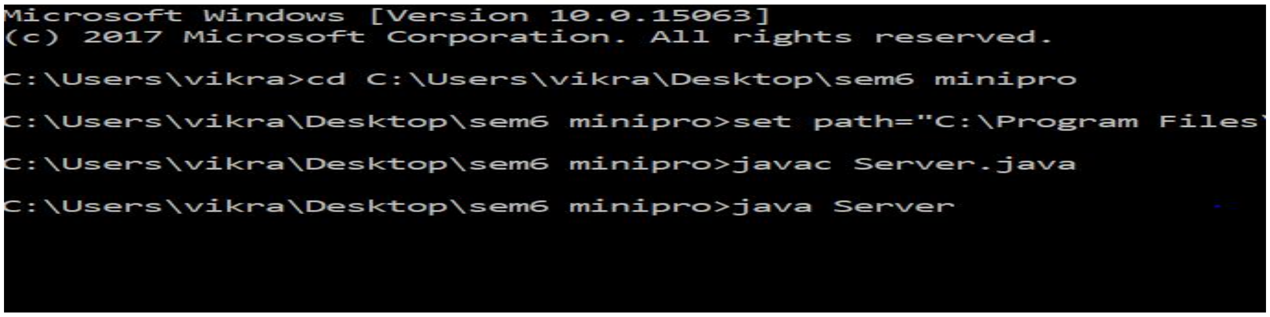
```
Command Prompt - java  Client

icrosoft Windows [Version 10.0.15063]
c) 2017 Microsoft Corporation. All rights reserved.

:\Users\vikra>cd C:\Users\vikra\Desktop\sem6 minipro

:\Users\vikra\Desktop\sem6 minipro>set path="C:\Program Files\Java\jdk-9.0.4\bin"

:\Users\vikra\Desktop\sem6 minipro>javac Client.java
ote: Client.java uses or overrides a deprecated API.
ote: Recompile with -Xlint:deprecation for details.

:\Users\vikra\Desktop\sem6 minipro>java Client
onnected
```

Figure 14 2nd Screenshot of Client Program



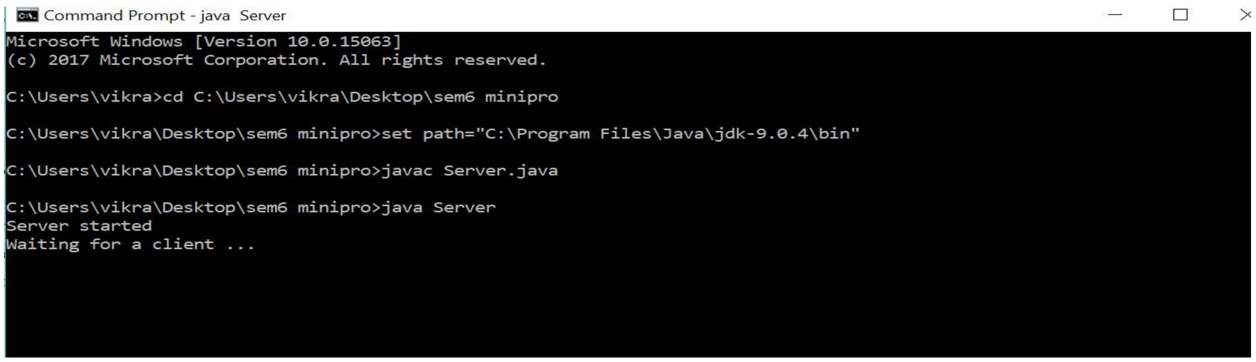
```
Command Prompt - java  Client

Exception in thread "main" java.lang.NullPointerException
        at Client.<init>(Client.java:43)
        at Client.main(Client.java:67)

C:\Users\vikra\Desktop\sem6 minipro>javac Client.java
Note: Client.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

C:\Users\vikra\Desktop\sem6 minipro>java Client
Connected
enter the string
goood should not be accepted
```

Figure 15 3rd Screenshot of Client Program



```
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\vikra>cd C:\Users\vikra\Desktop\sem6 minipro

C:\Users\vikra\Desktop\sem6 minipro>set path="C:\Program Files\

C:\Users\vikra\Desktop\sem6 minipro>javac Server.java

C:\Users\vikra\Desktop\sem6 minipro>java Server
```

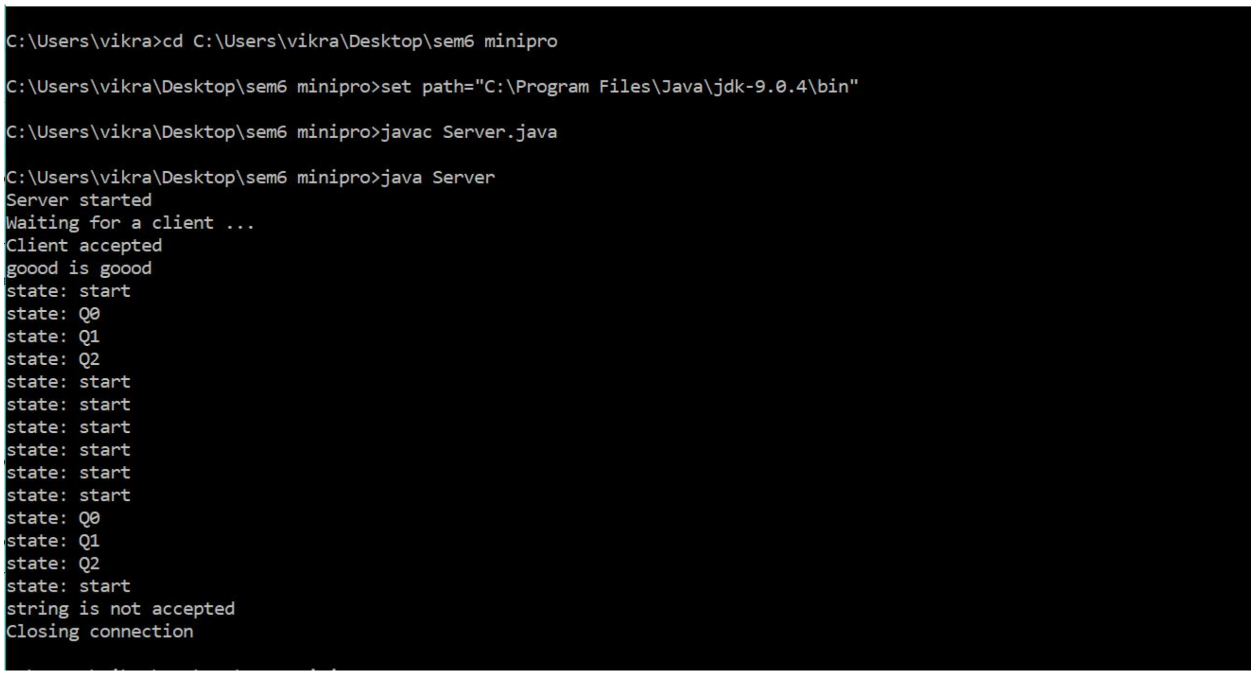
Figure 16 1st Screenshot of Server Program



Command Prompt - java Server

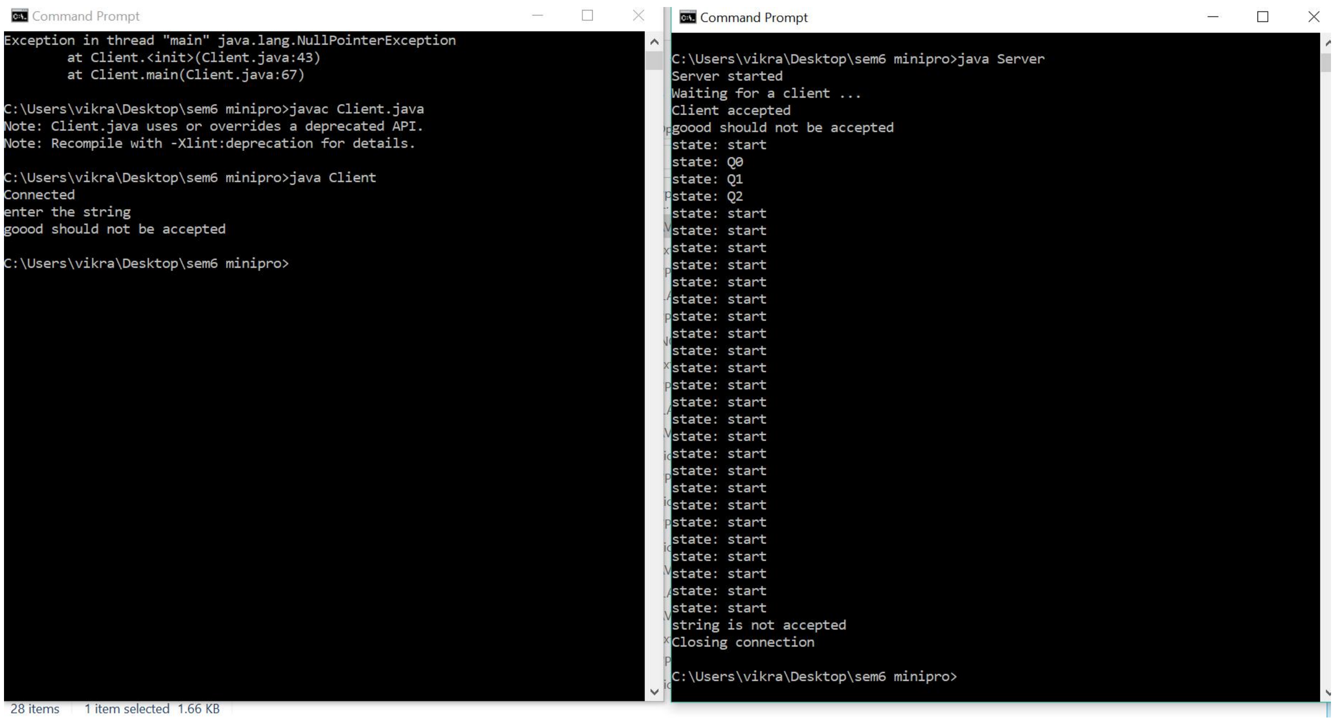
```
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\vikra>cd C:\Users\vikra\Desktop\sem6 minipro

C:\Users\vikra\Desktop\sem6 minipro>set path="C:\Program Files\Java\jdk-9.0.4\bin"

C:\Users\vikra\Desktop\sem6 minipro>javac Server.java

C:\Users\vikra\Desktop\sem6 minipro>java Server
Server started
Waiting for a client ...
```

Figure 17 2nd Screenshot of Server Program



```
C:\Users\vikra>cd C:\Users\vikra\Desktop\sem6 minipro

C:\Users\vikra\Desktop\sem6 minipro>set path="C:\Program Files\Java\jdk-9.0.4\bin"

C:\Users\vikra\Desktop\sem6 minipro>javac Server.java

C:\Users\vikra\Desktop\sem6 minipro>java Server
Server started
Waiting for a client ...
Client accepted
goood is goood
state: start
state: Q0
state: Q1
state: Q2
state: start
state: start
state: start
state: start
state: start
state: start
state: Q0
state: Q1
state: Q2
state: start
string is not accepted
Closing connection
```

Figure 18 3rd Screenshot of Server Program



Command Prompt

```
Exception in thread "main" java.lang.NullPointerException
        at Client.<init>(Client.java:43)
        at Client.main(Client.java:67)

C:\Users\vikra\Desktop\sem6 minipro>javac Client.java
Note: Client.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

C:\Users\vikra\Desktop\sem6 minipro>java Client
Connected
enter the string
goood should not be accepted

C:\Users\vikra\Desktop\sem6 minipro>
```

28 items   1 item selected 1.66 KB

Command Prompt

```
C:\Users\vikra\Desktop\sem6 minipro>java Server
Server started
Waiting for a client ...
Client accepted
goood should not be accepted
state: start
state: Q0
state: Q1
state: Q2
state: start
state: start
state: start
state: start
state: start
state: start
state: start
state: start
state: start
state: start
state: start
state: start
state: start
state: start
state: start
state: start
state: start
state: start
state: start
state: start
state: start
state: start
state: start
state: start
state: start
string is not accepted
Closing connection

C:\Users\vikra\Desktop\sem6 minipro>
```

Figure 19 Final output

CHAPTER 5

REQUIREMENT SPECIFICATION

5.1 HARDWARE SPECIFICATION

Processor	:	Pentium-III
Memory	:	128MB
Hard Disk	:	20GB

5.2 SOFTWARE SPECIFICATION

Operating System	:	WINDOWS, LINUX, FEDORA
Editor	:	Emacs, Vim, Sublime
Compiler	:	JAVA JDK 1. 8.0

CHAPTER 6

CONCLUSION

The objective of the project is achieved that was to build a Client- Server program to implement a DFA. In order to achieve this, I have used socket programming. The Program display each and every state of the strings goes through in a given DFA. And, it displays whether the given string is accepted by the DFA.