GAME.CS

Game.CS

```
public class Game
{
int numberOfPlayers;
List<Player> players = new List<Player>();
CardTable cardtable;
Deck deck = new Deck();
int currentPlayer = 0;
public Task nextTask;
private bool cheating = false;

public Game(CardTable c)
{
cardTable = c;
deck.Shuffle();
deck.ShowAllCards();
nextTask = Task.GetNumberOfPlayers;

public void AddPlayer(string n)
{
players.Add(new Player(n));
```

```
public void DoNextTask()
{
Console.WriteLine
("==========================");
if(nextTask == Task.GetNumberOfPlayers)
{
numberOfPlayers = cardTable.GetNumberOfPlayers();
nextTask = Task.GetNames;

else if(nextTask == Task.GetNames)
{
for( var count = 1; count<= numberOfPlayers; count ++)
{
var name = cardTable.GetPlayerName(count)
AddPlayer(name);

nextTask = Task.IntroducePlayers;

else if (next Task == Task.IntroducePlayers)
{
cardTable.Showplayers(players);
nextTask = Task.PlayerTurn;

else if (nextTask == Task.PlayerTurn)
```

```
{
cardTable.ShowHands(players)
Player player = players[currentPlayer];
if(player.status == PlayerStatus.active)
{
if(cardTable.OfferACard(player))
{
Card card = deck.DealTopCard();
player.cards.Add(card);
player.score = ScoreHand(player);
if(player.score > 21)
player.status = PlayerStatus.bust;
else if(player.score == 21)
player.status = PlayerStatus.win;
else
player.status = PlayerStatus.stay;

cardTable.ShowHand(player);
nextTask = Task.CheckforEnd)

else if (nextTask == Task.CheckForEnd)
if(!CheckActivePlayers())
{
Player winner = DoFinalScoring();
cardTable.AnnounceWinner(winner);
nextTask = Task.GameOver;
else
{
currentPlayer++;
if (currentPlayer > players.Count - 1)
currentPlayer = 0;
nextTask = Task.PlayerTurn;
```

```
public int ScoreHand(Player player)

int score = 0;
if(cheating == true && player.status == PlayerStatus.Active)
{
string response = null;
while(int.TryParse(response, out score) == false)
{
Console.Write("Ok, what should player" + player.name + "'s score
be?");
response = Console.ReadLine();

return score;

else
{
for each (Card card in player.cards)
{
string faceValue = card.ID.Remove(card.ID Length -1);
switch(faceValue)
{
case "K":
case "Q":
case "J":
score = score + 10;
break;
case "A":
```

```
score = score + 1;
break;
default:
score = score + int.Parse(faceValue);
break;

return score;

public bool CheckActivePlayers()
{
foreach (var player in players)
{
if (player.status == PlayerStatus.active)
return true;

return false;

public Player DoFinalScoring()
{
int highscore = 0;
foreach(var player in players)
```

PROGRAM.CS

Program.CS

```
static void Main (string[] args)
{
CardTable cardTable = new CardTable();
Game game = new Game(cardTable);

while(game.nextTask != Task.GameOver)
{
game.DoNextTask();
```

TASK.CS

Task.CS

```
public enum Task
{
GetNumberOfPlayers
GetNames,
IntroducePlayers,
PlayerTurn,
CheckForEnd,
GameOver
```

CARDTABLE.CS

```
public class Card Table

public CardTable()
{
Console.WriteLine("Setting Up Table....")

public void ShowPlayers(List<Player> players)
{
for (int i = 0; i < players.Count; i++)
{
players[i]. Introduce( i +1);

public int GetNumberOfPlayers()
{
Console.Write("How many players?");
string response = Console.ReadLine();
int numberOfPlayers;
while (int.TryParse(response, out numberOfPlayers)) == false ||
numberOfPlayers <2 || numberOfPlayers >8)
Console.WriteLine("Invalid number of players.");
Console.Write("How many players?");
response = Console.Readline();

return numberOfPlayers;
```

```
public string GetPlayerName (int playerNum)
{
Console.Write("Whats is the name of player #"+ playerNum + ?");
string response = Console.ReadLine();
while(response.Length <1)
{
Console.WriteLine("Invalid name.");
Console. Write("What is the name of player#" + playerNum + "?");
response = Console.Readline();

return response;

public bool OfferACard(Player player)
{
while (true)
Console.Write(player.name + ", do you want a card? (Y/N)?");
string response = Console.ReadLine();
if(response.To Upper().StartsWith("Y");
return true;
else if(response.ToUpper().StartsWith("N")
return false;
else
Console.WriteLine("Please Answer Y(es) or N(o)!");
```

```
public void ShowHand(Player player)

public void ShowHands

public void Announce Winner
```

CARD.CS

Card.Cs

```
public class Card
{
string ID
string name;
```

DECK.CS

Deck.CS

```
public class Deck
{

List<Card> cards = new List<Card>

public Deck()
{
Console.WriteLine
("******Building Deck...");
string [] suits = { "S", "H", "C", "D"};

for (int cardVal = 1; cardVal <=13; cardVal ++)
foreach (string cardSuit in suits)
{}
string cardName;
string cardLongName;

switch (cardVal)
{}
```

```
case 1:
cardName = "A";
cardLongName = "Ace";
break;
case 11:
cardName = "J";
cardLongName = "Jack";
break;
case 12:
cardName = "Q";
cardLongName = "Queen";
break;
case 13:
cardName = "K";
cardLongName = "King of";
break;
default:
cardName = cardVal.ToString();
cardLongname = cardVal.ToString() + "of" ;
break;
```

```
switch (cardSuit)
{}
case "S":
cardLongName += "Spades";
break;
case "H":
cardLongName += "Hearts";
break;
case "C":
cardLongName += "Clubs";
break;
case "D":
cardLongName =+ "Diamonds";
break;

cards.Add(new Card {ID = cardName + cardSuit, name
= cardLongName});
```

```
public void Shuffle ()
{}
Console.WriteLine("Shuffling cards...");
Random rng = new Random ();

for (int i = 0; i <cards.Count; i++)
{}
Card tmp = cards[i]
int swapindex = rng.Next(cards.Count);
cards[i] = cards [swapindex];
cards[swapindex] = tmp;

public void ShowAllCards()
{}
for (int i =0, i<cards.Count; i ++)
Console.Write(i+":"+ cards[i]. ID);
if (i< cards.Count -1)
Console.Write(" ");
else
Console.WriteLine("");
```

```
public Card DealTopCard()
{}
Card card = cards[cards.Count - 1];
cards.RemoveAt(cards.Count - 1);
return card;
```

PLAYER.CS

Player.CS

```
public class Player
{}
string name;
public List<Card> cards = new List<Card>();
public PlayerStatus status = PlayerStatus.active;
public int score;

public Player(string n)
{}
name = n;

public void Introduce(int playerNum)
{}
Console.WriteLine
("Hello, my name is " + name + " and I am player #" + playerNum);
```

PLAYERSTATUS.CS

PlayerStatus.CS

```
public enum PlayerStatus
{}
active,
stay,
bust,
win,
```