king Abdulaziz University
Faculty of Computing and Information Technology
Computer Science Department

CLIENT-SERVER KNOWLEDGE GAME

CPCS 371-COMPUTER NETWORKS 1
DR. OHOUD ALZAMZAMI

Group Members

Student Name	*ID*	*Section*
Hanadi Abdulrahim alsulami	*1706941*	*CAR*
Esraa Abdulrahim	*1707167*	*EAR*
Salwa Abbara	*1780293*	*EAR*
Mona Hafez	*1780735*	*EAR*
Aya Kazzaz	*1780367*	*EAR*

Table of Contents

Group Members *1*

1. Introduction : *4*

1.1. Discussing client – server applications 4

1.2. java TCP socket : 4

1.3. Explaining why we used threads in the game: 5

1.4. brief overview of GUI selected elements: 6

JFrame 6

JTextField to put the answer in it 6

1.5. Describing the knowledge quiz game : 6

1.6. Selected course and chapter material for the knowledge quiz game: 7

1.7. overview of the of the remaining report sections: 7

2. Knowledge game interaction diagram: *7*

2.1. Demonstrates the interaction between the server and the two clients 7

.2.2 Socket and thread pseudocode 9

3. Knowledge game implementation: *10*

4. Application run snapshots: *22*

5. Teamwork and Lessons learned: *24*

5.1. 24

5.2. What are the difficulties encountered? And how did you overcome them? 24

5.3. What did you learn from this project? 25

6. Conclusion: *25*

7. Appendix: *26*

8. Reference: *27*

Table of Figures :

FIGURE 1:INTERACTION BETWEEN THE SERVER AND THE TWO CLIENTS 8
FIGURE 2: SOCKET AND THREAD PSEUDOCODE 9
FIGURE 3: FIRST RUN PART 1 22
FIGURE 4: FIRST RUN PART 2 22
FIGURE 5: FIRST RUN PART 3 23
FIGURE 6: SECOND RUN PART 1 23
FIGURE 7: SECOND RUN PART 2 24

1. Introduction :

1.1.Discussing client – server applications

The Client-Server Knowledge Game is based on the client-server architecture. Which is a distributed application structure that divides workloads between servers and client where server provides resource or service to the client that request this service. In our game one host will work as a server that create multiple quiz games and the communication between the clients(two player), monitor the progress of the quiz and send error message if some rules of the game are violated and two other hosts will work as client that connect to the server to join and start the game session.

The server and two clients could be different processes on the same host.

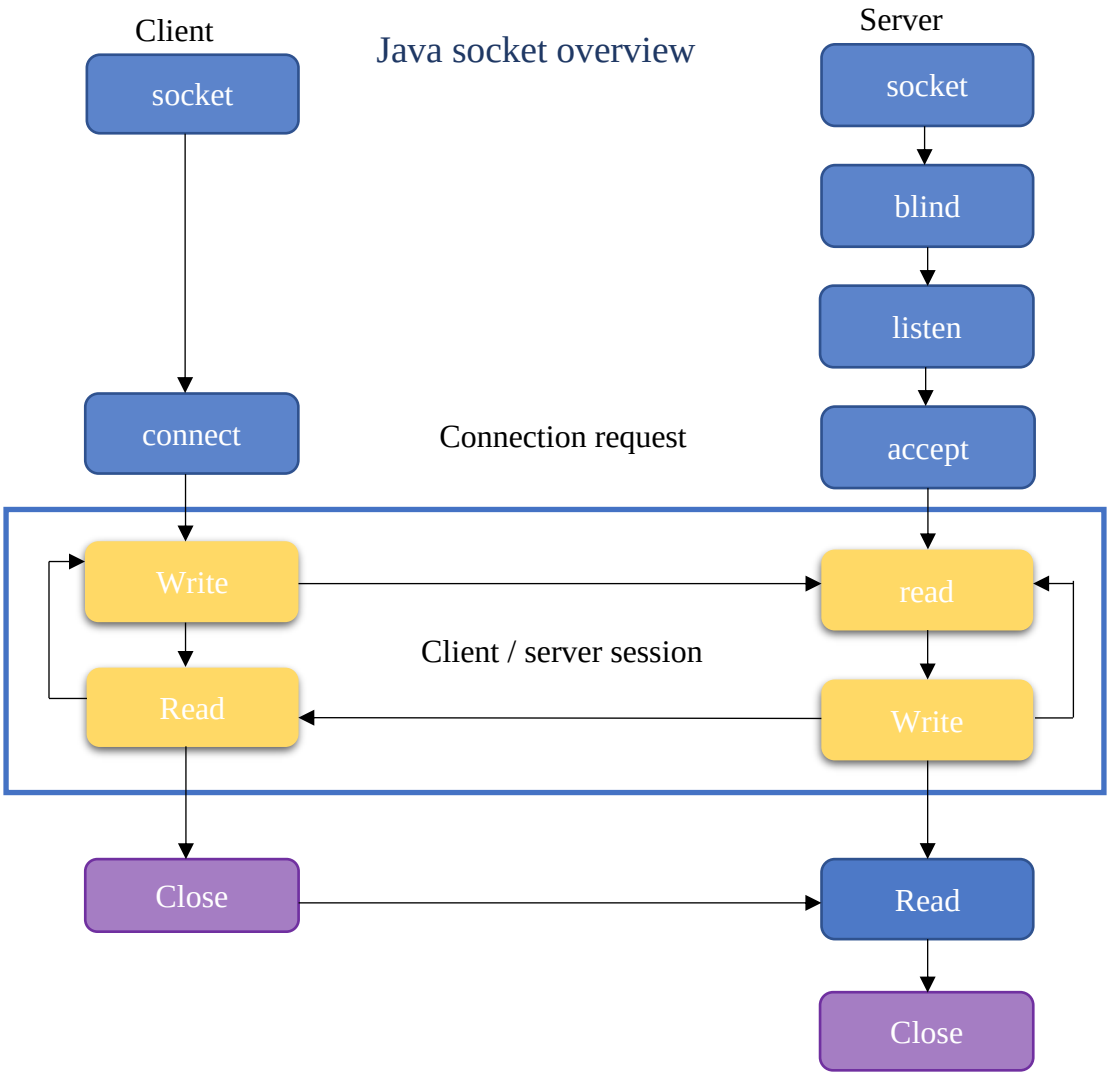
1.2. java TCP socket :

The term *socket* programming refers to writing programs that execute across multiple computers in which the devices are all connected to each other using a network.

There are two communication protocols that one can use for socket programming: User Datagram Protocol (UDP) and Transfer Control Protocol (TCP)**.**

The main difference between the two is that UDP is connectionless, meaning there is no session between the client and the server while TCP is connection-oriented, meaning an exclusive connection must first be established between client and server for communication to take place.

Check the figure below

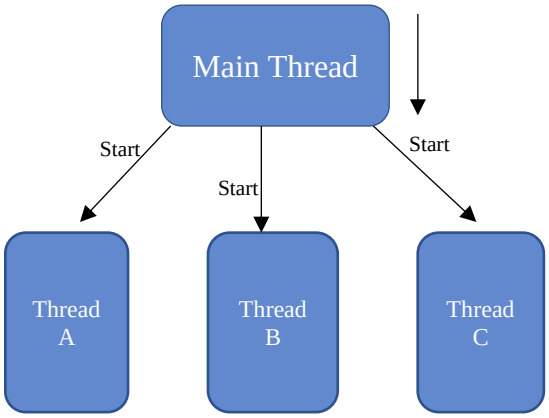


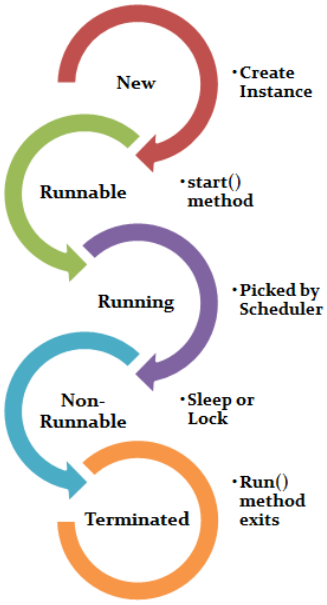
By definition, a socket is one endpoint of a two-way communication link between two programs running on different computers on a network. A socket is bound to a port number so that the transport layer can identify the application that data is destined to be sent to, and that's why we needed TCP connection, so we can play the game in across multiple computers in which the devices are all connected to each other using a network.

1.3.Explaining why we used threads in the game:

Multithreading is a Java feature that allows parallel or concurrent execution of parts of a program to get the max utilization of CPU as shown in figure 1 and figure 2 . Each part of such program is called a thread.

The server creates sockets and accepts connection from every two players to form a session ,each session is a thread that communicate with the two players and determines the statues of the game the server can establish any number of sessions .





In general , we can create the threads by using two mechanisms :

1. Extends the Thread class.
2. Implements the Runnable Interface.

In quiz Game ,the Server implements Extends the Thread class and When the game is started , the server Creates a thread by a new class that extends **Thread** class and create an instance of that class. The extending class must override **run()** method which is the entry point of new thread.

1.4.brief overview of GUI selected elements:

JFrame
JTextField to put the answer in it
JButton when you click on it the answer sent to server
JPanel
TextArea

1.5. Describing the knowledge quiz game :

The game start when the server creating a session for two players (server can run multiple sessions with each session having two players). The game consists of five rounds (one round in the game consist). In each round, the server sends a question to the first player and wait to receive her answer. When the player answers the question, the server sends feedback to both players. Next, server sends the following question to the second player, and so on.

The game server calculates the points for each player based on their answers (one point for each correct answer)The game end after finishing the fifth round or when one player answering

three of five question true while the other player answering three of five questions false and the remaining two questions less than the answered question.

1.6. Selected course and chapter material for the knowledge quiz game:

The chapter that we made questions of is CPU scheduling of operating system which is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.

Process scheduling is an essential part of a Multiprogramming operating systems. Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

The server will randomly initialize a list of five question out of 10 questions that the game has, to one client and another five question to the second client and the client will answer each question and gets points if the answer was true.

1.7. overview of the of the remaining report sections:

Later, in section 2 the knowledge game interaction diagram between the server and the two clients (players) will be presented. Also, in section 3 Knowledge game implementation will be explained. Later in section 4 the application run snapshots will be added and explained. And in section 5 the teamwork and lessons learned will be showed. Finally, the conclusion will be placed in section 6 and appendix in section 7.

2. Knowledge game interaction diagram:

2.1.Demonstrates the interaction between the server and the two clients

The following diagram demonstrates the interaction between the server and the two clients (players). Starting from establishing the connection between the two clients, then displaying the questions respectively between the players and receiving answers, in addition to keeping the score of each individually to determine the state of winning and tie.

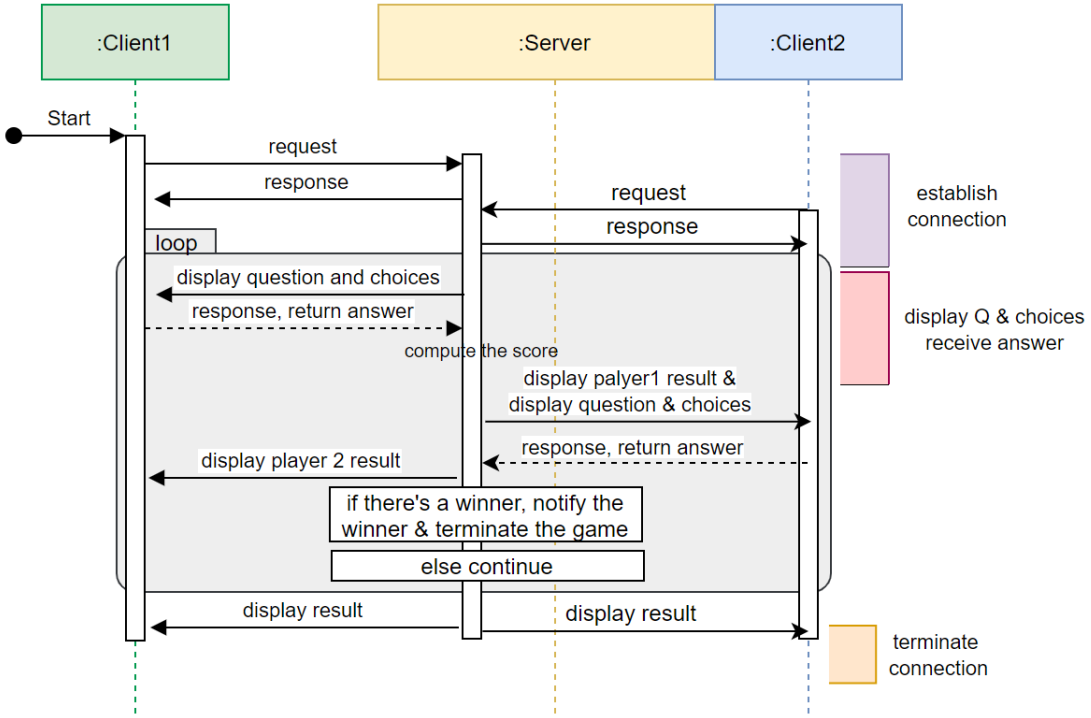


Figure 1:Interaction between the server and the two clients

2.2. Socket and thread pseudocode

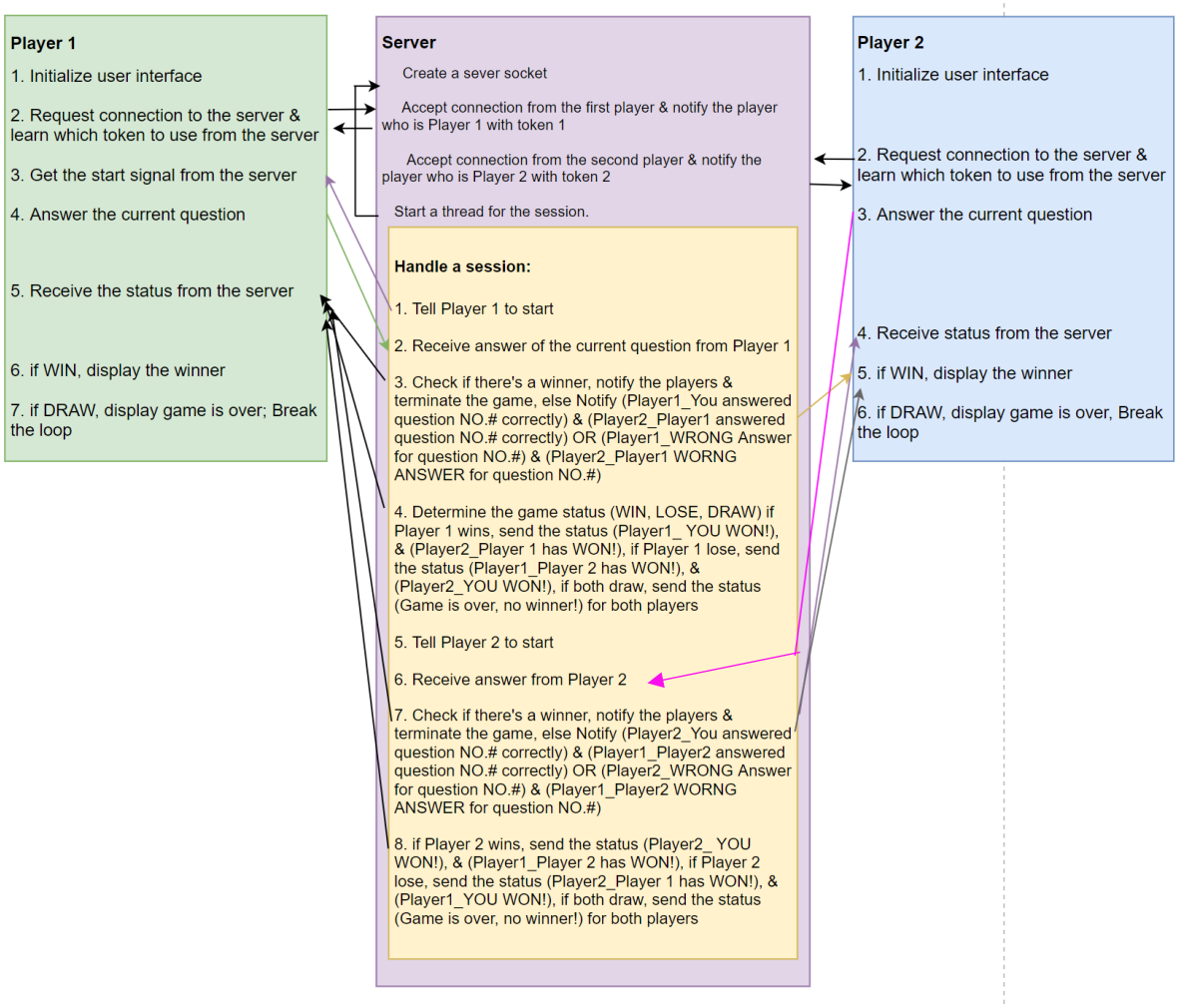


Figure 2: Socket and thread pseudocode

3. Knowledge game implementation:

Server

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */


import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Font;
import java.awt.TextArea;
import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.*;
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;


public class knowledgeGameServer
{

    /**
     * @param args the command line arguments
     * @throws java.lang.Exception
     */

    static BufferedReader input1;
    PrintWriter output1;

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

        ServerSocket listener = new ServerSocket(8050); //TCP socket
        int session = 1; //count number of session
```

```
        Date date = new Date();


        TextArea taLog = new TextArea(date + ":Server started at socket " + listener.getLocalPort()
    + "\n");


        java.io.File file = new java.io.File("CPCS361QuizCH5.txt");
        Scanner input = new Scanner(file);
        String question = "";
        //quiz array contain the quiz file
        String[][] quiz = new String[10][2];
        int j = 0;
        int k = 1;
        int n = 0;
        while (input.hasNext())
        {
           question = input.nextLine();
           if (!(k % 2 == 0))
           {
              quiz[j][0] = question;
           }
           else
           {

              quiz[j][1] = question;
              j++;
           }
           k++;
        }
        try
        {
           int counter = 0;
           while (true)
           {
              Game game = new Game(); //object of game
              ImageIcon icon = new ImageIcon("icon4.jpg");
              game.frame.setIconImage(icon.getImage());
              //GUI elements textArea,frame,panel,
              game.area.setFont(game.f);
              game.panel.setBackground(new Color(164, 209, 242));
              game.frame.getContentPane().setBackground(Color.LIGHT_GRAY);
              game.frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
              game.frame.setSize(800, 800);
              game.frame.setVisible(true);
              game.frame.setResizable(false);
```

```
            game.panel.add(game.area);
            game.frame.add(game.panel, BorderLayout.NORTH);
            game.frame.setVisible(true);
            if (counter == 0)
            {
               game.area.append(date + ":Server started at socket " + listener.getLocalPort() +
"\n");
            }
            game.takefile(quiz); //method in game take quiz file
            game.area.append("Wait for players to join session " + session + "\n");
            //Listen for a new connection request
            Socket socket1 = listener.accept();
            Game.Player player1 = game.new Player(socket1, '1');
            game.area.append(date + ": player " + player1.mark + " joined session " + session +
"\n");
            InetAddress ip = socket1.getInetAddress();
            game.area.append("player " + player1.mark + "'s IP address " + ip.getHostAddress() +
"\n");
            player1.output.println("waiting for palyer 2 to join!");
            Socket socket2 = listener.accept(); //Listen for a new connection request
            InetAddress ip2 = socket2.getInetAddress();
            Game.Player player2 = game.new Player(socket2, '2'); //after connecting player 2 the
thread is start for specific session
            game.area.append(date + ": player " + player2.mark + " joined session " + session +
"\n");
            game.area.append("player" + player2.mark + "'s IP address " + ip2.getHostAddress() +
"\n");
            game.area.append(date + ": Start a thread for session" + session + "\n");

            session++;
            //so we can make switches
            player1.setOpponent(player2);
            player2.setOpponent(player1);
            player2.output.println("waiting for palyer 1 to start the game");
            player2.opponent.output.println("Player 2 has joined. You start first!");
            game.currentPlayer = player1; //let game start with player1

            player1.start(); //run when all clients connected
            player2.start();
            counter++;

         }
      }
      finally
      {
         listener.close();
      }
```

```
    }
}

class Game
{
    static TextArea area = new TextArea(35, 90);
    static Font f = new Font("Serif", Font.PLAIN, 15);
    static JFrame frame = new JFrame("Knowledge Game Server");

    static JPanel panel = new JPanel();
    String quizFile[][];
    Player currentPlayer;
    final int count = 9; //number of question-1
    int counter = 0;
    //method that takes TextArea
    public void txt(TextArea area)
    {
        this.area = area;
    }
    //method that takes quiz file
    public void takefile(String file[][])
    {
        this.quizFile = file;

    }
    //check if the answer correct or not
    public boolean checkAnswer(String choice, int index)
    { //check if the answer true
        if (choice.equals(quizFile[index][1]))
        {
            return true;
        }
        else
        {
            return false;
        }
    }
    //check if the client choose  legal answer
    public synchronized boolean legalChoice(Player player, String choice, int counter)
    {

        if (player == currentPlayer )
        {
            currentPlayer.answerdQuestion++;
            if (checkAnswer(choice, counter))
            {
                player.output.println("you Answered question No." + (counter + 1) + "correctly!");
```

```
                currentPlayer.numberOfTrue++;
            }
            else
            {
                player.output.println("WRONG Answer for question No." + (counter + 1));
            }
            currentPlayer.otherPlayerMoved(choice, counter);
            return true;

        }
        return false;
    }
    //  The class for the helper threads in this multithreaded server to handel multiple session at the
same time
    //  application. A Player is identified by a character mark
    //  which is either '1' or '2'. For communication with the
    //  client the player has a socket with its input and output
    //  streams. Since only text is being communicated we use a
    //  Reader and a writer
    class Player extends Thread
    {

        char mark;
        Player opponent;
        Socket socket;
        int answerdQuestion = 0; //count to count the number of answered question
        BufferedReader input;
        PrintWriter output;
        int numberOfTrue = 0; //count to count the number of true



        //Constructs a handler thread for a given socket and mark
        public Player(Socket socket, char mark)
        {
            this.socket = socket;
            this.mark = mark;
            try
            {

                input = new BufferedReader(new InputStreamReader(this.socket.getInputStream()));
                output = new PrintWriter(this.socket.getOutputStream(), true);
                Date date = new Date();
                output.println(date + " Connecting to server");
                output.println("you are player " + this.mark);
            }
            catch (IOException e)
```

```
            {

            }
        }
        //method to switch turn between players
        public synchronized void switchPlayers(Player player)
        {
            player.opponent.resume();
            currentPlayer = currentPlayer.opponent; ///update current player and give the turn to the
opponent
            player.suspend();
        }
        //method to print the feedback from server to clients
        public void otherPlayerMoved(String choice, int counter)
        {

            currentPlayer.opponent.output.println(checkAnswer(choice, counter) ? "player " +
currentPlayer.mark + " answered question No." + (counter + 1) + "Correctly!" : "player " +
currentPlayer.mark + "WRONG ANSWER for question No." + (counter + 1));

        }
        //method to set oppenent
        public void setOpponent(Player opponent)
        {
            this.opponent = opponent;
        }
        //method to determine the winner or the tie and end the game
        public synchronized boolean endGame(Player player, int counter)
        {
            if (counter == 10)
            {
                if (player.numberOfTrue > player.opponent.numberOfTrue)
                { //if the number of true for current player greater than his opponent
                    player.output.println("you WON!");
                    player.opponent.output.println("player" + player.mark + "has WON!");
                }
                if (player.numberOfTrue == player.opponent.numberOfTrue)
                { //when both player have same score
                    player.output.println("Game is over,no winner!");
                    player.opponent.output.println("Game is over,no winner!");
                }
                return true;
            }
            else if (player.opponent.numberOfTrue > (player.numberOfTrue + (5 -
player.answerdQuestion)))
            {
```

```
            //if the first one answered 3 true question and his opponent answer 3  false question
then the game end with player one as a winner
               player.opponent.output.println("you WON!");
               player.output.println("player " + player.opponent.mark + " has WON!");
               return true;
            }
            else if (player.numberOfTrue > (player.opponent.numberOfTrue + (5 -
player.opponent.answerdQuestion)))
            {
               //if the first one answered 3 false question and his opponent answer 3  true question
then the game end with player one as a winner
               player.output.println("you WON!");
               player.opponent.output.println("player " + player.mark + " has WON!");
               return true;
            }

            return false;

         }
         //The run method of this thread
         @Override
         public void run()
         {


            String answer = "";
            try
            {
               currentPlayer.opponent.suspend();
               while (true)
               {
                  if (currentPlayer.opponent == null || currentPlayer == null )
                  {
                     counter = 0;
                     break;
                  }
                  if(counter!=10 &&!(currentPlayer.numberOfTrue >
(currentPlayer.opponent.numberOfTrue + (5 - currentPlayer.opponent.answerdQuestion)))){
                  area.append("Now it is player " + this.mark + " turn!\n");

                    currentPlayer.output.println("Its now your turn!");
                    currentPlayer.opponent.output.println("Its now player " + currentPlayer.mark + "
turn!");
                  }
                  else
                     break;
                  String s = quizFile[counter][0].replace(" ", "\n");
```

```
                output.println(s.replace("*", " "));
                answer = input.readLine(); //take the answer from the client
                legalChoice(this, answer, counter);
                counter = counter + 1;
                endGame(this, counter);


                switchPlayers(this);

            }

        }
        catch (IOException e)
        {}
        finally
        {
            try
            {
                input.close();
                output.close();
                socket.close();
            }
            catch (IOException e)
            {}
        }
    }
  }
}



/*

* To change this license header, choose License Headers in Project Properties.
* To change this template file, choose Tools | Templates
* and open the template in the editor.
*/
```

```
Client
*/
.To change this license header, choose License Headers in Project Properties *
To change this template file, choose Tools | Templates *
.and open the template in the editor *
/*

**/
*
author WinDows@ *
/*
;*.import java.awt
;*.import java.awt.event
;*.import java.io
;import java.net.InetAddress
;import java.net.Socket
;*.import javax.swing


public class knowledgeGameClient extends JFrame implements ActionListener
}
GUI elements textArea,frame,panel,JTextField,JLabel,JButton//
;private JFrame frame = new JFrame("knowledgeGameClient")
;private JLabel messageLabel = new JLabel("choose the correct answer")
;private JButton answer1
;private JButton answer2
;private JButton answer3
;private JButton answer4
;private JPanel panel2
;private JPanel panel3
;private JPanel intro
;private JPanel panel
;(70 ,35)TextArea area = new TextArea
;Font f = new Font("Serif", Font.PLAIN, 15)
;"" = String answer
private JLabel fixedTitle; //on the top of game
private static int PORT = 8050; //prefixed in both side
;private Socket socket
; private BufferedReader in
;private PrintWriter out


public knowledgeGameClient(String serverAddress) throws Exception
}
```

```
socket = new Socket(serverAddress, PORT); //initiate TCP connection to connect with
server for playing the Knowledge Game quiz
in = new BufferedReader(new InputStreamReader(socket.getInputStream())); //used for
receving information from server
out = new PrintWriter(socket.getOutputStream(), true); //used for sending information
from client to server
GUI element//
;ImageIcon icon = new ImageIcon("icon4.jpg")
;frame.setIconImage(icon.getImage())
;fixedTitle = new JLabel("Welcome to online based quiz!")
;fixedTitle.setFont(f)
;fixedTitle.setForeground(Color.BLACK)
;("-")answer1=new JButton
;("-")answer2=new JButton
;("-")answer3=new JButton
;("-")answer4=new JButton
;()panel2 = new JPanel
;()panel3 = new JPanel
;()intro = new JPanel
;()panel = new JPanel
;(200,200)panel.setSize
;intro.setBackground(Color.LIGHT_GRAY)
;panel.setBackground(Color.LIGHT_GRAY)
;panel.add(fixedTitle)
;frame.add(panel, BorderLayout.PAGE_START)
;area.setFont(f)
;panel2.setBackground(new Color(164, 209, 242))
;panel3.setBackground(new Color(164, 209, 242))
;(500,500)panel3.setSize
;(60)answer1.setAlignmentY
;answer1.setBackground(Color.LIGHT_GRAY)
;(60)answer2.setAlignmentY
;answer2.setBackground(Color.LIGHT_GRAY)
;(60)answer3.setAlignmentY
;answer3.setBackground(Color.LIGHT_GRAY)
;(60)answer4.setAlignmentY
;answer4.setBackground(Color.LIGHT_GRAY)
;frame.add(panel2, BorderLayout.NORTH)
;messageLabel.setFont(f)
;messageLabel.setForeground(Color.BLACK)
;panel3.add(messageLabel)
;panel3.add(answer1)
;answer1.setPreferredSize(new Dimension(60,60))
;panel3.add(answer2)
;answer2.setPreferredSize(new Dimension(60,60))
```

```
;panel3.add(answer3)
;answer3.setPreferredSize(new Dimension(60,60))
;panel3.add(answer4)
;;answer4.setPreferredSize(new Dimension(60,60))


;frame.add(intro, BorderLayout.SOUTH)
;answer1.addActionListener(this)
;answer2.addActionListener(this)
;answer3.addActionListener(this)
;answer4.addActionListener(this)

;frame.add(panel2)
;frame.add(panel3,BorderLayout.SOUTH)
;(800 ,600)frame.setSize
;frame.setVisible(true)
;frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)
{
Override@
public void actionPerformed(ActionEvent e)
}
answer = e.getActionCommand(); // a global variable
;out.println(answer)
{
The main thread of the client will listen for messages  //
message in which we receive our mark. Then we go into a //
.loop listening for response from the server //
when the the play() end the client exit from the server //
public void play() throws Exception
}
try
}

while (true)
}
;()String response = in .readLine
}if(response.endsWith("turn!"))
;("1")answer1.setText
;("2")answer2.setText
;("3")answer3.setText
;("4")answer4.setText
{

if (response.contains("?"))
}
```

```
;area.append(response + "\n")

;()String response1 = in .readLine
;()String response2 = in .readLine
;()String response3 = in .readLine
;()String response4 = in .readLine



area.append(response1 + "\n" + response2 + "\n" + response3 + "\n" +
;response4 + "\n")

{
else
}
;area.append(response + "\n")

{

;panel2.add(area)
;frame.add(panel2)
;frame.setLayout(null)


if (response.endsWith("WON!") || response.endsWith("no winner!") ||
response.endsWith("loss"))
}
;out.println("end")
;answer1.setVisible(false)
;answer2.setVisible(false)
;answer3.setVisible(false)
;answer4.setVisible(false)
;messageLabel.setText("Game end")
;break
{
{
{
finally
}
;()in .close
;()out.close
;()socket.close
{
{
**/
```

```
param args the command line arguments@ *
/*
run the client//
public static void main(String[] args) throws Exception
}

;[1]String serverAddress = (args.length == 0) ? "localhost" : args

;knowledgeGameClient client = new knowledgeGameClient(serverAddress)
;()client.play

{
```

4. Application run snapshots:

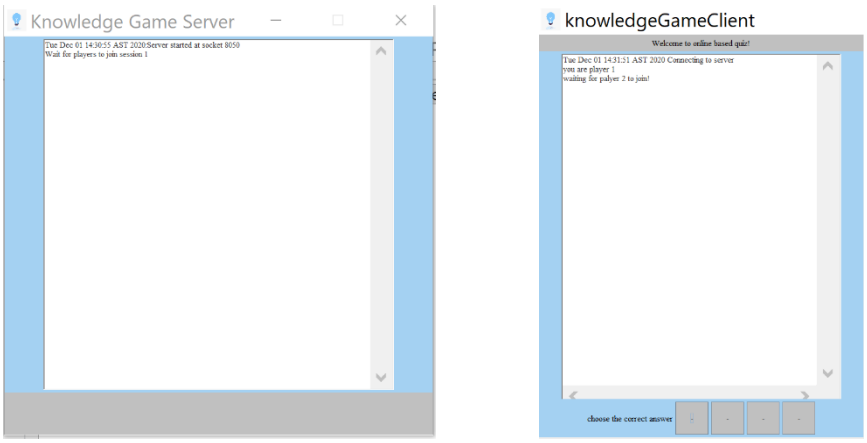


Figure 3: FIRST RUN PART 1

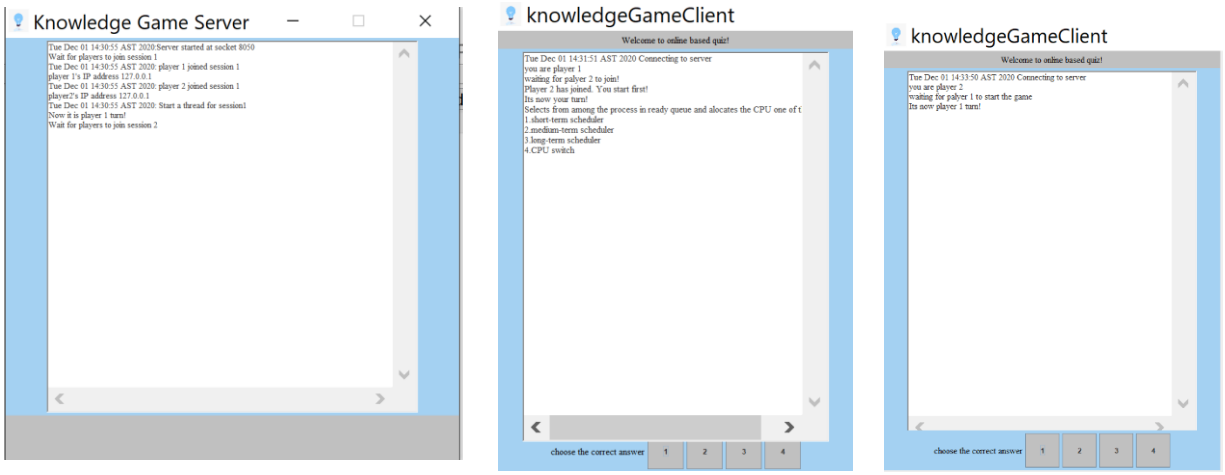


Figure 4: FIRST RUN PART 2

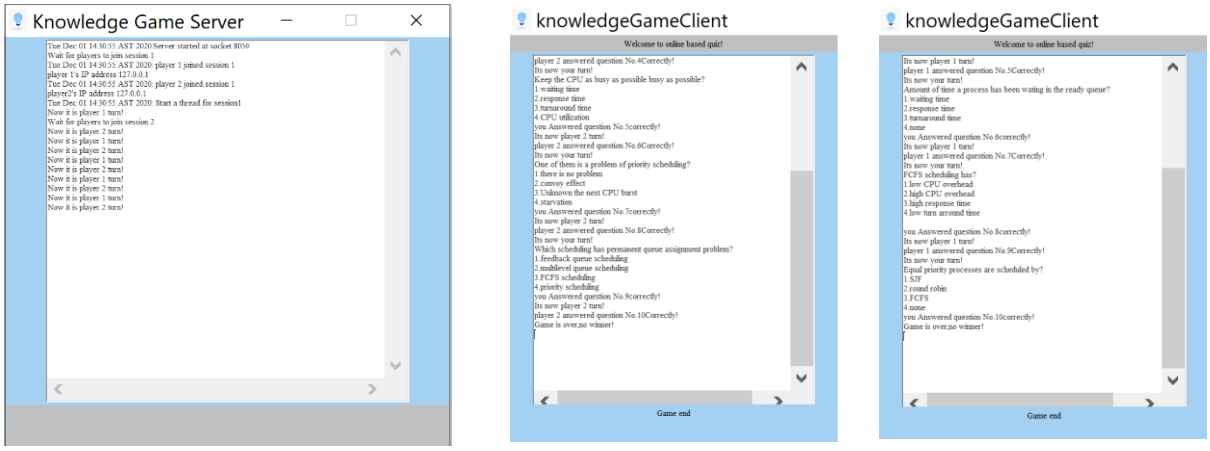


Figure 5: FIRST RUN PART 3

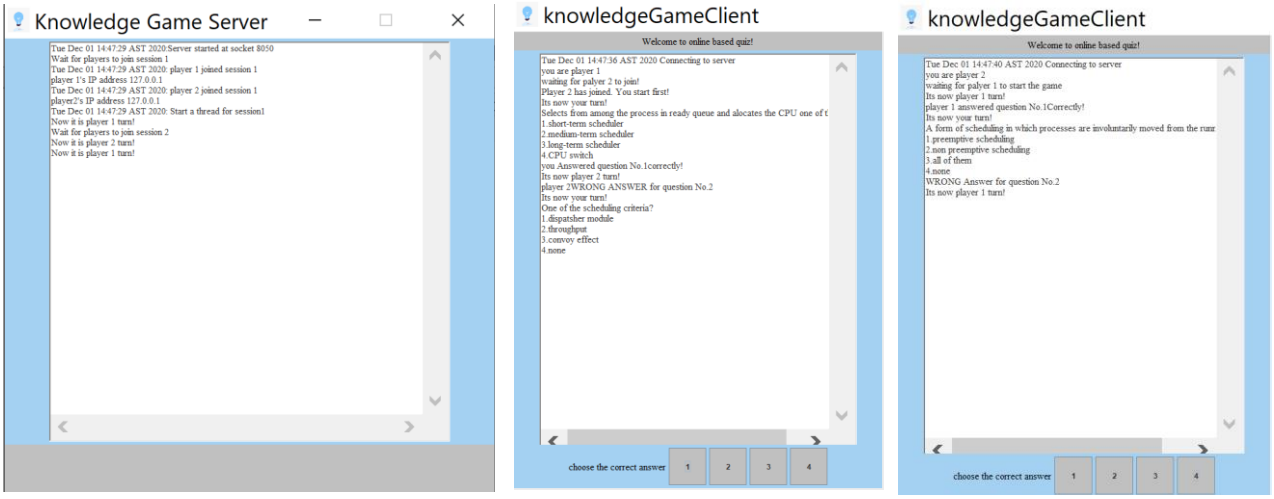


Figure 6: SECOND RUN PART 1

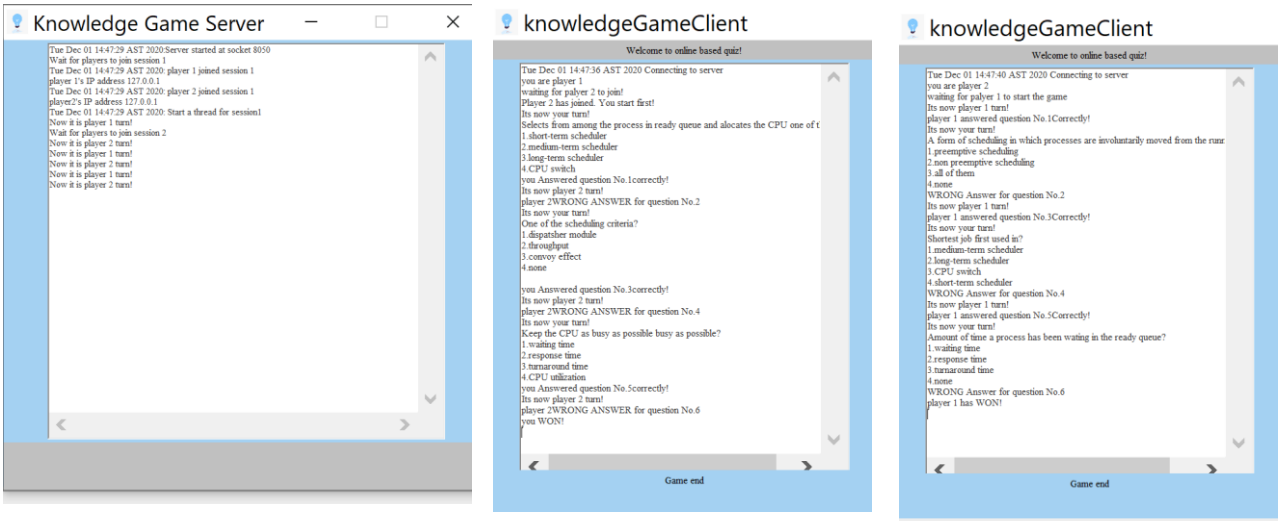


Figure 7: SECOND RUN PART 2

5. Teamwork and Lessons learned:

5.1. How the work is divided on the group members:

We all as a group helped each other whenever any of us faced a problem through this project, but we did divide the project work equally based on the difficulty.

Student name	The Work
Hanadi	***Participated in writing the report, covered topics no 1.2,1.3,1.4,1.6, 5,6***
Mona	***Participated in both coding and writing the report, covered 1.1,1.5,1.7***
Aya	***Participated in coding and correct the report***
Salwa	***Participated in coding and implement the GUI***
Esraa	***Participated in both coding and writing the report, covered topic no 2***

5.2. What are the difficulties encountered? And how did you overcome them?

The most difficult thing we have faced is writing the code and making it Run in a right way without any errors, also covering some topics was kind of hard because most of them were new to us, we did overcome them by searching and learning more about socket programming.

5.3. What did you learn from this project?

Socket programming more specifically java TCP socket and the basics of client-server Internet applications.

6. Conclusion:

The quiz game can be implemented to play it between different hosts (devices) by running in one device to be the server then running at any other two devices to be the clients , clients play against each other. That is applicable by using socket programming.

Socket programming is a way of connecting two nodes on a network to communicate with each other, One socket(node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the server.

7. Appendix:

Quiz file

Selects*from*among*the*process*in*ready*queue*and*alocates*the*CPU*one*of*them? 1.short-term*scheduler 2.medium-term*scheduler 3.long-term*scheduler 4.CPU*switch
1
A*form*of*scheduling*in*which*processes*are*involuntarily*moved*from*the*running*state? 1.preemptive*scheduling 2.non*preemptive*scheduling 3.all*of*them 4.none
1
One*of*the*scheduling*criteria? 1.dispatsher*module 2.throughput 3.convoy*effect 4.none
2
Shortest*job*first*used*in? 1.medium-term*scheduler 2.long-term*scheduler 3.CPU*switch 4.short-term*scheduler
2
Keep*the*CPU*as*busy*as*possible*busy*as*possible? 1.waiting*time 2.response*time 3.turnaround*time 4.CPU*utilization
4
Amount*of*time*a*process*has*been*wating*in*the*ready*queue? 1.waiting*time 2.response*time 3.turnaround*time 4.none
1
One*of*them*is*a*problem*of*priority*scheduling? 1.there*is*no*problem 2.convoy*effect the*next*CPU*burst 4.starvation*3.Unknown
4
FCFS*scheduling*has? 1.low*CPU*overhead 2.high*CPU*overhead 3.high*response*time 4.low*turn*arround*time
1
Which*scheduling*has*permanent*queue*assignment*problem? 1.feedback*queue*scheduling 2.multilevel*queue*scheduling 3.FCFS*scheduling 4.priority*scheduling
2
Equal*priority*processes*are*scheduled*by? 1.SJF 2.round*robin 3.FCFS 4.none
3

8. Reference:

1- Computer Networking: A Top Down Approach, 6th edition, Jim Kurose, Keith Ross Addison- Wesley March 2012.

2- Introduction to Java Programming, Comprehensive, 10th Edition, Y.Daniel Liang.

3- A Guide to Java Sockets, by baeldung, February 12, 2020.