

**פרויקט גמר**

**למילוי חלקי של הדרישות לקבלת תואר הנדסאי**

**הנדסת תוכנה**

**בהתמחות : מחשבים**

**נושא הפרוייקט : מעבדת תקשורת**

**שם הסטודנט/ית : ליאון טלנוב**

**העבודה בוצעה בהנחיית : מוטי פניקשווילי ואודי מלכה**

תוכן עניינים

[**הקדמה 3**](#_Toc40952246)

[**צילומי מסך: 4**](#_Toc40952247)

[**תרשים :UML 5**](#_Toc40952248)

[**פירוט מחלקות: 6**](#_Toc40952249)

[**קטעי קוד: 9**](#_Toc40952250)

# הקדמה

בפרויקט המעבדה מוצג קשר בין צד שרת לצד לקוח, בפרויקט שלי אעסוק בהזמנת תור למספרה מסוימת.

בפרויקט זה , התוכנה מדמה תוכנת הזמנות למספרה מסוימת של חודש מסוים. כלומר , מאפשרת ללקוח לבחור באחד משרותי המספרה ומאפשרת לו להזמין תור ליום ולשעה בה הוא מעוניין.

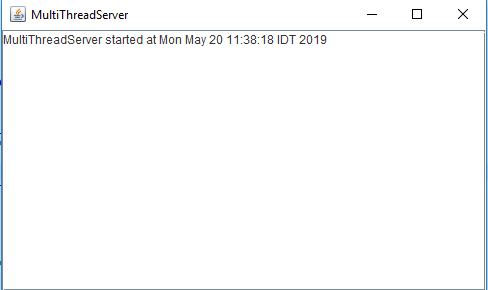
אם השעה והיום המבוקשים כבר תפוסים על ידי לקוח אחר , תוצג הודעה המודיעה ללקוח כי אינו יכול לבצע הזמנה זו.

בפרויקט נעשה שימוש בפעולה synchronized כדי לוודא שכאשר לקוח אחד מזמין תור ליום ושעה מסוימת , ההזמנה שלו תבוצע מתחילתה ועד סופה ללא הפרעה של לקוח אחר. כך נמנעת האפשרות ששני לקוחות מזמינים את אותו התור לאותו יום ושעה והלקוח הראשון אשר בחר בתור מסוים יקבל אותו בוודאות.

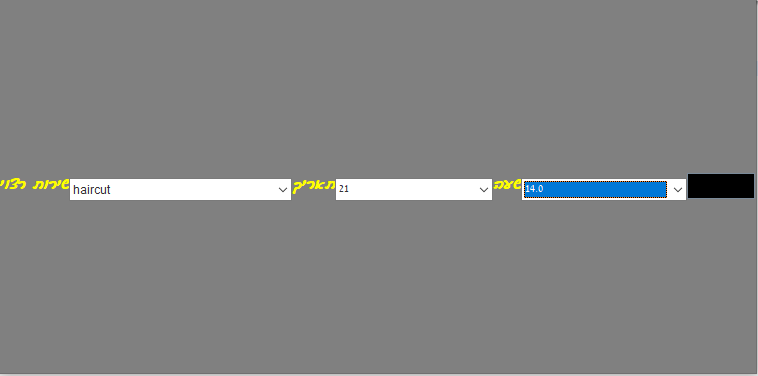
# צילומי מסך:

ראשית הפעל את הסרבר

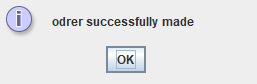
הרץ את המחלקה MultiThreadServerGUI ובעקבות זאת יופיע החלון הבא:

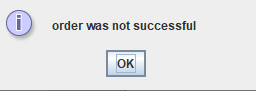


לאחר מכן , נריץ את ממשק המשתמש , בו המשתמש יכול להזמין תור לשירותים מסוימים שמציעה המספרה

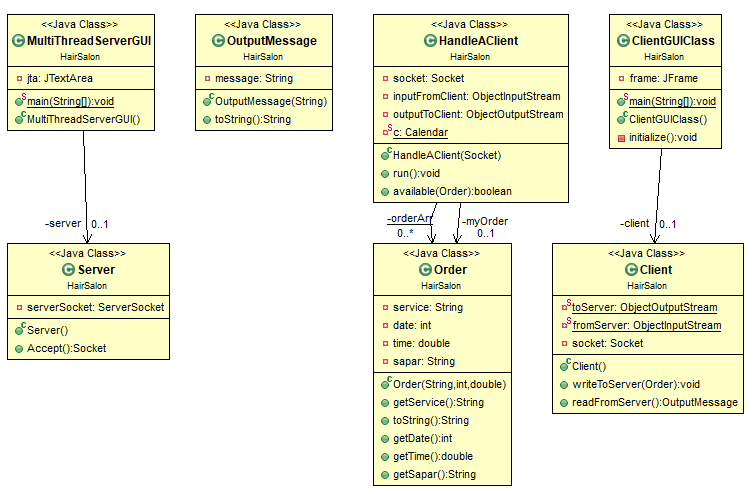


בשלב הבא יתכנו שתי אפשרויות , הזמנה בוצעה או הזמנה לא בוצעה





# תרשים :UML



# פירוט מחלקות:

**:Server מחלקת**

שרת עבור התוכנה.

|  |  |
| --- | --- |
| **שדה** | **פירוט** |
| **private** ServerSocket serverSocket | סוקט לחיבורי משתמשים  חדשים |

|  |  |
| --- | --- |
| **שיטה** | **פירוט** |
| **public** Socket Accept() | ממתין לחיבור שיבוצע לסוקט  ומקבל אותו. |
| **public** Server() | פעולה בונה. |

**:MultiThreadServerGUIמחלקת**

מחלקה גרפית של השרת היוצרת תהליך לשליחת הודעות ממשתמשים.

|  |  |
| --- | --- |
| **שדה** | **פירוט** |
| **private** JTextArea jta | שדה טקסט |
| **private** Server server | משתנה מסוג סרבר (בניית עצם.) |

|  |  |
| --- | --- |
| **שיטה** | **פירוט** |
| **public** MultiThreadServerGUI() | פעולה בונה היוצרת תהליכים לפי כמות  הסוקטים למשתמשים. |

**:HandleAClient מחלקת**

מחלקת עזר המטפלת בפנייה מהמשתמש אל השרת.

|  |  |
| --- | --- |
| **שדה** | **פירוט** |
| **private** Socket socket; | סוקט לחיבורים. |
| **private** ObjectInputStream inputFromClient; | זרם קלט הנתונים |
| **private** ObjectOutputStream outputToClient; | זרם פלט הנתונים. |
| **private** **static** Order [][] *orderArr* = **new** Order[31][11];; | מטריצת התורים של המספרה |
| **private** **static** Calendar *c* = Calendar.*getInstance*(); | לוח שנה |

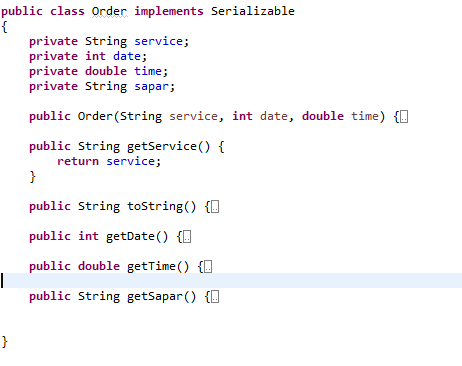
|  |  |
| --- | --- |
| **שיטה** | **פירוט** |
| **public** HandleAClient(Socket socket) | פעולה בונה. |
| **public** **void** run() | הפעולה האחראית על האזנה תמידית ללקוח וקישור בינו לבין השרת. |
|  |  |
| **public** **synchronized** **boolean** available(Order d) | הפעולה האחראית על הכנסת ההזמנה של הלקוח למטריצת התורים , פעולה המונעת התנגשות בין שני לקוחות עקב היותה syncronized |
|  |
|  |
|  |  |

**:Client מחלקת**

|  |  |
| --- | --- |
| **שדה** | **פירוט** |
| **private** Socket socket; | הסוקט שמחבר בין הלקוח לשרת |
| **private** **static** ObjectOutputStream *toServer*; | תקשורת עם הסרבר כלפי חוץ |
| **private** **static** ObjectInputStream *fromServer*; | תקשורת עם הסרבר כלפי פנים |

|  |  |
| --- | --- |
| **שיטה** | **פירוט** |
| **public** Client() | פעולה בונה המגדירה את הסוקט ומקבלת  ושולחת מידע אל הסרבר ומהסרבר. |
| **public** **void** writeToServer(Order d) | פעולה הכותבת ושולחת אל הסרבר את  ההזמנה |
| **public** OutputMessage readFromServer() | פעולה הקוראת את ההודעות מהסרבר |

**:Orderמחלקת**



# קטעי קוד:

1. **package** HairSalon;
2. **import** java.io.DataInputStream;
3. **import** java.io.DataOutputStream;
4. **import** java.io.IOException;
5. **import** java.io.ObjectInputStream;
6. **import** java.io.ObjectOutputStream;
7. **import** java.net.Socket;
9. **class** Client {
11. // IO streams
12. **private** **static** ObjectOutputStream toServer;
13. **private** **static** ObjectInputStream fromServer;
15. **private** Socket socket;
17. **public** Client() {
19. **try** {
20. // Create a socket to connect to the server
21. socket = **new** Socket("localhost", 8000);
23. // Create an output stream to send data
24. // to the server
25. toServer = **new** ObjectOutputStream(socket.getOutputStream());
27. // Create an input stream to receive data
28. // from the server
29. fromServer = **new** ObjectInputStream(socket.getInputStream());

32. } **catch** (IOException ex) {
33. }
34. }
36. **public** **void** writeToServer(Order d) {
37. **try** {
38. toServer.writeObject(d);;
39. toServer.flush();
40. } **catch** (IOException e) {
41. e.printStackTrace();
42. }
43. }
45. **public** OutputMessage readFromServer() {
46. **try** {
47. **try** {
48. **return** (OutputMessage)fromServer.readObject();
49. } **catch** (ClassNotFoundException e) {
50. // TODO Auto-generated catch block
51. e.printStackTrace();
52. }
53. } **catch** (IOException e) {
54. e.printStackTrace();
55. }
56. **return** **null**;
57. }
59. }

**ClientGUIClass**

1. **package** HairSalon;
3. **import** java.awt.EventQueue;
5. **import** javax.swing.JFrame;
6. **import** javax.swing.JLabel;
7. **import** javax.swing.JOptionPane;
9. **import** java.awt.BorderLayout;
10. **import** java.awt.GridBagLayout;
11. **import** java.awt.GridBagConstraints;
12. **import** net.miginfocom.swing.MigLayout;
13. **import** javax.swing.GroupLayout;
14. **import** javax.swing.GroupLayout.Alignment;
15. **import** com.jgoodies.forms.layout.FormLayout;
16. **import** com.jgoodies.forms.layout.ColumnSpec;
17. **import** com.jgoodies.forms.layout.RowSpec;
19. //import jdk.internal.org.jline.reader.Parser.ParseContext;
21. **import** com.jgoodies.forms.layout.FormSpecs;
22. **import** java.awt.Choice;
23. **import** java.awt.Font;
24. **import** java.awt.Color;
25. **import** java.awt.SystemColor;
26. **import** javax.swing.JButton;
27. **import** java.awt.event.ActionListener;
28. **import** java.util.Calendar;
29. **import** java.awt.event.ActionEvent;
30. **import** java.awt.CardLayout;
31. **import** javax.swing.BoxLayout;
33. **public** **class** ClientGUIClass {
34. **private** Client client = **new** Client();
35. **private** JFrame frame;
37. /\*\*
38. \* Launch the application.
39. \*/
40. **public** **static** **void** main(String[] args) {


44. EventQueue.invokeLater(**new** Runnable() {
45. **public** **void** run() {
46. **try** {
47. ClientGUIClass window = **new** ClientGUIClass();
48. window.frame.setVisible(**true**);
49. } **catch** (Exception e) {
50. e.printStackTrace();
51. }
52. }
53. });
54. }
56. /\*\*
57. \* Create the application.
58. \*/
59. **public** ClientGUIClass() {
60. initialize();
61. }
63. /\*\*
64. \* Initialize the contents of the frame.
65. \*/
66. **private** **void** initialize() {
67. frame = **new** JFrame();
68. frame.getContentPane().setBackground(Color.GRAY);
69. frame.setBounds(100, 100, 773, 412);
70. frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);
71. frame.getContentPane().setLayout(**new** BoxLayout(frame.getContentPane(), BoxLayout.X\_AXIS));
73. Choice selectedService = **new** Choice();
74. selectedService.add("haircut");
75. selectedService.add("beard");
76. selectedService.add("haircut+beard");
78. JLabel lbl\_service = **new** JLabel("\u05E9\u05D9\u05E8\u05D5\u05EA \u05E8\u05E6\u05D5\u05D9");
79. lbl\_service.setForeground(Color.YELLOW);
80. lbl\_service.setFont(**new** Font("Guttman Yad-Brush", Font.BOLD, 14));
81. frame.getContentPane().add(lbl\_service);
82. frame.getContentPane().add(selectedService);
84. Choice date = **new** Choice();
85. Calendar c = Calendar.getInstance();
86. //System.out.println("The Current Date is:" + c.getTime());
87. **for**(**int** i=c.get(Calendar.DATE);i<31;i++)
88. {
89. date.add(i+"");
90. }
92. JLabel lbl\_date = **new** JLabel("\u05EA\u05D0\u05E8\u05D9\u05DA");
93. lbl\_date.setForeground(Color.YELLOW);
94. lbl\_date.setFont(**new** Font("Guttman Yad-Brush", Font.BOLD, 14));
95. frame.getContentPane().add(lbl\_date);
96. frame.getContentPane().add(date);

99. Choice time = **new** Choice();
100. **for**(**double** i=10.0;i<21.0;i++)
101. {
102. time.add(i+"");

105. }
106. JLabel lbl\_time = **new** JLabel("\u05E9\u05E2\u05D4");
107. lbl\_time.setForeground(Color.YELLOW);
108. lbl\_time.setFont(**new** Font("Guttman Yad-Brush", Font.BOLD, 14));
109. frame.getContentPane().add(lbl\_time);
110. frame.getContentPane().add(time);





117. JButton btnDone = **new** JButton("\u05D4\u05D6\u05DE\u05DF");
118. btnDone.setBackground(Color.CYAN);
119. btnDone.addActionListener(**new** ActionListener() {
120. **public** **void** actionPerformed(ActionEvent e)
121. {
122. Order myOrder = **new** Order(selectedService.getSelectedItem(), Integer.parseInt(date.getSelectedItem()), Double.parseDouble(time.getSelectedItem()));
123. client.writeToServer(myOrder);
124. System.out.println("send to server : "+myOrder);
125. JOptionPane.showMessageDialog(**null**,((OutputMessage)client.readFromServer()).toString());
127. }
128. });
129. btnDone.setBackground(Color.BLACK);
130. btnDone.setForeground(Color.BLACK);
131. btnDone.setFont(**new** Font("Guttman Yad-Brush", Font.BOLD, 14));
132. frame.getContentPane().add(btnDone);
133. }
134. }

**HandleAClient**

1. **package** HairSalon;
3. **import** java.io.DataInputStream;
4. **import** java.io.DataOutputStream;
5. **import** java.io.IOException;
6. **import** java.io.ObjectInputStream;
7. **import** java.io.ObjectOutputStream;
8. **import** java.net.Socket;
9. **import** java.util.Calendar;
10. **import** java.util.StringTokenizer;
12. **class** HandleAClient **implements** Runnable {
13. **private** Socket socket; // A connected socket
15. **private** ObjectInputStream inputFromClient;
16. **private** ObjectOutputStream outputToClient;
17. **private** Order myOrder;
18. **private** **static** Calendar c = Calendar.getInstance();
19. **private** **static** Order [][] orderArr = **new** Order[31][11];;
21. **public** HandleAClient(Socket socket) {
22. **this**.socket = socket;
23. /\*
24. for (int i = 0; i < 30; i++) {
25. for (int j = 0; j < 11; j++) {
26. orderArr[i][j] = null;
27. }
28. }
29. \*/
31. }
33. **public** **void** run() {
34. OutputMessage opGood=**new** OutputMessage("odrer successfully made");
35. OutputMessage opFail=**new** OutputMessage("order was not successful");
36. **try** {
37. // Create data input and output streams
38. outputToClient = **new** ObjectOutputStream(socket.getOutputStream());
39. inputFromClient = **new** ObjectInputStream(socket.getInputStream());
41. // Continuously serve the client
42. **while** (**true**) {
43. // Receive order from the client
44. **try** {
45. myOrder = (Order) inputFromClient.readObject();
46. System.out.println(myOrder);
47. **if** (available(myOrder))
48. {
49. outputToClient.writeObject(opGood);
50. System.out.println("sent back to client : "+opGood);}
51. **else**
52. {
53. outputToClient.writeObject(opFail);
54. System.out.println("sent back to client : "+opFail);}
55. } **catch** (ClassNotFoundException e) {
56. // TODO Auto-generated catch block
57. e.printStackTrace();
58. }
60. // Send area back to the client
62. }
63. } **catch** (IOException e) {
64. System.err.println(e);
65. }
66. }
68. **public** **synchronized** **boolean** available(Order d) {
69. **if** (orderArr[d.getDate() - 1][(**int**) d.getTime() -10] == **null**)
70. {
71. orderArr[d.getDate() - 1][(**int**) d.getTime() -10] = d;
73. **for** (**int** i = 0; i < 30; i++) {
74. **for** (**int** j = 0; j < 11; j++) {
75. System.out.print("["+orderArr[i][j]+"]");
76. }
77. System.out.println("");
78. System.out.println("-------------------------");
79. }
81. **return** **true**;
82. } **else** {
84. **return** **false**;
85. }
86. }
88. }

**MultiThreadServerGui**

1. **package** HairSalon;
2. **import** java.io.\*;
3. **import** java.net.\*;
4. **import** java.util.\*;
5. **import** java.awt.\*;
6. **import** javax.swing.\*;
8. **public** **class** MultiThreadServerGUI **extends** JFrame {
10. **private** JTextArea jta = **new** JTextArea();
11. **private** Server server = **new** Server();

14. **public** **static** **void** main(String[] args) {
15. **new** MultiThreadServerGUI();
16. }
18. **public** MultiThreadServerGUI() {
19. // Place text area on the frame
20. setLayout(**new** BorderLayout());
21. add(**new** JScrollPane(jta), BorderLayout.CENTER);
22. setTitle("MultiThreadServer");
23. setSize(500, 300);
24. setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);
25. setVisible(**true**); // It is necessary to show the frame here!
27. jta.append("MultiThreadServer started at " + **new** Date() + '\n');


31. **int** clientNo = 1;
33. **while** (**true**) {
34. // Listen for a new connection request
35. Socket socket = server.Accept();
37. // Display the client number
38. jta.append("Starting thread for client " + clientNo + " at " + **new** Date() + '\n');
40. // Find the client's host name, and IP address
41. InetAddress inetAddress = socket.getInetAddress();
42. jta.append("Client " + clientNo + "'s host name is " + inetAddress.getHostName() + "\n");
43. jta.append("Client " + clientNo + "'s IP Address is " + inetAddress.getHostAddress() + "\n");
45. // Create a new task for the connection
46. Thread task = **new** Thread(**new** HandleAClient(socket));
47. task.start();
48. clientNo++;
49. }
50. }
51. }

**ORDER**

1. **package** HairSalon;
3. **import** java.io.Serializable;
5. **public** **class** Order **implements** Serializable
6. {
7. **private** String service;
8. **private** **int** date;
9. **private** **double** time;
10. **private** String sapar;
12. **public** Order(String service, **int** date, **double** time) {
13. **this**.service = service;
14. **this**.date = date;
15. **this**.time = time;
17. }
19. **public** String getService() {
20. **return** service;
21. }
23. @Override
24. **public** String toString() {
25. **return** "Order [service=" + service + ", date=" + date + ", time=" + time + "]";
26. }
28. **public** **int** getDate() {
29. **return** date;
30. }
32. **public** **double** getTime() {
33. **return** time;
34. }
36. **public** String getSapar() {
37. **return** sapar;
38. }

41. }

**OutputMessage**

1. **package** HairSalon;
3. **import** java.io.Serializable;
5. **public** **class** OutputMessage **implements** Serializable
6. {
7. **private** String message;
9. **public** OutputMessage(String s)
10. {
11. message=s;
13. }
15. @Override
16. **public** String toString() {
17. **return** message;
18. }
20. }

**SERVER**

1. **package** HairSalon;
3. **import** java.io.IOException;
4. **import** java.net.ServerSocket;
5. **import** java.net.Socket;
7. **class** Server {
9. **private** ServerSocket serverSocket;
11. **public** Server() {
12. **try** {
13. serverSocket = **new** ServerSocket(8000);
14. } **catch** (IOException e) {
15. e.printStackTrace();
16. }
17. }
19. **public** Socket Accept() {
20. **try** {
21. **return** serverSocket.accept();
22. } **catch** (IOException e) {
23. e.printStackTrace();
24. }
25. **return** **null**;
26. }
28. }