OOP ASSIGNMENT - Go Fish!

# Introduction

# Non-Delphi Source

Originally I wanted to make a networked card game. So that the users could play without seeing each other’s hands. So I developed some python to use sockets on a LAN, which was a valid method of networking on my old school network so I built this network handler class:

NetworkHandeler.py

1. **from** socket **import** \*
2. **import** time
4. port = 1234
6. **class** network:
7. host = ""
8. timeOut = 1
9. buffer = 1024
11. **def** \_\_init\_\_(self, host):
12. self.host = host
14. **def** Send(self, data):
15. addr = (self.host, port)
16. Sock = socket(AF\_INET, SOCK\_DGRAM)
17. Sock.sendto(data.encode("utf8", 'ignore'), addr)
18. Sock.close()
20. **def** SendData(self, data):
21. self.Send(data)
22. response = self.Listen()
23. **print**("Got response ", response)
24. **if** (response == "ACK"):
25. **return** "RECIVED ACK"
26. **else**:
27. **return** "NET FAIL"
29. **def** Listen(self):
30. addr = ("", port)
31. Sock = socket(AF\_INET, SOCK\_DGRAM)
32. Sock.bind(addr)
33. (data, hostAddr) = Sock.recvfrom(self.buffer)
34. data = data.decode("utf-8")
35. Sock.close()
36. **return** data
38. **def** ListenForData(self):
39. Data = self.Listen()
40. time.sleep(0.5)
41. self.Send("ACK")
42. **return** Data
44. **def** \_\_exit\_\_(self, exc\_type, exc\_val, exc\_tb):
45. """some clean up code probably should exist"""

48. n = network("172.20.24.65")
49. n.ListenForData()

I then planned for this to communicate to Delphi over a CSV file system as I had already done the hard work of building a decent file handling unit in Delphi and I had split on deliminator capabilities:

CSVHandler.py

1. **import** csv
2. **import** time

5. **class** CSV:
6. filename = ""
7. MsgID = 0
9. **def** \_\_init\_\_(self, filename):
10. self.filename = filename
11. self.CSVAppend("PY IS ALIVE")
13. **def** CSVRead(self):
14. rows = []
15. with open(self.filename, 'r') as csvfile:
16. csvreader = csv.reader(csvfile)
17. **for** row **in** csvreader:
18. rows.append(row)
19. csvfile.close()
20. self.MsgID = rows[-2][0]
21. # Note this is -2 because of the double line spacing bug
22. **return** rows
24. **def** CSVAppend(self, data):
25. with open(self.filename, 'a') as csvfile:
26. csvwriter = csv.writer(csvfile)
27. **for** i **in** range(0, len(data)):
28. row = [self.MsgID, time.time(), data]
29. csvwriter.writerow(row)
30. self.MsgID += 1
31. csvfile.close()
33. **def** CSVClear(self):
34. csvfile = open(self.filename, "w+")
35. csvfile.close()
37. **def** \_\_exit\_\_(self, exc\_type, exc\_val, exc\_tb):
38. self.CSVAppend("PY OBJ DESTROYED")

I then tried this on the schools network and promptly ran into fire wall problems…. So I started trying to get around the fire wall problems by building a node.js server and communicating over http.

1. **var** http = require('http');
2. **var** url = require('url');
3. **var** md5 = require('md5');
5. **function** User() {
6. **const** ID = md5(Math.random());
8. **this**.getUserID = **function** () {
9. **return** ID;
10. }
11. }
13. **function** Player(User, num) {
14. **var** number = num;
15. **var** hand = [];
16. **var** ranksWon = [];
18. **this**.addToHand = **function**(card) {
19. hand.push(card);
20. };
22. **this**.removeFromHand = **function** (card) {
23. **var** pos = hand.indexOf(card);
24. hand.splice(pos, 1);
25. };
27. **this**.getHand = **function** () {
28. **return** hand;
29. };
31. **this**.getID = **function** () {
32. **return** User.getUserID();
33. }
34. }
36. **function** Game(users) {
37. **var** deck = [];
38. **var** Finished = **false**;
40. **this**.players = users;
42. **this**.dealToPlayer = **function** (player) {
43. **for** (x = 0; x < 8; x++) {
44. **this**.players[player].addToHand(deck.pop());
45. }
46. };
48. **this**.getDataForUser = **function** (userID) {
49. **for** (i = 0; i < **this**.players.length; i++) {
50. **if** (userID == **this**.players[i].getUserID()) {
51. **var** player = **this**.players[i];
52. **break**
53. }
54. }
56. **return** "HAND: " + player.getHand().toString();
57. }

60. **for** (i = 0; i < 52; i++) {
61. deck[i] = i;
62. }
64. **for** (i = 0; i < numOfPlayers+1; i++) {
65. **this**.players[i] = **new** Player(i);
66. **this**.dealToPlayer(i);
67. }
69. }
71. **function** Lobby() {
72. **this**.waitingUsers = [];
73. **this**.users = [];
74. **this**.games = [];
76. **this**.maybeNewGame = **function** () {
77. **if** (**this**.waitingUsers.length > 1) {
79. **var** g = **new** Game(**this**.waitingUsers);
80. **this**.games.push(g);
81. **this**.waitingUsers.empty();
83. **return** **this**.games.indexOf(g);
84. } **else** {
85. **return** **false**
86. }
87. };
89. **this**.maybeRemoveGame = **function** () {
90. **for** (i = 0; i < **this**.games.length; i++) {
91. **if** (**this**.games[i].isFinished()) {
92. **this**.games.splice(i, 1);
93. }
94. }
95. };
96. }
98. //=========---Main Program---=========
100. lobby = **new** Lobby();
102. http.createServer(**function** (req, res) {
103. **var** q = url.parse(req.url, **true**);
104. **var** qdata = q.query;
106. **var** userID = qdata.userID;
107. **var** gameNum =  qdata.gameNum;
108. **var** userIndex = **null**;
110. **for**(**var** i = 0; i < lobby.users.length; i++) {
111. **if** (lobby.users[i].getUserID() == userID) {
112. userIndex = i;
113. **break**;
114. }
115. }
117. **if** (userIndex == **null**) {
118. **var** u = **new** User();
119. lobby.users.push(u);
120. userID = u.getUserID();
121. res.write("USERID: " + userID.toString());
122. lobby.waitingUsers.push(u);
123. } **else** {
124. res.write("USERID: " + userID + ";");
125. **var** u = lobby.users[userIndex];
126. **if** (!lobby.waitingUsers.includes(u)) { //Contains is not a function
127. lobby.waitingUsers.push(u);
128. } **else** {
129. res.write(lobby.games[gameNum].getDataForUser(userID));
130. }
131. }
133. **var** potentialGame = lobby.maybeNewGame();
135. **if** (potentialGame != **false**) {
136. res.write("INIT NEW GAME;")
137. **var** game = potentialGame;
138. **for** (i = 0; i < game.players.length; i++) {
139. **if** (userID == game.players.users[i].getUserID()) {
140. **var** user = game.players.users[i];
141. **break**
142. }
143. }
144. **var** playerNumber = game.players.indexOf(user);
145. res.write("PlayerNumber " + playerNumber.toString());
146. }
148. res.end();
150. }).listen(8080);

The program above does everything that it is meant to do, it’s just it needs to do a lot and I focused on the details that didn’t really matter the most first. So after Easter I started work with just making the single player version of the game.

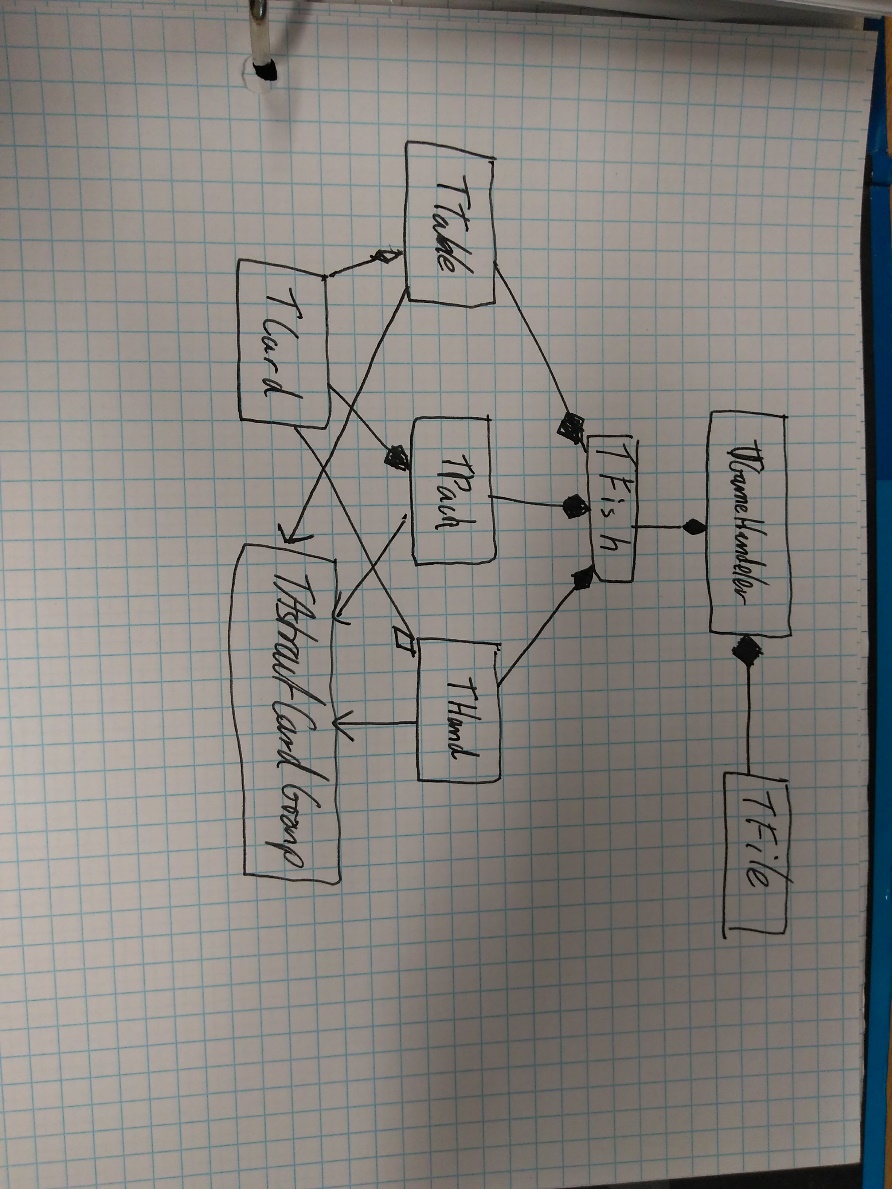
I didn’t have time for a class diagram but in essence, program creates a server object which creates a lobby object which handles an array of user objects which can be placed taken out of the lobby and placed into a game object, theory being that you could be able to have many game objects running at once.

I did successfully manage to get this set up on a raspberry pi at home with port forwarding.

# The Delphi Source.

The game has many classes, all custom writen, here is a summary:

|  |  |
| --- | --- |
| Class | Purpose |
| GameHandler | The idea was that this class could load, in different game objects and play those games, and save them for playing later. |
| Fish | This is game class with methods that are specific to GoFish, e.g. ask a user for a card of a specific rank. |
| Hand | This is the “players” it contains the cards they hold the ability to search through their hand etc. |
| Pack | Parent class of all of the cards. |
| Table | Where the players can place their books of cards |
| AbstractCardGroup | The abstract class that, hand, pack, table classes are based off |
| Cards | The objects |
| FileHandeler | Handles Files |



PCards:

1. **program** PCards;
3. {$APPTYPE CONSOLE}
4. {$R \*.res}
6. **uses**
7. System.SysUtils,
8. UCard **in** 'UCard.pas',
9. UPack **in** 'UPack.pas',
10. UHand **in** 'UHand.pas',
11. UExceptions **in** 'UExceptions.pas',
12. UFish **in** 'UFish.pas',
13. UAbstractCardGroup **in** 'UAbstractCardGroup.pas',
14. UGameHandeler **in** 'UGameHandeler.pas',
15. UFileHandeling **in** 'UFileHandeling.pas';
17. **var**
18. GameHandeler: TGameHandeler;
20. **begin**
21. randomize();
22. GameHandeler := TGameHandeler.create();
24. GameHandeler.play;
26. **writeln**('Press enter to quit');
27. readln;
29. GameHandeler.destroy;
30. exit;
31. **end**.

UCard:

1. **unit** UCard;
3. **interface**
5. **uses**
6. System.SysUtils;
8. **type**
9. Tcard = **class**
10. **private**
11. Rank, Suit: **integer**;
12. **public**
13. **constructor** create(r: **integer**; s: **integer**);
14. **constructor** createfromstr(r: **string**; s: **string**);
15. **function** GetRank: **integer**;
16. **function** GetExplicitRank: **string**;
17. **function** GetExplicitSuit: **string**;
18. **function** GetExplicitCard: **string**;
19. **function** GetSuit: **integer**;
20. **function** GetScore: **integer**;
22. **end**;
24. **implementation**
26. { Tcard }
27. **constructor** Tcard.create(r, s: **integer**);
28. **begin**
29. Suit := s;
30. Rank := r;
31. **end**;
33. **constructor** Tcard.createfromstr(r: **string**; s: **string**);
34. **begin**
35. **case** s[1] **of**
36. 'S':
37. Suit := 0;
38. 'H':
39. Suit := 1;
40. 'C':
41. Suit := 2;
42. 'D':
43. Suit := 3;
44. **else**
45. Suit := strtoint(s);
46. **end**;
48. **case** r[1] **of**
49. 'A':
50. Rank := 0;
51. 'J':
52. Rank := 10;
53. 'Q':
54. Rank := 11;
55. 'K':
56. Rank := 12;
57. **else**
58. Rank := strtoint(r);
59. **end**;
60. **end**;
62. **function** Tcard.GetExplicitCard: **string**;
63. **begin**
64. result := self.GetExplicitRank + ' of ' + self.GetExplicitSuit;
65. **end**;
67. **function** Tcard.GetExplicitRank: **string**;
68. **begin**
69. **case** self.Rank **of**
70. 0:
71. result := 'Ace';
72. 10:
73. result := 'Jack';
74. 11:
75. result := 'Queen';
76. 12:
77. result := 'King';
78. **else**
79. result := inttostr(self.Rank + 1);
80. **end**;
81. **end**;
83. **function** Tcard.GetExplicitSuit: **string**;
84. **begin**
85. **case** self.Suit **of**
86. 0:
87. result := 'Spades';
88. 1:
89. result := 'Hearts';
90. 2:
91. result := 'Clubs';
92. 3:
93. result := 'Diamonds';
94. **end**;
95. **end**;
97. **function** Tcard.GetRank: **integer**;
98. **begin**
99. result := self.Rank;
100. **end**;
102. **function** Tcard.GetScore: **integer**;
103. **begin**
104. result := (self.GetRank \* 4) + self.GetSuit;
105. **end**;
107. **function** Tcard.GetSuit: **integer**;
108. **begin**
109. result := self.Suit;
110. **end**;


114. **end**.

UAbstractCardGroup:

1. **unit** UAbstractCardGroup;
3. **interface**
4. **uses**
5. UCard;
7. **type**
8. TArrayOfString = Tarray<**string**>;
9. TCards = **array** **of** TCard;
11. TAbstractCardGroup = **class** abstract
12. **protected**
13. cards: TCards;
14. **public**
15. **function** GetSize: **integer**;
16. **function** findPos(rank, suit: **integer**): **integer**; Overload;
17. **function** findPos(card: tcard): **integer**; Overload;
18. **function** findCard(rank, suit: **integer**): tcard; Overload;
19. **function** findCard(card: tcard): tcard; Overload;
20. **procedure** addTo(card: tcard);
21. **end**;

24. **implementation**
26. { TAbstractCardGroup }
28. **procedure** TAbstractCardGroup.addTo(card: tcard);
29. **begin**
30. setlength(cards, length(cards) + 1);
31. cards[High(cards)] := card;
32. **end**;
34. **function** TAbstractCardGroup.findCard(card: tcard): tcard;
35. **var**
36. i: **integer**;
37. **begin**
38. **for** i := 0 **to** 51 **do**
39. **if** (cards[i].GetRank = card.GetRank) **and** (cards[i].GetSuit = card.GetSuit)
40. **then**
41. result := cards[i];
42. **end**;
44. **function** TAbstractCardGroup.findCard(rank, suit: **integer**): tcard;
45. **var**
46. i: **integer**;
47. **begin**
48. **for** i := 0 **to** 51 **do**
49. **if** (cards[i].GetRank = rank) **and** (cards[i].GetSuit = suit) **then**
50. result := cards[i];
51. **end**;
53. **function** TAbstractCardGroup.findPos(rank, suit: **integer**): **integer**;
54. **var**
55. i: **integer**;
56. **begin**
57. **for** i := 0 **to** 51 **do**
58. **if** (cards[i].GetRank = rank) **and** (cards[i].GetSuit = suit) **then**
59. result := i;
60. **end**;
62. **function** TAbstractCardGroup.findPos(card: tcard): **integer**;
63. **var**
64. i: **integer**;
65. **begin**
66. **for** i := 0 **to** length(cards) **do**
67. **if** (cards[i] = card) **then**
68. result := i;
69. **end**;
71. **function** TAbstractCardGroup.GetSize: **integer**;
72. **begin**
73. result := length(cards);
74. **end**;
76. **end**.

UPack:

1. **unit** UPack;
3. **interface**
5. **uses**
6. System.SysUtils,
7. UCard, UAbstractCardGroup, UExceptions, System.Generics.Collections,
8. System.StrUtils;
10. **type**
11. TPack = **class**(TAbstractCardGroup)
12. **private**
13. pack\_top: **integer**;
14. **public**
15. Debug: Boolean;
16. **constructor** create;
17. **function** draw: Tcard;
18. **procedure** shuffle;
20. **destructor** destroy;
21. **end**;
23. **implementation**
25. { Tpack }
27. **constructor** TPack.create();
28. **var**
29. I: **integer**;
30. **begin**
31. setlength(cards, 52);
33. **for** I := 0 **to** 51 **do**
34. cards[I] := Tcard.create(I **mod** 13, I **mod** 4);
36. pack\_top := 0;
37. **end**;
39. **destructor** TPack.destroy;
40. **var**
41. I: **integer**;
42. **begin**
43. **for** I := 0 **to** 51 **do**
44. **begin**
45. cards[I].Free;
46. **end**;
47. **end**;
49. **function** TPack.draw: Tcard; // This should be done using a stack
50. **begin**
51. result := cards[pack\_top];
52. **if** pack\_top < 52 **then**
53. **begin**
54. inc(pack\_top);
55. **end**
56. **else**
57. **begin**
58. EOutofCards;
59. **end**;
60. **end**;
62. **procedure** TPack.shuffle;
63. **var**
64. temp: Tcard;
65. random\_pos: **integer**;
66. I: **integer**;
67. **begin**
68. **for** I := 0 **to** 52 **do**
69. **begin**
70. random\_pos := I + random(52 - I); // like random range
71. temp := cards[random\_pos];
72. cards[random\_pos] := cards[I];
73. cards[I] := temp;
74. **end**;
75. pack\_top := 0;
76. **end**;
78. **end**.

UHand:

1. **unit** UHand;
3. **interface**
5. **uses**
6. UPack, UAbstractCardGroup, UCard, Generics.Collections;
8. **type**
9. Thand = **class** (TAbstractCardGroup)
10. **private**
11. **procedure** removeCard(card: tcard);
12. **function** GetHandSize: **integer**;
13. **public**
14. **constructor** Create;
15. **function** getcontents: TCards;
16. **function** placecard(index: **integer**): tcard; Overload;
17. **function** placecard(card: tcard): tcard; Overload;
18. **function** findCardByRank(rank: **integer**): tcard;
19. **function** howManyOfRank(rank: **integer**): **integer**;
20. **destructor** destroy;
22. **end**;
24. **implementation**
26. { Thand }
28. **function** Thand.howManyOfRank(rank: **integer**): **integer**;
29. **var**
30. i: **integer**;
31. **begin**
32. result := 0;
33. **for** I := 0 **to** length(cards)-1 **do** **begin**
34. **if** cards[i].GetRank = rank **then**
35. inc(result);
36. **end**;
37. **end**;
39. **constructor** Thand.Create;
40. **var**
41. i: **integer**;
42. **begin**
43. setlength(cards, 0);
44. **end**;
46. **destructor** Thand.destroy;
47. **begin**
49. **end**;

52. **function** Thand.findCardByRank(rank: **integer**): tcard;
53. **var**
54. i: **integer**;
55. **begin**
56. **for** I := 0 **to** length(cards)-1 **do** **begin**
57. **if** cards[i].GetRank = rank **then** **begin**
58. result := cards[i];
59. break;
60. **end**;
61. **end**;
62. **end**;

65. **function** Thand.getcontents: TCards;
66. **begin**
67. result := cards;
68. **end**;
70. **function** Thand.GetHandSize: **integer**;
71. **begin**
72. result := length(cards);
73. **end**;

76. **function** Thand.placecard(card: tcard): tcard;
77. **begin**
78. result := card;
79. self.removeCard(card);
80. **end**;
82. **procedure** Thand.removeCard(card: tcard);
83. **var**
84. i: **integer**;
85. **begin**
86. delete(cards, findPos(card), 1);
88. **end**;
90. **function** Thand.placecard(index: **integer**): tcard;
91. **begin**
92. result := cards[index];
93. delete(cards, index, 1);
94. **end**;
96. **end**.

UFish:

1. **unit** UFish;
3. **interface**
5. **uses**
6. UHand, UPack, UCard, UAbstractCardGroup, UExceptions;
8. **type**
10. TArrayOfHand = **array** **of** THand;
11. TArrayOfInterger = **array** **of** **integer**;
13. TTable = **class**(TAbstractCardGroup)
14. **private**
15. **constructor** create;
16. **end**;
18. TFish = **class**
19. **private**
20. Hands: TArrayOfHand;
21. Table: TTable;
22. Scores: TArrayOfInterger;
23. deck: TPack;
24. **public**
25. **constructor** create(numberOfPlayers: **integer**);
26. **function** checkForBook(playerNum: **integer**): **boolean**;
27. **function** AskForCard(rank: **integer**; Handfrom: THand; Handto: THand): TCards;
28. **function** GoFish(hand: THand): TCard;
30. **destructor** destroy;
32. **property** players: TArrayOfHand read Hands **write** Hands;
33. **property** pack: TPack read deck **write** deck;
34. **property** books: TArrayOfInterger read scores **write** scores;
36. **end**;
38. **implementation**
40. { Fish }
42. **function** TFish.AskForCard(rank: **integer**; Handfrom: THand; Handto: THand)
43. : TCards;
44. **var**
45. HandFromContents: TCards;
46. GotCardFromPlayer: **boolean**;
47. I, r: **integer**;
48. hf: TCard;
49. test: TCard;
50. **begin**
51. GotCardFromPlayer := **false**;
52. setlength(result, 0);
53. HandFromContents := Handfrom.getcontents;
54. i := 0;
55. **while** i < length(HandFromContents) **do**
56. **begin**
57. **if** (HandFromContents[I].getRank = rank) **then**
58. **begin**
59. setlength(result, length(result) + 1);
60. result[length(result) - 1] := HandFromContents[I];
61. test := Handfrom.placecard(I);
62. Handto.AddTo(test);
63. GotCardFromPlayer := **true**;
64. **end**;
65. inc(i);
66. **end**;
67. **end**;
69. **function** TFish.checkForBook(playerNum: **integer**): **boolean**;
70. **var**
71. I, j: **integer**;
72. hand: THand;
73. **begin**
74. **for** I := 0 **to** 12 **do**
75. **begin**
76. **if** (Hands[playerNum].howManyOfRank(I) = 4) **then**
77. **begin**
78. hand := Hands[playerNum];
79. inc(Scores[playerNum]);
80. **for** j := 0 **to** 3 **do**
81. Table.AddTo(hand.placecard(hand.findCardByRank(I)));
82. **end**;
83. **end**;
84. **end**;
86. **constructor** TFish.create(numberOfPlayers: **integer**);
87. **var**
88. I, x: **integer**;
89. **begin**
90. **if** numberOfPlayers > 4 **then**
91. **begin**
92. ETooManyPlayers;
93. **end**
94. **else**
95. **begin**
97. Table := TTable.create;
99. deck := TPack.create();
100. deck.shuffle;
102. setlength(Hands, numberOfPlayers);
103. setlength(Scores, numberOfPlayers);
105. **for** I := 0 **to** numberOfPlayers **do**
106. **begin**
107. scores[i] := 0;
108. Hands[I] := THand.create;
109. **for** x := 0 **to** 7 **do**
110. Hands[I].AddTo(deck.draw);
111. **end**;
112. **end**;
114. **end**;
116. **destructor** TFish.destroy;
117. **begin**
119. **end**;
121. **function** TFish.GoFish(hand: THand): TCard;
122. **var**
123. Card: TCard;
124. **begin**
125. Card := deck.draw;
126. hand.AddTo(Card);
127. result := Card;
128. **end**;
130. { TTable }
132. **constructor** TTable.create;
133. **begin**
134. setlength(cards, 0);
135. **end**;
137. **end**.
138. **unit** UGameHandeler;
140. **interface**
142. **uses**
143. System.SysUtils, classes, UCard, UFileHandeling, UFish,
144. System.Generics.Collections, UAbstractCardGroup, math;
146. **type**
147. TArrayOfString = Tarray<**string**>;
148. TCards = **array** **of** TCard;
150. TGameHandeler = **class**
151. **private**
152. Delimiter: **string**;
154. commands: TDictionary<**string**, **integer**>;
155. RankNames: TDictionary<**string**, **integer**>;
156. SuitNames: TDictionary<**string**, **integer**>;
158. game: TFish;
159. currentUser: **integer**;
161. **public**
162. **constructor** create;
163. **function** GetUserIn: TArrayOfString;
164. **procedure** man;
165. **procedure** ExecuteUserInstruction(cmd: TArrayOfString);
166. **procedure** Welcome;
167. **procedure** play;
168. **function** interpretCard(input: **string**): TCard;
169. **function** interpretRank(input: **string**): **integer**;
170. **procedure** displayBooks;
171. Destructor destroy;
172. **end**;
174. **const**
175. manFilePath = 'man.txt';
177. **implementation**
179. { TGameHandeler }
181. **function** TGameHandeler.GetUserIn: TArrayOfString;
182. **var**
183. userin: **string**;
184. **begin**
185. **while** length(userin) < 3 **do**
186. **begin**
187. **write**('~: ');
188. readln(userin);
189. **end**;
191. result := userin.Split([Delimiter]);
193. **end**;
195. **function** TGameHandeler.interpretCard(input: **string**): TCard;
196. **var**
197. SplitInput: Tarray<**string**>;
198. rank, suit: **integer**;
200. **begin**
201. SplitInput := input.Split([' ']);
202. rank := RankNames.Items[SplitInput[0]];
203. suit := SuitNames.Items[SplitInput[2]];
204. result := game.pack.findCard(rank, suit);
205. **end**;
207. **function** TGameHandeler.interpretRank(input: **string**): **integer**;
208. **begin**
209. result := RankNames.Items[input];
210. **end**;
212. **constructor** TGameHandeler.create;
213. **var**
214. playerNumber: **integer**;
215. **begin**
216. Delimiter := '; ';
218. RankNames := TDictionary<**string**, **integer**>.create();
220. RankNames.add('Ace', 0);
221. RankNames.add('ace', 0);
222. RankNames.add('Two', 1);
223. RankNames.add('two', 1);
224. RankNames.add('2', 1);
225. RankNames.add('Three', 2);
226. RankNames.add('three', 2);
227. RankNames.add('3', 2);
228. RankNames.add('Four', 3);
229. RankNames.add('four', 3);
230. RankNames.add('4', 3);
231. RankNames.add('Five', 4);
232. RankNames.add('five', 4);
233. RankNames.add('5', 4);
234. RankNames.add('Six', 5);
235. RankNames.add('six', 5);
236. RankNames.add('6', 5);
237. RankNames.add('Seven', 6);
238. RankNames.add('seven', 6);
239. RankNames.add('7', 6);
240. RankNames.add('Eight', 7);
241. RankNames.add('eight', 7);
242. RankNames.add('8', 7);
243. RankNames.add('Nine', 8);
244. RankNames.add('nine', 8);
245. RankNames.add('9', 8);
246. RankNames.add('Ten', 9);
247. RankNames.add('ten', 9);
248. RankNames.add('10', 9);
249. RankNames.add('Jack', 10);
250. RankNames.add('jack', 10);
251. RankNames.add('Queen', 11);
252. RankNames.add('queen', 11);
253. RankNames.add('King', 12);
254. RankNames.add('king', 12);
256. SuitNames := TDictionary<**string**, **integer**>.create();
258. SuitNames.add('Spades', 0);
259. SuitNames.add('spades', 0);
260. SuitNames.add('Hearts', 1);
261. SuitNames.add('hearts', 1);
262. SuitNames.add('Clubs', 2);
263. SuitNames.add('clubs', 2);
264. SuitNames.add('Diamonds', 3);
265. SuitNames.add('diamonds', 3);
267. commands := TDictionary<**string**, **integer**>.create();
269. commands.add('m', 0);
270. commands.add('man', 0);
271. commands.add('s', 1);
272. commands.add('save', 1);
273. commands.add('a', 2);
274. commands.add('ask', 2);
275. commands.add('A', 2);
276. commands.add('Ask', 2);
277. commands.add('q', 3);
278. commands.add('quit', 3);
280. Welcome;
282. **repeat**
283. **write**('How many players: ');
284. readln(playerNumber);
285. playerNumber := playerNumber - 1 // -= !!!!!
286. **until** (playerNumber > 0) **and** (playerNumber < 5);
288. game := TFish.create(playerNumber);
290. **end**;
292. **destructor** TGameHandeler.destroy;
293. **begin**
294. RankNames.destroy;
295. SuitNames.destroy;
296. commands.destroy;
297. game.destroy;
298. **end**;
300. **procedure** TGameHandeler.ExecuteUserInstruction(cmd: TArrayOfString);
301. **var**
302. cards: UAbstractCardGroup.TCards;
303. i: **integer**;
304. **begin**
305. **case** commands.Items[cmd[0]] **of**
306. 0:
307. **begin**
308. man;
309. ExecuteUserInstruction(GetUserIn);
310. **end**;
311. // 1:
312. // Add code to save game
314. 2:
315. **begin**
316. cards := game.AskForCard(interpretRank(cmd[1]),
317. game.players[strToint(cmd[2])], game.players[currentUser]);
318. **if** length(cards) = 0 **then**
319. **begin**
320. **writeln**('Player does not have card');
321. **writeln**('GO FISH');
322. **writeln**('Got card: ', game.GoFish(game.players[currentUser])
323. .GetExplicitCard);
325. **writeln**('');
326. **writeln**('---');
327. **writeln**('The following books have been won:');
328. displayBooks;
329. **end**
330. **else**
331. **begin**
332. **writeln**('Got cards from user');
333. **for** i := 0 **to** length(cards) - 1 **do**
334. **writeln**(cards[i].GetExplicitCard);
336. **writeln**('');
337. **writeln**('---');
338. **writeln**('The following books have been won:');
339. displayBooks;
340. **end**;
341. **end**;
342. 3:
343. game.destroy;
344. **end**;
346. **end**;
348. **procedure** TGameHandeler.displayBooks;
349. **var**
350. i: **integer**;
351. **begin**
352. **writeln**('The books won are as follows');
353. **for** i := 0 **to** length(game.players) **do**
354. **begin**
355. **writeln**('Score for player ', inttoStr(i), ': ', inttoStr(game.books[i]));
356. **end**;
357. **end**;
359. **procedure** TGameHandeler.man;
360. **var**
361. manFile: TFile;
362. mancontents: TArrayOfString;
364. **begin**
365. manFile := TFile.create(manFilePath);
366. manFile.printfile;
367. manFile.destroy;
368. **end**;
370. **procedure** TGameHandeler.play;
371. **var**
372. i, x: **integer**;
373. cardsInHand: UAbstractCardGroup.TCards;
374. totalBooksWon: **integer**;
376. **begin**
378. totalBooksWon := 0;
380. **while** 12 > totalBooksWon **do**
381. **begin**
383. **for** i := 0 **to** length(game.players) **do**
384. **begin**
385. currentuser := i;
387. **writeln**('----------------------------');
388. **writeln**('Player ', inttoStr(i), ' turn!');
390. **writeln**('Your hand contains: ');
392. cardsInHand := game.players[i].getcontents;
394. **for** x := 0 **to** length(cardsInHand) - 1 **do**
395. **writeln**(cardsInHand[x].GetExplicitCard);
397. **writeln**('');
398. ExecuteUserInstruction(GetUserIn);
400. **if** game.checkForBook(currentUser) **then**
401. **begin**
402. **writeln**('YOU GOT A BOOK!');
403. **writeln**('The scores are as follows');
404. displayBooks;
405. **end**;
407. **writeln**('Waiting for enter for next player');
408. readln;
410. **end**;
412. totalBooksWon := 0;
413. **for** i := 0 **to** length(game.books) **do** **begin**
414. totalBooksWon := totalBooksWon + 1
415. **end**;
416. **end**;
418. **writeln**('====================================');
419. **writeln**('-------------GAME OVER--------------');
420. **writeln**('====================================');
422. **writeln**('SCORES:');
423. displayBooks;
424. **end**;
426. **procedure** TGameHandeler.Welcome;
427. **const**
428. welcomeFilePath = 'ASCIIFISH.txt';
429. **var**
430. welcomeFile: TFile;
431. **begin**
432. welcomeFile := TFile.create(welcomeFilePath);
433. welcomeFile.printfile;
434. welcomeFile.destroy;
435. **end**;
437. **end**.
438. **unit** UFileHandeling;
440. **interface**
442. **uses**
443. sysutils;
445. **type**
446. TArrayOfString = Tarray<**string**>;
448. TFile = **class**
449. **public**
450. **constructor** Create(filename: **string**);
451. **function** getfilelength: **integer**;
452. **function** readfile: TArrayOfString;
453. **procedure** writefile(data: **string**);
454. **procedure** appendfile(data: **string**);
455. **procedure** printfile;
456. Destructor Destroy;
457. **private**
458. filename: **string**;
459. **end**;
461. **implementation**
463. { TFile }
465. **procedure** TFile.appendfile(data: **string**);
466. **var**
467. myfile: textfile;
468. **begin**
469. assignfile(myfile, filename);
470. append(myfile);
471. **writeln**(myfile, data);
472. closefile(myfile);
473. **end**;
475. **constructor** TFile.Create(filename: **string**);
476. **begin**
477. self.filename := filename;
478. **end**;
480. **destructor** TFile.Destroy;
481. **begin**
483. **end**;
485. **function** TFile.getfilelength(): **integer**;
486. **var**
487. myfile: textfile;
488. count: **integer**;
490. **begin**
491. assignfile(myfile, self.filename);
492. reset(myfile);
493. **while** **not** eof(myfile) **do**
494. **begin**
495. readln(myfile, count);
496. count := count + 1;
497. **end**;
498. result := count;
499. **end**;

502. **procedure** TFile.printfile();
503. **var**
504. filecontents: TArrayOfString;
505. i: **integer**;
506. **begin**
507. filecontents := self.readfile();
508. **for** i := 0 **to** length(filecontents) - 1 **do**
509. **begin**
510. **writeln**(filecontents[i])
511. **end**;
512. **end**;
514. **function** TFile.readfile(): TArrayOfString;
515. **var**
516. myfile: textfile;
517. line, contents: **string**;
518. count: **integer**;
519. i: **integer**;
521. **begin**
522. count := 0;
523. assignfile(myfile, self.filename);
524. reset(myfile);
525. **while** **not** eof(myfile) **do**
526. **begin**
527. readln(myfile, line);
528. count := count + 1;
529. **end**;
531. setlength(result, count);
533. reset(myfile);
534. **for** i := 0 **to** count - 1 **do**
535. **begin**
536. readln(myfile, line);
537. result[i] := line;
538. **end**;
540. closefile(myfile);
541. **end**;
543. **procedure** TFile.writefile(data: **string**);
544. **var**
545. myfile: textfile;
546. **begin**
547. assignfile(myfile, self.filename);
548. rewrite(myfile);
549. **write**(myfile, data);
550. closefile(myfile);
551. **end**;
553. **end**.

UExceptions:

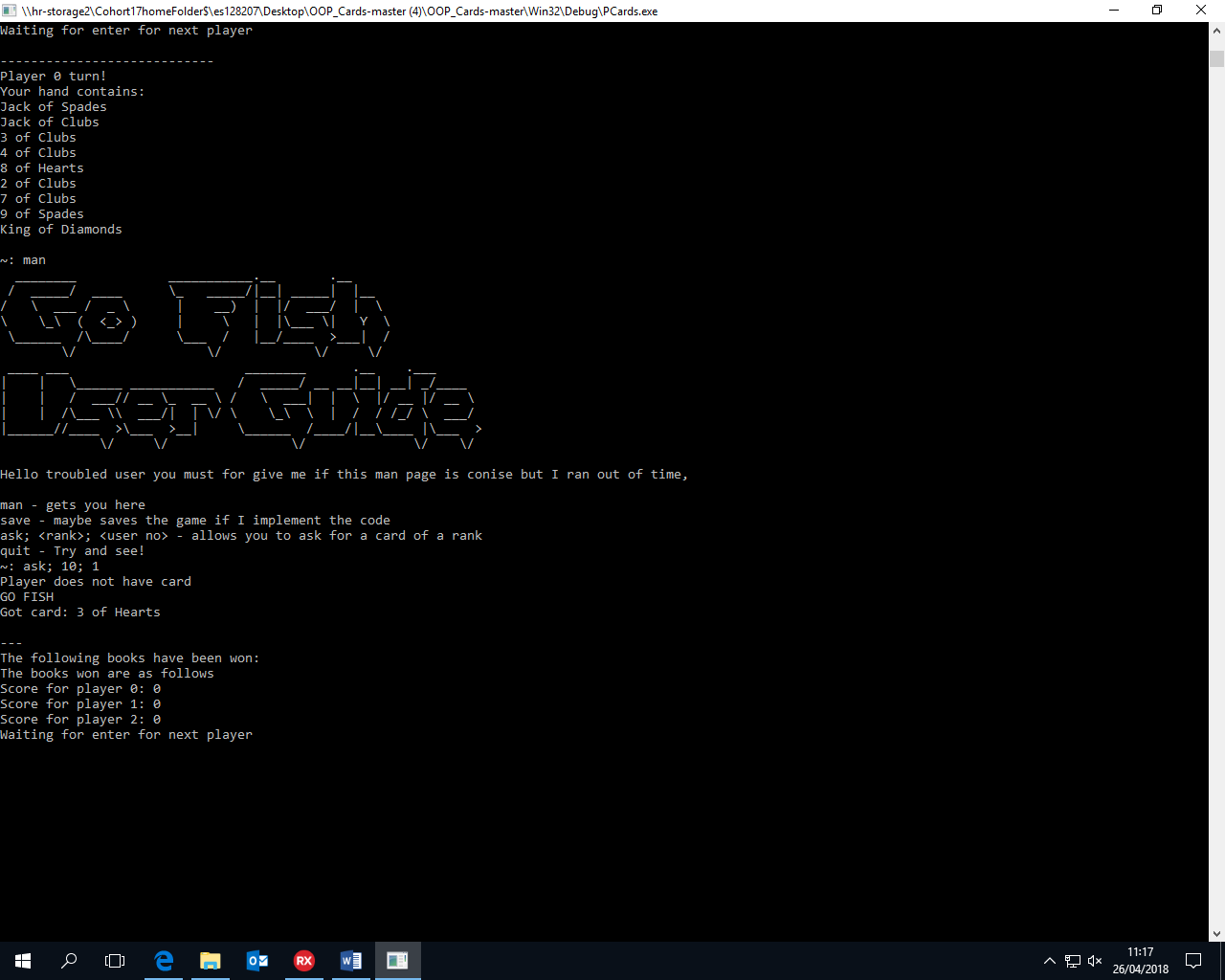
1. **unit** UExceptions;
3. **interface**
5. **uses**
6. System.SysUtils;
8. **procedure** EOutOfCards;
9. **procedure** ETooManyPlayers;
11. **implementation**
13. **procedure** EOutOfCards;
14. **begin**
15. **writeln**('Insufficent cards, can not draw requested card');
16. **raise** Exception.Create('Out of cards');
17. **end**;
19. **procedure** ETooManyPlayers;
20. **begin**
21. **writeln**('Too many players');
22. **raise** Exception.Create('Too many players');
23. **end**;
25. **end**.

# Screen shots

Here is an example of my code running, player 0 asking for kings from player 1 and getting the king of Diamonds added to their hand.

It then displays the scores at the bottom.

f

Here is another user, confused about how to use the program. They got the man page and they asked for a card the users didn’t have and so were told GO FISH!

# LESSONS LEARNT

I was too ambitious and prioritized the wrong stuff, eg building the network handlers before I built the actual game. This is a lesson that I will have to bear in mind when doing course work.

This resulted in insufficient time for testing and a program with probably many unknown bugs.