# Classes for card games (card, deck, and various hands)

using System;

using System.Collections.Generic;

using System.Text;

namespace CardGames

{

public class Card

{

private int rank;

private int suit;

public Card(int r, int s)

{

rank = r;

suit = s;

}

public int getRank()

{

return rank;

}

public int getSuit()

{

return suit;

}

public int getScore()

{

return (rank \* 4) + suit;

}

public string getRankText()

{

string rankText = "";

switch (rank)

{

case 0:

rankText = "Ace";

break;

case 1:

rankText = "Two";

break;

case 2:

rankText = "Three";

break;

case 3:

rankText = "Four";

break;

case 4:

rankText = "Five";

break;

case 5:

rankText = "Six";

break;

case 6:

rankText = "Seven";

break;

case 7:

rankText = "Eight";

break;

case 8:

rankText = "Nine";

break;

case 9:

rankText = "Ten";

break;

case 10:

rankText = "Jack";

break;

case 11:

rankText = "Queen";

break;

case 12:

rankText = "King";

break;

}

return rankText;

}

public char getRankSymbol()//this is for the ASCII part of the code below

{

string rankSymbol = "";

switch (rank)

{

case 0:

rankSymbol = "A";

break;

case 1:

rankSymbol = "2";

break;

case 2:

rankSymbol = "3";

break;

case 3:

rankSymbol = "4";

break;

case 4:

rankSymbol = "5";

break;

case 5:

rankSymbol = "6";

break;

case 6:

rankSymbol = "7";

break;

case 7:

rankSymbol = "8";

break;

case 8:

rankSymbol = "9";

break;

case 9:

rankSymbol = "T";

break;

case 10:

rankSymbol = "J";

break;

case 11:

rankSymbol = "Q";

break;

case 12:

rankSymbol = "K";

break;

}

return char.Parse(rankSymbol);

}

public string getSuitText()

{

string suitText = "";

switch (suit)

{

case 0:

suitText = "Clubs";

break;

case 1:

suitText = "Diamonds";

break;

case 2:

suitText = "Hearts";

break;

case 3:

suitText = "Spades";

break;

}

return suitText;

}

public string getCardText()

{

return getRankText() + " of " + getSuitText();

}

private char getSuitSymbol()//again, for the ASCII cards

{

Console.OutputEncoding = System.Text.Encoding.UTF8;

char suitSymbol = ' ';

switch (suit)

{

case 0:

suitSymbol = '♣';

break;

case 1:

suitSymbol = '♦';

break;

case 2:

suitSymbol = '♥';

break;

case 3:

suitSymbol = '♠';

break;

}

return suitSymbol;

}

public string[] generateCardASCII()//generates the card in ASCII with the rank and suit

{

string[] lines = new string[9];

lines[0] = ("┌─────────┐");

lines[1] = String.Format("│{0}{1} │", getRankSymbol(), getSuitSymbol()); // use two {} one for char, one for space or char

lines[2] = ("│ │");

lines[3] = ("│ │");

lines[4] = String.Format("│ {0} │", getSuitSymbol());

lines[5] = ("│ │");

lines[6] = ("│ │");

lines[7] = String.Format("│ {0}{1}│", getSuitSymbol(), getRankSymbol());

lines[8] = ("└─────────┘");

return lines;

}

}

public class Deck

{

public Card[] deckOfCards = new Card[52];

private int front;

public int Top //property to provide read-only access to front pointer

{

get

{

return front;

}

set

{

front = value;

}

}

private int rear;

public int Bottom //property to provide read-only access to rear pointer

{

get

{

return rear;

}

set

{

rear = value;

}

}

private int size;

// size is number of cards currently in pack

public int Size //property to provide read-only access to size field

{

get

{

return size;

}

set

{

size = value;

}

}

public Deck()

{

int sN; //the suit number for pack of card generation

int rN; //the rank number for pakc of card generation

for (rN = 0; rN < 13; rN++)

{

for (sN = 0; sN < 4; sN++)

{

//Card cardTemp = new Card(rN, sN);

//deckOfCards[cardTemp.getScore()] = cardTemp;

deckOfCards[rN \* 4 + sN] = new Card(rN, sN);

}

}

front = 0;

rear = 51;

size = 52;

}

public Deck(int i)

{

}//thi is a constructor for an empty deck

public Card getCard(int n)

{

return deckOfCards[n];//returns the card at that index

}

private void changeCard(int n, Card card)

{

deckOfCards[n] = card;

}

public void shuffleDeck()

{

Random r = new Random();//create new random

for (int n = deckOfCards.Length - 1; n > 0; --n)//loop through each card once

{

int k = r.Next(n + 1);

Card temp = deckOfCards[n]; //generate number and swap the current card with the card at that position.

deckOfCards[n] = deckOfCards[k];

deckOfCards[k] = temp;

}

}

public void displayDeck()

{

for (int i = 0; i < 52; i++)//itterate through every card

{

Console.WriteLine(deckOfCards[i].getCardText());//display

}

}

public void fillDeck()

{

int sN; //the suit number for pack of card generation

int rN; //the rank number for pakc of card generation

for (rN = 0; rN < 13; rN++)

{

for (sN = 0; sN < 4; sN++)

{

//Card cardTemp = new Card(rN, sN);

//deckOfCards[cardTemp.getScore()] = cardTemp;

deckOfCards[rN \* 4 + sN] = new Card(rN, sN);

}

}

}//own thing

public void dealDeck2Players( Deck hand1, Deck hand2)

{

for (int i = 0; i <= 25; i++)

{

hand1.changeCard(i, deckOfCards[2 \* i]);

hand2.changeCard(i, deckOfCards[2 \* i + 1]);

}

}//own thing

public bool IsEmpty()//check if deck is empty

{

return size == 0;

}

public bool IsFull()//check if deck is full

{

return size == 52;

}

public Card DealCard()//deals card from the deck

{

if (!IsEmpty())

{

Card ACard = deckOfCards[Top];

if (Top == 51)

Top = 0;

else

Top++;

Size--;

return ACard;

}

else

return null;

}

public void AddCard(Card ACard)//adds card back to the deck

{

if (Bottom == 51)

Bottom = 0;

else

Bottom++;

Size++;

deckOfCards[Bottom] = ACard;

}

public Card Card

{

get => default;

set

{

}

}

}

public class Hand

{

// Hand is a collection of previously-created cards, typically contained

// in Pack.Cards

// Hand does not create any Cards itself

protected List<Card> cards = new List<Card>();

public Card this[int i]

{

get { return cards[i]; }

// this provides read-only access to the List by index

}

protected int GetSize()

{

return cards.Count;

}

public int Size

{

get

{

return GetSize();

}

}

public void AddCard(Card card)//adds cards to the hand

{

cards.Add(card);

}

public int FindCard(int r, int s)

// find the position of a specified card in the hand

// returns -1 if not found

// useful in rummy-type games

{

int result = -1;

for (int i = 0; i <= Size; i++)

{

if ((cards[i].getRank() == r) && (cards[i].getSuit() == s))

{

result = i;

}

}

return result;

//returns -1 if not present

}

public Card First()

{

return cards[0];

}

public bool IsEmpty()

{

return Size == 0;

}

public Card Last()

{

return cards[Size - 1];

}

public Card RemoveCard(int i)

// remove card from the hand, by index

{

if (Size > i)

{

Card result = cards[i];

cards.RemoveAt(i);

return result;

}

else

{

return null;

}

}

public Card RemoveFirstCard()

{

if (!IsEmpty())

{

Card c = cards[0];

cards.RemoveAt(0);

return c;

}

else

{

return null;

}

}

public void Clear()

{

cards.Clear();

}

public void display1CardASCII()//uses the ASCII card from card class and dsiplays one card on the screen

{

Console.OutputEncoding = System.Text.Encoding.UTF8;

string[] cardDisp = cards[0].generateCardASCII();

for (int i = 0; i < 9; i++)

{

Console.WriteLine(cardDisp[i]);

}

}

public void display2CardASCII(Hand hand)//same as 1 card but 2 in line with eachother

{

Console.OutputEncoding = System.Text.Encoding.UTF8;

string[] card1Disp = cards[0].generateCardASCII();

string[] card2Disp = cards[1].generateCardASCII();

for (int i = 0; i < 9; i++)

{

Console.WriteLine(card1Disp[i] + " " + card2Disp[i]);

}

}

public void display3CardASCII(Hand hand)//same as 1 card but 3 in line with eachother

{

Console.OutputEncoding = System.Text.Encoding.UTF8;

string[] card1Disp = cards[0].generateCardASCII();

string[] card2Disp = cards[1].generateCardASCII();

string[] card3Disp = cards[2].generateCardASCII();

for (int i = 0; i < 9; i++)

{

Console.WriteLine(card1Disp[i] + " " + card2Disp[i] + " " + card3Disp[i]);

}

}

public Card Card

{

get => default;

set

{

}

}

}

public abstract class ScoringHand : Hand//class of a scoring hand

{

public abstract int getScore();

}

public class BaccaratHand : ScoringHand

{

public override int getScore()//this is for scoring a baccarat hand

{

int result = 0;

foreach (Card card in cards)

{

if (card.getRank() < 10)

{

result += card.getRank() + 1;

}

}

return result % 10;//result is the unit of the addition of A through 9

}

}

}

# File containing the Baccarat game classes including the UI class

using System;

using CardGames;

namespace Baccarat

{

public class Baccarat

{

private Deck deck;

private BaccaratHand \_PHand;

private BaccaratHand \_BHand;

private int getPHandScore()//get the score of the players hand

{

return \_PHand.getScore();

}

public int PScore

{

get { return getPHandScore(); }

}

private int getBHandScore()//get the score of the bankers hand

{

return \_BHand.getScore();

}

public int BScore

{

get { return getBHandScore(); }

}

//get the players and bankers hand

public BaccaratHand PHand

{

get { return \_PHand; }

}

public BaccaratHand BHand

{

get { return \_BHand; }

}

//constructor for the class, which sets up the hands from the deck

public Baccarat()

{

deck = new Deck();

\_PHand = new BaccaratHand();

\_BHand = new BaccaratHand();

}

//check for a natural win

public bool NaturalWin()

{

return ((PScore >= 8) || (BScore >= 8));

}

//deal cards to the players

public void DealGame()

{

deck.shuffleDeck();

PHand.AddCard(deck.DealCard());

PHand.AddCard(deck.DealCard());

BHand.AddCard(deck.DealCard());

BHand.AddCard(deck.DealCard());

}

//gets both players to draw cards if no player has a natural and within the rules

public void DrawCard()

{

if ((PScore >= 0) && (PScore <= 5))

{

PHand.AddCard(deck.DealCard());

int PDraw = PHand[2].getRank();

if (PDraw == 8)

{

PDraw = -2;

}

if (PDraw == 9)

{

PDraw = -1;

}

if ((PDraw >= 10) && (PDraw <= 13))

{

PDraw = 0;

}

if (BScore <= (PDraw / 2) + 3)

{

BHand.AddCard(deck.DealCard());

}

}

else

{

if ((BScore >= 0) && (BScore <= 5))

{

BHand.AddCard(deck.DealCard());

}

}

}

//deals and draws cards

public void Play()

{

DealGame();

DrawCard();

}

//set up the betting function including channging money, bets and who you are betting on

private int money = 100;//amount of money

public int Money

{

get { return money; }

set { money = value; }

}

private int bet = 0;//the bet amount

public int Bet

{

get { return bet; }

set { bet = value; }

}

private char betLocation = ' ';//where you are placing your bet

public char BetLocation

{

get { return betLocation; }

set { betLocation = value; }

}

//function for a legal bet changing the necesarry parameters to reflect this

public void MakeBet()

{

bool sucessful;

do

{

Console.Write("Current money is {0} \n" +

"Enter the bet amount --> ", Money);

int tempBet = int.Parse(Console.ReadLine());

Console.Write("Current bet is {0} \n" +

"Enter the bet position (B, P, T)--> ", Bet);

char tempPos = char.Parse(Console.ReadLine());

sucessful = (Bet < Money);

if (sucessful)

{

Bet = tempBet;

Money -= tempBet;

BetLocation = tempPos;

}

else

{

Console.WriteLine("You don't have enough money, please enter a lower amount");

}

} while (!sucessful || !(BetLocation == 'B' || BetLocation == 'P' || BetLocation == 'T'));

}

//distributes the money from winning the bet

public void ResolveTurn(bool win)

{

double winnings = 0;

if (win)

{

if (BetLocation == 'B')

{

winnings = 0.95 \* Bet;

}

else if(BetLocation == 'P')

{

winnings = 1.5 \* Bet;

}

else if(BetLocation == 'T')

{

winnings = 8 \* Bet;

}

Money += (int)Math.Floor(winnings);

Bet = 0;

}

}

}

public class BaccaratGameDisplaying

{

//class for the UI of the game

public void DisplayInLine(Baccarat game)

{

//displays players and bankers hand

Console.WriteLine("Player's cards are: ");

if (game.PHand.Size == 2)

{

game.PHand.display2CardASCII(game.PHand);

}

else

{

game.PHand.display3CardASCII(game.PHand);

}

Console.WriteLine("Banker's cards: ");

if (game.BHand.Size == 2)

{

game.BHand.display2CardASCII(game.PHand);

}

else

{

game.BHand.display3CardASCII(game.BHand);

}

Console.WriteLine("Player's score: " + game.PScore);

Console.WriteLine("Banker's score: " + game.BScore + "\n");

}

public void DisplayDraw(Baccarat game)

// displays outcome of draw phase of game

{

if (game.PHand.Size == 2)

{

Console.WriteLine("The player has stood");

Console.ReadLine();

}

if (game.PHand.Size == 3)

{

Console.WriteLine("The player has drawn the " + game.PHand[2].getCardText());

Console.ReadLine();

}

if (game.BHand.Size == 2)

{

Console.WriteLine("The banker has stood");

Console.ReadLine();

}

if (game.BHand.Size == 3)

{

Console.WriteLine("The banker has drawn the " + game.BHand[2].getCardText());

Console.ReadLine();

}

Console.WriteLine();

}

public char FinalResult(Baccarat game)

// displays outcome of game

{

char winChar = ' ';

if (game.PScore > game.BScore)

{

Console.WriteLine("The Player has won");

winChar = 'P';

}

if (game.BScore > game.PScore)

{

Console.WriteLine("The Banker has won");

winChar = 'B';

}

if (game.PScore == game.BScore)

{

Console.WriteLine("It's a tie");

winChar = 'T';

}

return winChar;

}

}

}

# File for the main program

using System;

using CardGames;

using Baccarat;

namespace Baccarat

{

class Program

{

static void Main(string[] args)

{

//create loop for game and set up the game

bool exit = false;

BaccaratGameDisplaying gameUI = new BaccaratGameDisplaying();

Baccarat game = new Baccarat();

while (exit == false)

{

//get the bet and start the game

game.MakeBet();

game.DealGame();

gameUI.DisplayInLine(game);

Console.ReadLine();

//check for natural win

if (game.NaturalWin())

{

Console.WriteLine("One player has a Natural: the game is over");

}

else

{

//if not, check and draw cards

game.DrawCard();

gameUI.DisplayDraw(game);

gameUI.DisplayInLine(game);

}

//display the result of the game

Console.ReadLine();

char winChar = gameUI.FinalResult(game);

Console.ReadLine();

//sort out the bets

if (game.BetLocation == winChar)

{

game.ResolveTurn(true);

}

else

{

game.ResolveTurn(false);

}

//display the money

Console.WriteLine("Money at the moment is {0}", game.Money);

//logic for second + round

Console.Write("Enter Y for another round--> ");

string choice = Console.ReadLine();

if (choice != "Y")

{

exit = true;

}

}

//end of game

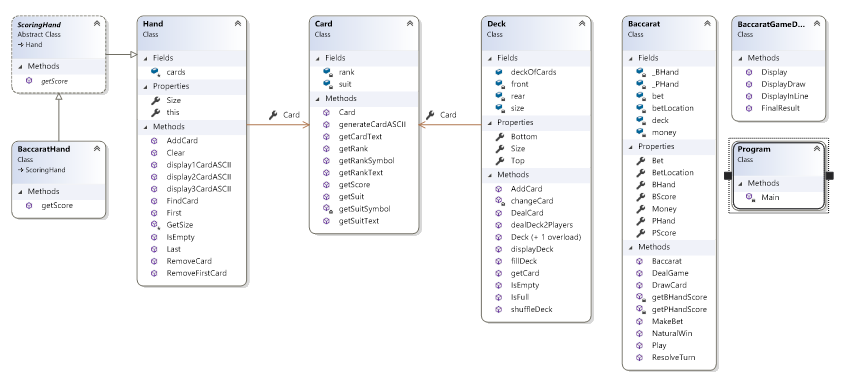
Console.WriteLine("Money at the end of the game is {0}", game.Money);

}

}

}

# Class diagrams



# Screenshots



