**Name: Muhammad Anas Baig**

**Enrollment No.: 01-134152-037**

**Section: BS(CS)-5A**

**ASSIGNMENT # 2**

**Visual Programming**

**BS(CS) – 5A**

**October 23, 2017**

**Source Code:**

**Form1.cs Code:**

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.ComponentModel;
4. **using** System.Data;
5. **using** System.Drawing;
6. **using** System.Linq;
7. **using** System.Text;
8. **using** System.Windows.Forms;
9. **using** System.IO;
10. **using** System.Collections;
12. **namespace** Assignment2
13. {
14. **public** partial **class** Form1 : Form
15. {
16. **public** Form1()
17. {
18. InitializeComponent();
19. displayTable();
20. }
22. **private** **void** button1\_Click(**object** sender, EventArgs e)
23. {
24. Form2 f = **new** Form2();
25. f.Show();
26. displayTable();
27. }
29. **private** **void** button2\_Click(**object** sender, EventArgs e)
30. {
31. Form3 f = **new** Form3();
32. f.Show();
33. displayTable();
34. }
36. **private** **void** button3\_Click(**object** sender, EventArgs e)
37. {
39. }
41. **private** **void** button3\_Click\_1(**object** sender, EventArgs e)
42. {
43. Form4 f = **new** Form4();
44. f.Show();
45. displayTable();
46. }
48. **private** **void** button4\_Click(**object** sender, EventArgs e)
49. {
50. Form5 f = **new** Form5();
51. f.Show();
52. displayTable();
53. }
55. **private** **void** button5\_Click(**object** sender, EventArgs e)
56. {
57. Form6 f = **new** Form6();
58. f.Show();
59. displayTable();
60. }
62. **private** **void** button6\_Click(**object** sender, EventArgs e)
63. {
64. DateTime tt = DateTime.Now;
65. label35.Text = Convert.ToString(tt.ToUniversalTime());
66. }
67. **private** **void** displayTable()
68. {
69. ArrayList tableList = **new** ArrayList(); //ArrayList to store list of tables
70. table t = **new** table();
71. tableList = t.readTableFileList(); //reads tables from file to list
73. **for** (**int** i = 0; i < tableList.Count; i++) //checks each table
74. {
75. **if** (i == 0)
76. {
77. label13.Text = ((tableList[i] **as** table).gameStatusProperty == 0) ? "✖" : "✔";
78. label23.Text = Convert.ToString((tableList[i] **as** table).gameStatusProperty);
79. label35.Text = ((tableList[i] **as** table).gameStatusProperty != 0)?Convert.ToString((tableList[i] **as** table).startTimeProperty.ToUniversalTime()):"-";
80. }
81. **if** (i == 1)
82. {
83. label14.Text = ((tableList[i] **as** table).gameStatusProperty == 0) ? "✖" : "✔";
84. label24.Text = Convert.ToString((tableList[i] **as** table).gameStatusProperty);
85. label36.Text = ((tableList[i] **as** table).gameStatusProperty != 0)?Convert.ToString((tableList[i] **as** table).startTimeProperty.ToUniversalTime()):"-";
86. }
87. **if** (i == 2)
88. {
89. label15.Text = ((tableList[i] **as** table).gameStatusProperty == 0) ? "✖" : "✔";
90. label25.Text = Convert.ToString((tableList[i] **as** table).gameStatusProperty);
91. label37.Text = ((tableList[i] **as** table).gameStatusProperty != 0)?Convert.ToString((tableList[i] **as** table).startTimeProperty.ToUniversalTime()):"-";
92. }
93. **if** (i == 3)
94. {
95. label16.Text = ((tableList[i] **as** table).gameStatusProperty == 0) ? "✖" : "✔";
96. label26.Text = Convert.ToString((tableList[i] **as** table).gameStatusProperty);
97. label38.Text = ((tableList[i] **as** table).gameStatusProperty != 0)?Convert.ToString((tableList[i] **as** table).startTimeProperty.ToUniversalTime()):"-";
98. }
99. **if** (i == 4)
100. {
101. label17.Text = ((tableList[i] **as** table).gameStatusProperty == 0) ? "✖" : "✔";
102. label27.Text = Convert.ToString((tableList[i] **as** table).gameStatusProperty);
103. label39.Text = ((tableList[i] **as** table).gameStatusProperty != 0)?Convert.ToString((tableList[i] **as** table).startTimeProperty.ToUniversalTime()):"-";
104. }
105. **if** (i == 5)
106. {
107. label18.Text = ((tableList[i] **as** table).gameStatusProperty == 0) ? "✖" : "✔";
108. label28.Text = Convert.ToString((tableList[i] **as** table).gameStatusProperty);
109. label40.Text = ((tableList[i] **as** table).gameStatusProperty != 0)?Convert.ToString((tableList[i] **as** table).startTimeProperty.ToUniversalTime()):"-";
110. }
111. **if** (i == 6)
112. {
113. label19.Text = ((tableList[i] **as** table).gameStatusProperty == 0) ? "✖" : "✔";
114. label29.Text = Convert.ToString((tableList[i] **as** table).gameStatusProperty);
115. label41.Text = ((tableList[i] **as** table).gameStatusProperty != 0)?Convert.ToString((tableList[i] **as** table).startTimeProperty.ToUniversalTime()):"-";
116. }
117. **if** (i == 7)
118. {
119. label20.Text = ((tableList[i] **as** table).gameStatusProperty == 0) ? "✖" : "✔";
120. label30.Text = Convert.ToString((tableList[i] **as** table).gameStatusProperty);
121. label42.Text = ((tableList[i] **as** table).gameStatusProperty != 0)?Convert.ToString((tableList[i] **as** table).startTimeProperty.ToUniversalTime()):"-";
122. }
123. **if** (i == 8)
124. {
125. label21.Text = ((tableList[i] **as** table).gameStatusProperty == 0) ? "✖" : "✔";
126. label31.Text = Convert.ToString((tableList[i] **as** table).gameStatusProperty);
127. label43.Text = ((tableList[i] **as** table).gameStatusProperty != 0)?Convert.ToString((tableList[i] **as** table).startTimeProperty.ToUniversalTime()):"-";
128. }
129. }
130. }
132. **private** **void** button7\_Click(**object** sender, EventArgs e)
133. {
134. displayTable();
135. Application.Exit();
136. }
138. **private** **void** label1\_Click(**object** sender, EventArgs e)
139. {
141. }
142. }
143. }

**Form2.cs Code:**

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.ComponentModel;
4. **using** System.Data;
5. **using** System.Drawing;
6. **using** System.Linq;
7. **using** System.Text;
8. **using** System.Windows.Forms;
10. **namespace** Assignment2
11. {
12. **public** partial **class** Form2 : Form
13. {
14. **public** Form2()
15. {
16. InitializeComponent();
17. }
19. **private** **void** label3\_Click(**object** sender, EventArgs e)
20. {
21. }
23. **private** **void** button1\_Click(**object** sender, EventArgs e)
24. {
25. player p = **new** player();
26. **if** (!p.createNewPlayer(**double**.Parse(textBox1.Text), textBox2.Text, textBox3.Text, **double**.Parse(textBox4.Text)))
27. {
28. label6.Text = "ERROR! Player-ID already registered.";
29. }
30. **else**
31. {
32. label6.Text = "DONE! Player succesfully registered.";
33. }
34. }
36. **private** **void** label6\_Click(**object** sender, EventArgs e)
37. {
39. }
40. }
41. }

**Form3.cs Code:**

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.ComponentModel;
4. **using** System.Data;
5. **using** System.Drawing;
6. **using** System.Linq;
7. **using** System.Text;
8. **using** System.Windows.Forms;
10. **namespace** Assignment2
11. {
12. **public** partial **class** Form3 : Form
13. {
14. **public** Form3()
15. {
16. InitializeComponent();
17. }
19. **private** **void** label2\_Click(**object** sender, EventArgs e)
20. {
22. }
24. **private** **void** button1\_Click(**object** sender, EventArgs e)
25. {
26. player p = **new** player();
27. **if** (radioButton3.Checked)
28. {
29. **if**(p.searchPlayer(textBox1.Text, 1))
30. {
31. label10.Text = Convert.ToString(p.userIDProperty);
32. label11.Text = p.firstNameProperty+" "+p.lastNameProperty;
33. label12.Text = Convert.ToString(p.cnicProperty);
34. label13.Text = Convert.ToString(p.wonProperty);
35. label14.Text = Convert.ToString(p.drawProperty);
36. label15.Text = Convert.ToString(p.lostProperty );
37. label16.Text = "DONE! Player found.";
38. }
39. **else**
40. {
41. label10.Text = "-";
42. label11.Text = "-";
43. label12.Text = "-";
44. label13.Text = "-";
45. label14.Text = "-";
46. label15.Text = "-";
47. label16.Text = "SORRY! Player not found.";
48. }
49. }
50. **if** (radioButton2.Checked)
51. {
52. **if** (p.searchPlayer(textBox1.Text, 2))
53. {
54. label10.Text = Convert.ToString(p.userIDProperty);
55. label11.Text = p.firstNameProperty + " " + p.lastNameProperty;
56. label12.Text = Convert.ToString(p.cnicProperty);
57. label13.Text = Convert.ToString(p.wonProperty);
58. label14.Text = Convert.ToString(p.drawProperty);
59. label15.Text = Convert.ToString(p.lostProperty);
60. label16.Text = "DONE! Player found.";
61. }
62. **else**
63. {
64. label10.Text = "-";
65. label11.Text = "-";
66. label12.Text = "-";
67. label13.Text = "-";
68. label14.Text = "-";
69. label15.Text = "-";
70. label16.Text = "SORRY! Player not found.";
71. }
72. }
73. **if** (radioButton1.Checked)
74. {
75. **if** (p.searchPlayer(textBox1.Text, 3))
76. {
77. label10.Text = Convert.ToString(p.userIDProperty);
78. label11.Text = p.firstNameProperty + " " + p.lastNameProperty;
79. label12.Text = Convert.ToString(p.cnicProperty);
80. label13.Text = Convert.ToString(p.wonProperty);
81. label14.Text = Convert.ToString(p.drawProperty);
82. label15.Text = Convert.ToString(p.lostProperty);
83. label16.Text = "DONE! Player found.";
84. }
85. **else**
86. {
87. label10.Text = "-";
88. label11.Text = "-";
89. label12.Text = "-";
90. label13.Text = "-";
91. label14.Text = "-";
92. label15.Text = "-";
93. label16.Text = "SORRY! Player not found.";
94. }
95. }
96. }
97. }
98. }

**Form4.cs Code:**

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.ComponentModel;
4. **using** System.Data;
5. **using** System.Drawing;
6. **using** System.Linq;
7. **using** System.Text;
8. **using** System.Windows.Forms;
10. **namespace** Assignment2
11. {
12. **public** partial **class** Form4 : Form
13. {
14. **public** Form4()
15. {
16. InitializeComponent();
17. }
19. **private** **void** button1\_Click(**object** sender, EventArgs e)
20. {
21. table t = **new** table();
22. **if** (t.assignNewTable(**double**.Parse(textBox1.Text)))
23. {
24. label3.Text = "DONE! Table assigned.";
25. }
26. **else**
27. {
28. label3.Text = "SORRY! Please wait, all tables are filled.";
29. }
30. }
31. }
32. }

**Form5.cs Code:**

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.ComponentModel;
4. **using** System.Data;
5. **using** System.Drawing;
6. **using** System.Linq;
7. **using** System.Text;
8. **using** System.Windows.Forms;
10. **namespace** Assignment2
11. {
12. **public** partial **class** Form5 : Form
13. {
14. **public** Form5()
15. {
16. InitializeComponent();
17. }
19. **private** **void** button1\_Click(**object** sender, EventArgs e)
20. {
21. table t = **new** table();
22. **if** (t.assignNewTable(**double**.Parse(textBox1.Text), **double**.Parse(textBox2.Text)))
23. {
24. label3.Text = "DONE! Table assigned.";
25. }
26. **else**
27. {
28. label3.Text = "SORRY! Please wait, all tables are filled.";
29. }
30. }
31. }
32. }

**Form6.cs Code:**

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.ComponentModel;
4. **using** System.Data;
5. **using** System.Drawing;
6. **using** System.Linq;
7. **using** System.Text;
8. **using** System.Windows.Forms;
10. **namespace** Assignment2
11. {
12. **public** partial **class** Form6 : Form
13. {
14. **public** Form6()
15. {
16. InitializeComponent();
17. }
19. **private** **void** label2\_Click(**object** sender, EventArgs e)
20. {
22. }
24. **private** **void** button1\_Click(**object** sender, EventArgs e)
25. {
26. table t = **new** table();
27. t.submitTableResults(**int**.Parse(textBox1.Text), 1);
28. }
30. **private** **void** button1\_Click\_1(**object** sender, EventArgs e)
31. {
32. table t = **new** table();
33. button2.Text = "Player-1: " + Convert.ToString(t.playerOneID(**int**.Parse(textBox1.Text)));
34. button3.Text = "Player-2: " + Convert.ToString(t.playerTwoID(**int**.Parse(textBox1.Text)));
35. **int** tab = **int**.Parse(textBox1.Text);
36. **if** (tab == 1 || tab == 2 || tab == 3 || tab == 4 || tab == 5 || tab == 6 || tab == 7 || tab == 8 || tab == 9)
37. {
38. label3.Text = "DONE! Table found.";
39. }
40. **else**
41. {
42. label3.Text = "ERROR! Table not found.";
43. }
44. }
46. **private** **void** button3\_Click(**object** sender, EventArgs e)
47. {
48. table t = **new** table();
49. t.submitTableResults(**int**.Parse(textBox1.Text), 2);
50. }
52. **private** **void** button4\_Click(**object** sender, EventArgs e)
53. {
54. table t = **new** table();
55. t.submitTableResults(**int**.Parse(textBox1.Text), 3);
56. }
57. }
58. }

**Player.cs Code:**

1. //by Muhammad Anas Baig-(01-134152-037)-BS(CS)-5A-VP
2. **using** System;
3. **using** System.Collections.Generic;
4. **using** System.Linq;
5. **using** System.Text;
6. **using** System.IO;
7. **using** System.Collections;
9. **class** player
10. {
11. **double** userID;
12. **string** firstName;
13. **string** lastName;
14. **double** cnic;
15. **int** won;
16. **int** draw;
17. **int** lost;
19. **public** **double** userIDProperty
20. {
21. **get**{    **return** userID;  }
22. **set**{    userID = value; }
23. }
24. **public** **string** firstNameProperty
25. {
26. **get**{    **return** firstName;   }
27. **set**{    firstName = value;  }
28. }
29. **public** **string** lastNameProperty
30. {
31. **get**{    **return** lastName;    }
32. **set**{    lastName = value;   }
33. }
34. **public** **double** cnicProperty
35. {
36. **get**{    **return** cnic;    }
37. **set**{    cnic = value;   }
38. }
39. **public** **int** wonProperty
40. {
41. **get**{    **return** won;     }
42. **set**{    won = value;    }
43. }
44. **public** **int** drawProperty
45. {
46. **get** {   **return** draw;    }
47. **set** {   draw = value;   }
48. }
49. **public** **int** lostProperty
50. {
51. **get**{    **return** lost;    }
52. **set**{    lost = value;   }
53. }
54. **public** player()
55. {
56. won = 0;
57. draw = 0;
58. lost = 0;
59. }
60. **public** **string** fullName(**double** id) //function that returns concatenation of firstname and lastname
61. {
62. ArrayList playerList = **new** ArrayList(); //player list
63. playerList = readPlayerFile(); //reading file to list
65. **for** (**int** i = 0; i < playerList.Count; i++) //checks each user in list
66. {
67. **if** ((playerList[i] **as** player).userID == id) //checks for the required user
68. {
69. **return** ((playerList[i] **as** player).firstName + " " + (playerList[i] **as** player).lastName); //returns concatenation of firstname and lastname
70. }
71. }
72. **return** (""); //if user not found
73. }
74. **public** **bool** createNewPlayer(**double** userID ,**string** firstName, **string** lastName, **double** cnic)
75. {
76. **if** (searchUniqueUserID(userID)) //not unique
77. {
78. **return** **false**;
79. }
80. **else**
81. {
82. **this**.userIDProperty = userID;
83. **this**.firstNameProperty = firstName;
84. **this**.lastNameProperty = lastName;
85. **this**.cnicProperty = cnic;
86. writePlayerFile(**this**); //appends the new player in player file
87. **return** **true**;
88. }
89. }
90. **public** **bool** searchUniqueUserID(**double** id) //before creating new user this methods checks either the userID is already taken or not
91. {
92. ArrayList playerList = **new** ArrayList(); //player list
93. playerList = readPlayerFile(); //reading file to list
95. **for** (**int** i = 0; i < playerList.Count; i++)  //checks each user in list
96. {
97. **if** ((playerList[i] **as** player).userID == id) //checks either userID is already taken or not
98. {
99. **return** **true**;
100. }
101. }
102. **return** **false**;
103. }
104. **public** **bool** searchPlayer(**string** phrase, **int** type) //to search a specific user in the system
105. {
106. ArrayList playerList = **new** ArrayList(); //player list
107. playerList = readPlayerFile(); //reading file to list
109. **if**(type == 1) //search by userID
110. {
111. **double** ID = **double**.Parse(phrase);
113. **for** (**int** i = 0; i < playerList.Count; i++) //checks each user in list
114. {
115. **if** ((playerList[i] **as** player).userID == ID) //checks for the requied userID
116. {
117. **this**.userIDProperty = (playerList[i] **as** player).userIDProperty;
118. **this**.firstNameProperty = (playerList[i] **as** player).firstNameProperty;
119. **this**.lastNameProperty = (playerList[i] **as** player).lastNameProperty;
120. **this**.cnicProperty = (playerList[i] **as** player).cnicProperty;
121. **this**.wonProperty = (playerList[i] **as** player).wonProperty;
122. **this**.drawProperty = (playerList[i] **as** player).drawProperty;
123. **this**.lostProperty = (playerList[i] **as** player).lostProperty;
124. **return** **true**;
125. }
126. }
127. }
128. **if** (type == 2) //search by name
129. {
130. **for** (**int** i = 0; i < playerList.Count; i++) //checks each user in list
131. {
132. **if** (((playerList[i] **as** player).firstName + " " + (playerList[i] **as** player).lastName) == phrase) //checks for the requied name(firstName + lastName)
133. {
134. **this**.userIDProperty = (playerList[i] **as** player).userIDProperty;
135. **this**.firstNameProperty = (playerList[i] **as** player).firstNameProperty;
136. **this**.lastNameProperty = (playerList[i] **as** player).lastNameProperty;
137. **this**.cnicProperty = (playerList[i] **as** player).cnicProperty;
138. **this**.wonProperty = (playerList[i] **as** player).wonProperty;
139. **this**.drawProperty = (playerList[i] **as** player).drawProperty;
140. **this**.lostProperty = (playerList[i] **as** player).lostProperty;
141. **return** **true**;
142. }
143. }
144. }
145. **if** (type == 3) //search by cnic
146. {
147. **double** num = **double**.Parse(phrase);
149. **for** (**int** i = 0; i < playerList.Count; i++) //checks each user in list
150. {
151. **if** ((playerList[i] **as** player).cnic == num) //check for the required cnic
152. {
153. **this**.userIDProperty = (playerList[i] **as** player).userIDProperty;
154. **this**.firstNameProperty = (playerList[i] **as** player).firstNameProperty;
155. **this**.lastNameProperty = (playerList[i] **as** player).lastNameProperty;
156. **this**.cnicProperty = (playerList[i] **as** player).cnicProperty;
157. **this**.wonProperty = (playerList[i] **as** player).wonProperty;
158. **this**.drawProperty = (playerList[i] **as** player).drawProperty;
159. **this**.lostProperty = (playerList[i] **as** player).lostProperty;
160. **return** **true**;
161. }
162. }
163. }
164. **return** **false**;
165. }
166. **public** **void** displayAllPlayers() //to display all players statistics
167. {
168. ArrayList playerList = **new** ArrayList(); //player list
169. playerList = readPlayerFile(); //reading file to list
170. Console.Write("--------------------------------------------------------------------------------");
171. Console.WriteLine( "DISPLAY ALL PLAYERS STATISTICS:");
172. Console.Write("--------------------------------------------------------------------------------");
174. **for** ( **int** i = 0; i < playerList.Count; i++ ) //checks each user in list
175. {
176. Console.WriteLine();
177. Console.WriteLine("----------------------");
178. Console.WriteLine("Player - " + (i+1) + " Statistics:");
179. Console.WriteLine("----------------------" );
180. Console.WriteLine("User ID:     " + (playerList[i] **as** player).userID);
181. Console.WriteLine("Name:        " + (playerList[i] **as** player).firstName + " " + (playerList[i] **as** player).lastName);
182. Console.WriteLine("CNIC:        " + (playerList[i] **as** player).cnic);
183. **int** gamesPlayed = ((playerList[i] **as** player).won + (playerList[i] **as** player).draw + (playerList[i] **as** player).lost); //number of games played
184. Console.WriteLine("SCORE:");
185. Console.WriteLine("------");
186. Console.WriteLine("Won:         " + (playerList[i] **as** player).won);
187. Console.WriteLine("Draw:        " + (playerList[i] **as** player).draw);
188. Console.WriteLine("Lost:        " + (playerList[i] **as** player).lost);
189. Console.WriteLine("----------------------");
190. Console.WriteLine("Total Games: " + gamesPlayed);
191. Console.WriteLine("----------------------");
192. }
193. }
194. **public** ArrayList readPlayerFile() //reads player file and returns ArrayList which containts all players data
195. {
196. ArrayList playerList = **new** ArrayList(); //to display all players statistics
197. StreamReader readPlayerFile = **new** StreamReader("Players.txt"); //reading file to list
198. player p;
200. **while** (!readPlayerFile.EndOfStream) //reading file till end
201. {
202. p = **new** player();
203. p.userID = **double**.Parse(readPlayerFile.ReadLine());
204. p.firstName = readPlayerFile.ReadLine();
205. p.lastName = readPlayerFile.ReadLine();
206. p.cnic = **double**.Parse(readPlayerFile.ReadLine());
207. p.won = **int**.Parse(readPlayerFile.ReadLine());
208. p.draw = **int**.Parse(readPlayerFile.ReadLine());
209. p.lost = **int**.Parse(readPlayerFile.ReadLine());
210. playerList.Add( p );
211. }
212. readPlayerFile.Close();
213. **return** playerList; //returning ArrayList which contains all players data
214. }
215. **public** **void** writePlayerFile(player p) //to add new player to the file
216. {
217. StreamWriter writePlayerFile = **new** StreamWriter("Players.txt", **true**); //appending the player file
219. writePlayerFile.WriteLine( p.userID );
220. writePlayerFile.WriteLine( p.firstName );
221. writePlayerFile.WriteLine( p.lastName );
222. writePlayerFile.WriteLine( p.cnic );
223. writePlayerFile.WriteLine( p.won );
224. writePlayerFile.WriteLine( p.draw );
225. writePlayerFile.WriteLine( p.lost );
227. writePlayerFile.Close();
228. }
229. **public** **void** writePlayerFileList(ArrayList playerList) //to write modified/updated data to file -> modify/update -> game Win/Loss
230. {
231. StreamWriter writePlayerFile = **new** StreamWriter("Players.txt"); //not opened in appended mode because all modified/updated data is to write to file -> modify/update -> game Win/Loss
233. **for** (**int** i = 0; i < playerList.Count; i++) //checks each user in list
234. {
235. writePlayerFile.WriteLine((playerList[i] **as** player ).userIDProperty);
236. writePlayerFile.WriteLine((playerList[i] **as** player).firstNameProperty);
237. writePlayerFile.WriteLine((playerList[i] **as** player).lastNameProperty);
238. writePlayerFile.WriteLine((playerList[i] **as** player).cnicProperty);
239. writePlayerFile.WriteLine((playerList[i] **as** player).wonProperty);
240. writePlayerFile.WriteLine((playerList[i] **as** player).drawProperty);
241. writePlayerFile.WriteLine((playerList[i] **as** player).lostProperty);
242. }
243. writePlayerFile.Close();
244. }
245. **public** **void** playerWon(**double** id) //takes userID and updates user's won games
246. {
247. ArrayList playerList = **new** ArrayList(); //player list
248. playerList = readPlayerFile(); //reading file to list
250. **for** (**int** i = 0; i < playerList.Count; i++) //checks each user in list
251. {
252. **if** ((playerList[i] **as** player).userIDProperty == id)
253. {
254. (playerList[i] **as** player).wonProperty = 1 + (playerList[i] **as** player).wonProperty; //increments in user's won games
255. }
256. }
257. writePlayerFileList(playerList);
258. }
259. **public** **void** playerLost(**double** id) //takes userID and updates user's won games
260. {
261. ArrayList playerList = **new** ArrayList(); //player list
262. playerList = readPlayerFile(); //reading file to list
264. **for** (**int** i = 0; i < playerList.Count; i++) //checks each user in list
265. {
266. **if** ((playerList[i] **as** player).userIDProperty == id) //checks if required user is found
267. {
268. (playerList[i] **as** player).lostProperty = 1 + (playerList[i] **as** player).lostProperty; //increments in user's won games
269. }
270. }
271. writePlayerFileList(playerList);
272. }
273. **public** **void** playerDraw(**double** id1, **double** id2) //takes userIDs of player1 and player2 and updates both user's draw games
274. {
275. ArrayList playerList = **new** ArrayList(); //player list
276. playerList = readPlayerFile(); //reading file to list
278. **for** (**int** i = 0; i < playerList.Count; i++) //checks each user in list
279. {
280. **if** ((playerList[i] **as** player).userIDProperty == id1)
281. {
282. (playerList[i] **as** player).drawProperty = 1 + (playerList[i] **as** player).drawProperty; //increments player's draw games
283. }
284. **if** ((playerList[i] **as** player).userIDProperty == id2)
285. {
286. (playerList[i] **as** player).drawProperty = 1 + (playerList[i] **as** player).drawProperty; //increments player's draw games
287. }
288. }
289. writePlayerFileList( playerList );
290. }
291. }
292. //by Muhammad Anas Baig-(01-134152-037)-BS(CS)-5A-VP

**Tables.cs Code:**

1. //by Muhammad Anas Baig-(01-134152-037)-BS(CS)-5A-VP
2. **using** System;
3. **using** System.Collections.Generic;
4. **using** System.Linq;
5. **using** System.Text;
6. **using** System.IO;
7. **using** System.Collections;
9. **class** table
10. {
11. **int** tableID; //stores tableID
12. **int** gameStatus; //stores game status i.e. 0->Empty Table, 1->One Player Assigned, 2->Two Players Assigned
13. DateTime startTime;
14. DateTime endTime;
15. **public** player playerOne = **new** player(); //player1 on table
16. **public** player playerTwo = **new** player(); //player2 on table
18. **public** **int** tableIDProperty
19. {
20. **get** { **return** tableID; }
21. **set** { tableID = value; }
22. }
23. **public** **int** gameStatusProperty
24. {
25. **get** { **return** gameStatus; }
26. **set** { gameStatus = value; }
27. }
28. **public** **double** playerOneProperty
29. {
30. **get** { **return** playerOne.userIDProperty; }
31. **set** { playerOne.userIDProperty = value; }
32. }
33. **public** **double** playerTwoProperty
34. {
35. **get** { **return** playerTwo.userIDProperty; }
36. **set** { playerTwo.userIDProperty = value; }
37. }
38. **public** DateTime startTimeProperty
39. {
40. **get** { **return** startTime; }
41. **set** { startTime = value; }
42. }
43. **public** DateTime endTimeProperty
44. {
45. **get** { **return** endTime; }
46. **set** { endTime = value; }
47. }
48. **public** **bool** searchUniqueTableID(**int** id) //while creating new table checks either the tableID in already assigned or not
49. {
50. ArrayList tableList = **new** ArrayList(); //ArrayList to store list of tables
51. tableList = readTableFileList(); //reads tables from file to list
53. **for** (**int** i = 0; i < tableList.Count; i++) //checks each table
54. {
55. **if** ((tableList[i] **as** table).tableID == id) //checks for the required tableID
56. {
57. **return** **true**;
58. }
59. }
60. **return** **false**;
61. }
62. **public** **void** createNewTable() //to add new table to system
63. {
64. Console.Write("--------------------------------------------------------------------------------");
65. Console.WriteLine("ADD NEW TABLE TO SYSTEM:");
66. Console.Write("--------------------------------------------------------------------------------");
68. **do**
69. {
70. Console.WriteLine("Enter New Table-ID:");
71. **this**.tableID = **int**.Parse(Console.ReadLine());
72. **if** (searchUniqueTableID(**this**.tableID)) //while creating new table checks either the tableID in already assigned or not
73. {
74. Console.WriteLine("ERROR! Table-ID already assigned, kindly choose another.");
75. }
76. }
77. **while** (searchUniqueTableID(**this**.tableID));
79. gameStatusProperty = 0; //game status i.e. 0->Empty Table, 1->One Player Assigned, 2->Two Players Assigned
80. playerOneProperty = 0;
81. playerTwoProperty = 0;
82. startTimeProperty = DateTime.Now;
83. endTimeProperty = DateTime.Now;
84. Console.WriteLine("Table Successfully Created.");
86. writeAddTableFile(**this**); //appends new to table to table file
87. }
88. **public** ArrayList readTableFileList() //reads table file to list and then returns list
89. {
90. ArrayList tableList = **new** ArrayList(); //ArrayList to store list of tables
91. StreamReader readTableFile = **new** StreamReader("Tables.txt"); //read file
92. table t;
94. **while** (!readTableFile.EndOfStream) //reads table file till end
95. {
96. t = **new** table();
97. t.tableIDProperty = **int**.Parse(readTableFile.ReadLine());
98. t.gameStatusProperty = **int**.Parse(readTableFile.ReadLine());
99. t.startTime = DateTime.Parse(readTableFile.ReadLine());
100. t.endTime = DateTime.Parse(readTableFile.ReadLine());
101. t.playerOneProperty = **double**.Parse(readTableFile.ReadLine());
102. t.playerTwoProperty = **double**.Parse(readTableFile.ReadLine());
103. tableList.Add(t);
104. }
105. readTableFile.Close();
106. **return** tableList;
107. }
108. **public** **void** writeAddTableFile(table t) //to add new table to the sysem by appending
109. {
110. StreamWriter writeTableFile = **new** StreamWriter("Tables.txt", **true**); //appending table file
112. writeTableFile.WriteLine(t.tableIDProperty);
113. writeTableFile.WriteLine(t.gameStatusProperty);
114. writeTableFile.WriteLine(t.startTimeProperty);
115. writeTableFile.WriteLine(t.endTimeProperty);
116. writeTableFile.WriteLine(t.playerOneProperty);
117. writeTableFile.WriteLine(t.playerTwoProperty);
119. writeTableFile.Close();
120. }
121. **public** **void** writeTableFileList(ArrayList tableList) //to write modified/updated data to file -> modify/update -> table status
122. {
123. StreamWriter writeTableFile = **new** StreamWriter("Tables.txt"); //not opened in appended mode because all modified/updated data is to write to file -> modify/update -> table status
125. **for** (**int** i = 0; i < tableList.Count; i++)
126. {
127. writeTableFile.WriteLine((tableList[i] **as** table).tableIDProperty);
128. writeTableFile.WriteLine((tableList[i] **as** table).gameStatusProperty);
129. writeTableFile.WriteLine((tableList[i] **as** table).startTimeProperty);
130. writeTableFile.WriteLine((tableList[i] **as** table).endTimeProperty);
131. writeTableFile.WriteLine((tableList[i] **as** table).playerOneProperty);
132. writeTableFile.WriteLine((tableList[i] **as** table).playerTwoProperty);
133. }
134. writeTableFile.Close();
135. }
136. **public** **bool** assignNewTable(**double** playerOneUserID) //to assign new table if one player comes
137. {
138. ArrayList tableList = **new** ArrayList(); //ArrayList to store list of tables
139. tableList = readTableFileList(); //reads tables from file to list
141. **for** (**int** i = 0; i < tableList.Count; i++) //checks each table
142. {
143. **if** ((tableList[i] **as** table).gameStatus == 0) //if table is empty
144. {
145. (tableList[i] **as** table).gameStatusProperty = 1; //game status i.e. 0->Empty Table, 1->One Player Assigned, 2->Two Players Assigned
146. (tableList[i] **as** table).startTimeProperty = DateTime.Now;
147. (tableList[i] **as** table).endTimeProperty = DateTime.Now;
148. (tableList[i] **as** table).playerOneProperty = playerOneUserID;
149. (tableList[i] **as** table).playerTwoProperty = 0;
150. writeTableFileList(tableList); //write again to file
151. **return** **true**;
152. }
153. **else** **if** ((tableList[i] **as** table).gameStatus == 1) //if table has 1 playyer then assign the new player to this table
154. {
155. (tableList[i] **as** table).gameStatusProperty = 2; //game status i.e. 0->Empty Table, 1->One Player Assigned, 2->Two Players Assigned
156. (tableList[i] **as** table).startTimeProperty = DateTime.Now;
157. (tableList[i] **as** table).endTimeProperty = DateTime.Now;
158. (tableList[i] **as** table).playerTwoProperty = playerOneUserID;
159. writeTableFileList(tableList); //write again to file
160. **return** **true**;
161. }
162. }
163. **return** **false**; //all tables filled
164. }
165. **public** **bool** assignNewTable(**double** playerOneUserID, **double** playerTwoUserID) //to assign new table if two players come
166. {
167. ArrayList tableList = **new** ArrayList(); //ArrayList to store list of tables
168. tableList = readTableFileList(); //reads tables from file to list
170. **for** (**int** i = 0; i < tableList.Count; i++) //checks each table
171. {
172. **if** ((tableList[i] **as** table).gameStatus == 0) //if table is empty then assign to them
173. {
174. (tableList[i] **as** table).gameStatusProperty = 2; //game status i.e. 0->Empty Table, 1->One Player Assigned, 2->Two Players Assigned
175. (tableList[i] **as** table).startTimeProperty = DateTime.Now;
176. (tableList[i] **as** table).endTimeProperty = DateTime.Now;
177. (tableList[i] **as** table).playerOneProperty = playerOneUserID;
178. (tableList[i] **as** table).playerTwoProperty = playerTwoUserID;
179. writeTableFileList(tableList); //write again to file
180. Console.WriteLine("Table Successfully Assigned.");
181. **return** **true**;
182. }
183. }
184. **for** (**int** i = 0; i < tableList.Count; i++) //checks each table
185. {
186. **if** ((tableList[i] **as** table).gameStatus == 1) //as no full table is empty so now it will check table where one player is assigned so that they can start game immediately
187. {
188. (tableList[i] **as** table).gameStatusProperty = 2; //game status i.e. 0->Empty Table, 1->One Player Assigned, 2->Two Players Assigned
189. (tableList[i] **as** table).startTimeProperty = DateTime.Now;
190. (tableList[i] **as** table).endTimeProperty = DateTime.Now;
191. (tableList[i] **as** table).playerOneProperty = playerOneUserID;
192. (tableList[i] **as** table).playerTwoProperty = playerTwoUserID;
193. writeTableFileList(tableList);
194. Console.WriteLine("Table Successfully Assigned.");
195. **return** **true**;
196. }
197. }
198. **return** **false**;
199. }
200. **public** **void** displayTableList() //to display all tables status
201. {
202. ArrayList tableList = **new** ArrayList(); //ArrayList to store list of tables
203. tableList = readTableFileList(); //reads tables from file to list
205. Console.Write("--------------------------------------------------------------------------------");
206. Console.WriteLine("DISPLAY ALL TABLES STATUS:");
207. Console.Write("--------------------------------------------------------------------------------");
209. **for** (**int** i = 0; i < tableList.Count; i++) //checks each table
210. {
211. Console.WriteLine();
212. Console.WriteLine("----------------------");
213. Console.WriteLine("Table-ID: " + (tableList[i] **as** table).tableID);
214. Console.WriteLine("----------------------");
215. **if** ((tableList[i] **as** table).gameStatus == 0) //game status i.e. 0->Empty Table, 1->One Player Assigned, 2->Two Players Assigned
216. {
217. Console.WriteLine("0 Players Assigned.");
218. }
219. **else** **if** ((tableList[i] **as** table).gameStatus == 1) //game status i.e. 0->Empty Table, 1->One Player Assigned, 2->Two Players Assigned
220. {
221. player p = **new** player();
222. Console.WriteLine("1 Player Assigned.");
224. **string** playerOneName = p.fullName((tableList[i] **as** table).playerOneProperty);
225. Console.WriteLine("1. Player-1 (ID-" + (tableList[i] **as** table).playerOneProperty + ") " + playerOneName);
226. Console.WriteLine("Start Time:" + (tableList[i] **as** table).startTimeProperty);
227. }
228. **if** ((tableList[i] **as** table).gameStatus == 2) //game status i.e. 0->Empty Table, 1->One Player Assigned, 2->Two Players Assigned
229. {
230. player p = **new** player();
231. Console.WriteLine("2 Players Assigned.");
233. **string** playerOneName = p.fullName((tableList[i] **as** table).playerOneProperty);
234. **string** playerTwoName = p.fullName((tableList[i] **as** table).playerTwoProperty);
235. Console.WriteLine("1. Player-1 (ID-" + (tableList[i] **as** table).playerOneProperty + ") " + playerOneName);
236. Console.WriteLine("2. Player-2 (ID-" + (tableList[i] **as** table).playerTwoProperty + ") " + playerTwoName);
237. Console.WriteLine("Start Time:" + (tableList[i] **as** table).startTimeProperty);
238. }
239. }
240. }
241. **public** **void** submitTableResults(**int** id, **int** result) //to submit game results and clear table status
242. {
243. ArrayList tableList = **new** ArrayList(); //ArrayList to store list of tables
244. tableList = readTableFileList(); //reads tables from file to list
246. **for** (**int** i = 0; i < tableList.Count; i++) //checks each table
247. {
248. **if** ((tableList[i] **as** table).tableID == id) //checks for the required table
249. {
250. player p = **new** player();
251. **if** (result == 1)
252. {
253. p.playerWon((tableList[i] **as** table).playerOneProperty); //updates won status of player1
254. p.playerLost((tableList[i] **as** table).playerTwoProperty); //updates lost status of player2
255. }
256. **else** **if** (result == 2)
257. {
258. p.playerWon((tableList[i] **as** table).playerTwoProperty); //updates won status of player2
259. p.playerLost((tableList[i] **as** table).playerOneProperty); //updates lost status of player1
260. }
261. **else** **if** (result == 3)
262. {
263. p.playerDraw((tableList[i] **as** table).playerOneProperty, (tableList[i] **as** table).playerTwoProperty); //updates draw status of both users
264. }
265. (tableList[i] **as** table).endTimeProperty = DateTime.Now;
267. StreamWriter writeGameLogFile = **new** StreamWriter("GameLog.txt", **true**);
268. writeGameLogFile.WriteLine((tableList[i] **as** table).tableIDProperty);
269. writeGameLogFile.WriteLine((tableList[i] **as** table).playerOneProperty);
270. writeGameLogFile.WriteLine((tableList[i] **as** table).playerTwoProperty);
271. writeGameLogFile.WriteLine((tableList[i] **as** table).startTimeProperty);
272. writeGameLogFile.WriteLine((tableList[i] **as** table).endTimeProperty);
273. writeGameLogFile.Close();
275. (tableList[i] **as** table).gameStatus = 0; //clears table status to empty
276. (tableList[i] **as** table).playerOneProperty = 0; //clears table player1 to empty
277. (tableList[i] **as** table).playerTwoProperty = 0; //clears table player2 to empty
278. //(tableList[i] as table).startTimeProperty = null; //clears table player2 to empty
279. //(tableList[i] as table).playerTwoProperty = 0; //clears table player2 to empty
280. writeTableFileList(tableList);
281. **return**;
282. }
283. }
284. }
285. **public** **double** playerOneID(**int** id) //to submit game results and clear table status
286. {
287. ArrayList tableList = **new** ArrayList(); //ArrayList to store list of tables
288. tableList = readTableFileList(); //reads tables from file to list
290. **for** (**int** i = 0; i < tableList.Count; i++) //checks each table
291. {
292. **if** ((tableList[i] **as** table).tableID == id) //checks for the required table
293. {
294. player p = **new** player();
295. **return** ((tableList[i] **as** table).playerOneProperty);
296. }
297. }
298. **return** 0;
299. }
300. **public** **double** playerTwoID(**int** id) //to submit game results and clear table status
301. {
302. ArrayList tableList = **new** ArrayList(); //ArrayList to store list of tables
303. tableList = readTableFileList(); //reads tables from file to list
305. **for** (**int** i = 0; i < tableList.Count; i++) //checks each table
306. {
307. **if** ((tableList[i] **as** table).tableID == id) //checks for the required table
308. {
309. player p = **new** player();
310. **return** ((tableList[i] **as** table).playerTwoProperty);
311. }
312. }
313. **return** 0;
314. }
315. **public** **void** displayGameLogFile() //to display game history
316. {
317. Console.Write("--------------------------------------------------------------------------------");
318. Console.WriteLine("DISPLAY GAME LOG HISTORY:");
319. Console.Write("--------------------------------------------------------------------------------");
321. **double** id;
322. **double** player1ID;
323. **double** player2ID;
324. DateTime start;
325. DateTime end;
327. StreamReader readGameLogFile = **new** StreamReader("GameLog.txt");
328. **while** (!readGameLogFile.EndOfStream) //reads game log file till end
329. {
330. id = **double**.Parse(readGameLogFile.ReadLine());
331. player1ID = **double**.Parse(readGameLogFile.ReadLine());
332. player2ID = **double**.Parse(readGameLogFile.ReadLine());
333. start = DateTime.Parse(readGameLogFile.ReadLine());
334. end = DateTime.Parse(readGameLogFile.ReadLine());
336. Console.WriteLine("T-ID:" + id + " | P1-ID:" + player1ID + " | P2-ID:" + player2ID + " | START:" + start + " | END:" + end);
337. }
338. }
339. }
340. //by Muhammad Anas Baig-(01-134152-037)-BS(CS)-5A-VP

**Program.cs Code:**

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Linq;
4. **using** System.Windows.Forms;
6. **namespace** Assignment2
7. {
8. **static** **class** Program
9. {
10. /// <summary>
11. /// The main entry point for the application.
12. /// </summary>
13. [STAThread]
14. **static** **void** Main()
15. {
16. Application.EnableVisualStyles();
17. Application.SetCompatibleTextRenderingDefault(**false**);
18. Application.Run(**new** Form1());
19. }
20. }
21. }