Московский государственный технический университет
им. Н.Э. Баумана

УТВЕРЖДАЮ:

Галкин В.А.
"__"_____2021 г.

Курсовая работа по курсу «Сетевые технологии в АСОИУ»
«Локальная безадаптерная сеть»

Описание программы
(вид документа)

печатная бумага
(вид носителя)

35
(количество листов)

ИСПОЛНИТЕЛИ:

студенты группы ИУ5-63Б
Назаров М.М. _____
Терентьев В.О. _____
Халимонов А.М. _____

"__"_____2021 г.

Москва – 2021

Оглавление

Класс MainWindow:Window

- класс, определяющий главное окно.

Переменные

chatWindow - переменная окна чата

string outcomePort - выбранный исходящий COM-порт

string incomePort - выбранный входящий COM-порт

string[] portNames - класс с COM-портами

string portName - переменная содержащая имя COм-порта

События

private void buttonConnection_Click(object sender, RoutedEventArgs e) - событие, возникающие при нажатие кнопки «Установить соединение»:

- *object sender* – объект, вызывающий событие;
- *RoutedEventArgs e* – аргументы для события;

private void comboBox_Initialized(object sender, EventArgs e) - событие, возникающее при первом взаимодействие с ним:

- *object sender* – объект, вызывающий событие;
- *RoutedEventArgs e* – аргументы для события;

private void comboBox1_Initialized(object sender, EventArgs e) - событие, возникающее при первом взаимодействие с ним:

- *object sender* – объект, вызывающий событие;
- *RoutedEventArgs e* – аргументы для события;

private void textBoxUserName_LostFocus(object sender, RoutedEventArgs e) - событие, происходящее при потере данным элементом логического фокуса:

- *object sender* – объект, вызывающий событие;
- *RoutedEventArgs e* – аргументы для события;

private void textBoxUserName_GotFocus(object sender, RoutedEventArgs e) - событие, происходящее при получении данным элементом логического фокуса:

- *object sender* – объект, вызывающий событие;
- *RoutedEventArgs e* – аргументы для события;

Методы и свойства

ResizeMode - свойство, указывающее, может ли быть изменен размер окна, и как.

D.IsChecked - свойство, которое Возвращает или задает значение, указывающее, находится ли ToggleButton во включенном состоянии.

comboBox.SelectedItem - свойство, которое возвращает или задает выделенный элемент в поле со списком ComboBox.

comboBox1.SelectedItem - свойство, которое возвращает или задает выделенный элемент в поле со списком ComboBox.

textBoxUserName.Text - свойство, которое возвращает или задает текстовое содержимое элемента управления TextBox.

chatWindow.Title - свойство, которое отвечает за получение или установку заголовка окна.

Application.Current.MainWindow.Hide() - метод, который делает окно невидимым.

chatWindow.Show() - метод, который открывает окно.

comboBox.Items.Add(portName) - метод, добавляющий элемент в ComboBox:

- portName - переменная COM-портами

Класс Chat : Window

- класс определяющий окно чата.

Переменные

static System.Windows.Controls.ListBox listBoxListOfUserToDisplay - переменная списка пользователей.

static Chat ths - переменная окна чат.

static bool checkExit = false - переменная проверки выхода.

static Dictionary<byte, string> dictionaryWithListOfUser - словарь со списком пользователей.

static Dictionary<string, ListBox> dictionaryWithListBox - словарь с ListBox’ами.

static Stack<ListBox> stackOfListBox - стек ListBox’ов.

static byte? thisUserAddress - переменная с адрессом данного пользователя.

static StackPanel staticVariableStackPanel - переменная типа StackPanel.

static Ellipse connectionStatus - переменная состояния статуса.

bool ckeck - переменная проверки наличия пользователя в списке пользователей.

string username - переменная содержащая имя пользователя.

string username1 - переменная содержащая имя пользователя для проверки правильности отображения.

byte? UserAddressToSendToDataLinkLayer - адрес пользователя, отправляющего сообщение.

string messageToSendToDataLinkLayer - сообщение отправляемое пользователем.

string userWithSpecialCharacter - имя пользователя с символом “*”.

string userWithoutSpecialCharacter - имя пользователя.

byte? UserAddressToSendToDataLinkLayer - адрес пользователя для отправки на канальный уровень.

События

private void button_Click(object sender, RoutedEventArgs e) - событие, возникающие при нажатие кнопки «Отправить»:

- *object sender* – объект, вызывающий событие;
- *RoutedEventArgs e* – аргументы для события;

private void textBox_LostFocus(object sender, RoutedEventArgs e) - событие, происходящее при потере данным элементом логического фокуса:

- *object sender* – объект, вызывающий событие;
- *RoutedEventArgs e* – аргументы для события;

private void textBox_KeyDown(object sender, KeyEventArgs e) - событие, происходящее при нажатие клавиши Enter:

- *object sender* – объект, вызывающий событие;
- *KeyEventArgs e* – аргументы для события;

private void Window_Closing(object sender, System.ComponentModel.CancelEventArgs e) - событие закрытия окна:

- *object sender* – объект, вызывающий событие;
- *System.ComponentModel.CancelEventArgs e* – аргументы для события;

private void textBox_GotFocus(object sender, RoutedEventArgs e) - событие, происходящее при получении данным элементом логического фокуса:

- *object sender* – объект, вызывающий событие;
- *RoutedEventArgs e* – аргументы для события;

private void listBox1_MouseDoubleClick(object sender, MouseButtonEventArgs e) - событие, которое вызывается при двойном щелчке мышью элемента управления.

- *object sender* – объект, вызывающий событие;
- *MouseButtonEventArgs e* – аргументы для события;

private void textBox_Initialized(object sender, EventArgs e) - событие, возникающее при загрузке окна:

- *object sender* – объект, вызывающий событие;
- *EventArgs e* – аргументы для события;

Методы и свойства

public static void List(Dictionary<byte, string> userLists, byte? userAddress) - функция отвечающая за добавление пользователей в список пользователей:

- Dictionary<byte, string> userLists – словарь со списком пользователей;
- byte? userAddress – адрес пользователя;

public static void inMessage(string message, byte userAddress, byte addressDeparture) - функция отображения входящего сообщений:

- string message – входящее сообщение для отображения;
- byte? userAddress – адрес пользователя отправившего сообщение;
- byte addressDeparture - адрес пользователя получающего сообщение.

public static void exit() - функция закрытия окна при разрывве соединения другим пользователем.

public static void connectionWait() - функция ожидания восстановления соединения.

public static void connectionRestored() - функция восстановления соединения.

dictionaryWithListOfUser.Keys - метод возвращения коллекции, содержащий ключи из словаря со списком пользователей.

listBoxListOfUserToDisplay.Items - метод возвращающий коллекция, используемую для созздания списка пользователей.

dictionaryWithListBox.Add(dictionaryWithListOfUser[b], stackOfListBox.Pop()) - метод добавление нового Listbox:

- dictionaryWithListOfUser[b] - элемент словаря со списком пользователей;
- stackOfListBox.Pop() - стек ListBox’ов.

listBoxListOfUserToDisplay.Items.Add(dictionaryWithListOfUser[b]) - метод добавления пользователя в список пользователей для отображения:

- dictionaryWithListOfUser[b] - элемент словаря со списком пользователей.

dictionaryWithListBox[username1].Items.Add(DateTime.Now.ToString("HH:mm") + " " + username + ": " + message) - функция вывода сообщения в чат:

- DateTime.Now.ToString("HH:mm") - метод выводящий время в часах и минутах;
- username - имя пользователя отправившего сообщение;
- message - полученное сообщение для отображения в чате..

listBoxListOfUserToDisplay.Items.Remove(lbl) - метод удаляющий пользователя из списка:

- lbl - имя пользователя для удаления из списка пользователей.

listBoxListOfUserToDisplay.Items.Insert(0, lbl + "*") - метод добавляющий пользователя на первую строчку в списке пользователей:

- lbl - имя пользователя для добавления в список пользователей.

textBox.Text - свойство, которое возвращает или задает текстовое содержимое элемента управления TextBox.

connectionStatus.Fill - метод, который изменяет цвет состояния статуса.

textBox.MaxLength - метод, который получает или задает максимальное число символов, которые могут быть вручную введены в текстовом поле.

listBoxListOfUserToDisplay.Items[0] - метод, добавляющий пользователяв список пользователей.

StackPanel.Children.Clear() - метод, удаляющий дочерний элемент из StackPanel.

dictionaryWithListBox[userWithSpecialCharacter].Height - метод, устанавливающий высоту элемента ListBox:

- userWithSpecialCharacter - имя пользователя с символом “*”.

StackPanel.Children.Add(dictionaryWithListBox[userWithSpecialCharacter]) - метод, добавляющий дочерний элемент в StackPanel:

- userWithSpecialCharacter - имя пользователя с символом “*”.

textBox.Text - свойство, которое возвращает или задает текстовое содержимое элемента управления TextBox.

ths.Close() - метод, закрывающий окно.

Application.Current.MainWindow.Show() - метод, показывающий главное окно.

e.Cancel - свойство, проверяющее подтверждение закрытия.

button.RaiseEvent - метод, вызывающий событие кнопки “Отправить”.

e.Key - свойство, проверяющий нажатие клавиши Enter

Класс Frame

– класс, описывающий кадр канального уровня.

Переменные

public enum Type : byte – перечисление типа *byte*, определяющее тип кадра (*Token, I, Link, Dis, ACK, Ret*).

public byte destination – переменная для адреса получателя.

public byte departure – переменная для адреса отправителя.

public Type? type – переменная для типа кадра.

public byte? data_length – переменная для длины поля данных кадра.

public byte[] data – массив байтов данных кадра.

Методы

public Frame(byte dep, Type type, byte? des = null, byte[] bytes = null) – конструктор класса Frame (кадра). Формирует кадр (объект класса Frame).

- *byte dep* – адрес отправителя;
- *Type type* – тип кадра;
- *byte? des* – адрес получателя;
- *byte[] bytes* – массив данных.

public Frame() – стандартный конструктор класса.

public bool TryConvertFromBytes(byte[] bytes) – метод, формирующий кадр из массива байтов. Возвращает значение True, при успешной попытке преобразовать массив байтов в кадр, или значение False, при неудачной попытке.

- *byte[] bytes* – массив байтов.

public static explicit operator byte[](Frame frame) – оператор преобразование кадра в массив байтов. Возвращает массив байтов.

- *Frame frame* – кадр для преобразования (объект класса Frame).

Класс DataLinkLayer

– класс, содержащий методы канального уровня.

Переменные

static int timeOut – переменная, содержащая значение тайм-аута.

static byte? userAddress – переменная для адреса получателя.

static string userNickname – переменная для текстового псевдонима пользователя.

static Queue<Frame> sendingFrames – очередь сообщений для отправки, ожидающих захвата маркера.

static bool flag – вспомогательная переменная (флаг) логического типа.

События

static AutoResetEvent waitACC – переменная события прихода кадра подтверждения успешной доставки сообщения.

static AutoResetEvent recdRet – переменная события прихода кадра на повторную отправку сообщения.

static AutoResetEvent Disc – переменная события прихода кадра разрыва соединения.

static AutoResetEvent Tkn – переменная события прихода маркера.

static AutoResetEvent Lnk – переменная события прихода кадра установки соединения.

Методы

static public void OpenConnection(string incomePortName, string outcomePortName, bool isMaster, string userName) – метод установки логического соединения.

- *string incomePortName* – имя входящего порта;
- *string outcomePortName* – имя исходящего порта;
- *bool isMaster* – является ли станция ведущей;
- *string userName* – текстовый псевдоним пользователя.

static public void SendFrameToConnection(byte[] bytes) – метод для отправки массива байтов на физический уровень.

- *byte[] bytes* – массив байтов.

static public void SendFrame(Frame frame) – метод для отправки кадра на канальном уровне.

- *Frame frame* – кадр для отправки.

static public void SendFramesToken() – метод для отправки кадров из очереди кадров при захвате маркера.

static public void SendMessage(byte? des, string mes) – метод для отправки информационного кадра (текстового сообщения).

- *byte? des* – адрес получателя;
- *string mes* – текстовое сообщение.

static public void CloseConnection() – метод для разъединения соединения на канальном уровне.

static public void HandleFrame(byte[] bytes) – метод обработки пришедшего массива байтов (кадра).

- *byte[] bytes* – пришедший массив байтов.

Класс CyclicCode

– класс, который содержит методы для кодирования, декодирования и обнаружения ошибок с использованием циклического кода.

Переменные

int g_vect – образующий вектор, используемый при кодировании и декодировании.

Методы и свойства

byte[] Coding() – метод для кодирования массива байтов с помощью циклического кода:

- *byte[] inputVect* – массив байтов для кодирования;

- *byte[] res* – закодированный массив, используемый для отправки в COM-порт;
- *int left_vect* – кодовый вектор, составленный из первых четырех разрядов байта;
- *int right_vect* – кодовый вектор, составленный из последних четырех разрядов байта.

(byte[], bool) Decoding() – метод для декодирования массива байтов и обнаружения ошибок с помощью циклического кода:

- *byte[] inputVect* – массив байтов для декодирования;
- *byte[] res* – декодированный массив, используемый для отправки на канальный уровень;
- bool hasError – переменная, хранящая информацию о наличии ошибок.

int CyclicCoding() – метод для кодирования кодового вектора циклическим кодом:

- *int info_vect* – вектор для кодирования;
- *int coded_vect* – закодированный вектор.

bool ErrorCheck () – метод обнаружения ошибок в кодовом векторе:

- *int coded_vect* – закодированный вектор для проверки.

int division() – метод, возвращающий остаток от деления кодового вектора на образующий.

Класс Connection

– класс, содержащий методы физического уровня.

Переменные

AutoResetEvent FrameIsRead – событие для синхронизации потоков, срабатывающее после чтения данных из COM-порта.

SerialPort incomePort – объект класса SerialPort, описывающий входящий COM-порт

SerialPort outcomePort – объект класса SerialPort, описывающий входящий COM-порт

bool isMaster – переменная, определяющая является ли текущая станция ведущей

События

static void RecieveBytes() – событие, возникающее при получении данных через входящий COM-порт:

- *object sender* – объект, вызывающий событие;
- *RoutedEventArgs e* – аргументы для события;
- *Thread myThread* – объект класса Thread, используемый для запуска нового потока.

Методы и свойства

string[] GetPortsNames() – метод, возвращающий список доступных COM-портов.

bool OpenPorts() – метод, используемый для установки параметров и открытия портов. Возвращает значение True, при успешном открытии:

- *string incomePortName* – имя входящего COM-порта;
- *string outcomePortName* – имя исходящего COM-порта;
- *bool _isMaster* – переменная, задающая значение переменной *isMaster*;
- *incomePort.Parity* – свойство входящего COM-порта, задающее тип проверочного бита;
- *incomePort.BaudRate* – свойство входящего COM-порта, задающее скорость передачи;
- *incomePort.StopBits* – свойство входящего COM-порта, задающее число стоповых битов;
- *incomePort.Handshake* – свойство входящего COM-порта, задающее протокол установления связи;
- *incomePort.ReadBufferSize* – свойство входящего COM-порта, задающее размер буфера чтения;
- *incomePort.ReceivedBytesThreshold* – свойство входящего COM-порта, задающее минимальное число байтов в буфере для чтения;
- *incomePort.ReadTimeout* – свойство входящего COM-порта, задающее максимальное время чтения.
- *outcomePort.Parity* – свойство исходящего COM-порта, задающее тип проверочного бита;

- *outcomePort.BaudRate* – свойство исходящего COM-порта, задающее скорость передачи;
- *outcomePort.StopBits* – свойство исходящего COM-порта, задающее число стоповых битов;
- *outcomePort.Handshake* – свойство исходящего COM-порта, задающее протокол установления связи;
- *outcomePort.WriteBufferSize* – свойство исходящего COM-порта, задающее размер буфера записи;
- *outcomePort.WriteTimeout* – свойство исходящего COM-порта, задающее максимальное время записи.

bool ClosePorts() – метод, используемый для закрытия портов. Возвращает значение True, при успешном закрытии.

void SendBytes() – метод, используемый для отправки кадра, через исходящий COM-порт:

- *byte[] outputVect* – отправляемый кадр;
- *byte[] codedVect* – отправляемый кадр после циклического кодирования.

void ReadBytes() – метод для чтения кадра из буфера COM-порта и отправки его на канальный уровень:

- byte[] inputVect – массив байтов, используемый для записи из COM-порта;
- int bytes – число байтов в буфере приёма COM-порта.

Листинг

Файл MainWindow.xaml

```
<Window x:Class="ChatTokenRing.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:ChatTokenRing"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800">
    <Grid>
        <CheckBox x:Name="D" Content="Ведущая станция" HorizontalAlignment="Left"
Margin="114,276,0,0" VerticalAlignment="Top" Height="30" Width="160"/>
        <Button x:Name="buttonConnection" Content="Установить соединение"
HorizontalAlignment="Left" Height="30" Margin="304,276,0,0" VerticalAlignment="Top"
Width="376" Click="buttonConnection_Click"/>
        <Label x:Name="label" Content="Исходящий COM-порт" HorizontalAlignment="Left"
Height="30" Margin="114,92,0,0" VerticalAlignment="Top" Width="160"/>
```

```
        <Label x:Name="label_Copy" Content="Входящий COM-порт" HorizontalAlignment="Left"
Height="30" Margin="114,153,0,0" VerticalAlignment="Top" Width="160"/>
        <Label x:Name="label_Copy1" Content="Имя пользователя" HorizontalAlignment="Left"
Height="30" Margin="114,217,0,0" VerticalAlignment="Top" Width="160"/>
        <TextBox x:Name="textBoxUserName" Text="Введите имя пользователя"
LostFocus="textBoxUserName_LostFocus"  HorizontalAlignment="Left" Height="30"
Margin="304,217,0,0" VerticalAlignment="Top" Width="376"
GotFocus="textBoxUserName_GotFocus"/>
        <ComboBox x:Name="comboBox" HorizontalAlignment="Left" Height="30"
Margin="304,92,0,0" VerticalAlignment="Top" Width="376"
Initialized="comboBox_Initialized"/>
        <ComboBox x:Name="comboBox1" HorizontalAlignment="Left" Height="30"
Margin="304,153,0,0" VerticalAlignment="Top" Width="376"
Initialized="comboBox1_Initialized"/>
    </Grid>
</Window>
```

Файл MainWindow.xaml.cs

```
using System;
using System.Windows;

namespace ChatTokenRing
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>


    public partial class MainWindow : Window
    {
        public Chat chatWindow;

        public MainWindow()
        {
            InitializeComponent();
            this.ResizeMode = System.Windows.ResizeMode.CanMinimize;
        }

        private void buttonConnection_Click(object sender, RoutedEventArgs e)
        {
            if ((comboBox.SelectedItem != null) && (comboBox1.SelectedItem != null) &&
(textBoxUserName.Text != "") && (textBoxUserName.Text != "Введите имя пользователя"))
            {
                if (comboBox.SelectedItem.ToString() ==
comboBox1.SelectedItem.ToString())
                {
                    MessageBox.Show("Выберите различные COM-порты и повторите попытку",
"Ошибка соединения", MessageBoxButton.OK);
                    return;
                }
                else
                {
                    string outcomePort = comboBox.SelectedItem.ToString();
                    string incomePort = comboBox1.SelectedItem.ToString();
                    chatWindow = new Chat();
                    if (D.IsChecked == true)
                    {
                        chatWindow.Title = "Чат (Вы вошли как: " + textBoxUserName.Text +
") Ведущая станция";
                    }
                    else
```

```
                    {
                        chatWindow.Title = "Чат (Вы вошли как: " + textBoxUserName.Text +
")";
                    }
                    chatWindow.ResizeMode = System.Windows.ResizeMode.CanMinimize;
                    DataLinkLayer.OpenConnection(incomePort, outcomePort,
(bool)D.IsChecked, textBoxUserName.Text.Trim(new char[] { '*' }));
                    Application.Current.MainWindow.Hide();
                    chatWindow.Show();
                }
            }
            else
            {
                if ((comboBox.SelectedItem == null) || (comboBox1.SelectedItem == null))
                {
                    MessageBox.Show("Выберите COM-порты и повторите попытку", "Ошибка
соединения", MessageBoxButton.OK);
                    return;
                }
                else
                {
                    MessageBox.Show("Введите имя пользователя и повторите попытку",
"Ошибка соединения", MessageBoxButton.OK);
                    return;
                }
            }
        }

        private void comboBox_Initialized(object sender, EventArgs e)
        {
            string[] portNames = Connection.GetPortsNames();
            foreach (string portName in portNames)
            {
                comboBox.Items.Add(portName);
            }
        }

        private void comboBox1_Initialized(object sender, EventArgs e)
        {
            string[] portNames = Connection.GetPortsNames();
            foreach (string portName in portNames)
            {
                comboBox1.Items.Add(portName);
            }
        }

        private void textBoxUserName_LostFocus(object sender, RoutedEventArgs e)
        {
            if (textBoxUserName.Text == "")
            {
                textBoxUserName.Text = "Введите имя пользователя";
            }
        }

        private void textBoxUserName_GotFocus(object sender, RoutedEventArgs e)
        {
            if (textBoxUserName.Text == "Введите имя пользователя")
            {
                textBoxUserName.Text = "";
            }
        }
    }
}
```

Файл Chat.xaml

```
<Window x:Class="ChatTokenRing.Chat"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:ChatTokenRing"
        mc:Ignorable="d"
        Title="Chat" Height="450" Width="800" Closing="Window_Closing">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>
        <TextBox x:Name="textBox" Text="Введите сообщение" LostFocus="textBox_LostFocus"
HorizontalAlignment="Left" Height="32" Margin="105,316,0,0" VerticalAlignment="Top"
Width="429" KeyDown="textBox_KeyDown" GotFocus="textBox_GotFocus"
Initialized="textBox_Initialized"/>
        <Button x:Name="button" Content="Отправка" HorizontalAlignment="Left"
Margin="572,316,0,0" VerticalAlignment="Top" Width="117" Height="32"
Click="button_Click"/>
        <ListBox x:Name="listBox1" HorizontalAlignment="Left" Height="220"
Margin="10,64,0,0" VerticalAlignment="Top" Width="72"
MouseDoubleClick="listBox1_MouseDoubleClick" Initialized="listBox1_Initialized"/>
        <StackPanel x:Name="StackPanel" HorizontalAlignment="Left" Height="224"
Margin="105,64,0,0" VerticalAlignment="Top" Width="584"
Initialized="StackPanel_Initialized">
        </StackPanel>
        <Ellipse Fill="Green" x:Name="ellipse" HorizontalAlignment="Left" Height="12"
Margin="39,328,0,0" Stroke="Black" VerticalAlignment="Top" Width="12"
RenderTransformOrigin="0.938,1.788" ToolTip="Статус соединения"/>
        <Label x:Name="label" Content="Все чаты:" HorizontalAlignment="Left"
Margin="10,33,0,0" VerticalAlignment="Top" Height="26" Width="72"/>
    </Grid>
</Window>
```

Файл Chat.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Shapes;

namespace ChatTokenRing
{
    /// <summary>
    /// Логика взаимодействия для Chat.xaml
    /// </summary>
    public partial class Chat : Window
    {
        static System.Windows.Controls.ListBox listBoxListOfUserToDisplay;
        static Chat ths;
        static bool checkExit = false;
        static Dictionary<byte, string> dictionaryWithListOfUser;
        static Dictionary<string, ListBox> dictionaryWithListBox;
        static Stack<ListBox> stackOfListBox;
        static byte? thisUserAddress;
        static StackPanel staticVariableStackPanel;
        static Ellipse connectionStatus;
```

```
        public Chat()
        {
            dictionaryWithListBox = new Dictionary<string, ListBox>();
            dictionaryWithListBox.Add("Общий", new ListBox());
            dictionaryWithListBox["Общий"].Height = 220;
            InitializeComponent();
            listBox1.Items.Add("Общий");
            listBox1.SelectedItem = listBox1.Items[0];
            StackPanel.Children.Add(dictionaryWithListBox["Общий"]);
            staticVariableStackPanel = StackPanel;
            listBoxListOfUserToDisplay = listBox1;
            ths = this;
            connectionStatus = ellipse;
            dictionaryWithListOfUser = new Dictionary<byte, string>();
            stackOfListBox = new Stack<ListBox>();
            for (int i = 0; i < 100; i++)
            {
                stackOfListBox.Push(new ListBox());
            }
        }

        public static void List(Dictionary<byte, string> userLists, byte? userAddress)
        {
            thisUserAddress = userAddress;
            dictionaryWithListOfUser = userLists;
            bool ckeck = false;
            foreach (byte b in dictionaryWithListOfUser.Keys)
            {
                ckeck = false;
                foreach (var item in listBoxListOfUserToDisplay.Items)
                {
                    if (item.ToString().TrimEnd(new char[] { '*' }) ==
dictionaryWithListOfUser[b])
                    {
                        ckeck = true;
                        break;
                    }
                }
                if (!ckeck &&
!listBoxListOfUserToDisplay.Items[0].ToString().Contains("*"))
                {
                    dictionaryWithListBox.Add(dictionaryWithListOfUser[b],
stackOfListBox.Pop());
                    listBoxListOfUserToDisplay.Dispatcher.Invoke(() =>
                    {

listBoxListOfUserToDisplay.Items.Add(dictionaryWithListOfUser[b]);
                    });
                    break;
                }
            }
        }

        public static void inMessage(string message, byte userAdress, byte
addressKydanado)
        {
            string username = "";
            string username1 = "";
            foreach (byte b in dictionaryWithListOfUser.Keys)
            {
                if (b == userAdress)
                {
                    username = dictionaryWithListOfUser[b];
                    break;
                }
```

```
            }
            if (addressKydanado == 0x7F)
            {
                username1 = "Общий";
            }
            else
            {
                username1 = username;
            }
            dictionaryWithListBox[username1].Dispatcher.Invoke(() =>
            {
                if (dictionaryWithListBox[username1] !=
staticVariableStackPanel.Children[0])
                {
                    foreach (var lbl in listBoxListOfUserToDisplay.Items)
                    {

                        if (lbl.ToString() == username1)
                        {
                            listBoxListOfUserToDisplay.Items.Remove(lbl);
                            listBoxListOfUserToDisplay.Items.Insert(0, lbl + "*");
                            break;
                        }
                    }
                }
            });
            dictionaryWithListBox[username1].Dispatcher.Invoke(() => {
dictionaryWithListBox[username1].Items.Add(DateTime.Now.ToString("HH:mm") + " " +
username + ": " + message); });
        }

        private void button_Click(object sender, RoutedEventArgs e)
        {
            byte? UserAddressToSendToDataLinkLayer = null;
            string messageToSendToDataLinkLayer = textBox.Text;
            foreach (var variableToIteratedictionaryWithListBox in dictionaryWithListBox)
            {
                if (variableToIteratedictionaryWithListBox.Value ==
StackPanel.Children[0])
                {
                    if (variableToIteratedictionaryWithListBox.Key == "Общий")
                    {
                        UserAddressToSendToDataLinkLayer = 0x7F;
                    }
                    else
                    {
                        foreach (var variableToIteratedictionaryWithListOfUser in
dictionaryWithListOfUser)
                        {
                            if (variableToIteratedictionaryWithListBox.Key ==
variableToIteratedictionaryWithListOfUser.Value)
                            {
                                UserAddressToSendToDataLinkLayer =
variableToIteratedictionaryWithListOfUser.Key;
                                break;
                            }
                        }
                    }
                    if (UserAddressToSendToDataLinkLayer != 0x7F &&
UserAddressToSendToDataLinkLayer != thisUserAddress)
                    {

variableToIteratedictionaryWithListBox.Value.Items.Add(DateTime.Now.ToString("HH:mm") + "
Вы: " + messageToSendToDataLinkLayer);
                    }
```

```
                    break;
                }
            }
            DataLinkLayer.SendMessage(UserAddressToSendToDataLinkLayer,
messageToSendToDataLinkLayer);
            textBox.Clear();
        }

        private void textBox_LostFocus(object sender, RoutedEventArgs e)
        {
            if (textBox.Text == "")
            {
                textBox.Text = "Введите сообщение";
            }
        }

        private void textBox_KeyDown(object sender, KeyEventArgs e)
        {
            if (e.Key == Key.Enter)
            {
                button.RaiseEvent(new
RoutedEventArgs(System.Windows.Controls.Primitives.ButtonBase.ClickEvent));
            }
        }

        private void Window_Closing(object sender, System.ComponentModel.CancelEventArgs
e)
        {
            if (checkExit)
            {
                Application.Current.MainWindow.Show();
                checkExit = false;
            }
            else
            {
                if (MessageBox.Show("Вы уверены, что хотите закрыть окно?",
"Подтверждение закрытия", MessageBoxButton.YesNo) == MessageBoxResult.No)
                {
                    e.Cancel = true;
                }
                else
                {
                    e.Cancel = false;
                    DataLinkLayer.CloseConnection();
                    Application.Current.MainWindow.Show();
                }
            }
        }

        public static void exit()
        {
            connectionStatus.Dispatcher.Invoke(() => { connectionStatus.Fill =
Brushes.Red; });
            if (MessageBox.Show("Другой пользователь вышел из программы, разрыв
соединения", "Разрыв соединения", MessageBoxButton.OK) == MessageBoxResult.OK)
            {
                ths.Dispatcher.Invoke(() => { checkExit = true; ths.Close(); });
            }
        }

        private void textBox_GotFocus(object sender, RoutedEventArgs e)
        {
            if (textBox.Text == "Введите сообщение")
            {
                textBox.Text = "";
```

```
            }
        }

        private void listBox1_MouseDoubleClick(object sender, MouseButtonEventArgs e)
        {
            string userWithSpecialCharacter =
listBoxListOfUserToDisplay.SelectedItem.ToString().Trim(new char[] { '*' });
            string userWithoutSpecialCharacter =
listBoxListOfUserToDisplay.SelectedItem.ToString();
            StackPanel.Children.Clear();
            dictionaryWithListBox[userWithSpecialCharacter].Height = 220;
            StackPanel.Children.Add(dictionaryWithListBox[userWithSpecialCharacter]);
            listBoxListOfUserToDisplay.Items.Remove(userWithoutSpecialCharacter);
            listBoxListOfUserToDisplay.Items.Insert(0, userWithSpecialCharacter);
            listBoxListOfUserToDisplay.SelectedItem =
listBoxListOfUserToDisplay.Items[0];
        }

        private void textBox_Initialized(object sender, EventArgs e)
        {
            textBox.MaxLength = 127;
        }

        public static void connectionWait()
        {
            connectionStatus.Dispatcher.Invoke(() => { connectionStatus.Fill =
Brushes.Red; });
        }

        public static void connectionRestored()
        {
            connectionStatus.Dispatcher.Invoke(() => { connectionStatus.Fill =
Brushes.Green; });
        }
    }
}
```

Файл DataLinkLayer.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace ChatTokenRing
{
    /// <summary>
    /// Класс описывающий кадр
    /// </summary>
    class Frame
    {
        /// <summary>
        /// Тип кадра
        /// </summary>
        public enum Type : byte
        {
            Token,  // Кадр-маркер в направленном маркерном кольце
            I,      // Информационный кадр
            Link,   // Кадр установки соединения
            Dis,    // Кадр разрыва соединения
            ACK,    // Кадр подтверждения безошибочного приема кадра
            Ret,    // Кадр запроса повторения последнего отправленного кадра
        }
```

```
        public byte destination;            // Адрес получателя
        public byte departure;              // Адрес отправителя
        public Type? type = null;           // Тип кадра
        public byte? data_length = null;    // Длина поля данных
        public byte[] data = null;          // Данные

        /// <summary>
        /// Конструктор кадра
        /// </summary>
        public Frame(byte dep, Type type, byte? des = null, byte[] bytes = null)
        {
            departure = dep;
            this.type = type;
            switch (this.type)
            {
                case Type.Token:
                    if (des == null)
                    {
                        destination = 0x7F; // Адрес получателя широковещательный
                    }
                    else
                    {
                        if (des != 0x7F)
                        {
                            throw new Exception("Ошибка: адрес не широковещательный");
                        }
                        else
                        {
                            destination = (byte)des;
                        }
                    }
                    if (bytes != null)
                    {
                        throw new Exception("Ошибка: есть какие-то данные");
                    }
                    break;

                case Type.I:
                    if (des == null)
                    {
                        throw new Exception("Ошибка: нет адреса получателя");
                    }
                    else
                    {
                        destination = (byte)des;

                        if (bytes == null)
                        {
                            throw new Exception("Ошибка: нет данных");
                        }
                        else
                        {
                            if (bytes.Length > 255)
                            {
                                throw new Exception("Ошибка: данные не помещаются в
кадр");

                            }
                            else
                            {
                                data_length = (byte)bytes.Length;
                                data = bytes;
                            }
                        }
                    }
```

```
            break;

        case Type.Link:
            if (des == null)
            {
                destination = 0x7F; // Адрес получателя широковещательный
            }
            else
            {
                if (des != 0x7F)
                {
                    throw new Exception("Ошибка: адрес не широковещательный");
                }
                else
                {
                    destination = (byte)des;
                }
            }
            if (bytes == null)
            {
                throw new Exception("Ошибка: нет данных");
            }
            else
            {
                if (bytes.Length > 255)
                {
                    throw new Exception("Ошибка: данные не помещаются в кадр");
                }
                else
                {
                    data_length = (byte)bytes.Length;
                    data = bytes;
                }
            }
            break;

        case Type.Dis:
            if (des == null)
            {
                destination = 0x7F; // Адрес получателя широковещательный
            }
            else
            {
                if (des != 0x7F)
                {
                    throw new Exception("Ошибка: адрес не широковещательный");
                }
                else
                {
                    destination = (byte)des;
                }
            }
            if (bytes != null)
            {
                throw new Exception("Ошибка: есть какие-то данные");
            }
            break;

        case Type.ACK:
            if (des == null)
            {
                throw new Exception("Ошибка: нет адреса получателя");
            }
            else
            {
```

```
                destination = (byte)des;

                if (bytes != null)
                {
                    throw new Exception("Ошибка: есть какие-то данные");
                }
            }
            break;

        case Type.Ret:
            if (des == null)
            {
                throw new Exception("Ошибка: нет адреса получателя");
            }
            else
            {
                destination = (byte)des;

                if (bytes != null)
                {
                    throw new Exception("Ошибка: есть какие-то данные");
                }
            }
            break;

        default:
            throw new Exception("Ошибка: неверный тип кадра");
            break;
    }
}

public Frame() { }

/// <summary>
/// Преобразование массива байтов в кадр (true - успешно, false - не удалось)
/// </summary>
public bool TryConvertFromBytes(byte[] bytes)
{
    if (bytes[0] != 0xFF) // Проверка на стартовый байт
    {
        // Ошибка: неверный стартовый байт
        return false;
    }
    else
    {
        destination = bytes[1];
        departure = bytes[2];

        if (bytes[3] > 5)
        {
            // Ошибка: недопустимый тип кадра
            return false;
        }
        type = (Type)bytes[3];

        byte i = 0;
        if ((type == Type.I) || (type == Type.Link))
        {
            data_length = bytes[4];

            if (data_length > bytes.Length - 6)
            {
                // Ошибка: длина данных больше чем массив байтов
                return false;
            }
```

```
                    data = new byte[(byte)data_length];
                    for (i = 0; i < data_length; ++i)
                    {
                        data[i] = bytes[i + 5];
                    }
                    ++i;
                }

                if (bytes[i + 4] != 0xFF) // Проверка на стоповый байт
                {
                    // Ошибка: неверный стоповый байт
                    return false;
                }
                return true;
            }
        }

        /// <summary>
        /// Преобразование кадра в массив байтов
        /// </summary>
        public static explicit operator byte[](Frame frame)
        {
            byte[] bytes;
            if (frame.data == null)
            {
                bytes = new byte[5];
                bytes[0] = 0xFF; // Стартовый байт
                bytes[1] = frame.destination;
                bytes[2] = frame.departure;
                bytes[3] = (byte)frame.type;
                bytes[4] = 0xFF; // Стоповый байт
            }
            else
            {
                bytes = new byte[5 + (byte)frame.data_length + 1];
                bytes[0] = 0xFF; // Стартовый байт
                bytes[1] = frame.destination;
                bytes[2] = frame.departure;
                bytes[3] = (byte)frame.type;
                bytes[4] = (byte)frame.data_length;
                for (byte i = 0; i < bytes[4]; ++i)
                {
                    bytes[i + 5] = (byte)frame.data[i];
                }
                bytes[bytes[4] + 5] = 0xFF; // Стоповый байт
            }
            return bytes;
        }
    }
abstract class DataLinkLayer
    {
        static int timeOut = 2000; // Начальный тайм-аут
        static byte? userAddress = null; // Адрес пользователя
        static string userNickname; // Никнейм пользователя
        static Queue<Frame> sendingFrames = new Queue<Frame>(); // Буфер ожидающих к
отправлению сообщений (ждущих маркер)
        static AutoResetEvent waitACC = new AutoResetEvent(false); // Событие прихода
кадра подтверждения успешной доставки сообщения
        static AutoResetEvent recdRet = new AutoResetEvent(false); // Событие прихода
кадра на повторную отправку сообщения
        static AutoResetEvent Disc = new AutoResetEvent(false); // Событие прихода кадра
разрыва соединения
        static AutoResetEvent Tkn = new AutoResetEvent(false); // Событие прихода маркера
```

```
        static AutoResetEvent Lnk = new AutoResetEvent(false); // Событие прихода кадра
установки соединения
        static bool flag = false;

        /// <summary>
        /// Установка логического соединения
        /// </summary>
        static public void OpenConnection(string incomePortName, string outcomePortName,
bool isMaster, string userName)
        {
            userAddress = null; // Обнуление статических переменных
            userNickname = userName; // Получение никнейма с пользовательского уровня
            sendingFrames = new Queue<Frame>(); // Обнуление статических переменных
            waitACC = new AutoResetEvent(false);
            recdRet = new AutoResetEvent(false);
            Disc = new AutoResetEvent(false);
            Tkn = new AutoResetEvent(false);
            Lnk = new AutoResetEvent(false);
            timeOut = 2000; // Задание статических переменных
            flag = false;

            Connection.OpenPorts(incomePortName, outcomePortName, isMaster); // Установка
физического соединения
            if (isMaster) // Если станция ведущая
            {
                userAddress = 1;
                SendFrame(new Frame((byte)userAddress, Frame.Type.Link, bytes:
Encoding.UTF8.GetBytes("[1, " + userNickname + ']'))); // Отправка Link кадра
                SendFrame(new Frame((byte)userAddress, Frame.Type.Token)); // Отправка
маркера
                SendFramesToken();
            }
            else
            {
                Chat.connectionWait(); // Вывод сообщения о установлении соединения...
                Lnk.WaitOne();
                Chat.connectionRestored(); // Вывод сообщения что соединение установлено
            }
        }

        /// <summary>
        /// Отправка кадра на физический уровень
        /// </summary>
        static public void SendFrameToConnection(byte[] bytes)
        {
            if (!Disc.WaitOne(1)) // Если кадр разрыва соединения не приходил
            {
                Connection.SendBytes(bytes);
            }
            else
            {
                Disc.Set();
            }
        }

        /// <summary>
        /// Отправка кадра
        /// </summary>
        static public void SendFrame(Frame frame)
        {
            if ((frame.type == Frame.Type.ACK) || (frame.type == Frame.Type.Ret) ||
(frame.type == Frame.Type.Dis)) // Служебные кадры передаются без владения маркера
            {
                SendFrameToConnection((byte[])frame);
            }
```

```
            else
            {
                sendingFrames.Enqueue(frame); // Сохранение кадра для отправления когда
будет захвачен маркер
            }
        }

        /// <summary>
        /// Отправка кадров при наличии маркера
        /// </summary>
        static public void SendFramesToken()
        {
            Frame tmp;
            do
            {
                tmp = sendingFrames.Dequeue();
                if ((tmp.destination != 0x7F) || (tmp.type == Frame.Type.Link) ||
(tmp.type == Frame.Type.Token)) // Для кадров с подтверждением успешной доставки
                {
                    waitACC.Reset();
                    recdRet.Reset();
                    SendFrameToConnection((byte[])tmp); // Отправка кадра на физический
уровень
                    bool flg = false;
                    while (!(AutoResetEvent.WaitAny(new WaitHandle[] { waitACC, recdRet
}, timeOut) == 0)) // Ожидание кадра успешной доставки или запроса на повторную отправку
в случае ошибки
                    {
                        if (!Disc.WaitOne(1))
                        {
                            if (!flg)
                            {
                                flg = true;
                                Chat.connectionWait(); // Соединение потеряно...
Восстановление соединения...
                            }
                            SendFrameToConnection((byte[])tmp); // Повторная отправка
кадра
                        }
                        else
                        {
                            Disc.Set();
                            return;
                        }
                    }
                    if (flg)
                    {
                        Chat.connectionRestored(); // Соединение восстановлено
                    }
                }
                else // Для кадров без подтверждения успешной доставки
                {
                    SendFrameToConnection((byte[])tmp); // Отправка кадра на физический
уровень
                }
            } while (tmp.type != Frame.Type.Token); // Выполнять пока маркер не отдан
        }

        /// <summary>
        /// Отправка сообщения с пользовательского уровня
        /// </summary>
        static public void SendMessage(byte? des, string mes)
        {
            SendFrame(new Frame((byte)userAddress, Frame.Type.I, des,
Encoding.UTF8.GetBytes(mes)));
```

```
        }

        /// <summary>
        /// Разъединение логического соединения
        /// </summary>
        static public void CloseConnection()
        {
            SendFrame(new Frame((byte)userAddress, Frame.Type.Dis)); // Отправка кадра
разрыва соединения
            Disc.Set(); // Событие разрыва соединения
            Connection.ClosePorts(); // Разрыв физического соединения
        }

        /// <summary>
        /// Обработка пришедшего кадра
        /// </summary>
        static public void HandleFrame(byte[] bytes)
        {
            (byte[], bool) decoded = CyclicCode.Decoding(bytes); // Декодирование
пришедших байтов циклическим кодом
            if (decoded.Item1.Length > 4) // Если пришел пакет а не какие-нибудь помехи
(минимальный размер пакета в кольце)
            {
                if ((decoded.Item1[0] == 0xFF) && (decoded.Item1[decoded.Item1.Length -
1] == 0xFF)) // Если пришел пакет а не какие-нибудь помехи (проверка по стартовому и
стоповому байту)
                {
                    Frame frame = new Frame();
                    if (decoded.Item2 || (!frame.TryConvertFromBytes(decoded.Item1))) //
Если при декодировании циклическим кодом была выявлена ошибка или не удалась попытка
восстановить кадр из массива байтов, то отправляем запрос на повторную отправку
                    {
                        if (frame.type != null)
                        {
                            if (!((frame.type == Frame.Type.ACK) || (frame.type ==
Frame.Type.Ret) || (frame.type == Frame.Type.Dis)))
                            {
                                SendFrame(new Frame((byte)userAddress, Frame.Type.Ret,
des: frame.departure)); // Запрос на повторную отправку
                            }
                        }
                    }
                    else
                    {
                        switch (frame.type)
                        {
                            case Frame.Type.Token: // Обработка маркера
                                Tkn.Set(); // Событие прихода маркера

                                SendFrame(new Frame((byte)userAddress, Frame.Type.ACK,
des: frame.departure)); // Отправка кадра подтверждения безошибочного приема кадра

                                frame.departure = (byte)userAddress;
                                SendFrame(frame);
                                SendFramesToken(); // Отправка сообщений с захваченным
маркером

                                // Ожидание возвращения маркера (проверка целостности
соединения)
                                bool flg = false;
                                while (!Tkn.WaitOne(timeOut))
                                {
                                    if (!Disc.WaitOne(1))
                                    {
                                        if (!flg)
```

```
                                    {
                                        flg = true;
                                        Chat.connectionWait(); // Соединение
потеряно... Восстановление соединения...
                                    }
                                }
                                else
                                {
                                    Disc.Set();
                                    return;
                                }
                            }
                            if (flg)
                            {
                                Chat.connectionRestored(); // Соединение
восстановлено
                            }
                            break;

                        case Frame.Type.I: // Обработка информационного кадра
                            if ((frame.destination == 0x7F) || (frame.destination ==
(byte)userAddress)) // Если кадр предназначен этой станции
                            {
                                if (frame.destination != 0x7F)
                                {
                                    if (frame.departure == (byte)userAddress)
                                    {
                                        waitACC.Set(); // Нет смысла посылать кадр об
успешной доставке самому себе
                                    }
                                    else
                                    {
                                        SendFrame(new Frame((byte)userAddress,
Frame.Type.ACK, des: frame.departure)); // Отправка кадра подтверждения безошибочного
приема кадра
                                    }
                                }

                                Chat.inMessage(Encoding.UTF8.GetString(frame.data, 0,
frame.data.Length), frame.departure, frame.destination); // Передача сообщения на
пользовательский уровень

                                if ((frame.destination == 0x7F) && (frame.departure
!= userAddress))
                                {
                                    SendFrame(frame);
                                }
                            }
                            else
                            {
                                SendFrameToConnection((byte[])frame);
                            }
                            break;

                        case Frame.Type.Link: // Обработка кадра установки соединения
                            Lnk.Set(); // Событие прихода кадра установки соединения

                            Dictionary<byte, string> users = new Dictionary<byte,
string>(); // Словарь пользователей

                            try // Если нет ошибок при обработке списка
                            {
                                string[] items = Encoding.UTF8.GetString(frame.data,
0, frame.data.Length).Split(new string[] { "][" },
StringSplitOptions.RemoveEmptyEntries);
```

```
                                    foreach (string item in items)
                                    {
                                        string[] tmp = item.Trim('[', ']').Split(new
string[] { ", " }, StringSplitOptions.RemoveEmptyEntries);
                                        users.Add(Convert.ToByte(tmp[0]), tmp[1]);
                                    }
                                }
                                catch // Если произошла ошибка - запрос на повторную
отправку
                                {
                                    SendFrame(new Frame((byte)userAddress,
Frame.Type.Ret, des: frame.departure)); // Запрос на повторную отправку
                                    return;
                                }

                                if (userAddress == null)
                                {
                                    userAddress = (byte?)(users.Last().Key + 1);
                                }

                                // Если пользователь с таким именем уже есть
                                bool flgg;
                                if (!users.ContainsKey((byte)userAddress))
                                {
                                    do
                                    {
                                        flgg = false;
                                        foreach (var us in users)
                                        {
                                            if (us.Value == userNickname)
                                            {
                                                userNickname += " (1)";
                                                flgg = true;
                                                break;
                                            }
                                        }
                                    } while (flgg);

                                    users.Add((byte)userAddress, userNickname);
                                    frame.data = Encoding.UTF8.GetBytes(string.Join(null,
users));
                                    frame.data_length = (byte?)frame.data.Length;
                                }

                                Chat.List(users, userAddress); // Передача списка
пользователей на пользовательский уровень

                                timeOut = (int)(users.Count * 1.5 * 1000); // Значение
тайм-аута зависит от количества пользователей
                                SendFrame(new Frame((byte)userAddress, Frame.Type.ACK,
des: frame.departure)); // Отправка кадра подтверждения безошибочного приема кадра
                                frame.departure = (byte)userAddress;
                                if (userAddress != 1)
                                {
                                    SendFrame(frame);
                                }
                                else // Отправка заполненного списка пользователей всем
станциям
                                {
                                    if (!flag)
                                    {
                                        flag = true;
                                        SendFrame(frame);
                                    }
                                }
```

```
                                break;

                            case Frame.Type.Dis: // Обработка кадра разрыва соединения
                                SendFrameToConnection((byte[])frame); // Отправка кадра
разрыва соединения
                                Disc.Set(); // Событие прихода кадра разрыва соединения

                                Connection.ClosePorts(); // Разрыв соединения на
физическом уровне
                                Chat.exit(); // Выход из приложения на пользовательском
уровне
                                break;

                            case Frame.Type.ACK: // Обработка кадра подтверждения
безошибочного приема кадра
                                if (userAddress != null)
                                {
                                    if (frame.destination == (byte)userAddress) // Если
кадр предназначался этой станции
                                    {
                                        waitACC.Set(); // Событие прихода кадра
подтверждения безошибочного приема кадра
                                    }
                                    else // Если кадр предназначался НЕ этой станции
                                    {
                                        SendFrame(frame);
                                    }
                                }
                                else // Если кадр предназначался НЕ этой станции
                                {
                                    SendFrame(frame);
                                }
                                break;

                            case Frame.Type.Ret: // Обработка кадра запроса повторения
последнего отправленного кадра
                                if (frame.destination == (byte)userAddress) // Если кадр
предназначался этой станции
                                {
                                    recdRet.Set(); // Событие прихода кадра запроса
повторения последнего отправленного кадра
                                }
                                else // Если кадр предназначался НЕ этой станции
                                {
                                    SendFrame(frame);
                                }
                                break;
                        }
                    }
                }
            }
        }
    }
}
```

Файл CyclicCode.cs

```
namespace ChatTokenRing
{
    abstract class CyclicCode
    {
        static int g_vect = 0b1011;

        static public byte[] Coding(byte[] inputVect)
```

```
        {
            byte[] res = new byte[inputVect.Length * 2];

            for (int i = 0; i < inputVect.Length; i++)
            {
                int left_vect = (inputVect[i] & 0b11110000) >> 4;
                int right_vect = inputVect[i] & 0b00001111;

                res[i * 2] = (byte)CyclicCoding(left_vect);
                res[i * 2 + 1] = (byte)CyclicCoding(right_vect);
            }
            return res;
        }
        static public (byte[], bool) Decoding(byte[] inputVect)
        {
            byte[] res = new byte[inputVect.Length / 2];
            try
            {
                bool hasError = false;

                for (int i = 0; i < inputVect.Length; i += 2)
                {
                    int left_vect = inputVect[i];
                    int right_vect = inputVect[i + 1];
                    if (!hasError)
                    {
                        if (!ErrorCheck(left_vect) || !ErrorCheck(right_vect))
                        {
                            hasError = true;
                        }
                    }

                    res[i / 2] = (byte)(((left_vect & 0b01111000) << 1) | ((right_vect &
0b01111000) >> 3));
                }
                return (res, hasError);
            }
            catch
            {
                return (res, true);
            }
        }

        static int CyclicCoding(int info_vect)
        {
            int coded_vect = info_vect << 3;

            coded_vect = coded_vect | division(coded_vect);

            return coded_vect;
        }

        static bool ErrorCheck(int coded_vect)
        {
            if (division(coded_vect) == 0)
            {
                return true;
            }
            else
            {
                return false;
            }
        }

        static int division(int vect)
```

```
        {
            for (int i = 6; i > 2; i--)
            {
                if (vect >> i == 1)
                {
                    vect = vect ^ (g_vect << (i - 3));
                }
            }
            return vect;
        }
    }
}
```

Файл Connection.cs

```
using System;
using System.IO.Ports;
using System.Threading;

namespace ChatTokenRing
{
    abstract class Connection
    {
        static object slocker = new object();
        static object glocker = new object();

        private static AutoResetEvent FrameIsRead = new AutoResetEvent(false);

        static SerialPort incomePort;
        static SerialPort outcomePort;
        static bool isMaster;

        public static string[] GetPortsNames()
        {
            return SerialPort.GetPortNames();
        }

        /// <summary>
        /// Открытие портов
        /// </summary>
        public static bool OpenPorts(string incomePortName, string outcomePortName, bool
_isMaster)
        {
            // Создаем объекты портов.
            incomePort = new SerialPort(incomePortName);
            outcomePort = new SerialPort(outcomePortName);

            isMaster = _isMaster;

            // Настраиваем порты.
            incomePort.Parity = Parity.Even;
            incomePort.BaudRate = 9600;
            incomePort.StopBits = StopBits.Two;

            incomePort.Handshake = Handshake.RequestToSend;
            incomePort.ReadBufferSize = 1024;
            incomePort.DataReceived += new SerialDataReceivedEventHandler(RecieveBytes);
            incomePort.ReceivedBytesThreshold = 4;
            incomePort.ReadTimeout = 500;
            incomePort.DtrEnable = true;


            outcomePort.Parity = Parity.Even;
            outcomePort.BaudRate = 9600;
```

```
    outcomePort.StopBits = StopBits.Two;

    outcomePort.Handshake = Handshake.RequestToSend;
    outcomePort.WriteBufferSize = 1024;
    outcomePort.WriteTimeout = 500;
    outcomePort.DtrEnable = true;

    // Открываем порты.
    if (!incomePort.IsOpen)
    {
        incomePort.Open();
    }
    if (!outcomePort.IsOpen)
    {
        outcomePort.Open();
    }

    while (!outcomePort.DsrHolding || !incomePort.DsrHolding)
    {
        Thread.Sleep(50);
    }

    return (incomePort.IsOpen && outcomePort.IsOpen);
}

/// <summary>
/// Закрытие портов
/// </summary>
public static bool ClosePorts()
{
    // Закрываем порты.
    incomePort.Close();
    outcomePort.Close();

    return (!incomePort.IsOpen && !outcomePort.IsOpen);
}

/// <summary>
/// Отправка байтов
/// <summary>
public static void SendBytes(byte[] outputVect)
{
    byte[] codedVect = CyclicCode.Coding(outputVect);
    lock (slocker)
    {
        Thread.Sleep(10);
        if (outcomePort.IsOpen && incomePort.IsOpen && outcomePort.DsrHolding)
        {
            try
            {
                outcomePort.Write(codedVect, 0, codedVect.Length);
            }
            catch
            {
                Console.WriteLine("Connectrion is closed!");
            }
        }
    }
}

/// <summary>
/// Ивент на получение байтов
/// </summary>
static void RecieveBytes(object sender, SerialDataReceivedEventArgs e)
{
```

```
            Thread myThread = new Thread(new ThreadStart(ReadBytes));
            myThread.Start(); // запускаем поток
            FrameIsRead.WaitOne();
        }

        /// <summary>
        /// Считывание байтов
        /// </summary>
        static void ReadBytes()
        {
            byte[] inputVect = new byte[0];
            lock (glocker)
            {
                if (outcomePort.IsOpen && incomePort.IsOpen)
                {
                    int bytes = incomePort.BytesToRead;
                    inputVect = new byte[bytes];
                    incomePort.Read(inputVect, 0, bytes);
                }
            }
            FrameIsRead.Set();
            DataLinkLayer.HandleFrame(inputVect);
        }
    }
}
```