

UML & OOP

Case 1: Restaurant Table Booking

Case about restaurant table booking is presented now.

All the steps from requirements (now use cases mainly) to finished application are shown.

Case 2: Blackjack Card Game

Case concerning card game is presented here.

Case 1. Restaurant Table Booking

Requirements document

Version 0.01

Topic

Restaurant table booking



Introduction

Sometimes it is time to book a table from a restaurant – for one evening.

The app that is to be created has to show free tables: user can then book some of those tables to next evening.

User gets confirmation and that table is to be set reserved.

Restaurant desk person can see the reservation and keep it reserved until the right customers comes...

Use cases

Actors

Customer

Desk person

Use case documentations

Name of the use case

Booking a table

Actor

Customer

Precondition

App has been launched

Defitinion of the use case

Customer lists all the free tables by clicking a button

Customer then chooses one of free tables and confirms booking

Exception: no free tables

Exceptions

Customer is shown a message that there are no free tables

5

Postcondition

App is in idle state, can be closed

Use case name

Confirming

Actors

Employee

Preconditions

App is open, tables are listed and new booking is shown

Defitinion

Employee confirms the bookind by clicking a button

Postcondition

App stays in idle mode

Customer data: name

Table data: number, state

Design document

Version 0.5

By searching for nouns from use case texts we find these classes:

Customer

Restaurant

Table

Booking

Class definitions and relationships

Customer
-name: String
<<create>>+Customer() <<create>>+Customer(n: String) +setName(n: String): void +getName(): String

Table
-nr: int -state: int
<<create>>+Table(n: int, s: int) <<create>>+Table() +setState(t: int): void +setNr(n: int): void +getState(): int +getNr(): int

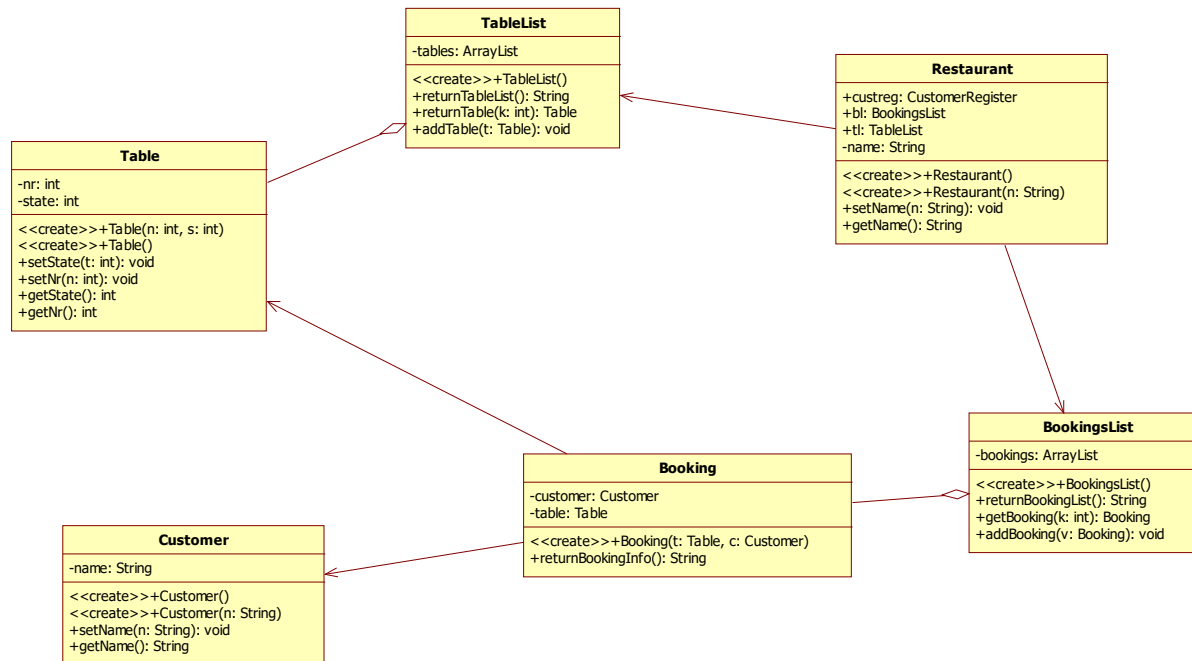
Restaurant
+custreg: CustomerRegister +bl: BookingsList +tl: TableList -name: String
<<create>>+Restaurant() <<create>>+Restaurant(n: String) +setName(n: String): void +getName(): String

BookingsList
-bookings: ArrayList
<<create>>+BookingsList() +returnBookingList(): String +getBooking(k: int): Booking +addBooking(v: Booking): void

TableList
-tables: ArrayList
<<create>>+TableList() +returnTableList(): String +returnTable(k: int): Table +addTable(t: Table): void

Booking
-customer: Customer -table: Table
<<create>>+Booking(t: Table, c: Customer) +returnBookingInfo(): String

Relationships



Implementation

Classes

```
class Customer
{
    private String name;

    public Customer()
    {

    }

    public Customer(String n)
    {
        name = n;
    }

    public void setName(String n)
    {
        name = n;
    }

    public String getName()
    {
        return name;
    }

}
```

```
class Table
{
    public Table(int n, int s)
    {
        nr = n;
        state = s;
    }
}
```

```
    }  
  
    public Table()  
    {  
  
    }  
  
    public void setState(int t)  
    {  
        state = t;  
    }  
  
    public void setNr(int n)  
    {  
        nr = n;  
    }  
  
    public int getState()  
    {  
        return state;  
    }  
  
    public int getNr()  
    {  
        return nr;  
    }  
  
    // Members  
    int nr;  
    int state;  
}
```

```
class Booking
{
    private Customer customer;
    private Table table;

    public Booking(Table t, Customer c)
    {
        customer = c;
        table = t;
    }

    public String returnBookingInfo()
    {
        String info = "Date and time: " + DateTime.Now;
        info += " " + customer.getName() + " " + table.getNr();

        return info;
    }
}
```

```
class CustomerRegister
{
    ArrayList customers;

    public CustomerRegister()
    {
        customers = new ArrayList();
    }

    public String returnCustomerList()
    {
        String list = "";
        for (int k = 0; k < customers.Count; k++)
        {
            Customer c;
            c = (Customer) customers[k];
        }
    }
}
```

```
        list += c.getName();
    }
    return list;
}

public Customer returnCustomer(int k)
{
    return (Customer) customers[k];
}

public Customer returnCustomer(String n)
{
    Customer temp = null;
    foreach (Customer c in customers)
    {
        if (c.getName().Equals(n))
            temp = c;
    }
    return temp;
}

public void addCustomer(Customer t)
{
    customers.Add(t);
}
}
```

```
class TableList
{
    private ArrayList tables;
    public TableList()
    {
        tables = new ArrayList();
    }
}
```

```

        for (int k = 0; k < 10; k++)
        {
            Table t = new Table(k, 1);
            tables.Add(t);
        }

    }

    public String returnTableList()
    {
        String list = "";
        for (int k = 0; k < tables.Count; k++)
        {
            Table t = new Table();
            t = (Table)tables[k];
            list += t.getNr() + " " + t.getState() + "\n";
        }
        return list;
    }

    public Table returnTable(int k)
    {
        return (Table)tables[k];
    }

    public void addTable(Table t)
    {
        tables.Add(t);
    }

}

```

```
class BookingsList
{
    ArrayList bookings;

    public BookingsList()
    {
        bookings = new ArrayList();
    }

    public String returnBookingList()
    {
        String list = "";
        for (int k = 0; k < bookings.Count; k++)
        {
            Booking c;
            c = (Booking) bookings[k];
            list += c.returnBookingInfo() + "\n";
        }
        return list;
    }

    public Booking getBooking(int k)
    {
        return (Booking) bookings[k];
    }

    public void addBooking(Booking v)
    {
        bookings.Add(v);
    }
}
```

```
class Restaurant
{
    public CustomerRegister custreg;
    public BookingsList bl;
    public TableList tl;

    public Restaurant()
    {
        custreg = new CustomerRegister();
        bl = new BookingsList();
        tl = new TableList();
    }

    public Restaurant(String n)
    {
        custreg = new CustomerRegister();
        bl = new BookingsList();
        tl = new TableList();
        name = n;
    }

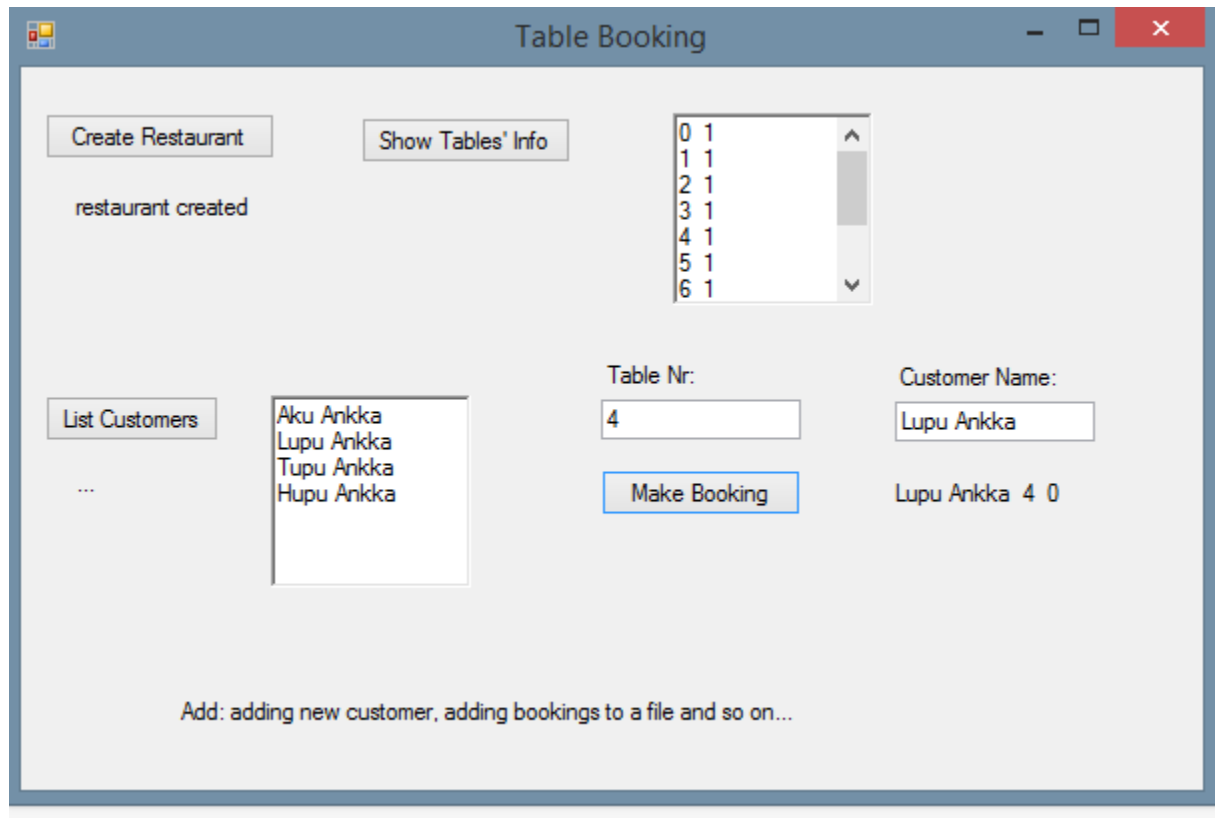
    private String name;

    public void setName(String n)
    {
        name = n;
    }

    public String getName()
    {
        return name;
    }
}
```

16

Gui prototype



Create the app!

Make it more versatile and personal!: some hints here:

17

Restaurant

GUI

Create this

Richtextbox

Form1

Create Restaurant Show Tables' Info

restaurant created

List Customers label2

Aku Ankka
Lupu Ankka
Tupu Ankka
Hupu Ankka

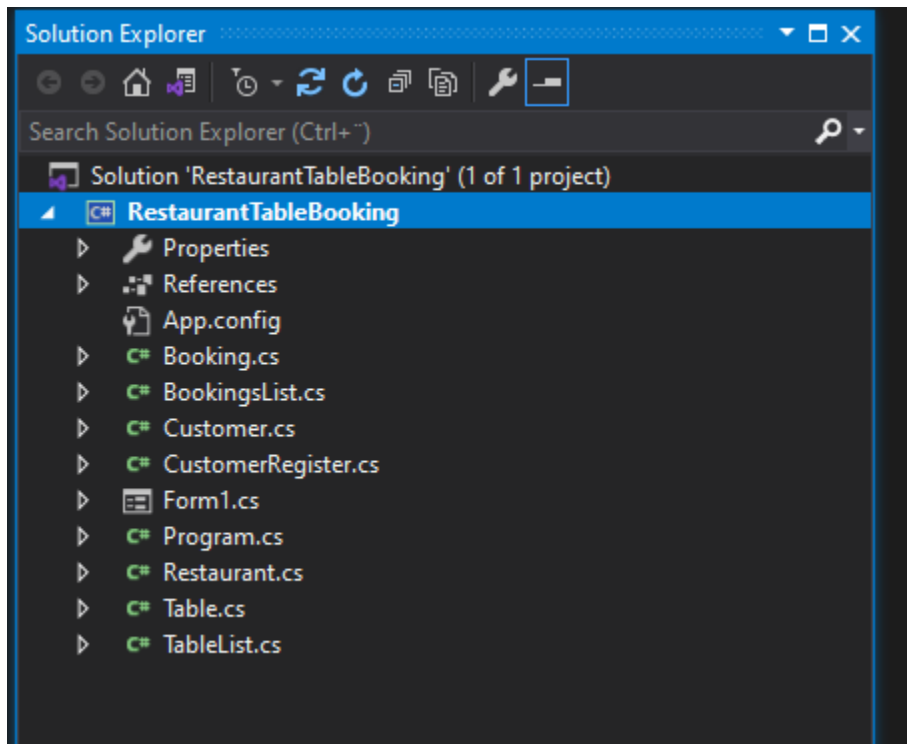
Table Nr: 2 Make Booking

Customer Name: Aku Ankka

Aku Ankka 2 0

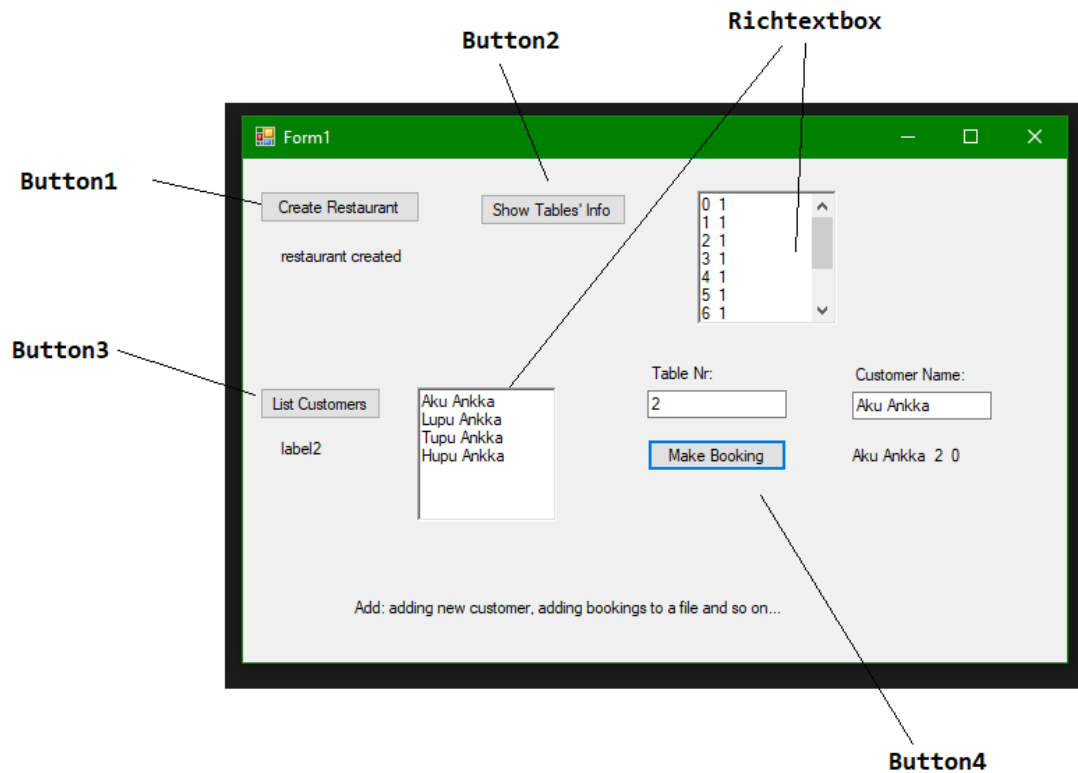
Add: adding new customer, adding bookings to a file and so on...

Add these classes to the project (take from console project)



Add these codes

Check button names here first



```
namespace RestaurantTableBooking
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        Restaurant rest;
```

```

private void button1_Click(object sender, EventArgs e)
{
    rest = new Restaurant();
    label1.Text = "restaurant created";
}
private void button2_Click(object sender, EventArgs e)
{
    richTextBox1.Text = rest.tl.returnTableList();
}
private void button3_Click(object sender, EventArgs e)
{
    String path = "customersFile.txt";
    String allNames = "";
    try
    {
        using (StreamReader sr = File.OpenText(path))
        {
            string s = "";
            while ((s = sr.ReadLine()) != null)
            {
                rest.custreg.addCustomer(new Customer(s));
                allNames += s + "\n";
            }
        }
    }
    catch (System.SystemException ee)
    {
        label2.Text = "Error - check the folder!!!";
    }
    richTextBox2.Text = allNames;
}

private void button4_Click(object sender, EventArgs e)
{
    int tableNr = Convert.ToInt16(textBox1.Text);
    Table bookedTable = rest.tl.returnTable(tableNr);
    rest.tl.returnTable(tableNr).setState(0);
}

```

```
String customerName = textBox2.Text;
Customer booker = rest.custreg.returnCustomer(customerName);

label3.Text = booker.getName() + " " + bookedTable.getNr() +
    " " + bookedTable.getState();

rest.bl.addBooking(new Booking(bookedTable, booker));    }  }
```

Test!

Add new features!

Case 2: BlackJack Card Game

BlackJack Card Game

Introduction

Equally well known as Twenty-One.

the popularity of Blackjack dates from World War I, its roots go back to the 1760s in France, where it is called Vingt-et-Un (French for 21). Today, Blackjack is the one card game that can be found in every American gambling casino. As a popular home game, it is played with slightly different rules.

The Pack The standard 52-card pack is used, but in most casinos several decks of cards are shuffled together.

Object of the Game Each participant attempts to beat the dealer by getting a count as close to 21 as possible, without going over 21.

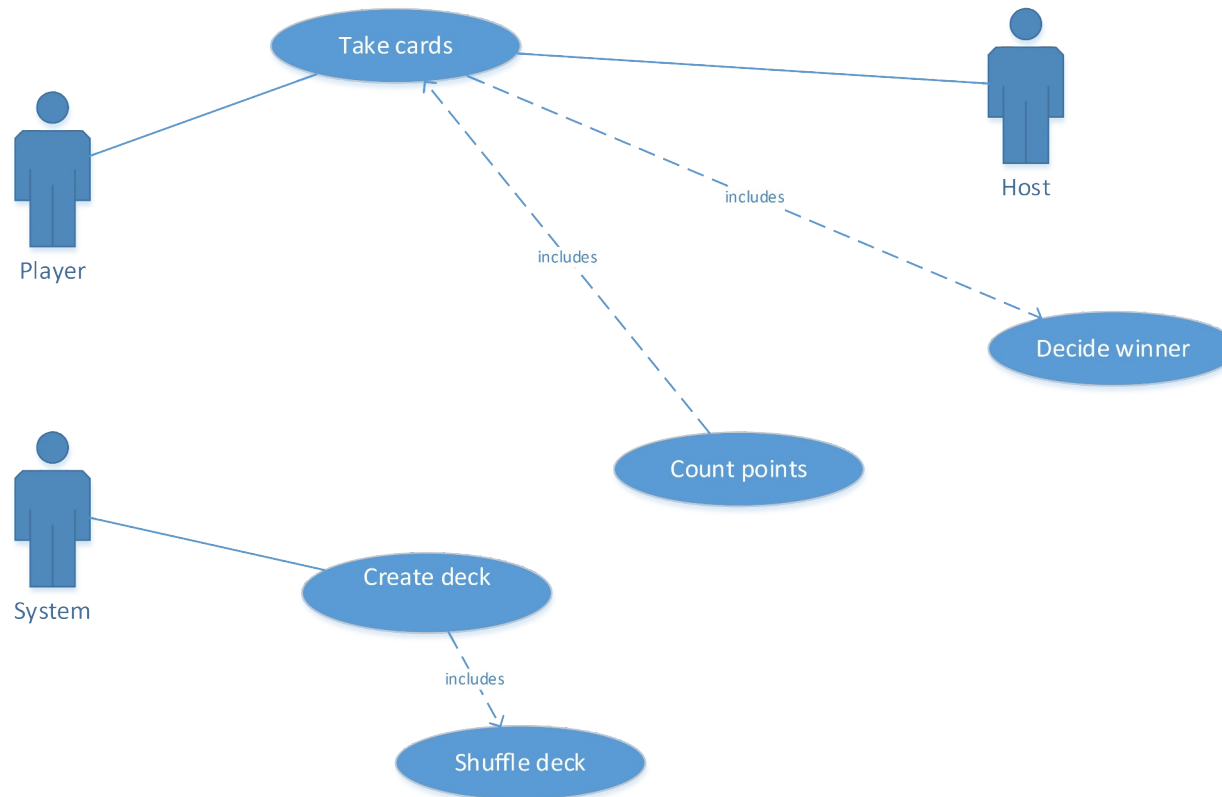
Betting is possible if wanted.

The goal is the get 21 points. Ace is here 1.

If host gets same points as player, host wins.

Requirements

Use case: functional requirements



Name	Take cards
Actors	Player, host
Preconditions	Deck is created and shuffled
Explanation	<p>Player takes cards from the deck one by one by clicking a button. Points and card images are shown on a form. If points are 21, player is winner and message is shown.</p> <p>If points are over 21 host has won.</p> <p>If points are 18 – 20, host may start taking cards.</p> <p>Exception: Deck is empty</p>
Exceptions	Deck is empty: if deck is empty, it has to be filled again and shuffled. Depending on the situation player's and host's cards are taken with or not.
Postconditions	A new round can be started.

Design

From use case 1 we get this scenario:

Player takes cards from the deck one by one by clicking a button. Points and card images are shown on a form. If points are 21, player is winner and message is shown.

If points are over 21 host has won.

If points are 18 – 20, host may start taking cards.

We can underline nouns that are class candidates:

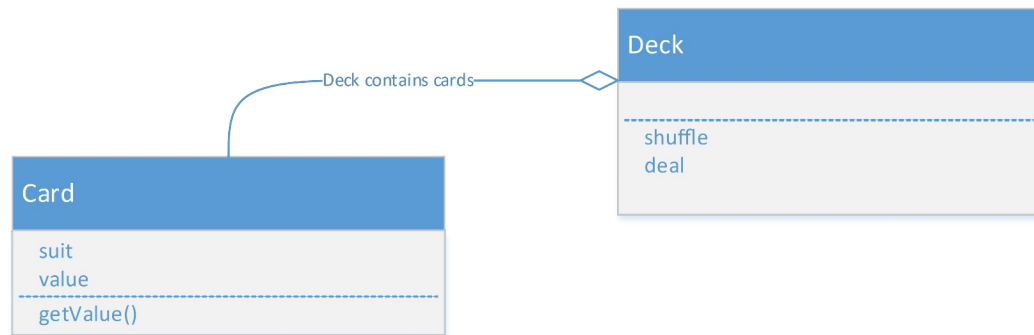
Player takes cards from the deck one by one by clicking a button. Points and card images are shown on a form. If points are 21, player is winner and message is shown.

If points are over 21 host has won.

If points are 18 – 20, host may start taking cards.

26

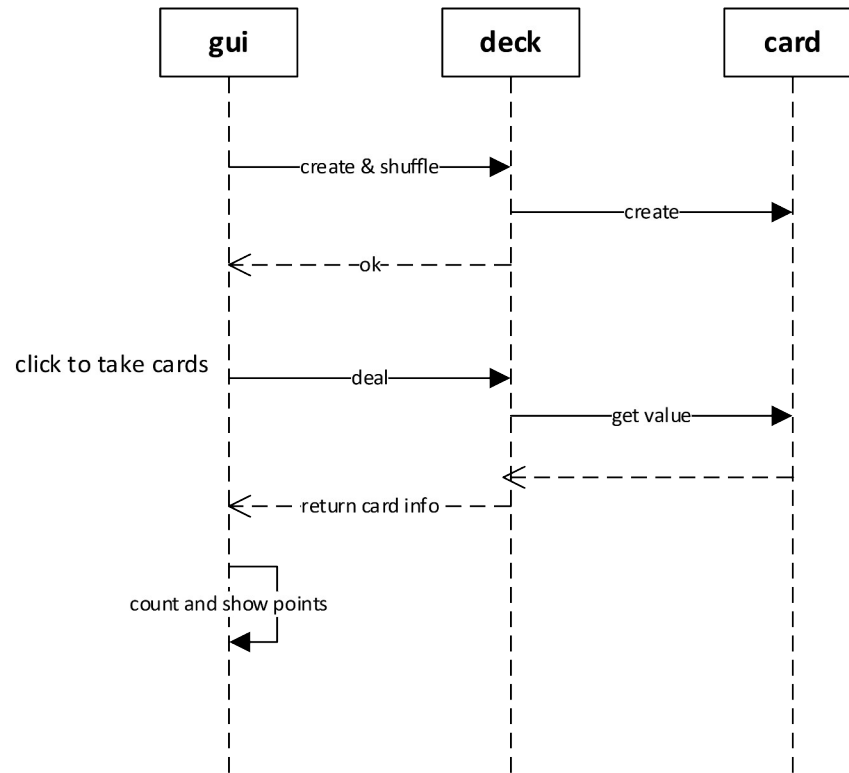
Class diagram



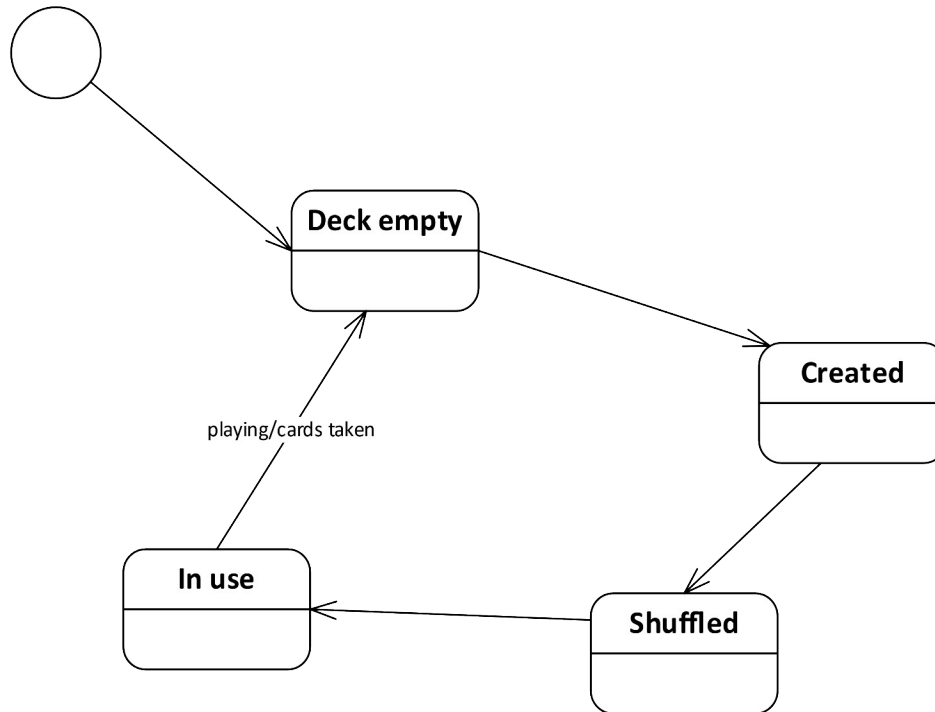
Gui prototype



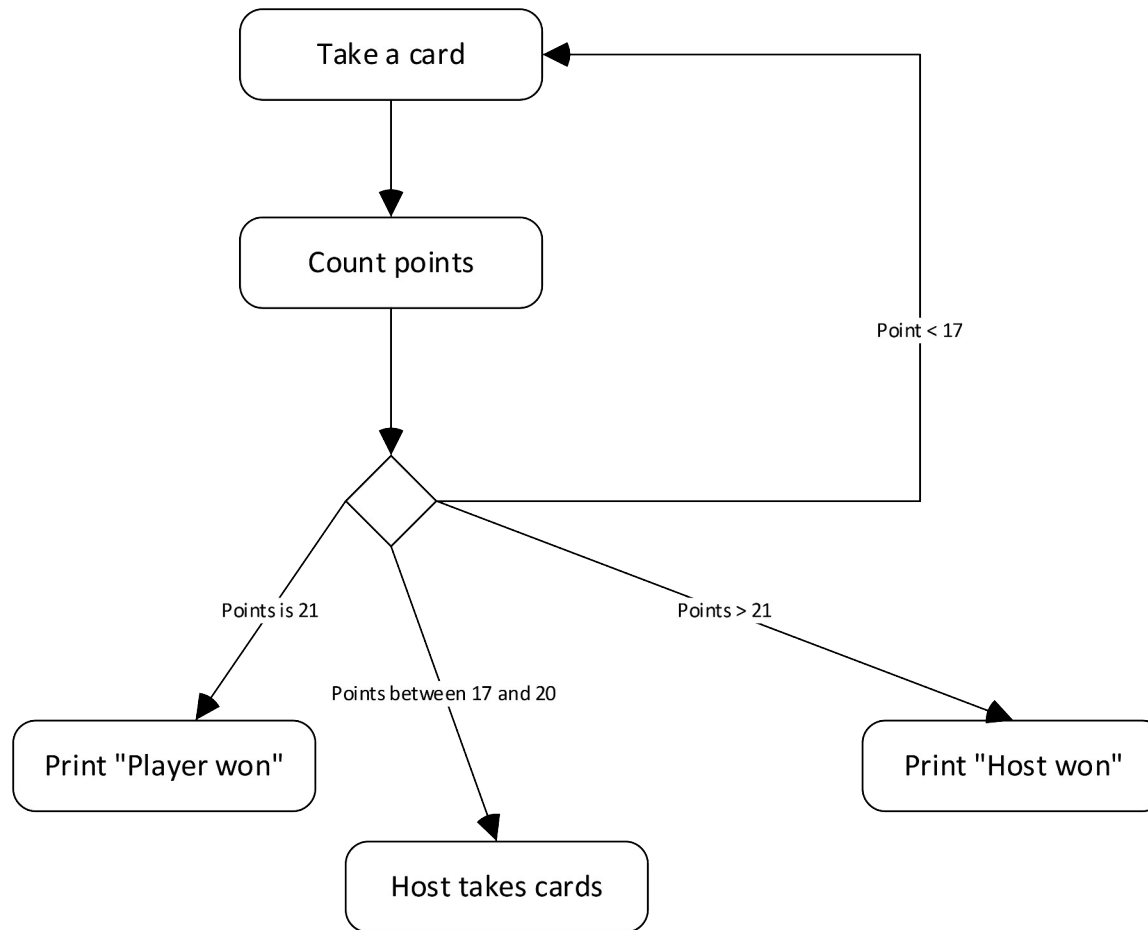
Sequence diagram



State chart



Activity diagram



Final classes in code

```
class Card
{
    private String suit;
    private String value;
    public Card(String m, String a)
    {
        suit = m;
        value = a;
    }

    public String getFileName()
    {
        String name = "" + suit + value + ".png";
        return name;
    }

    public String returnSuit()
    {
        return suit;
    }

    public String returnValue()
    {
        return value;
    }

    public String returnCardInfo()
    {
        String[] suits = { "Club", "Spade", "Heart", "Diamond" };
        String[] values = { "Ace", "2", "3", "4", "5", "6", "7", "8", "9", "10", "Jack", "Queen", "King" };
        int ind1 = Convert.ToInt16(suit);
        int ind2 = Convert.ToInt16(value)-1;
        String cardSuit = suits[ind1];
        String cardValue = values[ind2];
        String cardInfo = cardSuit + " " + cardValue;
        return cardInfo;
    }
}
```

```
    }
}
```

```
class Deck
{
    ArrayList cards = new ArrayList();

    public Deck()
    {
        int k = 0;
        for (int m = 0; m < 4; m++)
            for (int a = 1; a < 14; a++)
            {
                cards.Add(new Card("" + m, "" + a));
                k++;
            }
    }

    public String printDeck()
    {
        String allCards = "";
        foreach (Card c in cards)
            allCards += c.returnCardInfo() + "\n";

        return allCards;
    }

    public int deckSize()
    {
        return cards.Count;
    }

    public Card getFromTop()
    {

```











```
        Card temp = (Card) cards[0];
        cards.RemoveAt(0);
        return temp;
    }

    public String getThisCardFileName(int n)
    {
        Card x = (Card)cards[n];
        return x.GetFileName();
    }

















    public void shuffle()
    {
        Random rr = new Random();
        for (int i = 0; i < 1000; i++)
        {
            int x = rr.Next(0, 52);
            int y = rr.Next(0, 52);
            Card temp = (Card)cards[x];
            cards[x] = (Card)cards[y];
            cards[y] = temp;
        }
    }
}
```

Cards

Cards are in png-format, added to subfolder cards:

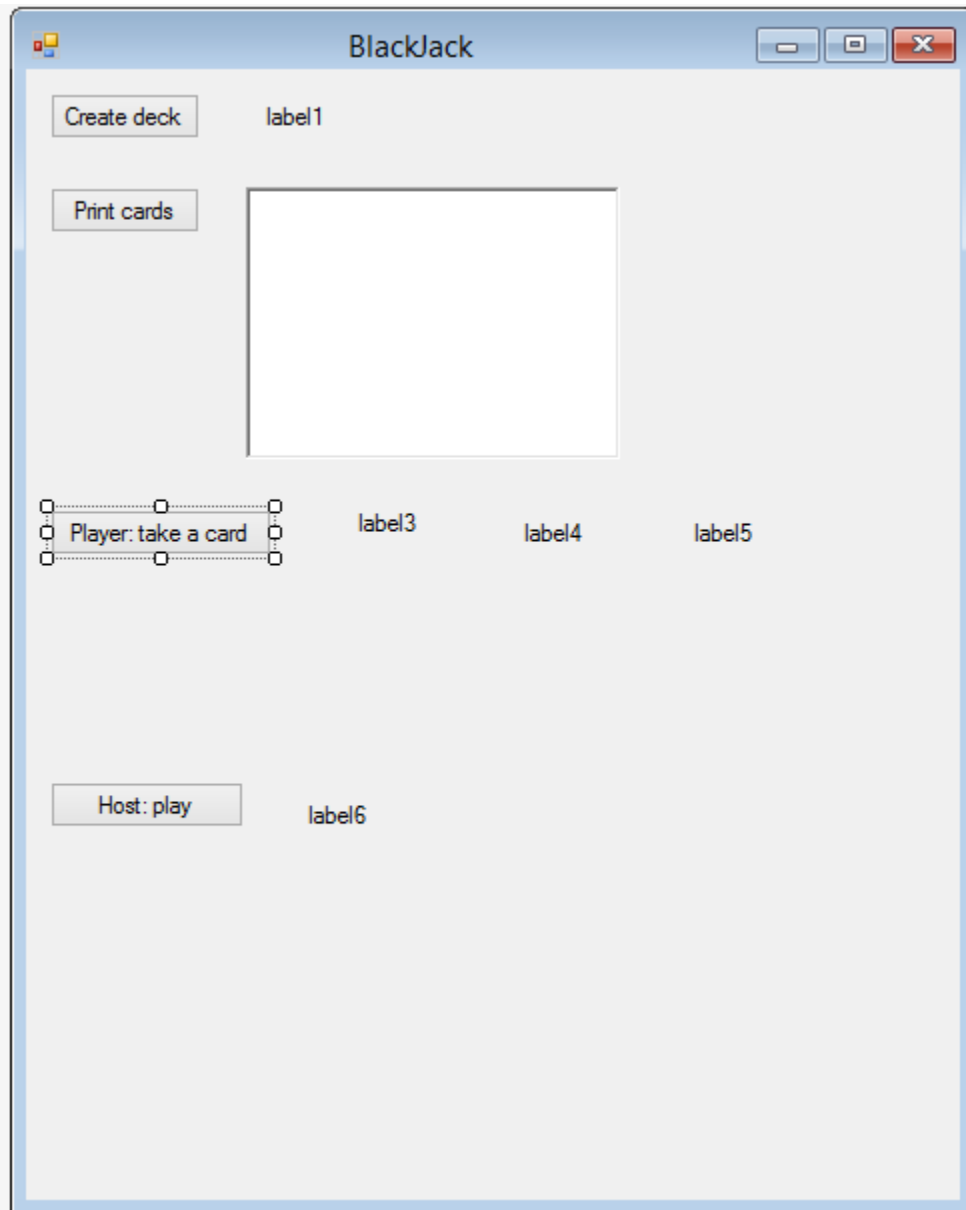
Desktop ▸ OO2017 ▸ BlackJack2017 ▸ BlackJack2017 ▸ bin ▸ Debug ▸ cards				
Name	Date	Type	Size	Tags
 01.png	1.4.2017 16:04	PNG File	1 KB	
 02.png	1.4.2017 16:04	PNG File	1 KB	
 03.png	1.4.2017 16:04	PNG File	1 KB	
 04.png	1.4.2017 16:04	PNG File	1 KB	
 05.png	1.4.2017 16:04	PNG File	1 KB	
 06.png	1.4.2017 16:04	PNG File	1 KB	
 07.png	1.4.2017 16:04	PNG File	1 KB	
 08.png	1.4.2017 16:04	PNG File	1 KB	

Cards are named so that the 1. Number tells the suit and 2. Value the card's value.

Desktop ▸ OO2017 ▸ BlackJack2017 ▸ BlackJack2017 ▸ bin ▸ Debug ▸ cards							
							
01.png	02.png	03.png	04.png	05.png	06.png	07.png	08.png
							
12.png	013.png	13.png	14.png	15.png	16.png	17.png	18.png

35

Final gui



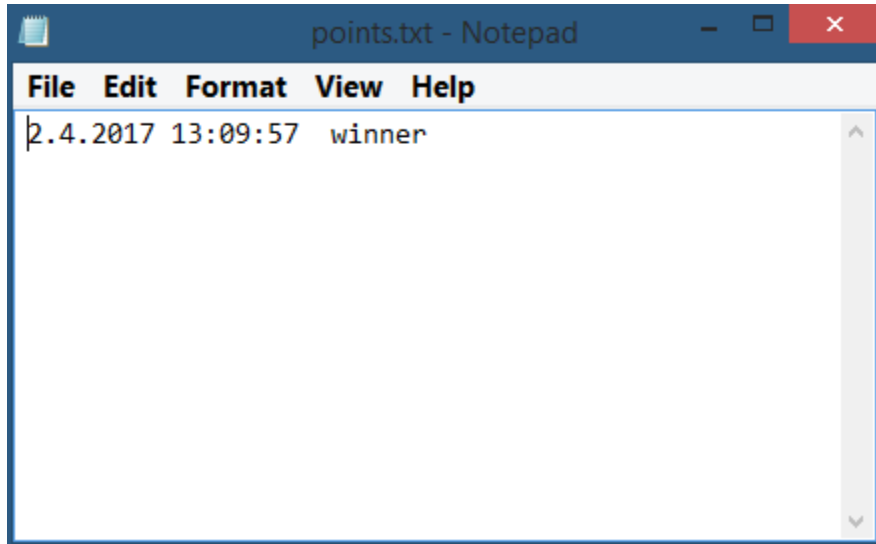
Using of exceptions

Here, if deck is empty when you try to take a card, an exception is thrown - program does not crash but gives a message about the situation:

```
Card taken = null;
try
{
    taken = deck.getFromTop();
}
catch (Exception ex)
{
    label13.Text = "deck is empty!! " + ex.Message;
}
```

File IO

Date and time and winner are added to the file, points.txt that is saved to debug-folder:



Function that handles file IO:

```
static String saveToFile(int w)
{
    String winner = "";
    if (w == 1)
        winner = "player";
    else
        winner = "host";
    String now = "" + DateTime.Now;
    String message = now + "  winner";

    String filename = "points.txt";
    FileStream fs;
    try
```

```
{  
    using (fs = new FileStream(filename, FileMode.Append,  
        FileAccess.Write))  
        using (StreamWriter sw = new StreamWriter(fs))  
        {  
            sw.WriteLine(message);  
        }  
    }  
    catch (System.Exception ee)  
    {  
        return ("Error - check the folder!!!");  
    }  
    return (winner);  
}
```

Test:



What can be added to the next version?

Betting

Starting a new round

Choosing the value of ace (1 or 11 or 14)

And so on.

Appendix 1: Whole code

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace BlackJack2017
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();

            PictureBox[] boxes = new PictureBox[10];

            private void Form1_Load(object sender, EventArgs e)
            {
                for (int k = 0; k < 10; k++)
                {
```

```

        boxes[k] = new PictureBox();
        boxes[k].SizeMode = PictureBoxSizeMode.StretchImage;
        boxes[k].Height = 70;
        boxes[k].Width = 50;
        boxes[k].Parent = this;
    }
}

Deck deck;

private void button1_Click(object sender, EventArgs e)
{
    deck = new Deck();
    deck.shuffle();

    label1.Text = "Deck ok";
}

private void button3_Click(object sender, EventArgs e)
{
    richTextBox1.Text = deck.printDeck();
}

static String saveToFile(int w)
{
    String winner = "";
    if (w == 1)
        winner = "player";
    else
        winner = "host";
    String now = "" + DateTime.Now;
    String message = now + " " + winner;

    String filename = "points.txt";
    FileStream fs;
    try

```

```

    {
        using (fs = new FileStream(filename, FileMode.Append,
            FileAccess.Write))
            using (StreamWriter sw = new StreamWriter(fs))
            {
                sw.WriteLine(message);
            }
    }
    catch (System.Exception ee)
    {
        return ("Error - check the folder!!!");
    }
    return (winner);
}

String path = "cards\\";
int points1 = 0;
int count1 = -1;
private void button5_Click(object sender, EventArgs e)
{
    count1++;
    Card taken = null;
    try
    {
        taken = deck.getFromTop();
    }
    catch (Exception ex)
    {
        label3.Text = "deck is empty!! " + ex.Message;
    }
    points1 += Convert.ToInt16(taken.returnValue());

    label3.Text = "" + points1;
    String picFile = path + taken.GetFileName();
    //deck.getThisCardFileName(0);
    label4.Text = picFile + " " + deck.deckSize();
    boxes[count1].Top = 260;
    boxes[count1].Left = 10 + count1 * 60;
    boxes[count1].Image = Image.FromFile(picFile);
}

```

```

        if (points1 >= 18 && points1 < 21)
            label5.Text = "Host's turn!";
        else if (points1 == 21)
        {
            label5.Text = saveToFile(1);
        }
        else if (points1 > 21)
            label5.Text = saveToFile(0);

        count2++;
    }
    int count2;
    int points2 = 0;
    int place = 0;
    private void button6_Click(object sender, EventArgs e)
    {
        place++;

        count2++;
        Card taken = deck.getFromTop();
        points2 += Convert.ToInt16(taken.returnValue());

        label6.Text = "" + points2;
        String picFile = path + taken.GetFileName();
        boxes[count2].Top = 400;
        boxes[count2].Left = 10 + place * 60;
        boxes[count2].Image = Image.FromFile(picFile);

        if (points2 >= points1 && points2 < 21)
            label6.Text = saveToFile(0);
        else if (points2 > 21)
            label6.Text = saveToFile(1);
    }
}

```

Thank you!

Feedback is welcome!