Министерство образования и науки Российской Федерации Федеральное государственное автономное образовательное

учреждение высшего образования

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

(ФГАОУ ВО «ЮФУ»)

Институт компьютерных технологий и информационной безопасности

ОТЧЕТ ПО ЛАБАРАТОРНОЙ РАБОТЕ №5  
по дисциплине «Объектно ориентированное программирование»

**КЛАССЫ, ОБЪЕКТЫ, НАСЛЕДОВАНИЕ В С#**

Выполнил  
студент группы КТбо2-1 И.А. Быстриченко

Принял

Таганрог 2020

СОДЕРЖАНИЕ

[1 Задание 5](#_Toc5341)

[1.1 Класс Card 5](#_Toc7180)

[1.2 Класс Player 5](#_Toc27098)

[1.3 Классы RandomCardPlayer, BiggestCardPlayer и SmallestCardPlayer 6](#_Toc1293)

[1.4 Класс MainPlayer 6](#_Toc22927)

[1.5 Класс GameTable 6](#_Toc3561)

[1.6 Класс GameWindow 7](#_Toc11952)

[1.7 Класс DeckLoader 7](#_Toc22666)

[1.8 Класс IntUtils 7](#_Toc9566)

[1.9 Класс PlayerFactory 7](#_Toc21224)

[1.10 Класс IdentifierGenerator 7](#_Toc29861)

[2 Используемые математические зависимости 8](#_Toc7436)

[3 Диаграмма классов 9](#_Toc5358)

[4 Листинг программы 10](#_Toc13073)

1. Задание

Определить класс Карта (Card), Карточный стол(GameTable)

Методы класса возвращают масть и достоинство. Второй

включает массив (32) объектов первого. Методы класса:

− перемешивания колоды;

− сравнения 2-х карт по достоинству при условии, что масти

одинаковы;

− создания 2-х мест и раздачи равного количества карт;

− моделирования упрощенного розыгрыша взятки: на стол

выкладываются по одной карте от каждого из 2-х игроков;

первая выложенная карта определяет масть; выигрывает

карта, старшая по достоинству (картинки старше простых

карт; козырной масти нет).

Требуется разработать программу на языке C# содержащую иерархию наследования. Реализовать функционирующую карточную игру

* 1. Класс Card

Служит для определения карт в игровой колоде и колодах игроков.

Содержит поля \_rank(достоинство) и \_suit(масть), а так же методы для получения значений этих полей.

* 1. Класс Player

Класс представляет из себя абстрактную модель игрока.

Класс содержит 3 поля: \_name, \_deck и \_winnings. То, что в них хранится понятно из названия.

Также в классе содержатся 2 абстрактных метода TakeGameCard и TakeGameCardBySuit и 7 публичных GetName, GetDeck, TakeCard, AddCard, GetWinnings, IncWinnings, DecWinnings. Именно эти 2 метода будут определять стиль игры игрока в классах наследниках.

* 1. Классы RandomCardPlayer, BiggestCardPlayer и SmallestCardPlayer

Классы RandomCardPlayer, BiggestCardPlayer и SmallestCardPlayer наследуются от класса Player и определяют его абстрактные методы.

Игрок RandomCardPlayer если есть выбор из нескольких карт, вытащит рандомную, BiggestCardPlayer вытащит карту с наибольшим рангом, SmallestCardPlayer с наименьшим рангом.

* 1. Класс MainPlayer

Класс MainPlayer отличается от классов, описанных в п1.3., лишь тем, что действиями игрока управляет пользователь

* 1. Класс GameTable

Класс GameTable служит для обеспечения игрового процесса.

У класса в полях имеется указатель на колоду, массив указателей на игроков и количество игроков.

Также класс содержит 8 публичных и 1 приватный метод.

К публичным относится функции:

ShuffleDeck, перемешивающая колоду карт. Перемешивание колоды описано в разделе “Используемые математические зависимости”

Статические функции CompareCards и CompareCardRanks о предназначении которых можно понять из названий. CompareCards выбрасывает ошибку при сравнении карт разных мастей.

Функция GiveCards, раздающая карты из колоды игрокам.

Функция GetNumberOfPlayers и GetPlayersData для получения значений полей.

Функция Play, вызывающая приватную функцию DoPlay, если у игроков есть карты. В противним случае, она выбрасывает исключение.

* 1. Класс GameWindow

Класс GameWindow- основной класс процесса игры наследующийся от win form. При инициализации этого компонента создается игровое поле с игроками.

* 1. Класс DeckLoader

Класс DeckLoader предназначен для ввода колоды карт с файла, для чего имеет единственный статический метод LoadDeck.

* 1. Класс IntUtils

Класс IntUrils предоставляет удобную функцию генерации рандомного числа в указанных пределах.

* 1. Класс PlayerFactory

Фабрика классов Player

* 1. Класс IdentifierGenerator

Генератор идентификаторов игроков

1. Используемые математические зависимости

Перемешивание колоды в классе gameTable осуществляется следующим способом:

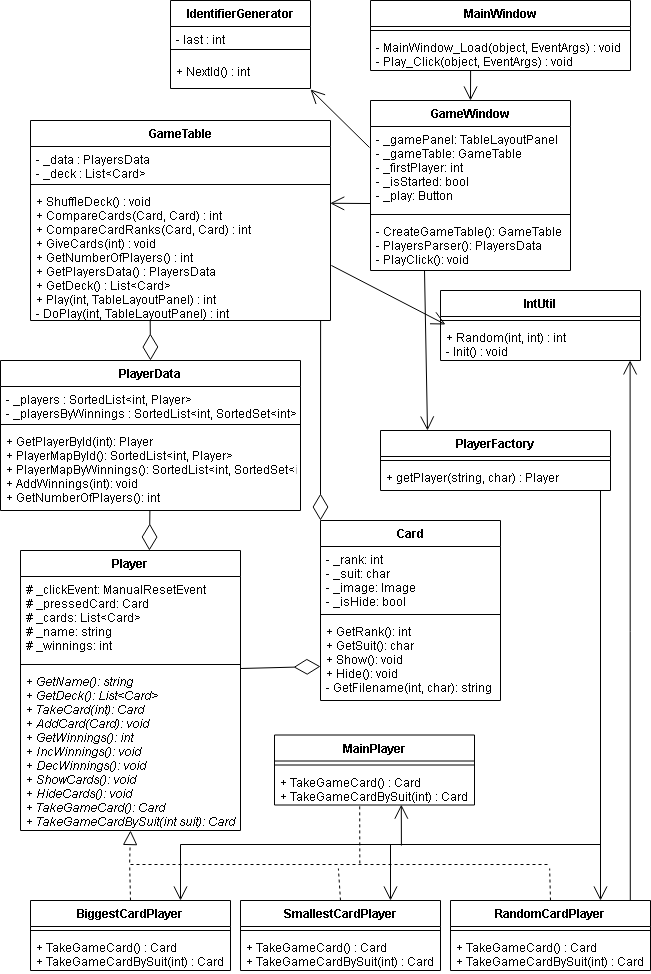
Допустим у нас колода длины N.

Тогда на первом шаге из колоды карт достается рандомная карта в диапазоне [0, N-1] и ставится в конец колоды.

Теперь колода состоит из N-1 неперемешанных карт и одной рандомно полученной на предыдущем шаге.

Повторяем предыдущие шаги вытаскивая на n-ом шаге рандомную карту в диапазоне [0, N-n] до тех пор, пока часть карт, вытащенных рандомно не станет по размеру равной N-1, и получаем перемешанную колоду карт

1. Диаграмма классов



1. Листинг программы

**Program.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace lab6

{

static class Program

{

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new MainWindow());

}

}

}

**Card.h**

using System;

using System.Collections;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using lab6.Properties;

namespace lab6

{

class Card : Button

{

private int \_rank;

private char \_suit;

private Image \_image;

private bool \_isHide;

public Card(int rank, char suit)

{

\_rank = rank;

\_suit = suit;

\_image = (Bitmap)Resources.ResourceManager.GetObject(GetFilename(rank, suit));

SetStyle(ControlStyles.UserPaint, true);

Hide();

Size = new Size(Image.Width, Image.Height);

FlatStyle = FlatStyle.Flat;

FlatAppearance.BorderSize = 0;

}

public int GetRank()

{

return \_rank;

}

public char GetSuit()

{

return \_suit;

}

public void Show()

{

\_isHide = false;

Image = new Bitmap(\_image, new Size((int)(\_image.Width \* 1.5), (int)(\_image.Height \* 1.5)));

Enabled = true;

}

public void Hide()

{

\_isHide = true;

Image = new Bitmap(Resources.card\_back, new Size((int)(Resources.card\_back.Width \* 1.5), (int)(Resources.card\_back.Height \* 1.5)));

Enabled = false;

}

private static string GetFilename(int rank, char suit)

{

string filename = "card\_";

switch (suit)

{

case '1': filename = filename + "clubs\_"; break;

case '2': filename = filename + "diamonds\_"; break;

case '3': filename = filename + "hearts\_"; break;

case '4': filename = filename + "spades\_"; break;

}

if (rank > 1 && rank < 10)

{

filename = filename + "0" + rank;

}

else

{

switch (rank)

{

case 10:

filename = filename + rank; break;

case 11:

filename = filename + "J"; break;

case 12:

filename = filename + "Q"; break;

case 13:

filename = filename + "K"; break;

case 14:

filename = filename + "A"; break;

}

}

return filename;

}

}

}

**Player.cs**

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace lab6

{

abstract class Player : FlowLayoutPanel

{

protected ManualResetEvent \_clickEvent = new ManualResetEvent(false);

protected Card \_pressedCard;

protected List<Card> \_cards;

protected string \_name;

protected int \_winnings;

public Player(string name)

{

\_name = name;

\_cards = new List<Card>();

\_winnings = 0;

AutoSize = true;

}

public string GetName()

{

return \_name;

}

public ref readonly List<Card> GetDeck()

{

return ref \_cards;

}

public void AddCard(Card card)

{

card.Click += (sender, e) =>

{

\_pressedCard = (Card)sender;

\_clickEvent.Set();

};

\_cards.Add(card);

Controls.Add(card);

Location = new Point(Location.X - (card.Width / 2), Location.Y);

if (Location.X < 0)

{

Location = new Point(0, Location.Y);

}

}

public Card TakeCard(int index)

{

Card \_lastTakedCard = \_cards[index];

BeginInvoke((MethodInvoker)(() =>

{

Controls.Remove(\_cards[index]);

\_cards.RemoveAt(index);

Location = new Point((Parent.Size.Width - \_cards.Count \* \_lastTakedCard.Width) / 2, Location.Y);

if (Location.X < 0)

{

Location = new Point(0, Location.Y);

}

}));

return \_lastTakedCard;

}

public int GetWinnings()

{

return \_winnings;

}

public void IncWinnings()

{

\_winnings++;

}

public void DecWinnings()

{

\_winnings--;

}

public void ShowCards()

{

foreach (var i in \_cards)

{

i.Show();

}

}

public void HideCards()

{

foreach (var i in \_cards)

{

i.Hide();

}

}

public abstract Card TakeGameCard();

public abstract Card TakeGameCardBySuit(int suit);

}

}

**RandomCardPlayer.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab6

{

class RandomCardPlayer : Player

{

public RandomCardPlayer(string name) : base(name) { }

public override Card TakeGameCard()

{

return TakeCard(IntUtil.Random(0, \_cards.Count - 1));

}

public override Card TakeGameCardBySuit(int suit)

{

int count = 0;

for (int i = 0; i < \_cards.Count; i++)

{

if (\_cards[i].GetSuit() == suit)

{

count++;

}

}

if (count == 0)

{

return TakeGameCard();

}

int rand = IntUtil.Random(1, count);

int index = -1;

for (int i = 0; i < \_cards.Count && rand > 0; i++)

{

if (\_cards[i].GetSuit() == suit)

{

rand--;

if (rand == 0)

{

index = i;

}

}

}

return TakeCard(index);

}

}

}

**BiggestCardPlayer.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab6

{

class RandomCardPlayer : Player

{

public RandomCardPlayer(string name) : base(name) { }

public override Card TakeGameCard()

{

return TakeCard(IntUtil.Random(0, \_cards.Count - 1));

}

public override Card TakeGameCardBySuit(int suit)

{

int count = 0;

for (int i = 0; i < \_cards.Count; i++)

{

if (\_cards[i].GetSuit() == suit)

{

count++;

}

}

if (count == 0)

{

return TakeGameCard();

}

int rand = IntUtil.Random(1, count);

int index = -1;

for (int i = 0; i < \_cards.Count && rand > 0; i++)

{

if (\_cards[i].GetSuit() == suit)

{

rand--;

if (rand == 0)

{

index = i;

}

}

}

return TakeCard(index);

}

}

}

**SmallestCardPlayer.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab6

{

class RandomCardPlayer : Player

{

public RandomCardPlayer(string name) : base(name) { }

public override Card TakeGameCard()

{

return TakeCard(IntUtil.Random(0, \_cards.Count - 1));

}

public override Card TakeGameCardBySuit(int suit)

{

int count = 0;

for (int i = 0; i < \_cards.Count; i++)

{

if (\_cards[i].GetSuit() == suit)

{

count++;

}

}

if (count == 0)

{

return TakeGameCard();

}

int rand = IntUtil.Random(1, count);

int index = -1;

for (int i = 0; i < \_cards.Count && rand > 0; i++)

{

if (\_cards[i].GetSuit() == suit)

{

rand--;

if (rand == 0)

{

index = i;

}

}

}

return TakeCard(index);

}

}

}

**MainPlayer.cs**

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace lab6

{

class MainPlayer : Player

{

public MainPlayer(string name) : base(name) { ShowCards();}

public override Card TakeGameCard()

{

\_clickEvent.Reset();

\_clickEvent.WaitOne();

int index = 0;

for (int i = 0; i < \_cards.Count; i++)

{

if (\_cards[i].GetRank() == \_pressedCard.GetRank() && \_cards[i].GetSuit() == \_pressedCard.GetSuit())

{

index = i;

break;

}

}

return TakeCard(index);

}

public override Card TakeGameCardBySuit(int suit)

{

int count = 0;

for (int i = 0; i < \_cards.Count; i++)

{

if (\_cards[i].GetSuit() == suit)

{

count++;

}

}

if (count == 0)

{

return TakeGameCard();

}

foreach (var i in \_cards)

{

if (i.GetSuit() != suit)

{

i.BeginInvoke((MethodInvoker)(() =>

{

i.Hide();

}));

}

}

\_clickEvent.Reset();

\_clickEvent.WaitOne();

int index = 0;

for (int i = 0; i < \_cards.Count; i++)

{

if (\_cards[i].GetRank() == \_pressedCard.GetRank() && \_cards[i].GetSuit() == \_pressedCard.GetSuit())

{

index = i;

break;

}

}

foreach (var i in \_cards)

{

if (i.GetSuit() != suit)

{

i.BeginInvoke((MethodInvoker)(() =>

{

i.Show();

}));

}

}

return TakeCard(index);

}

}

}

**PlayersData.cs**

using System;

using System.Collections.Generic;

using System.Data;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab6

{

class PlayersData

{

private SortedList<int, Player> \_players;

private SortedList<int, SortedSet<int>> \_playersByWinnings;

public PlayersData(SortedList<int, Player> players)

{

\_players = players;

\_playersByWinnings = new SortedList<int, SortedSet<int>>();

\_playersByWinnings.Add(0, new SortedSet<int>());

foreach (var player in \_players)

{

\_playersByWinnings[0].Add(player.Key);

}

}

public Player GetPlayerById(int id)

{

return \_players[id];

}

public ref readonly SortedList<int, Player> PlayerMapById()

{

return ref \_players;

}

public ref readonly SortedList<int, SortedSet<int>> PlayerMapByWinnings()

{

return ref \_playersByWinnings;

}

public void AddWinnings(int id)

{

\_playersByWinnings[\_players[id].GetWinnings()].Remove(id);

\_players[id].IncWinnings();

if (!\_playersByWinnings.ContainsKey(\_players[id].GetWinnings()))

{

\_playersByWinnings.Add(\_players[id].GetWinnings(), new SortedSet<int>());

}

\_playersByWinnings[\_players[id].GetWinnings()].Add(id);

}

public int GetNumberOfPlayers() {

return \_players.Count;

}

}

}

**PlayerFactory.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab6

{

class PlayerFactory

{

public static Player getPlayer(string name, char type)

{

Player player;

switch (type)

{

case 'm': player = new MainPlayer(name); break;

case 'r': player = new RandomCardPlayer(name); break;

case 'b': player = new BiggestCardPlayer(name); break;

case 's': player = new SmallestCardPlayer(name); break;

default: throw new Exception("Invalid input. Try again!\nFormat: \"Name type\"");

}

return player;

}

}

}

**GameTable.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace lab6

{

class GameTable

{

private List<Card> \_deck;

private PlayersData \_data;

public GameTable(PlayersData data, List<Card> deck)

{

\_data = data;

\_deck = deck;

}

public void ShuffleDeck()

{

\_deck.Sort((x, y) => IntUtil.Random(-1, 1));

}

public static int CompareCards(Card card1, Card card2)

{

if (card1.GetSuit() == card2.GetSuit())

{

return CompareCardRanks(card1, card2);

}

else

{

throw new Exception("Сomparison of cards of different suits");

}

}

public static int CompareCardRanks(Card card1, Card card2)

{

return card1.GetRank() - card2.GetRank();

}

public void GiveCards(int number)

{

if (number < 0 || number > \_deck.Count / \_data.GetNumberOfPlayers())

{

throw new Exception("Wrong number of cards!");

}

SortedList<int, Player> map = \_data.PlayerMapById();

foreach (var i in map)

{

for (int j = 0; j < number; j++)

{

i.Value.AddCard(\_deck[0]);

\_deck.RemoveAt(0);

}

}

}

public int GetNumberOfPlayers()

{

return \_data.GetNumberOfPlayers();

}

public PlayersData GetPlayersData()

{

return \_data;

}

public ref readonly List<Card> GetDeck()

{

return ref \_deck;

}

public int Play(int firstPlayerIndex, TableLayoutPanel panel)

{

if (\_data.PlayerMapById().ToList()[firstPlayerIndex].Value.GetDeck().Count == 0)

{

throw new Exception("Players don't have cards");

}

return DoPlay(firstPlayerIndex, panel);

}

private int DoPlay(int firstPlayerIndex, TableLayoutPanel panel)

{

List<KeyValuePair<int, Player>> list = \_data.PlayerMapById().ToList();

ManualResetEvent e = new ManualResetEvent(false);

Card biggestCard = list[firstPlayerIndex].Value.TakeGameCard();

char suit = biggestCard.GetSuit();

int winner = firstPlayerIndex;

e.Reset();

panel.BeginInvoke((MethodInvoker)(() =>

{

biggestCard.Show();

Label l1 = new Label();

l1.Size = biggestCard.Size;

l1.Image = biggestCard.Image;

panel.Controls.Add(l1, 0, list.Count - firstPlayerIndex - 1);

e.Set();

}));

e.WaitOne();

int i = (firstPlayerIndex + 1) % list.Count;

while (i != firstPlayerIndex)

{

Card playerCard = list[i].Value.TakeGameCardBySuit(suit);

e.Reset();

panel.BeginInvoke((MethodInvoker)(() =>

{

playerCard.Show();

Label l2 = new Label();

l2.Size = playerCard.Size;

l2.Image = playerCard.Image;

panel.Controls.Add(l2, 0, list.Count - i - 1);

e.Set();

}));

e.WaitOne();

if (playerCard.GetSuit() == suit && CompareCards(playerCard, biggestCard) > 0)

{

biggestCard = playerCard;

winner = i;

}

i++;

if (i == list.Count)

{

i = 0;

}

}

\_data.AddWinnings(list[winner].Key);

return list[winner].Key;

}

}

}

**IntUtil.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab6

{

public static class IntUtil

{

private static Random random;

private static void Init()

{

if (random == null) random = new Random();

}

public static int Random(int min, int max)

{

Init();

return random.Next(min, max);

}

}

class IdentifierGenerator

{

private int \_last;

public IdentifierGenerator()

{

\_last = 1;

}

public int NextId()

{

return \_last++;

}

}

}

**DeckLoader.cs**

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab6

{

class DeckLoader

{

public static List<Card> LoadDeck()

{

List<Card> deck = new List<Card>();

for (int i = 2; i < 15; i++)

{

for (int j = 1; j < 5; j++)

{

string rank = "";

switch (i)

{

case 11:

rank += "J"; break;

case 12:

rank += "Q"; break;

case 13:

rank += "K"; break;

case 14:

rank += "A"; break;

default: rank += i; break;

}

Card temp = CardParser(rank, "" + j);

deck.Add(temp);

}

}

return deck;

}

private static Card CardParser(string rank, string suit)

{

int cardRank = -1;

char cardSuit = ' ';

if (rank == "10")

{

cardRank = 10;

}

else if (rank.Length == 1)

{

if (rank[0] >= '2' && rank[0] <= '9')

{

cardRank = rank[0] - '0';

}

else

{

switch (rank[0])

{

case 'J': cardRank = 11; break;

case 'Q': cardRank = 12; break;

case 'K': cardRank = 13; break;

case 'A': cardRank = 14; break;

}

}

}

if (suit.Length == 1 && suit[0] >= '1' && suit[0] <= '4')

{

cardSuit = suit[0];

}

if (cardRank == -1 || cardSuit == ' ')

{

throw new Exception("Card parser exception");

}

return new Card(cardRank, cardSuit);

}

};

}

**MainWindow.cs**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace lab6

{

public partial class MainWindow : Form

{

public MainWindow()

{

InitializeComponent();

}

private void MainWindow\_Load(object sender, EventArgs e)

{

FormBorderStyle = FormBorderStyle.FixedSingle;

}

private void Play\_Click(object sender, EventArgs e)

{

GameWindow gw = new GameWindow();

gw.Closed += Exit\_Click;

gw.FormBorderStyle = FormBorderStyle.FixedSingle;

gw.Owner = this;

gw.Show();

this.Hide();

}

private void Rules\_Click(object sender, EventArgs e)

{

MessageBox.Show("Игроки ходят по очереди." +

"\nПервый игрок выкладывет любую карту," +

"\nдругие должны выкладывать карту этой же масти." +

"\nВыигрывает раунд тот игрок, который выложил наибольшую карту данной масти" +

"\nЕсли карты данной масти нет, то игрок выкладывает любую" +

"\nПри этом он автоматически проигрывает раунд.");

}

private void Exit\_Click(object sender, EventArgs e)

{

Application.Exit();

}

}

}

**GameWindow.cs**

using System;

using System.Collections;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Runtime.InteropServices;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Windows.Forms;

using lab6.Properties;

namespace lab6

{

public partial class GameWindow : Form

{

private TableLayoutPanel \_gamePanel;

private GameTable \_gameTable;

private int \_firstPlayer = 1;

private bool \_isStarted = false;

private Button \_play;

public GameWindow()

{

InitializeComponent();

\_gameTable = CreateGameTable();

\_gameTable.ShuffleDeck();

try

{

\_gameTable.GiveCards(6);

}

catch (Exception e)

{

Console.WriteLine(e);

}

\_gameTable.GetPlayersData().PlayerMapById()[1].ShowCards();

}

private GameTable CreateGameTable()

{

PlayersData data = PlayersParser();

List<Card> deck = DeckLoader.LoadDeck();

GameTable gameTable = new GameTable(data, deck);

Card fakeCard = new Card(2, '1');

\_gamePanel = new TableLayoutPanel

{

ColumnCount = 1,

RowCount = 2,

AutoSize = true,

Location = new Point((Size.Width - fakeCard.Width) / 2, Size.Height / 2 - fakeCard.Height)

};

\_play = new Button()

{

Location = new Point(675, 220),

Size = new Size(113, 61),

TabIndex = 0,

Text = "Play",

UseVisualStyleBackColor = true

};

\_play.Click += PlayClick;

gameTable.GetPlayersData().PlayerMapById()[1].Location = new Point(Size.Width / 2, Size.Height - fakeCard.Height - 48);

gameTable.GetPlayersData().PlayerMapById()[2].Location = new Point(Size.Width / 2, 0);

Controls.Add(gameTable.GetPlayersData().PlayerMapById()[1]);

Controls.Add(gameTable.GetPlayersData().PlayerMapById()[2]);

Controls.Add(\_play);

Controls.Add(\_gamePanel);

return gameTable;

}

private PlayersData PlayersParser()

{

SortedList<int, Player> players = new SortedList<int, Player>();

IdentifierGenerator gen = new IdentifierGenerator();

players.Add(gen.NextId(), PlayerFactory.getPlayer("Player", 'm'));

players.Add(gen.NextId(), PlayerFactory.getPlayer("Bot", 'r'));

return new PlayersData(players);

}

private void PlayClick(object sender, EventArgs e)

{

SortedList<int, Player> players = \_gameTable.GetPlayersData().PlayerMapById();

if (!\_isStarted)

{

\_isStarted = true;

\_gamePanel.Controls.Clear();

(new Thread(() =>

{

int winner = \_gameTable.Play(\_firstPlayer, \_gamePanel);

\_firstPlayer = (\_firstPlayer + 1) % \_gameTable.GetNumberOfPlayers();

switch (winner)

{

case 1:

PlayerScore.BeginInvoke((MethodInvoker) (() =>

{

PlayerScore.Text = "" + players[winner].GetWinnings();

}));

break;

case 2:

EnemyScore.BeginInvoke((MethodInvoker) (() =>

{

EnemyScore.Text = "" + players[winner].GetWinnings();

}));

break;

}

if (players[winner].GetDeck().Count == 0)

{

\_play.BeginInvoke((MethodInvoker) (() => { \_play.Hide(); }));

if (players[1].GetWinnings() > players[2].GetWinnings())

{

MessageBox.Show("ОГО!\nТы выиграл!");

}

else if (players[1].GetWinnings() < players[2].GetWinnings())

{

MessageBox.Show("О НЕТ!\nТы проиграл(\nПовезет в другой раз");

}

else

{

MessageBox.Show("НИЧЬЯ :|");

}

}

\_isStarted = false;

})).Start();

}

}

}

}