**פרויקט גמר**

**למילוי חלקי של הדרישות לקבלת תואר**

**הנדסאי הנדסת תוכנה**

**בהתמחות: מחשבים**

נושא הפרויקט: נותן מענה בו זמנית למשלוחי פיצה (הזמנת מגש- ולהכין מגש )

שם הסטודנט: ליאל גלזר

העבודה בוצעה בהנחיית: שרה שנהב

ה צ ה ר ת ס ט ו ד נ ט

אני הסטודנט ליאל גלזר מס' ת.ז 213375314 החתום מטה, מצהיר בזאת שכל עבודת הגמר/ הפרויקט המוגש/ת בחוברת זו היינו/ה פרי עבודתי בלבד.

על בסיס הנחייתו של המנחה ותוך הסתמכות על מקורות הידע והמידע האחרים המצויים ביבליוגרפיה המובאת בחוברת זאת.

אני מודע לאחריות שהנני מקבל על עצמי ע"י חתימתי על הצהרה זו שכל הנאמר בה הינו אמת ורק אמת.

חתימת מגיש החוברת : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

אישור המנחה :

הנני מאשר הגשת החוברת להערכה \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

תוכן עניינים

מבוא............................................................................... עמ' 4

סביבת פיתוח.................................................................... עמ' 4

דרישות מערכת מהמשתמש................................................ עמ' 4

UML.............................................................................. עמ' 4

מדריך למשתמש............................................................... עמ' 5-6

בעיות שהסנכרון פותר........................................................ עמ' 6

פירוט מחלקות.................................................................. עמ' 7-8

קטעי קוד......................................................................... עמ' 8-15

מבוא

המטרה היא להזמין ולהכין משלוחי פיצה בו זמנית על בר משותף, עובדים מכניסים פיצות ובמקביל הלקוחות מוצאים מבר -כביכול הבר משותף ליצרנים ולצרכנים

הלקוח יכול להיות או עובד או מזמין , במידה והא מזמין המערכת בודקת אם קיימת פיצה בבר (הבר מוגבל ל-4 פיצות ),במידה וקיימת פיצה -הוא מקבל אותה והבר מתעדכן ומידה ולא מוצגת לו הודעה שכרגע הוא צריך להמתין בסבלנות וברגע שיש מגש מוכן הוא מיד לוקח ,וכן עובד ברגע שהוא מעונין להכניס לבר פיצה נוספת המערכת בודקת האם יש מקום להכניס -אם כן מכניסה ומעדכנת ובמידה ולא מודיעה לעובד שאליו להמתין וברגע שיש מקום המערכת מכניסה את הפיצה

**מטרת הפרויקט** הייתה לפתור את בעיית הצרכן יצרן על ידי בר משותף וסנכרון ביניים .

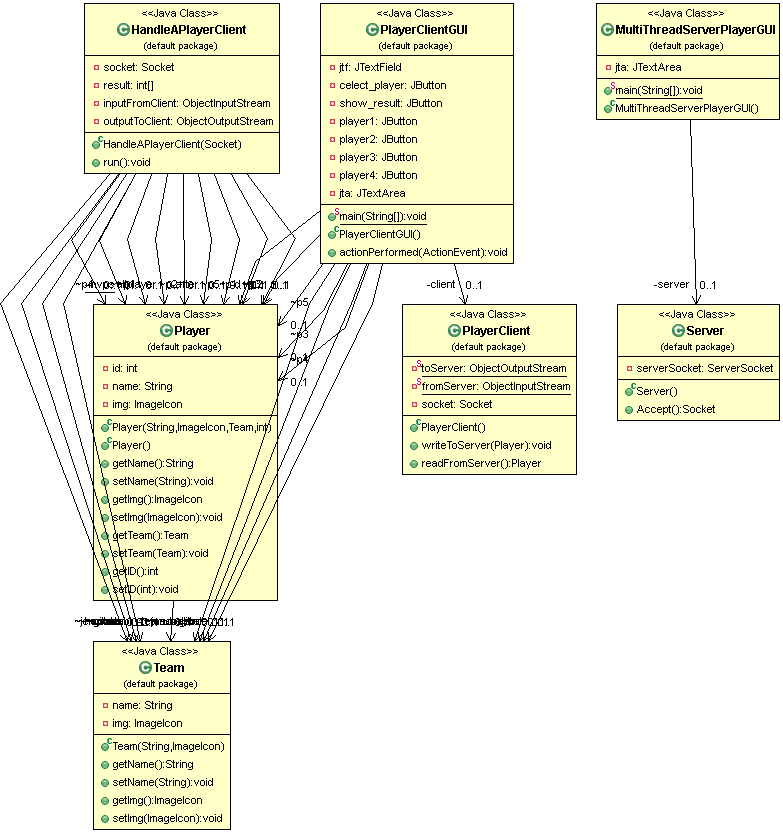
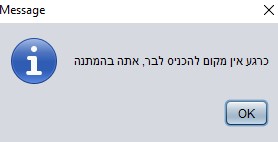
סביבת פיתוח

* Windows 7/10
* Eclipse Java

דרישות מערכת מהמשתמש

* Windows 7/10
* Java

UML



מדריך למשתמש:

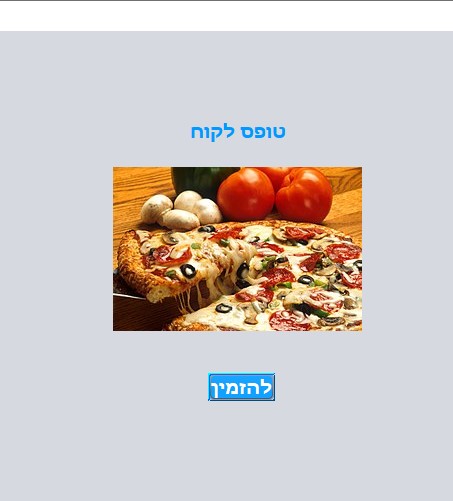
מסך אחד של לקוח ובו:

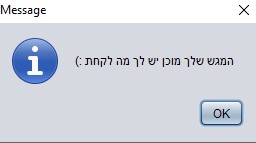
אופציה לבחור את מי אתה עובד או לקוח (יצרן או צרכן )



**ברגע שאתה לוחץ על אחד מהכפתורים הללו , נפתח טופס בהתאם.**

**במידה ואתה לקוח :**

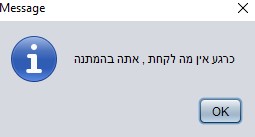


**במידה וקיימת פיצה בבר :**

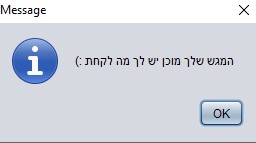
**זאת הודעה על כך שההזמנה בוצעה בהצלחה (והבר כמובן מתעדכן לאחריה)**

**במידה ולא קיימת פיצה בבר :**

**תוצג ההודעה הנ"ל :**

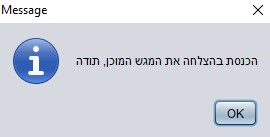


**וברגע שיתווסף לבר פיצה ישר תתבצע ההזמנה תוצג ההודעה :**



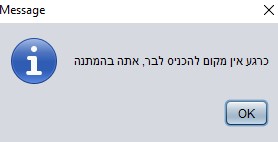
**במידה ואתה עובד :**

**במידה ויש מקום בבר להכניס פיצה אז מוצגת ההודעה :**

****

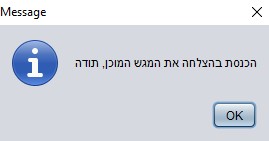
**זאת הודעה על כך הכנסה לבר בוצעה בהצלחה (והבר כמובן מתעדכן לאחריה)**

**במידה ואין מקום להכניס פיצה לבר (הבר מלא) תוצג ההודעה :**



**וברגע שתתבצע הזמנה בהצלחה מהבר – הלקוח ייקח פיצה מהבר**

**ומיכון שעכשיו יש מקום פנוי להכניס פיצה תתבצע ההכנסה של הפיצה מצד העובד תוצג ההודעה :**



בעיות שהסנכרון פותר:

1. לקוח שהוא-עובד אחד לוחץ על "הכנס פיצה לבר" ולקוח אחר שהוא לקוח-לוחץ על "הזמן פיצה מהבר" ואז הבר צריך לסכרן כל הזמן ולהציג הודעות בהתאם

צריך לדאוג לסנכרון שלא ייווצר מצב שגם הצרכן וגם היצרן יעמדו על אותו מקום והלקוח ייקח פיצה שלא קיימת והעובד יכניס פיצה שאין מקום בבר (ממש בעיית הצרכן יצרן )

אחרי שהוא יציג את התוצאות ללקוח עובד או ללקוח מזמין הוא יתפנה ללקוח השני, יכניס את הפיצה או יצאי בתאמם ויסנכרן את הבר ויציג את ההודעות המתאימות.

לכן אני משתמשת פה במחלקה שאני בניתי אותה בעצמי והיא מממשת את bloockQueue וממנה יצרתי מופע שהוא יציג את הבר ( שהוא synchronized)

1. מבצעים בדיקת הכנסה והוצאה , עבור כל הכנסה בודקים האם יש מקום בבר ואם כן מפעילים את הפונקציה שמכניסה לבר פיצה ומודיעה בתאם,

אם לא נעשה את זה בסנכרון האיבר נכניס גם במצב שאין מקום -הבר מלא

וכן אם ההוצאה שלא ייווצר מצב שייקחו פיצה שלא קיימת כי המידה ולא קיימת פיצה תיווצר הודעה אשר מודיעה על המתנה וברגע שתכנס פיצה המזין ייקח אותה עצם הסנכרון אנחנו דואגים שאם יכנסו עוד נתונים בזמן שהוצאנו ובדקנו אותם לא יכנסו ישירות לבר אלא יחכו שהתהליך יסתיים,

פירוט מחלקות:

מחלקת Client

מחלקה זו אחראית על שליחת מידע מהקליינט ושליחת מידע לקליינט ויצירת הקשר עם השרת לקרא לכתוב וכו'.

|  |  |
| --- | --- |
| פירוט | פונקציה |
| פונקציה אשר אחראית לכתוב מהלקוח לשרת | **writeToServer(String s)** |
| פונקציה אשר אחראית לקרא מה חזר מהשרת | **readFromServer()** |

מחלקת Serverb

מחלקה שרת היא השרת עצמו וממתינה שיצרו איתה קשר -ולאחר מכאן פועלת בהתאם .

|  |  |
| --- | --- |
| פירוט | פונקציה |
| פונקציה שמפעילים שממתינה ליצירת הקשר עם הלקוח | **public Serverb()** |

מחלקת ConnectedwithClient

מחלקה אשר מטפלת במידע שהקליינט שולח לסרבר ויורשת מTRED.

|  |  |
| --- | --- |
| פירוט | פונקציה |
| פונקציה אשר מתקבלת ממשק ההליכון. פונקציה זו תמיד מאזינה לבקשות הקליינט. | **public** **void** run() |

מחלקת SynchQueue<String> :

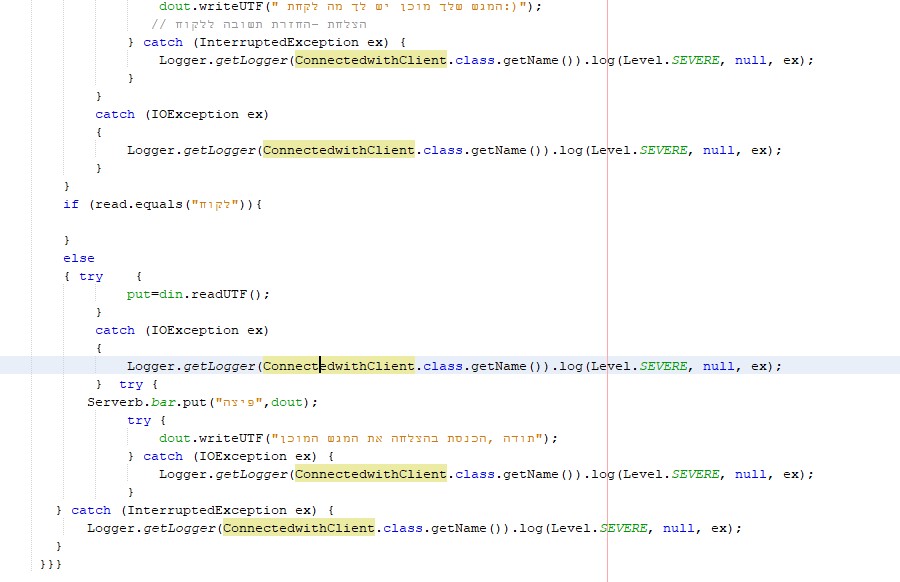
מחלקה אשר מממשת ממש את הרעיון של blockQue כבכבכב

|  |  |
| --- | --- |
| פירוט | פונקציה |
| פונקציה אשר מאתחלת את התור שהוא משמש כבר היא מגדירה לו גודל | **public SynchQueue(int limit)** |
| בודק אם אפשר להכניס מגש פיצה ופועלת בתאם ואם אפשר מכניסה אחרת ממתינה שמישו ייקח פיצה ואז יהיה מקום והיא תכניס את הפיצה הנוכחית ותחזיר הודעה בהתאם | **public synchronized void put(String item,DataOutputStream out)** |
| בודק אם אפשר לקחת מגש פיצה ופועלת בתאם ואם אפשר מוצאיה אחרת ממתינה שמישו יכניס פיצה ואז יהיה מה לקחת והיא תוציא את הפיצה הנוכחית ותחזיר הודעה בהתאם | **public synchronized String take(DataOutputStream out)** |

קטעי קוד

מחלקת :Client





מחלקת Serverb



מחלקת HandleAPlayerClient

**class** HandleAPlayerClient **implements** Runnable {

**private** Socket socket; // A connected socket

**private** **int** result[];

**private** ObjectInputStream inputFromClient;

**private** ObjectOutputStream outputToClient;

**private** Player d;

**private** Player after;

**private** **static** Vector<Player> *mvps*=**new** Vector<Player>();

Team gilboa= **new** Team("gilboa", **new** ImageIcon("img/gilboa.jpg"));

Team hapoal= **new** Team("hapoal tel aviv", **new** ImageIcon("img/hapoal.png"));

Team jerusalem= **new** Team("jerusalem", **new** ImageIcon("img/jerusalem.png"));

Team maccabi= **new** Team("maccabi haifa", **new** ImageIcon("img/maccabi.png"));

Player p1= **new** Player("james feldin", **new** ImageIcon("img/feldin.jpg"), jerusalem,1);

Player p2= **new** Player("tomer ginat", **new** ImageIcon("img/ginat.jpg"), hapoal,2);

Player p3= **new** Player("jastin tilman", **new** ImageIcon("img/tilman.jpg"), gilboa,3);

Player p4= **new** Player("james young", **new** ImageIcon("img/young.jpg"), maccabi,4);

Player p5= **new** Player("", **null**, **null**,0);//for draw

Player[] allplayer=**new** Player[] {p1, p2,p3,p4};

**public** HandleAPlayerClient(Socket socket) {

**this**.socket = socket;

**this**.result=**new** **int**[4];

**for**(**int** i=0; i<4; i++)

**this**.result[i]=0;

}

**public** **void** run() {

**try** {

// Create data input and output streams

outputToClient = **new** ObjectOutputStream(socket.getOutputStream());

inputFromClient = **new** ObjectInputStream(socket.getInputStream());

// Continuously serve the client

**while** (**true**) {

// Receive sentence from the client

**try** {

d = (Player) inputFromClient.readObject();

**if**(d.getID()==0)//show result

{

**this**.result[0]=0;**this**.result[1]=0;**this**.result[2]=0;**this**.result[3]=0;

**for**(Player a:*mvps*)

{

**if**(a.getID()==1)

**this**.result[0]++;

**else** **if**(a.getID()==2)

**this**.result[1]++;

**else** **if**(a.getID()==3)

**this**.result[2]++;

**else**

**this**.result[3]++;

}

**int** max\_index=0;

**for**(**int** j=0; j<4; j++)

{

**if**(result[j]>result[max\_index])

max\_index=j;

}

**boolean** isdraw= **false**;

**for**(**int** j=0; j<4; j++)

{

**if**( j!= max\_index &&result[j]==result[max\_index])

isdraw=**true**;

}

// Send revised sentence back to the client

**if**(isdraw)

outputToClient.writeObject(p5);

**else**

outputToClient.writeObject(allplayer[max\_index]);

}

**else**//add mvp

{

*mvps*.add(d);

Thread t=**new** Thread()

{

**public** **void** run() {

Player p;

//אם לא נעשה את זה בסכנרון האיבר שנוציא בפעולה הזאת לא בהרח יכנס למקום שאליו הוא נכנס בהתחלה בוקטור, עצם הסנכרון אנחנו דואגים שאם יכנסו עוד נתונים בזמן שהוצאנו ובדקנו את הוקטור הם לא יכנסוו ישירות לוקטור אלא יחכו שהתהליך יסתיים, שהוקטור שהוצאנו יכנס במחזרה למקומו ורק אז יכנסו לוקטור

**synchronized**(*mvps*) {

p= *mvps*.elementAt(*mvps*.size()-1);

*mvps*.removeElementAt(*mvps*.size()-1);

**try** {

//אפשר להגדיל את שמן ההשעיה כדי להמחיש שזה עובד

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**if**(p.getID()>0)

*mvps*.add(p);

}// synchronized

}// run

};// thread

t.start();

} // else

//System.out.println(mvps.toString());

}

**catch** (ClassNotFoundException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

} **catch** (IOException e) {

System.***err***.println(e);

}

}

}

מחלקת Server

**class** Server {

**private** ServerSocket serverSocket;

**public** Server() {

**try** {

serverSocket = **new** ServerSocket(8000);

} **catch** (IOException e) {

e.printStackTrace();

}

}

**public** Socket Accept() {

**try** {

**return** serverSocket.accept();

} **catch** (IOException e) {

e.printStackTrace();

}

**return** **null**;

}

}

מחלקת MultiThreadServerPlayerGUI

**public** **class** MultiThreadServerPlayerGUI **extends** JFrame {

**private** JTextArea jta = **new** JTextArea();

**private** Server server = **new** Server();

**public** **static** **void** main(String[] args) {

**new** MultiThreadServerPlayerGUI();

}

**public** MultiThreadServerPlayerGUI() {

// Place revised sentence on the frame

setLayout(**new** BorderLayout());

add(**new** JScrollPane(jta), BorderLayout.***CENTER***);

setTitle("MultiThreadServer");

setSize(500, 300);

setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

setVisible(**true**); // It is necessary to show the frame here!

jta.append("MultiThreadServer started at " + **new** Date() + '\n');

**int** clientNo = 1;

**while** (**true**) {

// Listen for a new connection request

Socket socket = server.Accept();

// Display the client number

jta.append("Starting thread for client " + clientNo + " at " + **new** Date() + '\n');

// Find the client's host name, and IP address

InetAddress inetAddress = socket.getInetAddress();

jta.append("Client " + clientNo + "'s host name is " + inetAddress.getHostName() + "\n");

jta.append("Client " + clientNo + "'s IP Address is " + inetAddress.getHostAddress() + "\n");

// Create a new task for the connection

Thread task = **new** Thread(**new** HandleAPlayerClient(socket));

task.start();

clientNo++;

}

}

}

מחלקת Team

**public** **class** Team **implements** Serializable{

**private** String name;

**private** ImageIcon img;

**public** Team(String name, ImageIcon img) {

**this**.name = name;

**this**.img = img;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** ImageIcon getImg() {

**return** img;

}

**public** **void** setImg(ImageIcon img) {

**this**.img = img;

{

מחלקת: Player

**public** **class** Player **implements** Serializable{

**private** **int** id;

**private** String name;

**private** ImageIcon img;

**private** Team team;

**public** Player(String name, ImageIcon img, Team team, **int** id) {

**this**.id=id;

**this**.name = name;

**this**.img = img;

**this**.team = team;

}

**public** Player() {

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** ImageIcon getImg() {

**return** img;

}

**public** **void** setImg(ImageIcon img) {

**this**.img = img;

}

**public** Team getTeam() {

**return** team;

}

**public** **void** setTeam(Team team) {

**this**.team = team;

}

**public** **int** getID() {

**return** id;

}

**public** **void** setID(**int** id) {

**this**.id = id;

}

}