**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук

Департамент программной инженерии

|  |  |  |
| --- | --- | --- |
| СОГЛАСОВАНО  Д. ф. -м. наук, профессор департамента программной инженерии факультета компьютерных наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ М.И. Кумсков  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2018 г. |  | УТВЕРЖДАЮ  Академический руководитель образовательной программы «Программная инженерия»  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.В. Шилов  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2018 г. |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | ***Подп. и дата*** |  | | ***Инв. № дубл.*** |  | | ***Взам. инв. №*** |  | | ***Подп. и дата*** |  | | ***Инв. № подл*** | RU.17701729.04.15-01 81 01-1 | | **пРОГРАММА СТАТИСТИЧЕСКОГО АНАЛИЗА ФИНАНСОВЫХ КОТИРОВОК ПРОВЕРКИ ПАТТЕРНОВ “Японских свечей”**  **Текст программы**  **ЛИСТ УТВЕРЖДЕНИЯ**  **RU.17701729.04.15-01 81 01-1** | | |
|  |  | |
| Исполнитель:  студент группы БПИ172  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Иванов М.В. /  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2018 г. | |
|  | | |
|  | |  |

**2018**

В программе задействованы классы:

|  |  |
| --- | --- |
| Класс | Назначение |
| Base | Класс, отвечающий за чтение вводимых данных |
| Candle | Класс, отвечающий за составление структуры свечи. |
| MainWindow | Класс, связывающий все элементы с главным окном |
| PatternsToList | Класс, отвечающий за составление структуры паттерна. |
| Stat | Класс, отвечающий за сбор статистики работы паттернов. |
| UserControlChart | Класс отвечающий за отрисовку графика |
| UserControlCreate | Класс отвечающий за просмотр загруженных данных |
| UserControlHome | Класс реализующий алгоритмы поиска паттернов японских свечей |
| UserControlStat | Класс собирающий статистику по найденным паттернам |

Класс Base:

public class Base

{

public string TICKER { get; set; }

public string PER { get; set; }

public DateTime DATA;

public int TIME { get; set; }

public double OPEN { get; set; }

public double HIGH { get; set; }

public double LOW { get; set; }

public double CLOSE { get; set; }

public static int kol { get; set; }

public static List<Base> ParseCsv(string file)

{

List<Base> res = new List<Base>();

try

{

var parser = new Microsoft.VisualBasic.FileIO.TextFieldParser(file);

parser.TextFieldType = Microsoft.VisualBasic.FileIO.FieldType.Delimited;

parser.SetDelimiters(new string[] { ";" });

parser.ReadFields();

while (!parser.EndOfData)

{

string[] row = parser.ReadFields();

Base base1 = new Base();

base1.TICKER = row[0];

base1.PER = row[1];

base1.DATA = new DateTime(int.Parse(row[2].Substring(0, 4)), int.Parse(row[2].Substring(4, 2)), int.Parse(row[2].Substring(6, 2)));

base1.TIME = int.Parse(row[3]);

base1.OPEN = double.Parse(row[4], CultureInfo.InvariantCulture);

base1.HIGH = double.Parse(row[5], CultureInfo.InvariantCulture);

base1.LOW = double.Parse(row[6], CultureInfo.InvariantCulture);

base1.CLOSE = double.Parse(row[7], CultureInfo.InvariantCulture);

res.Add(base1);

kol++;

}

return res;

}

catch

{

return null;

}

}

public static Candle[] ParseCandle(string file, int k)

{

Candle[] cndl = new Candle[k];

int z = 0;

try

{

var parser = new Microsoft.VisualBasic.FileIO.TextFieldParser(file);

parser.TextFieldType = Microsoft.VisualBasic.FileIO.FieldType.Delimited;

parser.SetDelimiters(new string[] { ";" });

parser.ReadFields();

while (!parser.EndOfData)

{

string[] row = parser.ReadFields();

Candle can = new Candle();

can.time = new DateTime(int.Parse(row[2].Substring(0, 4)), int.Parse(row[2].Substring(4, 2)), int.Parse(row[2].Substring(6, 2)), int.Parse(row[3].Substring(0, 2)), 0, 0);

can.open = double.Parse(row[4], CultureInfo.InvariantCulture);

can.high = double.Parse(row[5], CultureInfo.InvariantCulture);

can.close = double.Parse(row[7], CultureInfo.InvariantCulture);

can.low = double.Parse(row[6], CultureInfo.InvariantCulture);

can.volume = row[1];

cndl[z] = can;

z++;

}

return cndl;

}

catch

{

return cndl;

}

}

}

Класс Candle:

public class Candle

{

public DateTime time;

public double open;

public double high;

public double close;

public double low;

public string volume;

public string patternName;

public bool work;

}

Класс MainWindow

public partial class MainWindow : Window

{

public static string filename;

public static List<Base> Bases;

public MainWindow()

{

InitializeComponent();

}

private void ButtonOpenMenu\_Click(object sender, RoutedEventArgs e)

{

ButtonCloseMenu.Visibility = Visibility.Visible;

ButtonOpenMenu.Visibility = Visibility.Collapsed;

}

private void ButtonCloseMenu\_Click(object sender, RoutedEventArgs e)

{

ButtonCloseMenu.Visibility = Visibility.Collapsed;

ButtonOpenMenu.Visibility = Visibility.Visible;

}

private void ListViewMenu\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

UserControl usc = null;

GridMain.Children.Clear();

try {

switch (((ListViewItem)((ListView)sender).SelectedItem).Name)

{

case "Download":

OpenFileDialog dlg = new OpenFileDialog();

dlg.DefaultExt = ".csv";

dlg.Filter = "CSV Files (\*.csv)|\*.csv";

if (dlg.ShowDialog() == true)

{

filename = dlg.FileName;

}

break;

case "DrawGraph":

usc = new UserControlChart();

GridMain.Children.Add(usc);

break;

case "FindPatterns":

usc = new UserControlHome();

GridMain.Children.Add(usc);

break;

case "ShowLikeTable":

if (filename != null)

{

usc = new UserControlCreate();

GridMain.Children.Add(usc);

}

break;

case "Statistic":

usc = new UserControlStat();

GridMain.Children.Add(usc);

break;

default:

break;

}

}

catch

{

}

}

private void FindPatterns\_Selected(object sender, RoutedEventArgs e)

{

}

private void Information\_Selected(object sender, RoutedEventArgs e)

{

}

private void Button\_Click(object sender, RoutedEventArgs e)

{

this.Close();

}

private void Button\_Click\_1(object sender, RoutedEventArgs e)

{

WindowState = WindowState.Minimized;

}

}

Класс PatternsToList:

public class PatternsToList

{

public string patternname { get; set; }

public DateTime pattertime { get; set; }

public string Work { get; set; }

}

Класс Stat:

public class Stat

{

public string name { get; set; }

public int quantity { get; set; }

public int Work { get; set; }

public int NotWork { get; set; }

}

Класс UserControlCreat:

public partial class UserControlCreate : UserControl

{

public List<Base> Bases;

public UserControlCreate()

{

InitializeComponent();

Data\_Show();

}

private void Data\_Show()

{

Bases = Base.ParseCsv(MainWindow.filename);

Datagrid1.ItemsSource = Bases;

}

}

Класс UserControlHome

public partial class UserControlHome : UserControl

{

public UserControlHome()

{

InitializeComponent();

DataPat.ItemsSource = FindPatterns(CandleArray);

}

public Candle[] CandleArray = Base.ParseCandle(MainWindow.filename, Base.ParseCsv(MainWindow.filename).Count);

public static List<PatternsToList> FindPatterns(Candle[] CandleArray)

{

List<PatternsToList> pot = new List<PatternsToList>();

try

{

for (int i = 4; i < CandleArray.Length - 5; i++)

{

int kol = 0, kol1 = 0;

//4 предыдущие свечи

if (CandleArray[i - 4].close < CandleArray[i - 4].open)

kol++;

if ((CandleArray[i - 3].close < CandleArray[i - 3].open) && (CandleArray[i - 4].close != CandleArray[i - 3].close))

kol++;

if ((CandleArray[i - 2].close < CandleArray[i - 2].open) && (CandleArray[i - 2].close != CandleArray[i - 3].close))

kol++;

if ((CandleArray[i - 1].close < CandleArray[i - 1].open) && (CandleArray[i - 1].close != CandleArray[i - 2].close))

kol++;

//4 следующие свечи

if (CandleArray[i + 4].close > CandleArray[i + 4].open)

kol1++;

if ((CandleArray[i + 3].close > CandleArray[i + 3].open) && (CandleArray[i + 4].close != CandleArray[i + 3].close))

kol1++;

if ((CandleArray[i + 2].close > CandleArray[i + 2].open) && (CandleArray[i + 2].close != CandleArray[i + 3].close))

kol1++;

if ((CandleArray[i + 1].close > CandleArray[i + 1].open) && (CandleArray[i + 1].close != CandleArray[i + 2].close))

kol1++;

//молот

if ((kol >= 3) && (CandleArray[i].open > CandleArray[i].close))

{

if (((CandleArray[i].open - CandleArray[i].close) < ((CandleArray[i].close - CandleArray[i].low) \* 0.5)) &&

((CandleArray[i].high - CandleArray[i].open) < (CandleArray[i].open - CandleArray[i].close)))

{

CandleArray[i].patternName = "Молот";

if (kol1 >= 3)

CandleArray[i].work = true;

else

CandleArray[i].work = false;

PatternsToList kek = new PatternsToList();

kek.patternname = CandleArray[i].patternName;

kek.pattertime = CandleArray[i].time;

if (CandleArray[i].work == true)

kek.Work = "Сработал";

else

kek.Work = "Не Сработал";

pot.Add(kek);

}

}

//Падающая звезда

if ((kol < 2) && (CandleArray[i].open < CandleArray[i].close))

{

if (((CandleArray[i].close - CandleArray[i].open) < ((CandleArray[i].high - CandleArray[i].close) \* 0.5))

&& ((CandleArray[i].open - CandleArray[i].low) < (CandleArray[i].close - CandleArray[i].open)))

{

CandleArray[i].patternName = "Падающая звезда";

if (kol1 < 2)

CandleArray[i].work = true;

else

CandleArray[i].work = false;

PatternsToList kek = new PatternsToList();

kek.patternname = CandleArray[i].patternName;

kek.pattertime = CandleArray[i].time;

if (CandleArray[i].work == true)

kek.Work = "Сработал";

else

kek.Work = "Не Сработал";

pot.Add(kek);

}

}

//Повешенный

if ((kol < 2) && (CandleArray[i].open > CandleArray[i].close))

{

if (((CandleArray[i].open - CandleArray[i].close) < ((CandleArray[i].close - CandleArray[i].low) \* 0.5))

&& ((CandleArray[i].high - CandleArray[i].open) < (CandleArray[i].open - CandleArray[i].close)))

{

CandleArray[i].patternName = "Повешанный";

if (kol1 < 2)

CandleArray[i].work = true;

else

CandleArray[i].work = false;

PatternsToList kek = new PatternsToList();

kek.patternname = CandleArray[i].patternName;

kek.pattertime = CandleArray[i].time;

if (CandleArray[i].work == true)

kek.Work = "Сработал";

else

kek.Work = "Не Сработал";

pot.Add(kek);

}

}

//Дожи

if (Math.Abs(CandleArray[i].close - CandleArray[i].open) < 0.02 \* (CandleArray[i].high - CandleArray[i].low))

{

if (((kol1 < 2) && (kol < 2)) || ((kol1 >= 3) && (kol >= 3)))

{

CandleArray[i].patternName = "Дожи";

CandleArray[i].work = true;

}

else

{

CandleArray[i].patternName = "Дожи";

CandleArray[i].work = false;

}

PatternsToList kek = new PatternsToList();

kek.patternname = CandleArray[i].patternName;

kek.pattertime = CandleArray[i].time;

if (CandleArray[i].work == true)

kek.Work = "Сработал";

else

kek.Work = "Не Сработал";

pot.Add(kek);

}

//бычье поглощение

if ((CandleArray[i].open > CandleArray[i].close) && (kol >= 3))

{

if (CandleArray[i + 1].open < CandleArray[i + 1].close)

{

if ((CandleArray[i + 1].close > CandleArray[i].open) && (CandleArray[i + 1].open < CandleArray[i].close))

{

if (kol1 >= 3)

{

if (CandleArray[i].patternName == null)

{

CandleArray[i].patternName = "Бычье поглощение";

CandleArray[i].work = true;

}

else

{

CandleArray[i + 1].patternName = "Бычье поглощение";

CandleArray[i + 1].work = true;

}

}

else

{

if (CandleArray[i].patternName == null)

{

CandleArray[i].patternName = "Бычье поглощение";

CandleArray[i].work = false;

}

else

{

CandleArray[i + 1].patternName = "Бычье поглощение";

CandleArray[i + 1].work = false;

}

}

PatternsToList kek = new PatternsToList();

kek.patternname = CandleArray[i].patternName;

kek.pattertime = CandleArray[i].time;

if (CandleArray[i].work == true)

kek.Work = "Сработал";

else

kek.Work = "Не Сработал";

pot.Add(kek);

}

}

}

//медвежье поглощение

if ((CandleArray[i].open < CandleArray[i].close) && (kol1 >= 3))

{

if (CandleArray[i + 1].open > CandleArray[i + 1].close)

{

if ((CandleArray[i + 1].close < CandleArray[i].open) && (CandleArray[i + 1].open > CandleArray[i].close))

{

if (kol1 < 2)

{

if (CandleArray[i].patternName == null)

{

CandleArray[i].patternName = "Медвежье поглощение";

CandleArray[i].work = true;

}

else

{

CandleArray[i + 1].patternName = "Медвежье поглощение";

CandleArray[i + 1].work = true;

}

}

else

{

if (CandleArray[i].patternName == null)

{

CandleArray[i].patternName = "Медвежье поглощение";

CandleArray[i].work = false;

}

else

{

CandleArray[i + 1].patternName = "Медвежье поглощение";

CandleArray[i + 1].work = false;

}

}

PatternsToList kek = new PatternsToList();

kek.patternname = CandleArray[i].patternName;

kek.pattertime = CandleArray[i].time;

if (CandleArray[i].work == true)

kek.Work = "Сработал";

else

kek.Work = "Не Сработал";

pot.Add(kek);

}

}

}

//Бычий захват за пояс

if ((kol >= 3) && (CandleArray[i].close > CandleArray[i].open))

{

if (((CandleArray[i].close - CandleArray[i].open) > ((CandleArray[i].high - CandleArray[i].close) \* 2)) && ((CandleArray[i].open - CandleArray[i].low) == 0))

{

CandleArray[i].patternName = "Бычий захват за пояс";

if (kol1 >= 3)

CandleArray[i].work = true;

else

CandleArray[i].work = false;

PatternsToList kek = new PatternsToList();

kek.patternname = CandleArray[i].patternName;

kek.pattertime = CandleArray[i].time;

if (CandleArray[i].work == true)

kek.Work = "Сработал";

else

kek.Work = "Не Сработал";

pot.Add(kek);

}

}

//Медвежий захват за пояс

if ((kol < 2) && (CandleArray[i].close < CandleArray[i].open))

{

if (((CandleArray[i].open - CandleArray[i].close) > ((CandleArray[i].close - CandleArray[i].low) \* 2)) && ((CandleArray[i].high - CandleArray[i].open) == 0))

{

CandleArray[i].patternName = "Медвежий захват за пояс";

if (kol1 < 2)

CandleArray[i].work = true;

else

CandleArray[i].work = false;

PatternsToList kek = new PatternsToList();

kek.patternname = CandleArray[i].patternName;

kek.pattertime = CandleArray[i].time;

if (CandleArray[i].work == true)

kek.Work = "Сработал";

else

kek.Work = "Не Сработал";

pot.Add(kek);

}

}

}

}

catch

{

return pot;

}

return (pot);

}

private void Button\_Click(object sender, RoutedEventArgs e)

{

}

}

Класс UserControlStat:

public partial class UserControlStat : UserControl

{

List<Stat> LAK = new List<Stat>();

public UserControlStat()

{

InitializeComponent();

List<Stat> LAK1 = FindStat();

dat.ItemsSource = LAK1;

}

public Candle[] CandleArray1 = Base.ParseCandle(MainWindow.filename, Base.ParseCsv(MainWindow.filename).Count);

public List<Stat> FindStat()

{

int KolM = 0, KolPA = 0, KolPO = 0, KolD = 0, KolBP = 0, KolMP = 0, KolBZ = 0, KolMZ = 0;

int KolMT = 0, KolPAT = 0, KolPOT = 0, KolDT = 0, KolBPT = 0, KolMPT = 0, KolBZT = 0, KolMZT = 0;

PatternsToList[] pot;

pot = UserControlHome.FindPatterns(CandleArray1).ToArray();

try

{

for (int i = 0; i < pot.Length; i++)

{

if (pot[i].patternname == "Молот")

{

KolM++;

if (pot[i].Work == "Сработал")

KolMT++;

}

else if (pot[i].patternname == "Падающая звезда")

{

KolPA++;

if (pot[i].Work == "Сработал")

KolPAT++;

}

else if (pot[i].patternname == "Повешанный")

{

KolPO++;

if (pot[i].Work == "Сработал")

KolPOT++;

}

else if (pot[i].patternname == "Дожи")

{

KolD++;

if (pot[i].Work == "Сработал")

KolDT++;

}

else if (pot[i].patternname == "Бычье поглощение")

{

KolBP++;

if (pot[i].Work == "Сработал")

KolBPT++;

}

else if (pot[i].patternname == "Медвежье поглощение")

{

KolMP++;

if (pot[i].Work == "Сработал")

KolMPT++;

}

else if (pot[i].patternname == "Бычий захват за пояс")

{

KolBZ++;

if (pot[i].Work == "Сработал")

KolBZT++;

}

else if (pot[i].patternname == "Медвежий захват за пояс")

{

KolMZ++;

if (pot[i].Work == "Сработал")

KolMZT++;

}

}

Stat M = new Stat();

M.name = "Молот";

M.quantity = KolM;

M.Work = KolMT;

M.NotWork = KolM - KolMT;

LAK.Add(M);

Stat PA = new Stat();

PA.name = "Падающая звезда";

PA.quantity = KolPA;

PA.Work = KolPAT;

PA.NotWork = KolPA - KolPAT;

LAK.Add(PA);

Stat PO = new Stat();

PO.name = "Повешанный";

PO.quantity = KolPO;

PO.Work = KolPOT;

PO.NotWork = KolPO - KolPOT;

LAK.Add(PO);

Stat D = new Stat();

D.name = "Дожи";

D.quantity = KolD;

D.Work = KolDT;

D.NotWork = KolD - KolDT;

LAK.Add(D);

Stat BP = new Stat();

BP.name = "Бычье поглощение";

BP.quantity = KolBP;

BP.Work = KolBPT;

BP.NotWork = KolBP - KolBPT;

LAK.Add(BP);

Stat MP = new Stat();

MP.name = "Медвежье поглощение";

MP.quantity = KolMP;

MP.Work = KolMPT;

MP.NotWork = KolMP - KolMPT;

LAK.Add(MP);

Stat BZ = new Stat();

BZ.name = "Бычий захват за пояс";

BZ.quantity = KolBZ;

BZ.Work = KolBZT;

BZ.NotWork = KolBZ - KolBZT;

LAK.Add(BZ);

Stat MZ = new Stat();

MZ.name = "Медвежий захват за пояс";

MZ.quantity = KolMZ;

MZ.Work = KolMZT;

MZ.NotWork = KolMZ - KolMZT;

LAK.Add(MZ);

return LAK;

}

catch

{

return LAK;

}

}

Класс UserControlChart

public partial class UserControlChart : System.Windows.Controls.UserControl

{

System.Windows.Forms.DataVisualization.Charting.Chart chartForCandle; // это адрес для нашего чарта

public string pathToHistory = null; // это адрес для строки с путём к данным

private Candle[] CandleArray = Base.ParseCandle(MainWindow.filename, Base.ParseCsv(MainWindow.filename).Count); // это массив свечек

public UserControlChart() //окно загружается.

{

InitializeComponent();

// вызываем метод для создания чарта

}

public void CreateChart() // метод создающий чарт

{

// создаём чарт от Win Forms

chartForCandle = new System.Windows.Forms.DataVisualization.Charting.Chart();

// привязываем его к хосту.

hostChart.Child = chartForCandle;

hostChart.Child.Show();

// на всякий случай чистим в нём всё

chartForCandle.Series.Clear();

chartForCandle.ChartAreas.Clear();

// создаём область на чарте

chartForCandle.ChartAreas.Add("ChartAreaCandle");

chartForCandle.ChartAreas.FindByName("ChartAreaCandle").CursorX.IsUserSelectionEnabled = true; // разрешаем пользователю изменять рамки представления

chartForCandle.ChartAreas.FindByName("ChartAreaCandle").CursorX.IsUserEnabled = true; //чертa

chartForCandle.ChartAreas.FindByName("ChartAreaCandle").CursorY.AxisType = System.Windows.Forms.DataVisualization.Charting.AxisType.Secondary; // ось У правая

// создаём для нашей области коллекцию значений

chartForCandle.Series.Add("SeriesCandle");

// назначаем этой коллекции тип "Свечи"

chartForCandle.Series.FindByName("SeriesCandle").ChartType = System.Windows.Forms.DataVisualization.Charting.SeriesChartType.Candlestick;

// назначаем ей правую линейку по шкале Y

chartForCandle.Series.FindByName("SeriesCandle").YAxisType = System.Windows.Forms.DataVisualization.Charting.AxisType.Secondary;

// помещаем нашу коллекцию на ранее созданную область

chartForCandle.Series.FindByName("SeriesCandle").ChartArea = "ChartAreaCandle";

// наводим тень

chartForCandle.Series.FindByName("SeriesCandle").ShadowOffset = 2;

chartForCandle.Series.FindByName("SeriesCandle").BorderWidth = 3;

for (int i = 0; i < chartForCandle.ChartAreas.Count; i++)

{ // Делаем курсор по Y красным и толстым

chartForCandle.ChartAreas[i].CursorX.LineColor = System.Drawing.Color.Red;

chartForCandle.ChartAreas[i].CursorX.LineWidth = 2;

}

// подписываемся на события изменения масштабов

chartForCandle.AxisScrollBarClicked += chartForCandle\_AxisScrollBarClicked; // событие передвижения курсора

chartForCandle.AxisViewChanged += chartForCandle\_AxisViewChanged; // событие изменения масштаба

chartForCandle.CursorPositionChanged += chartForCandle\_CursorPositionChanged; // событие выделения диаграммы

}

public void buttonRew\_Click(object sender, RoutedEventArgs e) // кнопка delete

{

chartForCandle.Visible = false; // прячем чарт

}

private void Start\_Click(object sender, RoutedEventArgs e)

{

CreateChart();

Thread Worker = new Thread(StartPaintChart);

Worker.IsBackground = true;

Worker.Start();

}

private void StartPaintChart() // метод вызывающийся в новом потоке, для прорисовки графика

{

LoadCandleOnChart();

}

private void LoadCandleOnChart() // прогрузить загруженные свечки на график

{

if (CandleArray == null)

{//если наш массив пуст по каким-то причинам

return;

}

for (int i = 0; i < CandleArray.Length; i++)

{// отправляем наш массив по свечкам на прорисовку

LoadNewCandle(CandleArray[i], i);

}

}

private void LoadNewCandle(Candle newCandle, int numberInArray) // добавить одну свечу на график

{

try

{

if (!this.CheckAccess())

{// перезаходим в метод потоком формы, чтобы не было исключения

this.Dispatcher.Invoke(new Action<Candle, int>(LoadNewCandle), newCandle, numberInArray);

return;

}

// забиваем новую свечку

chartForCandle.Series.FindByName("SeriesCandle").Points.AddXY(numberInArray, newCandle.low, newCandle.high, newCandle.open, newCandle.close);

// подписываем время

chartForCandle.Series.FindByName("SeriesCandle").Points[chartForCandle.Series.FindByName("SeriesCandle").Points.Count - 1].AxisLabel = newCandle.time.ToString();

// разукрышиваем в привычные цвета

if (newCandle.close > newCandle.open)

{

if (newCandle.close != newCandle.open)

chartForCandle.Series.FindByName("SeriesCandle").Points[chartForCandle.Series.FindByName("SeriesCandle").Points.Count - 1].Color = System.Drawing.Color.Green;

}

else

{

chartForCandle.Series.FindByName("SeriesCandle").Points[chartForCandle.Series.FindByName("SeriesCandle").Points.Count - 1].Color = System.Drawing.Color.Red;

}

if (chartForCandle.ChartAreas.FindByName("ChartAreaCandle").AxisX.ScrollBar.IsVisible == true)

{// если уже выбран какой-то диапазон

chartForCandle.ChartAreas.FindByName("ChartAreaCandle").AxisX.ScaleView.Scroll(chartForCandle.ChartAreas[0].AxisX.Maximum); // сдвигаем представление вправо

}

ChartResize(); // Выводим нормальные рамки

}

catch

{

return;

}

}

private void ChartResize() // устанавливает границы представления по оси У

{

try

{

if (CandleArray == null)

{

return;

}

int startPozition = 0; // первая отображаемая свеча

int endPozition = chartForCandle.Series.FindByName("SeriesCandle").Points.Count; // последняя отображаемая свеча

if (chartForCandle.ChartAreas[0].AxisX.ScrollBar.IsVisible == true)

{// если уже выбран какой-то диапазон, назначаем первую и последнюю исходя из этого диапазона

startPozition = Convert.ToInt32(chartForCandle.ChartAreas.FindByName("ChartAreaCandle").AxisX.ScaleView.Position);

endPozition = Convert.ToInt32(chartForCandle.ChartAreas.FindByName("ChartAreaCandle").AxisX.ScaleView.Position) +

Convert.ToInt32(chartForCandle.ChartAreas.FindByName("ChartAreaCandle").AxisX.ScaleView.Size);

}

chartForCandle.ChartAreas.FindByName("ChartAreaCandle").AxisY2.Maximum = GetMaxValueOnChart(CandleArray, startPozition, endPozition);

chartForCandle.ChartAreas.FindByName("ChartAreaCandle").AxisY2.Minimum = GetMinValueOnChart(CandleArray, startPozition, endPozition);

chartForCandle.Refresh();

}

catch (System.InvalidOperationException)

{

return;

}

catch

{

return;

}

}

private double GetMinValueOnChart(Candle[] Book, int start, int end) // берёт минимальное значение из массива свечек

{

double result = double.MaxValue;

for (int i = start; i < end && i < Book.Length; i++)

{

if (Book[i].low < result)

{

result = Book[i].low;

}

}

return result;

}

private double GetMaxValueOnChart(Candle[] Book, int start, int end) // берёт максимальное значение из массива свечек

{

double result = 0;

for (int i = start; i < end && i < Book.Length; i++)

{

if (Book[i].high > result)

{

result = Book[i].high;

}

}

return result;

}

private void Rewind() // перемотка

{

try

{

chartForCandle.Visible = false; // открываем чарт

}

catch

{

return;

}

}

// события

void chartForCandle\_CursorPositionChanged(object sender, System.Windows.Forms.DataVisualization.Charting.CursorEventArgs e) // событие изменение отображения диаграммы

{

ChartResize();

}

void chartForCandle\_AxisViewChanged(object sender, System.Windows.Forms.DataVisualization.Charting.ViewEventArgs e) // событие изменение отображения диаграммы

{

ChartResize();

}

void chartForCandle\_AxisScrollBarClicked(object sender, System.Windows.Forms.DataVisualization.Charting.ScrollBarEventArgs e) // событие изменение отображения диаграммы

{

ChartResize();

}

private void hostChart\_ChildChanged(object sender, System.Windows.Forms.Integration.ChildChangedEventArgs e)

{

}

}