

מכללת אדיר במרום

פרויקט מעבדה – חישבו מקבילי ומבוזר

הנדסאי – תוכנה

התוכנה: בחירת כסאות למשחק כדורגל שימוש שרת – לקוח

**שם הסטודנט:** רועי מני

**תעודת זהות:** 322906595

**שם המנחה:** מוטי פינקש

**שנת לימודים:** התשפ"ג 2023

תוכן עניינים

[מבוא: 3](#_Toc133217820)

[עמדת הפיתוח: 3](#_Toc133217821)

[דרישות המערכת: 3](#_Toc133217822)

[בעיות העלולות לקרות במערכת: 3](#_Toc133217823)

[פירוט של הבעיות: 4](#_Toc133217824)

[פתרונות לבעיות המערכת: 4](#_Toc133217825)

[תרשים UML של המערכת: 5](#_Toc133217826)

[פירוט על המחלקות: 5](#_Toc133217827)

[מדריך למשתמש: 10](#_Toc133217828)

[קוד המערכת: 13](#_Toc133217829)

[קוד של הserver: 13](#_Toc133217830)

[קוד של ה:Color 16](#_Toc133217831)

[קוד של ה:ClientGui 16](#_Toc133217832)

[קוד של :Client 18](#_Toc133217833)

[קוד של :Chairs 22](#_Toc133217834)

[ביבליוגרפיה: 23](#_Toc133217835)

# מבוא:

מעבדה זו עוסקת בחישוב מקבילי ומבוזר ותקשורת.

המערכת פותחה בסביבת העבודה IntelliJ IDEA 2022.3 ובשפת ב .java  
מטרת המעבדה היא להרות שימוש בסנכרון ותזמון תהליכונים במערכת .MultiThreaded

המעבדה שלנו מדמה הזמנת כרטיסים למשחק כדורגל שבו מספר רב של משתמשים מנסים להזמין מקומות ישיבה למשחק שאמור להתקיים, בזמן אמת אפשר לראות איזה מקומות תפוסים, איזה פנויים, והאם עכשיו משתמש מזמין כיסאות.

איך המשתמש רואה את זה:

* בן אדם מתחבר למערכת עם כמות כסף מסוימת .
* נפתח חלון אשר מראה את כל המצב של הכיסאות:
  + אם כיסא בצבע אדום זה אומר שקנו את הכיסא.
  + אם כיסא בצבע לבן אפשר לקנות הכיסא.
  + אם כיסא בצבע אפור זאת אומרת שקונים כיסא ברגע זה עכשיו.
  + אם כיסא בצבע ירוק קנית את הכיסא.
* ולפי הכללים האלה אתה יכול לקנות כיסאות.

איך המערכת עובדת מאחורה:

* המערכת כל פעם מאזינה ללקוחות שמתחברים למערכת.
* המערכת מעדכנת את כל הלקוחות לפי הבחירה של הבן אדם האם לקנות או לא.

# עמדת הפיתוח:

* Windows 11
* IntelliJ IDEA 2022.3

דרישות המערכת:

* windows 10/11
* IntelliJ IDEA

# בעיות העלולות לקרות במערכת:

* בעיית מרוץ תהליכים Race Condition
* בעיית הקיפאון Dead Lock

# פירוט של הבעיות:

* מרוץ תהליכים Race condition הוא מצב שבו מספר תהליכונים או תהליכים מנסים בו זמנית להשיג גישה למשאב משותף.
* בעיית הקיפאון Dead Lock מצב שבו שתי פעולות מחכות כל אחת לסיומה של האחרת, ובכך נוצר מצב בו אף אחת לא מסתיימת. בהנדסת תוכנה מצב זה בא לידי ביטוי כאשר שני תהליכים מחכים אחד לאחר לשחרור משאב מסוים, או לקיפאון בתור מעגלי.

קיפאון יכול להיווצר כאשר התנאים הבאים מתקיימים:

- כשאחד המשאבים אינו שיתופי, רק תהליך אחד יכול להשתמש במשאב זה.

- כל תהליך מחזיק משאב ומחכה למשאב נוסף המוחזק ע"י התהליך האחר.

- משאבים לא משוחררים לאחר סיום השימוש בהם.

- שני תהליכים מחכים כל אחד למשאב המוחזק על ידי התהליך האחר.

# פתרונות לבעיות המערכת:

- Race condition אנחנו משתמשים בפעולה ChangeBuyMessage אשר נמצא בשרת כלומר, כל לקוח יכול לבקש גישה לפעולה זו בו זמנית אבל רק בקשה אחת תטופל בכל פעם.

- בעיית הקיפאון Dead Lock פתרונות אפשריים למצב קיפאון:

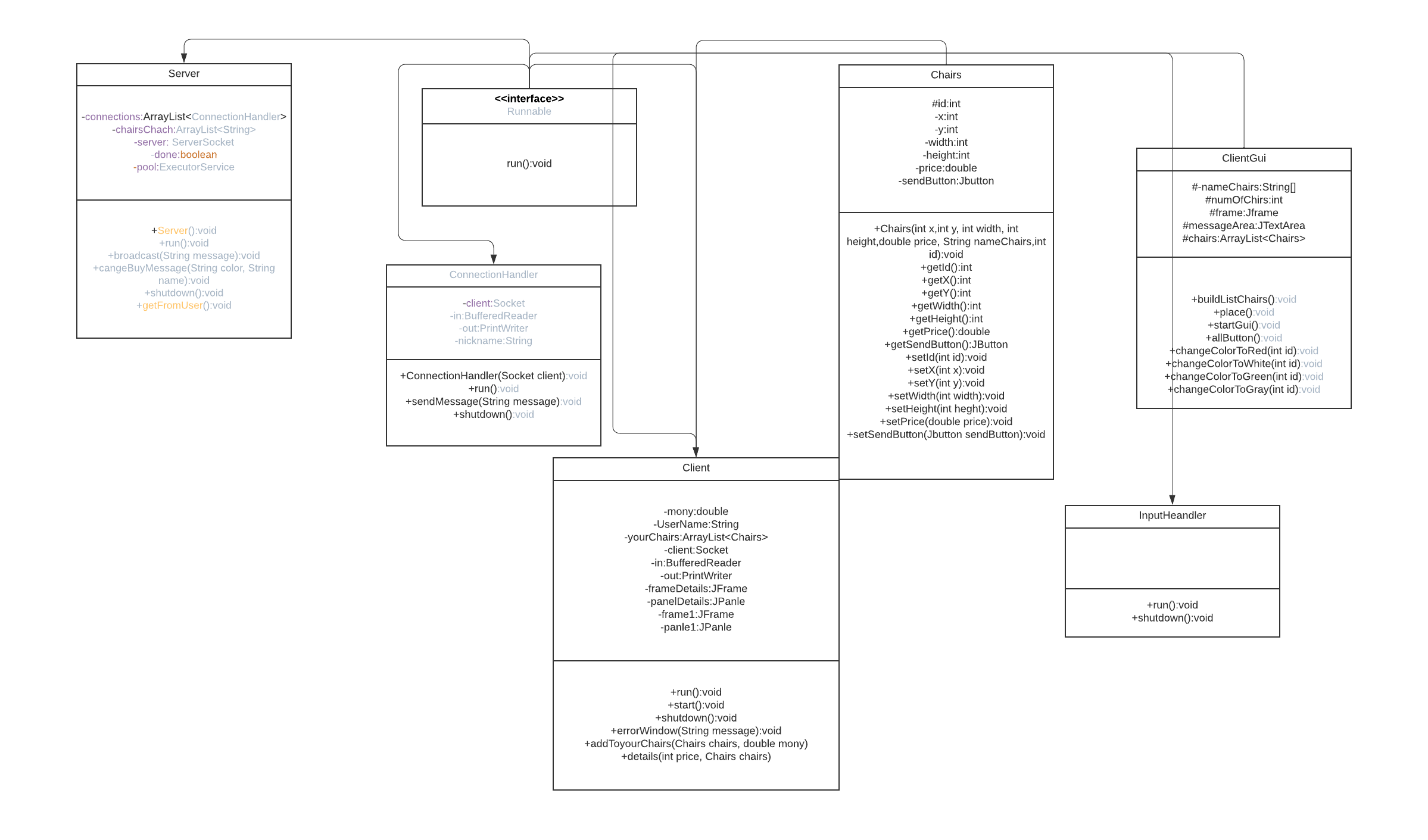
- ניתן לסדר משאבים קריטיים ולדרוש שימוש בסדר הנתון.

- ניתן לדרוש שתהליך יבקש גישה למשאב רק אם אינו מחזיק במשאב אחר.

- שימוש ב- wait ו - notify , לדרוש מתהליך לחכות למשאב ולדרוש מהתהליך שמשתמש במשאב להודיע כאשר הוא מסיים את השימוש בו.

מצב של קיפאון לא יכול להיווצר במערכת כיוון כל אחד מהלקוחות הוא ישות בפני עצמו ואינם צריכים להתחלק במשאבים, המשאב היחיד הקריטי הוא פעולת השידור הממומשת בשרת והיא מסונכרנת לכן, גם אם מספר משתמשים מנסים לגשת לפעולה זו רק הודעה ממשתמש אחד כל פעם תישלח באמצעות הפעולה.

# תרשים UML של המערכת:



# פירוט על המחלקות:

כל המשתנים של המחלקה Server

|  |  |
| --- | --- |
| מכילה בתוך את כל החיבורים לשרת | private ArrayList<ConnectionHandler> connections |
| משתנה המחזיק את כל הכיסאות שקנו | private ArrayList<String> chairsChach = new ArrayList<>(); |
| משתנה שמחזיק את כל הכיסאות שיכולים לקנו עכשיו | private ArrayList<String> nowChairsChach = new ArrayList<>(); |
| משתנה הסוקט של השרת, לקביעת הפורט שהשרת ישתמש בו | private ServerSocket server; |
| משתנה שמחזיר באם השרת פתוח או לא | private boolean done; |
| משתנה אשר יוצר שרשור ללקוח נכנס אם שאר הלקוחות | private ExecutorService pool; |

כל הפעולות של המחלקה Server

|  |  |
| --- | --- |
| פונקציית הפעולה הבונה, בה מאותחלים התכונות | public Server() |
| פונקציה הדואגת שהשרת שלנו כל הזמן עובד | public void run() |
| פונקציה המקבלת הודאה ושולחת לכולם | public void broadcast(String message) |
| פונקציה מקבלת הודאה ושם של משתמש ושולחת לכולם את ההודאה חוץ משהשם של המשתמש | public void changeBuyMessage(String color, String name) |
| השרת מנתק את כל המשתמשים מהשרת | public void shutdown() |

כל המשתנים של המחלקה ConnectionHandle

|  |  |
| --- | --- |
| משתנה הסוקט של הלקוח שאליו מקושר החיבור | private Socket client; |
| משתנה המקבל מידע מן הלקוח | private BufferedReader in; |
| משתנה השולח מידע אל הלקוח | private PrintWriter out; |
| משתנה שמחזיק את השם משתמש של אותו חיבור | private String nickname; |

כל הפעולות על המחלקה ConnectionHandle

|  |  |
| --- | --- |
| הפעולה הבונה, מקבלת משתנה מסוג סוקט של הלקוח, ומאתחלת בתכונה | public ConnectionHandler(Socket client) |
| פעולת ההרצה של התהליך, שם מתקיימים כל הפעולות שהלקוח רוצה לעשות. משם נשלח ומתקבל מידע מן ואל הלקוח. | public void run() |
| הפונקציה מקבלת מהמשתמש מידע ועל פי זה פועלת | public void getFromUser() |
| הפונקציה מקבלת הודאה ושולחת את זה ללקוח | public void sendMessage(String message) |
| הפונקציה מנתקת את המשתמש מהשרת | public void shutdown() |

כל המשתנים של המחלקה ClientGui

|  |  |
| --- | --- |
| במשתנה אזה אני מחזיק את השמות של כל הכיסאות | protected String[] nameChairs |
| במשתנה אזה אני מחזיק את מספר הכיסאות הקיימים | protected int numberOfChairs |
| משתנה המחזיק את החלון המרכזי של הזמנת כרטטיסים | protected JFrame frame = new JFrame"buy Chairs"); |
| המשתנה מיועד לשמש להצגת מידע | protected JTextArea messageArea = new JTextArea(8, 40); |
| משתנה המחזיק את כל הכיסאות הקיימים | protected ArrayList<Chairs> chairs = new ArrayList<>(); |

כל הפעולות של המחלקה ClientGui

|  |  |
| --- | --- |
| הפונקציה בונה את כל הפרמטרים של הכיסאות(את המיקום ...) | public void buildListChairs() |
| הפונקציה מוסיפה ללוח את הכיסא | public void place() |
| הפונקציה יוצרת את הלוח המרכזי של תחילת השימוש | public void startGui() |
| הפונקציה בונה את כל הכיסאות | public void allButton() |
| הפונקציה מקבלת ID ו משנה את צבע הכפתור שID שלו שווה למשתנה לאדום | public void changeColorToRed(int id) |
| הפונקציה מקבלת ID ו משנה את צבע הכפתור שID שלו שווה למשתנה ללבן | public void changeColorToWhite(int id) |
| הפונקציה מקבלת ID ו משנה את צבע הכפתור שID שלו שווה למשתנה לירוק | public void changeColorToGreen(int id) |
| הפונקציה מקבלת ID ו משנה את צבע הכפתור שID שלו שווה למשתנה לאפור | public void changeColorTogray(int id) |

כל המשתנים של המחלקה Chairs

|  |  |
| --- | --- |
| משתנה שמחזיק את ID של הכיסא | protected int id; |
| משתנה אשר מחזיק את ציר ה x של הכפתור | private int x; |
| משתנה אשר מחזיק את ציר ה y של הכפתור | private int y; |
| משתנה אשר מחזיק את הגובה של הכפתור | private int width; |
| משתנה אשר מחזיק את האורך של הכפתור | private int height; |
| משתנה שמחזיק את העלות של הכיסא | private double price; |
| משתנה שמחזיק את הכפתור של הכיסא | private JButton sendButton ; |
| משתנה שמחזיק את הצבע של הכיסא | Color color; |

כל הפעולות של המחלקה Chairs

|  |  |
| --- | --- |
| הפונקציה מקבלת את כל המידע על הכיסא ובונה את הכיסא | public Chairs(int x, int y, int width,int height, double price, String nameChairs, int id) |
| הפונקציה מחזירה את ID של הכיסא | public int getId() |
| הפונקציה מעדכנת את ID של הכיסא | public void setId(int id) |
| הפונקציה מחזירה את מיקום ה x של הכפתור | public int getX() |
| הפונקציה מעדכנת את מיקום ה x של הכפתור | public void setX(int x) |
| הפונקציה מחזירה את מיקום ה y של הכפתור | public int getY() |
| הפונקציה מעדכנת את מיקום ה y של הכפתור | public void setY(int y) |
| הפונקציה מחזירה את האורך של הכפתור | public int getWidth() |
| הפונקציה מעדכנת את האורך של הכפתור | public void setWidth(int width) |
| הפונקציה מחזירה את הגובה של הכפתור | public int getHeight() |
| הפונקציה מעדכנת את הגובה של הכפתור | public void setHeight(int height) |
| הפונקציה מחזירה את המחיר של הכיסא | public double getPrice() |
| הפונקציה מעדכנת את המחיר של הכיסא | public void setPrice(double price) |
| הפונקציה מחזירה את הכפתור של הכיסא | public JButton getSendButton() |
| הפונקציה מעדכנת לכפתור חדש | public void setSendButton(JButton sendButton) |
| הפונקציה מחזירה את הצבע של הכיסא | public Color getColor() |
| הפונקציה מעדכנת את הצבע של הכיסא | public void setColor(Color color) |

כל המשתנים של המחלקה Client

|  |  |
| --- | --- |
| משתנה המחזיק את הכסף של המשתמש | private double mony |
| משתנה שמחזיק את השם משתמש | private String UserName |
| משתנה שמחזיק את כל הכיסאות של אותו משתמש | private ArrayList<Chairs> yourChairs = new ArrayList<>() |
| משתנה הסוקט של הלקוח שאליו מקושר החיבור | private Socket client |
| משתנה המקבל מידע מן השרת | private BufferedReader in |
| משתנה השולח מידע אל השרת | private PrintWriter out |
| משתנה המחזיק את חלון הפרטים של הכיסאות | private JFrame frameDetails = new JFrame("Details  ") |
| משתנה המחזיק את הפאנל של חלון הפרטים | private JPanel panelDetails = new JPanel() |
| משתנה המחזיק את חלון השגיאה | private JFrame frame1 = new JFrame("Error Window Example") |
| משתנה המחזיק את הפאנל של חלון השגיאה | private JPanel panel1 = new JPanel() |

כל הפעולות של המחלקה Client

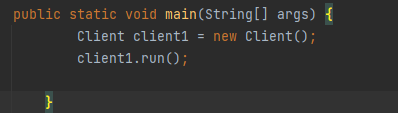
|  |  |
| --- | --- |
| פונקציה שמנסה להתחבר לשרת וכל הזמן מאזינה לו ועל פי כך פועלת | public void run() |
| פונקציה אשר מבקשת מהמשתמש את השם שלו | public void start() |
| הפונקציה מנתקת את המשתמש מהשרת | public synchronized void shutdown() |
| פונקציה שמקבלת הודאה ושולחת אותה לשרת | public void errorWindow(String message) |
| פונקציה המקבלת כיסא וכמה הוא עולה ומוסיפה אותו לקניית הכיסאות של אותו מערך | public void addToYourChairs(Chairs chairs, double mony) |
| פונקציה אשר פותחת חלון ומראה את המחיר של הכיסא ושואלת אם אתה רוצה לקנות את זה | public void details(int price, Chairs chairs) |

כל הפעולות של המחלקה InputHandler

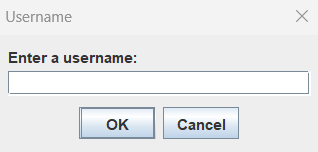
|  |  |
| --- | --- |
| פונקציה שמקבלת מה שאתה עושה (עם אתה קונה כיסאות...)ועל פי כך היא פועלת(עם צריך לעדכן את השרת או לשלוח אודה שגיאה...) | public void run() |
| הפונקציה מנתקת את המשתמש מהשרת | public void shutdown() |

מדריך למשתמש:יש להיכנס לקוד ולהריץ קודם כל במחלקת Server ואז ללכת למחלקת Client ולהריץ וגם ללכת למחלקת Main ולהריץ אותה.

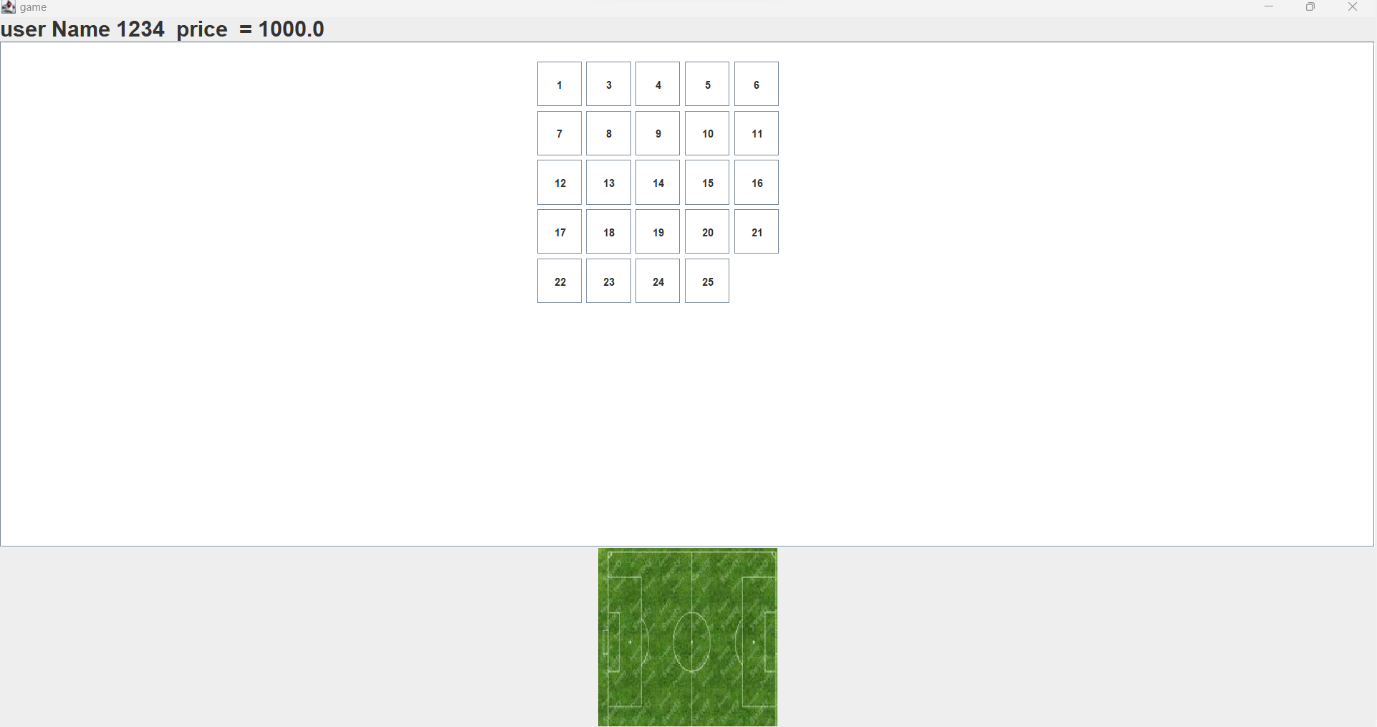
אם רוצים להריץ עוד לקוחות אפשר ליצור מחלקה חדשה ולכתוב את השורות הבאות



אחרי שהרצנו את הכול יופיע המסך הבא



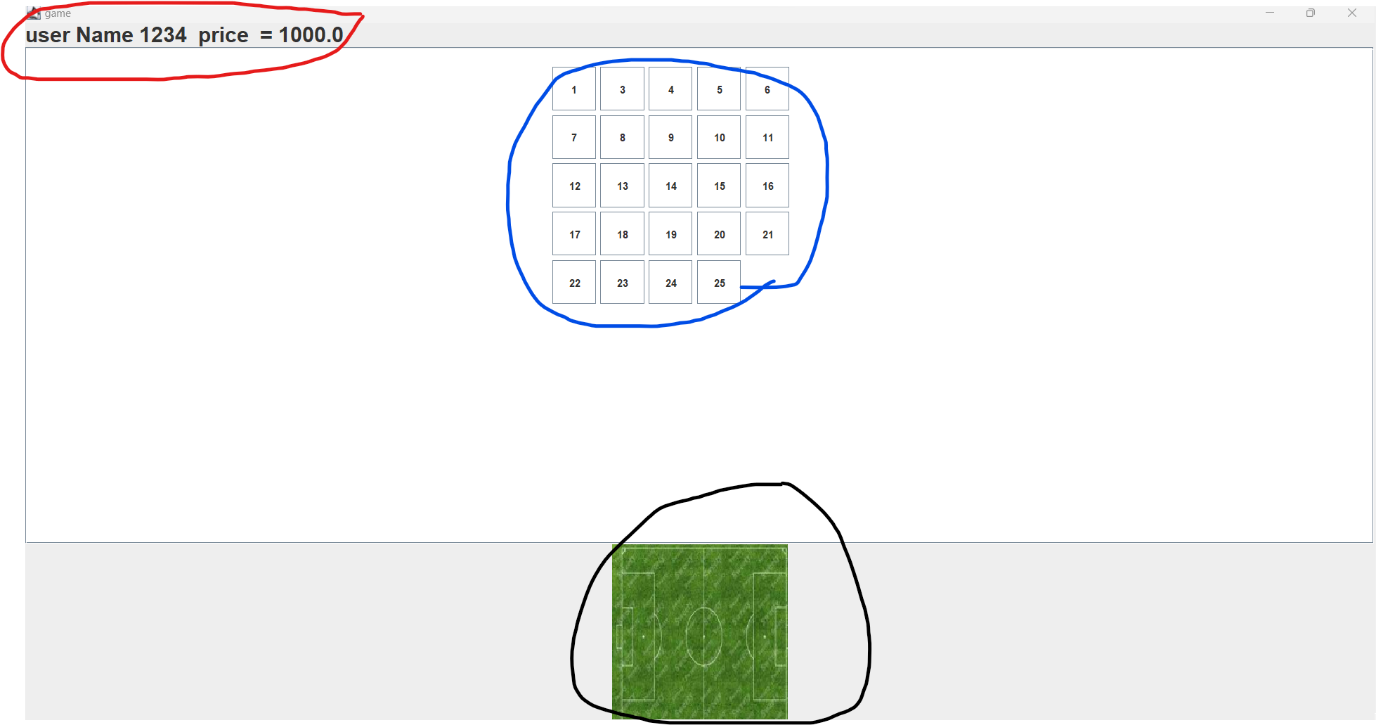
כאן מה שאנחנו צריכים לעשות זה להכניס את השם משתמש שלנו וללחוץ על .Ok

יפתח לנו החלון של כל הכיסאות .

יהיה אפשר לראות שבצד שמא למעלה אנחנו מחוברים וכמה כסף יש לנו.

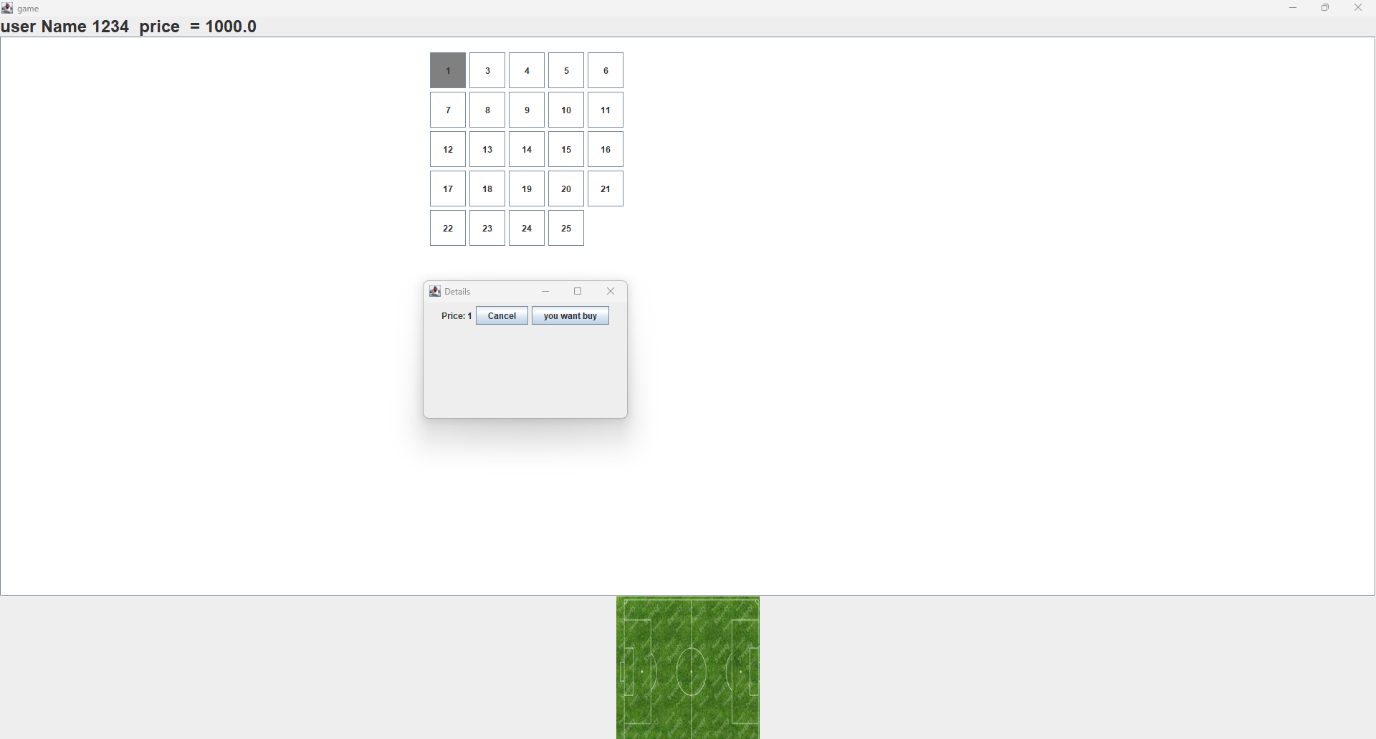
באמצע יהיה אפשר לראות את כל הכיסאות.

ולמטה יהיה אפשר לראות את המגרש.



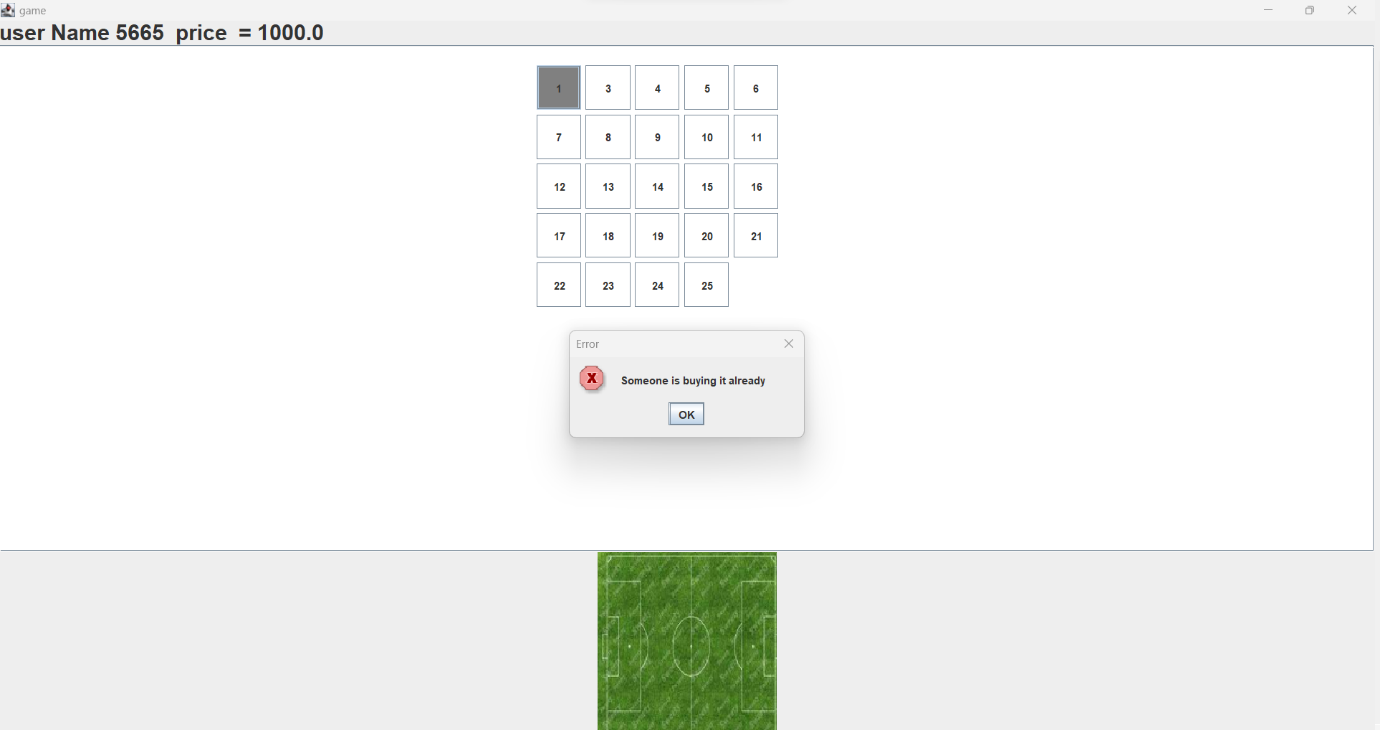
בזמן לחיצה על כיסא מסוים יפתח חלון קנייה והכיסא יהפוך לאפור, ושום משתמש לא יוכל בזמן אזה אף יוכל לקנות את הכיסאות שבחרת אלה אם תבטל, ואז הכיסא יהפוך חזרה ללבן.

אחרי שתקנה את הכיסאות לשאר המשתמשים הכיסא יצבע לאדום ולך הכיסא יצבע לירוק.

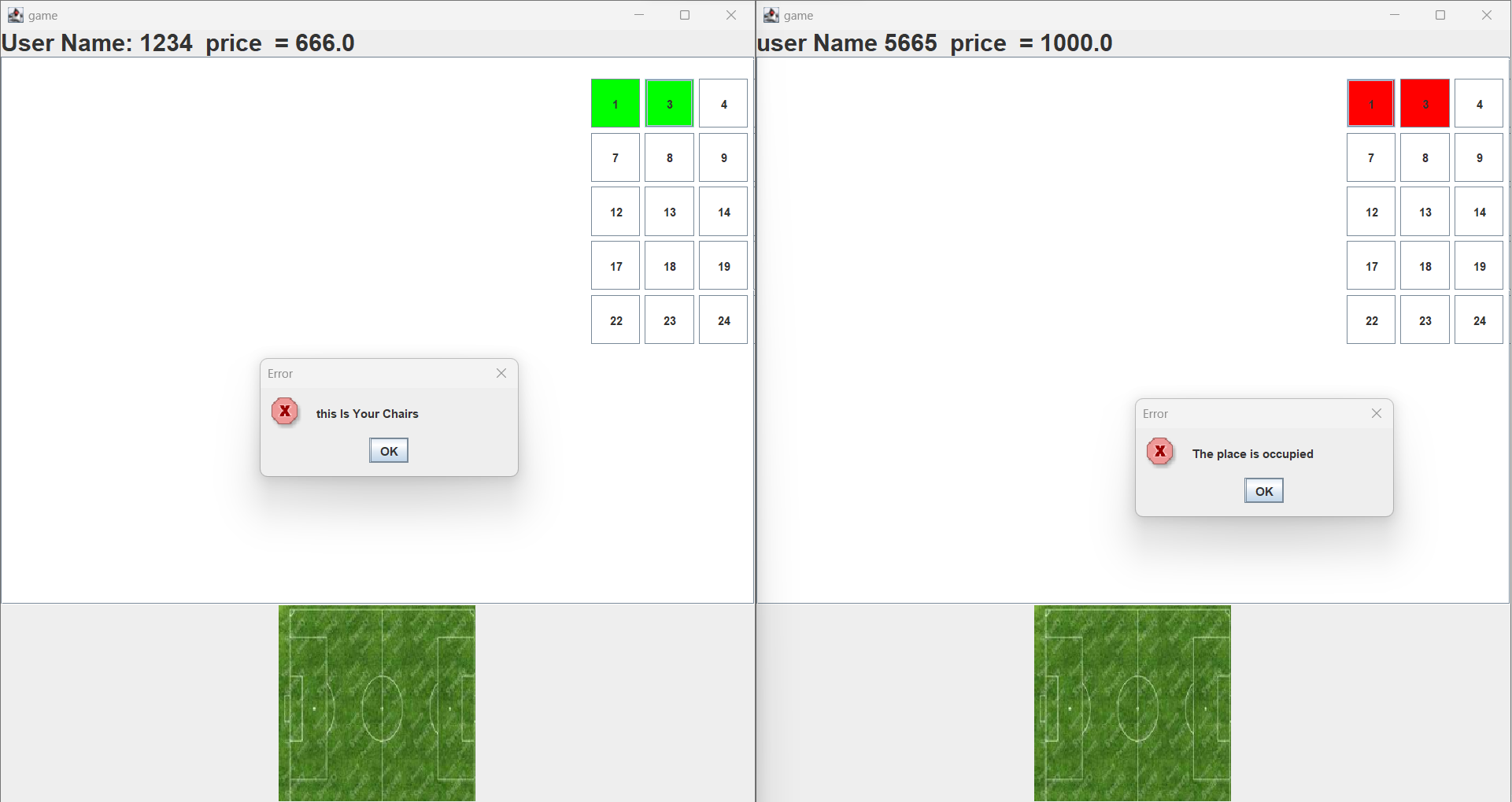
זה כאשר בחרתי את כיסא אחד.



זה מה שקורה למשתמש החדש.



זה כאשר המשתמש החדש ינסה לקנות את אותו כיסא.



וזה מה שקורה אחרי שאני קונה את הכיסאות(קניתי סתם עוד כיסאות) ואני לוחץ על הכיסאות הקנויים.

# קוד המערכת:

# קוד של הserver:

package org.example;  
  
  
  
import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStreamReader;  
import java.io.PrintWriter;  
import java.net.ServerSocket;  
import java.net.Socket;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.concurrent.ExecutorService;  
import java.util.concurrent.Executors;  
import java.util.regex.Matcher;  
import java.util.regex.Pattern;  
  
public class Server implements Runnable {  
  
 private ArrayList<ConnectionHandler> connections;  
 private ArrayList<String> chairsChach = new ArrayList<>();  
 private ArrayList<String> nowChairsChach = new ArrayList<>();  
 private ServerSocket server;  
 private boolean done;  
 private ExecutorService pool;  
  
 public Server() {  
 connections = new ArrayList<>();  
 done = false;  
 }  
  
 @Override  
 public void run() {  
 try {  
 server = new ServerSocket(8080);  
 pool = Executors.*newCachedThreadPool*();  
 while (!done) {  
 Socket client = server.accept();  
 ConnectionHandler handler = new ConnectionHandler(client);  
 connections.add(handler);  
 pool.execute(handler);  
 }  
 } catch (Exception e) {  
 shutdown();  
 }  
 }  
  
 // This method broadcasts a message to all connected clients  
 public void broadcast(String message) {  
 for (ConnectionHandler ch : connections) {  
 if (ch != null) {  
 ch.sendMessage(message );  
 }  
 }  
 }  
  
 public void changeBuyMessage(String color, String name){  
 for(ConnectionHandler ch : connections){  
 if (ch != null && !ch.nickname.equals(name)) {  
 ch.sendMessage(color);  
 }  
 }  
 }  
  
 public void shutdown() {  
 try {  
 done = true;  
 pool.shutdown();  
 if (!server.isClosed()) {  
 server.close();  
 }  
 for (ConnectionHandler ch : connections) {  
 ch.shutdown();  
 }  
 } catch (IOException e) {  
  
 }  
  
 }  
  
  
 class ConnectionHandler implements Runnable {  
 private Socket client;  
  
 private BufferedReader in;  
 private PrintWriter out;  
 private String nickname;  
  
 public ConnectionHandler(Socket client) {  
 this.client = client;  
 }  
  
 @Override  
 public void run() {  
 try {  
 out = new PrintWriter(client.getOutputStream(), true);  
 in = new BufferedReader(new InputStreamReader(client.getInputStream()));  
 // send prompt for nickname to client  
 out.println("Please enter a nickname: ");  
 // read nickname from client  
 nickname = in.readLine();  
  
 System.*out*.println(nickname + " connected!");  
 if(!chairsChach.isEmpty()){  
 for (int i = 0; i < chairsChach.size();i++){  
 out.println("RED" + chairsChach.get(i));  
 }  
 }  
 if(!nowChairsChach.isEmpty()){  
 for (int i = 0; i < nowChairsChach.size();i++){  
 out.println("gray" + nowChairsChach.get(i));  
 }  
 }  
 broadcast(nickname + " joined the chat!");  
 getFromUser();  
 } catch (IOException e) {  
 shutdown();  
 }  
 }  
  
 public void getFromUser() throws IOException {  
 String message = null;  
 while ((message = in.readLine()) != null) {  
  
 if (message.startsWith("/quit")) {  
 broadcast(nickname + " left the chat! ");  
 shutdown();  
 }  
  
 changeBuyMessage(message, this.nickname);  
 if(message.contains("gray")){  
 Pattern pattern = Pattern.*compile*("\\d+");  
 Matcher matcher = pattern.matcher(message);  
 if (matcher.find()) {  
 String extractedNumber = matcher.group();  
 nowChairsChach.add(extractedNumber);  
 }  
 }  
 if(message.contains("White")){  
 Pattern pattern = Pattern.*compile*("\\d+");  
 Matcher matcher = pattern.matcher(message);  
 if (matcher.find()) {  
 String extractedNumber = matcher.group();  
 nowChairsChach.remove(extractedNumber);  
 }  
 }  
 if(message.contains("RED")){  
 Pattern pattern = Pattern.*compile*("\\d+");  
 Matcher matcher = pattern.matcher(message);  
 if (matcher.find()) {  
 String extractedNumber = matcher.group();  
 nowChairsChach.remove(extractedNumber);  
 chairsChach.add(extractedNumber);  
 }  
 }  
  
 }  
  
 }  
  
 public void sendMessage(String message) {  
 out.println(message);  
 }  
  
 public void shutdown() {  
 try {  
 in.close();  
 out.close();  
 if (!client.isClosed()) {  
 client.close();  
 }  
  
 connections.remove(this);  
 } catch (IOException e) {  
 }  
 }  
  
 }  
  
 public static void main(String[] args) {  
 Server server = new Server();  
 Thread t = new Thread(server);  
 t.start();  
 }  
}

# קוד של ה:Color

package org.example;  
  
public enum Color {  
 *RED*,  
 *GREEN*,  
 *WHITE*,  
 *GRAY*}

# קוד של ה:ClientGui

package org.example;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.Color;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.util.ArrayList;  
  
public class ClientGui{  
  
 protected String[] nameChairs = {"1", "2", "3", "4", "5", "6" , "7", "8","9" , "10", "11","12" , "13", "14","15", "16", "17" , "18", "19","20" , "21", "22","23" , "24", "25"};  
 protected int numberOfChairs = nameChairs.length;  
 protected JLabel nameLabel;  
 protected JFrame frame = new JFrame("buy Chairs");  
 protected JTextArea messageArea = new JTextArea(8, 40);  
 protected ArrayList<Chairs> chairs = new ArrayList<>();  
  
 public void buildListChairs(){  
 int x = 600;  
 int y = 50;  
 int width = 50;  
 int height = 50;  
 chairs.add(new Chairs(x,y,width,height,1, nameChairs[0],0));  
 x = x + 55;  
 for(int i = 2; i < numberOfChairs; i++){  
 chairs.add(new Chairs(x,y,width,height,333, nameChairs[i], i - 1));  
 x = x + 55;  
 if(i % 5 == 0) {  
 y = y + 55;  
 x = 600;  
 }  
 }  
 }  
 public void place(){  
 for(int i = 0; i < chairs.size(); i++){  
 chairs.get(i).getSendButton().setBackground(Color.*WHITE*);  
 frame.add(chairs.get(i).getSendButton());  
 }  
 ImageIcon imageIcon = new ImageIcon("C:\\ProjectServer\\src\\main\\java\\org\\example\\football.jpg");  
 Image image = imageIcon.getImage().getScaledInstance(200, 200, Image.*SCALE\_SMOOTH*);  
 ImageIcon resizedImageIcon = new ImageIcon(image);  
 JLabel imageLabel = new JLabel(resizedImageIcon);  
 frame.add(imageLabel, BorderLayout.*SOUTH*);  
 }  
 public void startGui(){  
 allButton();  
 frame.setTitle("game ");  
 frame.setVisible(true);  
 }  
  
 public void allButton(){  
 buildListChairs();  
 messageArea.setEditable(false);  
 place();  
 frame.getContentPane().add(new JScrollPane(messageArea), BorderLayout.*CENTER*);  
 frame.pack();  
 }  
 public void changeColorToRed(int id){  
 chairs.get(id).setColor(org.example.Color.*RED*);  
 chairs.get(id).getSendButton().setBackground(Color.*RED*);  
  
 }  
 public void changeColorToWhite(int id){  
 chairs.get(id).setColor(org.example.Color.*WHITE*);  
 chairs.get(id).getSendButton().setBackground(Color.*WHITE*);  
 }  
 public void changeColorToGreen(int id){  
 chairs.get(id).setColor(org.example.Color.*GREEN*);  
 chairs.get(id).getSendButton().setBackground(Color.*GREEN*);  
 }  
 public void changeColorTogray(int id){  
 chairs.get(id).setColor(org.example.Color.*GRAY*);  
 chairs.get(id).getSendButton().setBackground(Color.*GRAY*);  
 }  
  
  
}

# קוד של :Client

package org.example;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.awt.event.WindowAdapter;  
import java.awt.event.WindowEvent;  
import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStreamReader;  
import java.io.PrintWriter;  
import java.net.Socket;  
import java.util.ArrayList;  
  
public class Client extends ClientGui implements Runnable {  
 private double mony = 1000;  
 private String UserName ;  
 private ArrayList<Chairs> yourChairs = new ArrayList<>();  
 private Socket client;  
 private BufferedReader in;  
 private PrintWriter out;  
 private JFrame frameDetails = new JFrame("Details");  
 private JPanel panelDetails = new JPanel();  
 private JFrame frame1 = new JFrame("Error Window Example");  
 private JPanel panel1 = new JPanel();  
  
 @Override  
 public void run() {  
 try {  
 client = new Socket("127.0.0.1", 8080);  
 out = new PrintWriter(client.getOutputStream(), true);  
 in = new BufferedReader(new InputStreamReader(client.getInputStream()));  
  
 start();  
 startGui();  
 //here i build Thread that run and listen to server  
 InputHandler inputHandler = new InputHandler();  
 Thread t = new Thread(inputHandler);  
 t.start();  
 String inMessage;  
 while ((inMessage = in.readLine()) != null) {  
 if(inMessage.contains("RED")){  
 for(Chairs ch: chairs){  
 String numberId = String.*valueOf*(ch.id);  
 if(inMessage.contains(numberId)){  
 changeColorToRed(ch.id);  
 }  
 }  
 }else if(inMessage.contains("White")){  
 for(Chairs ch: chairs){  
 String numberId = String.*valueOf*(ch.id);  
 if(inMessage.contains(numberId)){  
 changeColorToWhite(ch.id);  
 }  
 }  
 }else if(inMessage.contains("gray")){  
 for(Chairs ch: chairs) {  
 String numberId = String.*valueOf*(ch.id);  
 if (inMessage.contains(numberId)) {  
 changeColorTogray(ch.id);  
 }  
 }  
 }  
 System.*out*.println(inMessage);  
 }  
  
 } catch (IOException e) {  
 shutdown();  
 }  
 }  
 //this i ask for the user: name  
 public void start() throws IOException {  
 while (true) {  
 String input = JOptionPane.*showInputDialog*(frame, "Enter a username:", "Username", JOptionPane.*PLAIN\_MESSAGE*);  
 if (input != null && input.length() > 0) {  
 UserName = input;  
 nameLabel = new JLabel("user Name " + UserName + " " +"price = " + mony);  
 nameLabel.setFont(new Font("Arial", Font.*BOLD*, 24));  
 frame.add(nameLabel, BorderLayout.*NORTH*);  
 out.println(input);  
 break;  
 }  
 }  
 }  
  
 public synchronized void shutdown() {  
 try {  
 in.close();  
 out.close();  
 if (!client.isClosed()) {  
 client.close();  
 }  
 } catch (IOException e) {  
 // ignore  
 }  
 }  
  
 public void errorWindow(String message) {  
 frame1.getContentPane().add(panel1);  
 frame1.getContentPane().add(panel1);  
 frame1.setSize(300, 200);  
 frame1.setLocationRelativeTo(null);  
 frame1.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
  
 JOptionPane.*showMessageDialog*(frame1, message, "Error", JOptionPane.*ERROR\_MESSAGE*);  
 }  
  
 public void addToYourChairs(Chairs chairs, double mony) {  
 this.mony = this.mony - mony;  
 nameLabel.setText("User Name: " + UserName + " " +"price = " + this.mony);  
 out.println("RED" + chairs.id );  
 yourChairs.add(chairs);  
 changeColorToGreen(chairs.id);  
 System.*out*.println(this.mony);  
 }  
  
 public void details(int price, Chairs chairs) {  
 frameDetails.getContentPane().add(panelDetails);  
 frameDetails.getContentPane().add(panelDetails);  
 frameDetails.setSize(300, 200);  
 frameDetails.setLocationRelativeTo(null);  
 frameDetails.setDefaultCloseOperation(JFrame.*DO\_NOTHING\_ON\_CLOSE*);  
 frameDetails.addWindowListener(new WindowAdapter() {  
 public void windowClosing(WindowEvent e) {  
  
 }  
 });  
  
 JLabel priceLabel = new JLabel("Price: " + price);  
 panelDetails.add(priceLabel);  
  
 JButton buttonExit = new JButton("Cancel");  
  
 JButton button = new JButton("you want buy");  
  
 buttonExit.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 changeColorToWhite(chairs.id);  
 out.println("White" + chairs.id);  
 panelDetails.removeAll();  
 frameDetails.dispose();  
 }  
 });  
  
 button.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 // Update the price based on the value entered in the text field  
 try {  
 addToYourChairs(chairs, chairs.getPrice());  
  
  
 } catch (NumberFormatException ex) {  
 // Display an error message if the value entered in the text field is not a valid integer  
 JOptionPane.*showMessageDialog*(frameDetails, "Invalid price");  
 return;  
 }  
 panelDetails.removeAll();  
 frameDetails.dispose();  
 }  
 });  
 panelDetails.add(buttonExit);  
 panelDetails.add(button);  
 frameDetails.setVisible(true);  
  
 }  
  
 class InputHandler implements Runnable {  
 public void run() {  
 frame.addWindowListener(new WindowAdapter() {  
 public void windowClosing(WindowEvent e) {  
 out.println("/quit");  
 shutdown();  
 }  
 });  
 for (Chairs ch : chairs) {  
 ch.getSendButton().addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 if(ch.color == Color.*GRAY*){  
 errorWindow("Someone is buying it already");  
 }else if(ch.color == Color.*GREEN*){  
 errorWindow("this Is Your Chairs");  
 }else if(ch.color == Color.*RED*){  
 errorWindow("The place is occupied");  
 }  
 else{  
 if(ch.getPrice() > mony){  
 errorWindow("you need more mony");  
 }  
 else{  
 out.println("gray " + ch.id);  
 changeColorTogray(ch.id);  
 details((int) ch.getPrice(), ch);  
 }  
 }  
 }  
 });  
 }  
 }  
  
 public void shutdown() {  
 try {  
 in.close();  
 out.close();  
 if (!client.isClosed()) {  
 client.close();  
 }  
 System.*exit*(0);  
 } catch (IOException e) {  
 // ignore  
 }  
 }  
 }  
  
 public static void main(String[] args) {  
 Client client1 = new Client();  
 client1.run();  
  
 }  
}

# קוד של :Chairs

package org.example;  
  
import javax.swing.\*;  
import java.awt.\*;  
  
public class Chairs {  
 protected int id;  
 private int x;  
 private int y;  
 private int width;  
 private int height;  
  
 private double price;  
 private JButton sendButton ;  
 Color color;  
 public Chairs(int x, int y, int width,int height, double price, String nameChairs, int id){  
 this.x = x;  
 this.y = y;  
 this.width = width;  
 this.height = height;  
 this.price = price;  
 color = Color.*WHITE*;  
 this.id = id;  
 sendButton = new JButton(nameChairs);  
 sendButton.setBounds(x, y, width, height);  
 }  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
 public int getX() {  
 return x;  
 }  
  
 public void setX(int x) {  
 this.x = x;  
 }  
  
 public int getY() {  
 return y;  
 }  
  
 public void setY(int y) {  
 this.y = y;  
 }  
  
 public int getWidth() {  
 return width;  
 }  
  
 public void setWidth(int width) {  
 this.width = width;  
 }  
  
 public int getHeight() {  
 return height;  
 }  
  
 public void setHeight(int height) {  
 this.height = height;  
 }  
  
 public double getPrice() {  
 return price;  
 }  
  
 public void setPrice(double price) {  
 this.price = price;  
 }  
  
 public JButton getSendButton() {  
 return sendButton;  
 }  
  
 public void setSendButton(JButton sendButton) {  
 this.sendButton = sendButton;  
 }  
  
 public Color getColor() {  
 return color;  
 }  
  
 public void setColor(Color color) {  
 this.color = color;  
 }  
  
  
}

# ביבליוגרפיה:

Threads (אין תאריך, בא מתוך וויקיפדיה).

<https://en.wikipedia.org/wiki/Thread_(computing)>

Multithreading (אין תאריך, בא מתוך וויקיפדיה).

<https://en.wikipedia.org/wiki/Multithreading_(computer_architecture)>

race condition (אין תאריך, בא מתוך וויקיפדיה).

<https://en.wikipedia.org/wiki/Race_condition>

deadlock  (אין תאריך, בא מתוך וויקיפדיה).

<https://en.wikipedia.org/wiki/Deadlock>

resource starvation (אין תאריך, בא מתוך וויקיפדיה).

<https://en.wikipedia.org/wiki/Starvation_(computer_science)>