Igor Kohsin, 8.06.2022r

Podstawy programowania - projekt, sprawozdanie ilustrujące funkcjonalność programu

Analizator wyników pomiarów temperatury

Założenia projektowe:

Zadaniem jest napisanie programu, który:

1. Otwiera plik tekstowy z zapisanymi wynikami pomiarów temperatury.

```
private void button_zaladuj_Click(object sender, EventArgs e)
{

    Stream myStream = null;
    OpenFileDialog openFileDialog1 = new OpenFileDialog();
    openFileDialog1.InitialDirectory = "/.";
    openFileDialog1.Filter = "txt files (*.txt)|*.txt|All files (*.*)|*.*";
    if (openFileDialog1.ShowDialog() == DialogResult.OK && openFileDialog1.FileName.Contains(".txt"))
    {
        try
        {
            if ((myStream = openFileDialog1.OpenFile()) != null)
            {
                using (myStream)
                {
                    button_zaladuj.AccessibleRole = AccessibleRole.None;
                    button_zaladuj.Enabled = false;
                    button_zapisz.AccessibleRole = AccessibleRole.Default;
                    label_main_zaladowanyplik.ForeColor = Color.Cyan;

                    //label_komunikat.Text = "Wczytano pliki " + openFileDialog1.FileName.Substring(openFileDialog1.FileName.LastI

                    label_main_zaladowanyplik.Text = openFileDialog1.FileName.Substring(openFileDialog1.FileName.LastIndexOf(@"\")

                    string file = File.ReadAllText(openFileDialog1.FileName);
                    string[] readData = file.Split(';').ToArray();
                    label_komunikat.Text = "";
```

2. Dokonuje parsowania zawartego w nim tekstu, aby wyodrębnić wartości liczbowe pomiarów.

```
        double number;

        if (!double.TryParse(readData[0], out number))
            tytul = readData[0];

        for (int i = 0; i < readData.Length - 1; i++)
        {
            if (double.TryParse(readData[i], out number))
                samples.Add(number);
        }
        calculateValues();
        myStream.Close();
    }
}
```

3. Oblicza niektóre parametry pomiaru (wartość średnią, wariancję, wartość minimalną oraz maksymalną).

```
public void calculateValues()
{
    quantity = samples.Count;

    for (int i = 0; i < samples.Count; i++)
    {
        suma += samples[i];
    }
    average = Math.Round(suma / samples.Count, accuracy);
    samples.Sort();
    maxValue = Math.Round(samples[samples.Count - 1], accuracy);
    minValue = Math.Round(samples[0], accuracy);

    for (int j = 0; j < samples.Count; j++)
    {
        temp += Math.Pow(samples[j] - average, 2);
    }
    variance = Math.Round(temp / samples.Count, accuracy);

    ilosc.Text = quantity.ToString();
    srednia.Text = average.ToString();
    wariancja.Text = variance.ToString();
    max.Text = maxValue.ToString();
    min.Text = minValue.ToString();

}
```

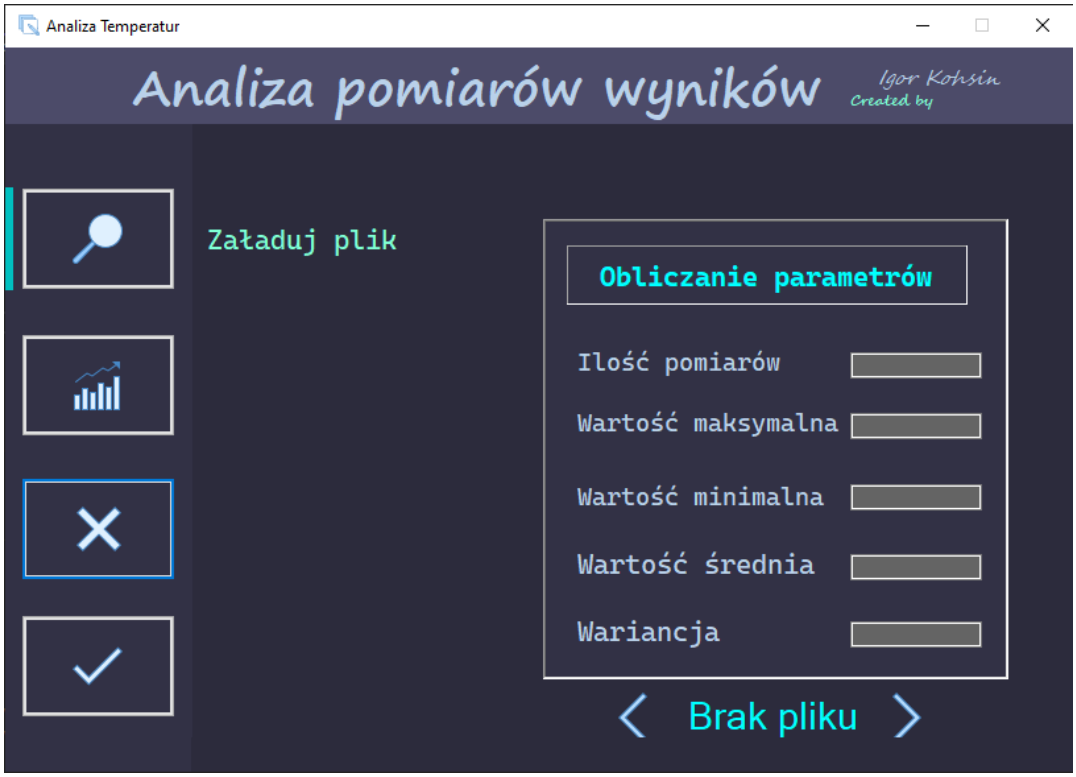
4. Umożliwia zapisanie obliczonych parametrów do pliku tekstowego.

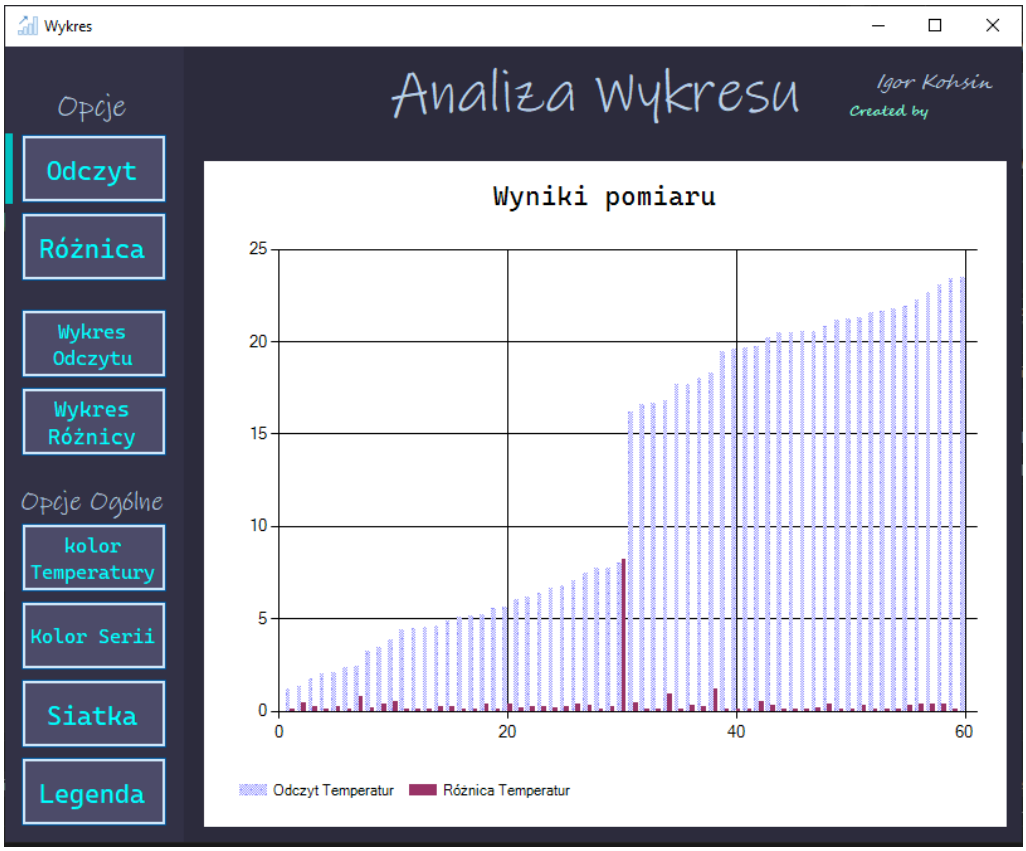
```
private void button_zapisz_Click(object sender, EventArgs e)
{
    SaveFileDialog saveFileDialog1 = new SaveFileDialog();
    saveFileDialog1.Filter = "txt files (*.txt)|*.txt";
    if (saveFileDialog1.ShowDialog() != System.Windows.Forms.DialogResult.OK)
    {
        MessageBox.Show("Plik nie został zapisany", "Info", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }

    else
    {

        List<string> zapisz = new List<string> { };

        zapisz.Add("Wartość maksymalna: " + max.Text);
        zapisz.Add("Wartość minimalna: " + min.Text);
        zapisz.Add("Wartość średnia: " + srednia.Text);
        zapisz.Add("Wariancja: " + wariancja.Text);

        try
        {
            System.IO.File.AppendAllLines(saveFileDialog1.FileName, zapisz);
            MessageBox.Show("Zapisano pomyślnie", "Zapisano", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }

        catch (Exception)
        {
            MessageBox.Show("Error", "Error", MessageBoxButtons.OK);
        }

    }
}
```

5. Umożliwia wizualizację różnicy wartości kolejnych pomiarów na wykresie (kontrolka Chart), na wygląd którego użytkownik może wpływać (np. wyłączając niektóre serie danych, wyłączając wyświetlane legendy, zmieniając kolor itp.).

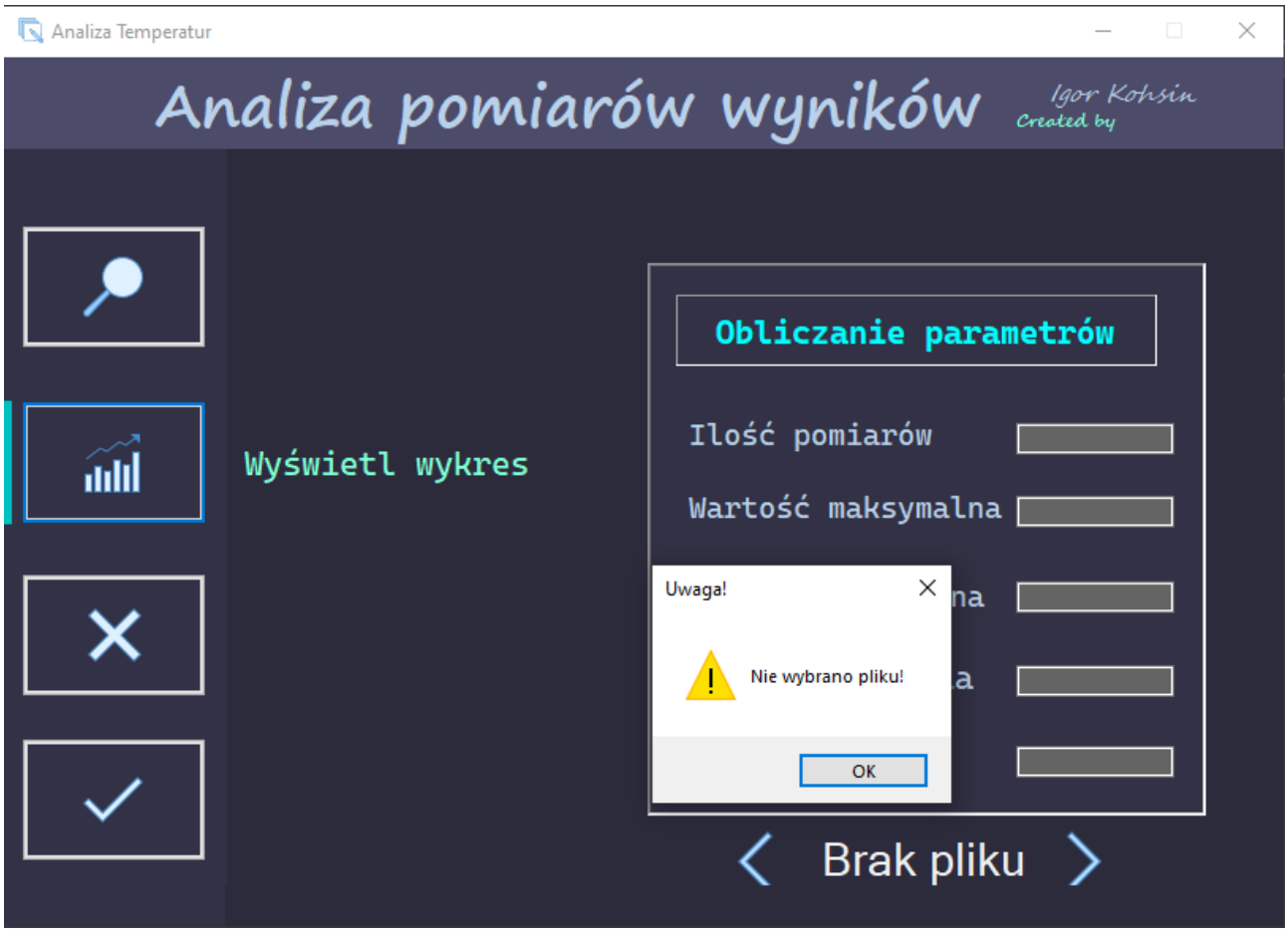
Ważne jest, aby program był user-friendly, czyli:

- charakteryzował się estetyczną stroną graficzną,

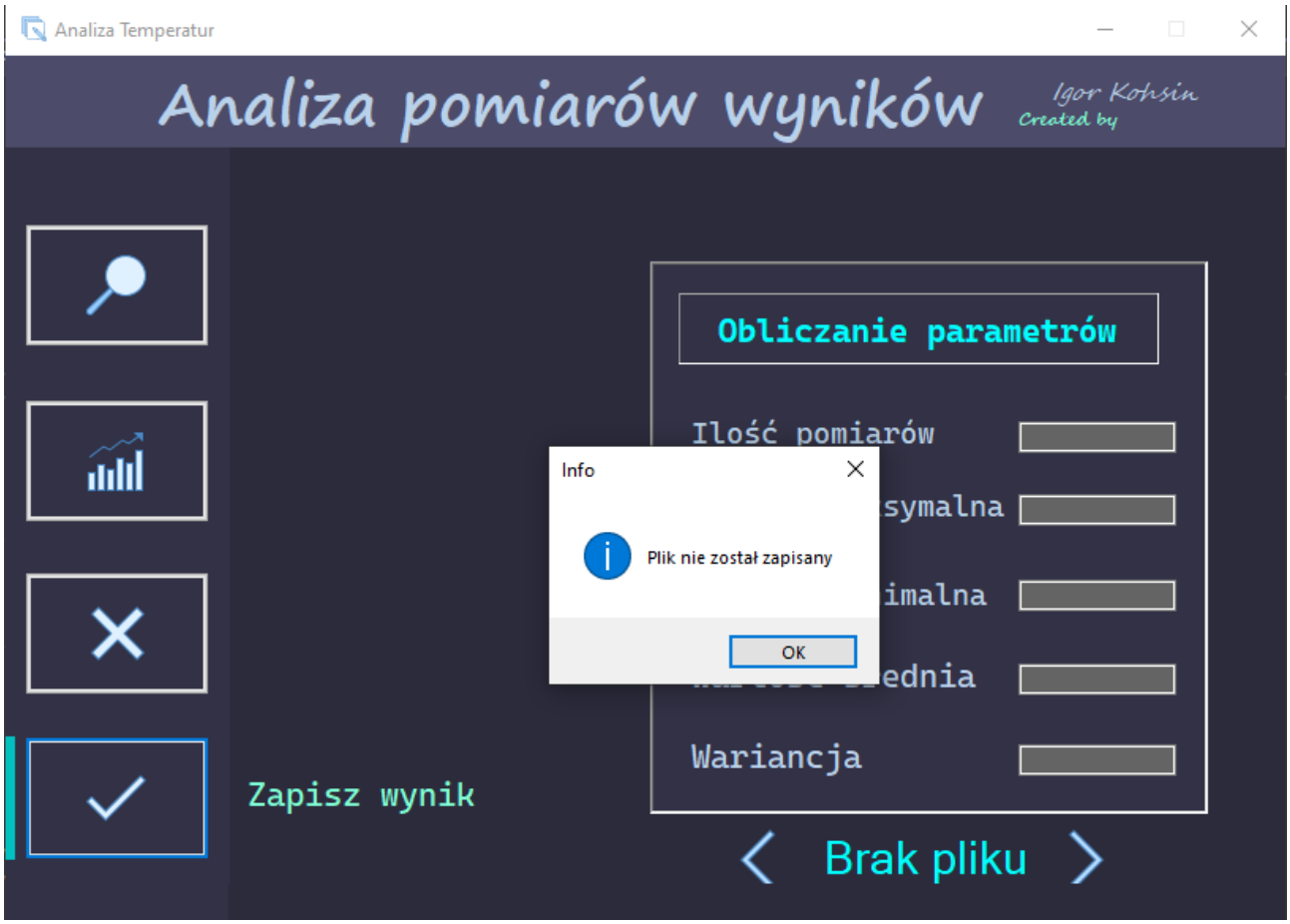




- nie kończył awaryjnie działania przy wystąpieniu każdego błędu (staramy się przewidzieć możliwe błędy i obsłużyć wyjątki),

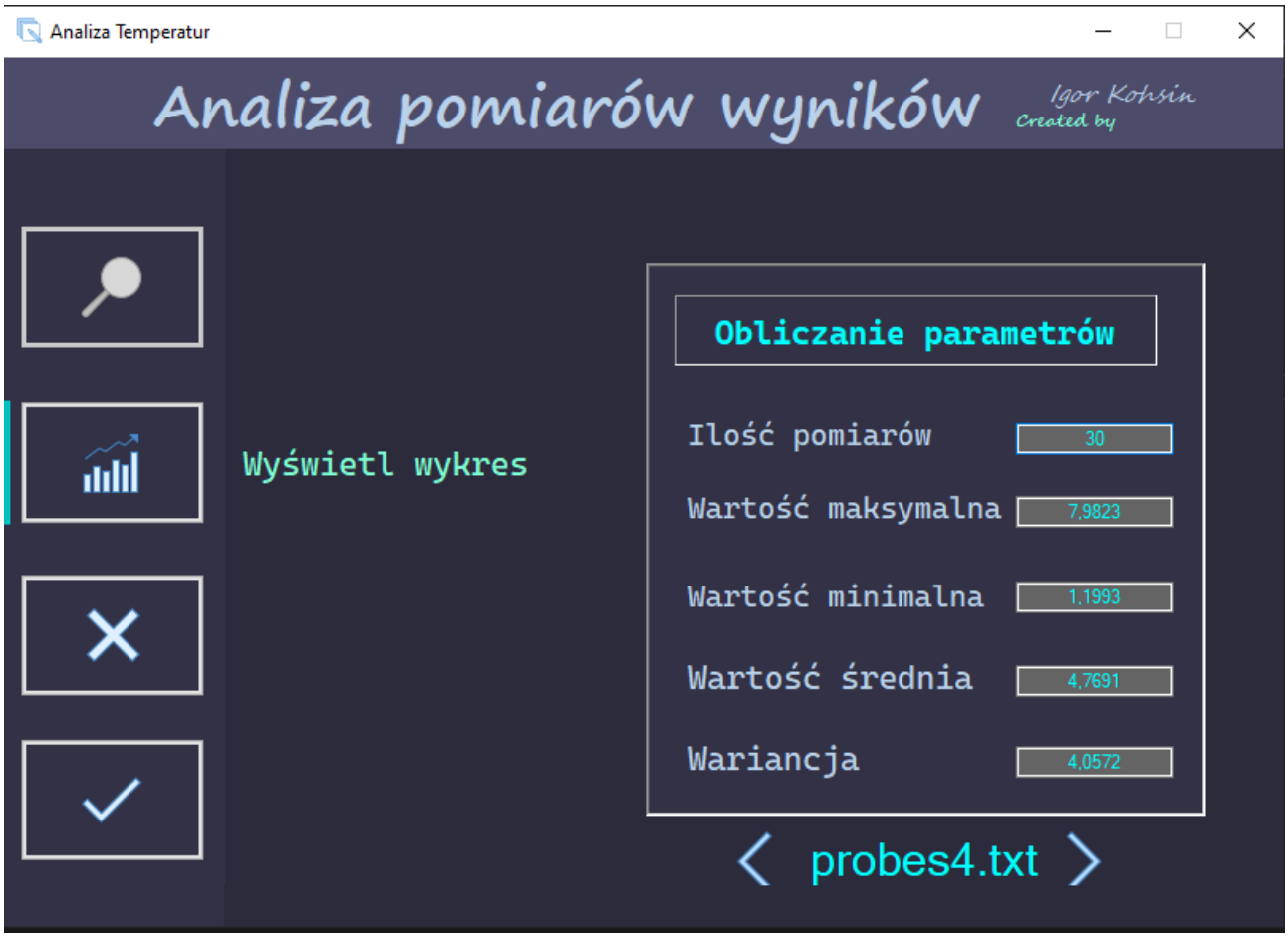


~ nie można wyświetlić wykresu bez załadowanego pliku
~ brak wyboru pliku również skutkuje powyższym oknem



~ brak wybranego pliku do zapisania skutkuje powyższym oknem

- nie pozwalał użytkownikowi na działania, których jeszcze nie da się wykonać, przykładowo, na otwarcie wykresu, zanim załadowane zostaną dane pomiarowe;



~ wybranie pliku blokuje przycisk ładowania pliku, tym samym wgranie kolejnego pliku; Aby wgrać inny plik, należy najpierw wyzerować dane

Omówienie kodu aplikacji

```
//---Przyciski Klikanie---///
0 references
private void button_zaladuj_Click(object sender, EventArgs e)
{

    Stream myStream = null;
    OpenFileDialog openFileDialog1 = new OpenFileDialog();
    openFileDialog1.InitialDirectory = "/.";
    openFileDialog1.Filter = "txt files (*.txt)|*.txt|All files (*.*)|*.*";
    if (openFileDialog1.ShowDialog() == DialogResult.OK && openFileDialog1.FileName.Contains(".txt"))
    {
        try
        {
            if ((myStream = openFileDialog1.OpenFile()) != null)
            {
                using (myStream)
                {
                    button_zaladuj.AccessibleRole = AccessibleRole.None;
                    button_zaladuj.Enabled = false;
                    button_zapisz.AccessibleRole = AccessibleRole.Default;
                    label_main_zaladowanyplik.ForeColor = Color.Cyan;

                    //label_komunikat.Text = "Wczytano pliki " + openFileDialog1.FileName.Substring(openFileDialog1.FileName.LastIndexOf(@"\") + 1);

                    label_main_zaladowanyplik.Text = openFileDialog1.FileName.Substring(openFileDialog1.FileName.LastIndexOf(@"\") + 1);

                    string file = File.ReadAllText(openFileDialog1.FileName);
                    string[] readData = file.Split(';').ToArray();
                    label_komunikat.Text = "";


                    double number;

                    if (!double.TryParse(readData[0], out number))
                        tytul = readData[0];

                    for (int i = 0; i < readData.Length - 1; i++)
                    {
                        if (double.TryParse(readData[i], out number))
                            samples.Add(number);
                    }
                    calculateValues();
                    myStream.Close();
                }
            }
        }
        catch (Exception ex)
        {

            MessageBox.Show("Błąd: Nie można było wczytać pliku.\nOriginal error: " + ex.Message, "Błąd!", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }
    else if (openFileDialog1.FileName == "")
    {
        MessageBox.Show("Nie wybrano pliku!", "Uwaga!", MessageBoxButtons.OK, MessageBoxIcon.Warning);
    }
    else
    {
        MessageBox.Show("Błąd: Wybrano zły typ pliku!", "Błąd!", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }


}
```

Metoda ta odpowiada za wgrywanie pliku, gdzie przy użyciu tru catch wgrywany jest plik o odpowiednim formacie. W przypadku problemu, program odpowie oknem dialogowym z odpowiednią informacją.

```
private void button_wyswietl_Click(object sender, EventArgs e)
{
    if (label_main_zaladowanyplik.Text == "Brak pliku")
    {
        MessageBox.Show("Nie wybrano pliku!", "Uwaga!", MessageBoxButtons.OK, MessageBoxIcon.Warning);
    }
    else
    {
        Wykres wykres = new Wykres(samples, tytul);
        wykres.Show();
        //button_wyswietl.Enabled = false;

    }
}
```

Metoda ta odpowiada za przycisk wyświetlający wykres, w przypadku braku pliku wyświetlone zostanie okno z informacją o nieprawidłowości.

```
public void calculateValues()
{
    quantity = samples.Count;

    for (int i = 0; i < samples.Count; i++)
    {
        suma += samples[i];
    }
    average = Math.Round(suma / samples.Count, accuracy);
    samples.Sort();
    maxValue = Math.Round(samples[samples.Count - 1], accuracy);
    minValue = Math.Round(samples[0], accuracy);

    for (int j = 0; j < samples.Count; j++)
    {
        temp += Math.Pow(samples[j] - average, 2);
    }
    variance = Math.Round(temp / samples.Count, accuracy);

    ilosc.Text = quantity.ToString();
    srednia.Text = average.ToString();
    wariancja.Text = variance.ToString();
    max.Text = maxValue.ToString();
    min.Text = minValue.ToString();

}
```

Metoda ta odpowiada za obliczanie wartości wprowadzanych próbek. Jej zadaniem jest odpowiednia konwersja danych, jak również przetworzenie ich by mogły zostać w odpowiednim miejscu wyświetlone jako tekst.

```
private void button_wyczysc_Click(object sender, EventArgs e)
{
    button_zaladuj.Enabled = true;
    label_main_zaladowanyplik.Text = "Brak pliku";
    ilosc.Text = "";
    max.Text = "";
    min.Text = "";
    srednia.Text = "";
    wariancja.Text = "";

}
```

Metoda wyżej odpowiada za czyszczenie okien wyświetlających dane, oraz modyfikuje dostępność pliku ładowania pliku - w przypadku dodania pliku zostaje on zablokowany, natomiast dzięki funkcji czyszczenia zostaje on z powrotem możliwy do wciśnięcia, a tym samym dane ponownie możliwe do wprowadzenia.

```
private void button_zapisz_Click(object sender, EventArgs e)
{
    SaveFileDialog saveFileDialog1 = new SaveFileDialog();
    saveFileDialog1.Filter = "txt files (*.txt)|*.txt";
    if (saveFileDialog1.ShowDialog() != System.Windows.Forms.DialogResult.OK)
    {
        MessageBox.Show("Plik nie został zapisany", "Info", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }

    else
    {

        List<string> zapisz = new List<string> { };

        zapisz.Add("Wartość maksymalna: " + max.Text);
        zapisz.Add("Wartość minimalna: " + min.Text);
        zapisz.Add("Wartość średnia: " + srednia.Text);
        zapisz.Add("Wariancja: " + wariancja.Text);

        try
        {
            System.IO.File.AppendAllLines(saveFileDialog1.FileName, zapisz);
            MessageBox.Show("Zapisano pomyślnie", "Zapisano", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }

        catch (Exception)
        {
            MessageBox.Show("Error", "Error", MessageBoxButtons.OK);
        }

    }
}
```

Metoda odpowiada za zapisywanie pliku z wgranymi próbkami do nowego pliku z wyliczonymi wartościami. Odpowiedzią na zapisanie są okna dialogowe - o pomyślnym zapisaniu, lub w przypadku nieprawidłowości o błędzie.

```
                    //---Przyciski po najechaniu---///
0 references
private void button_zaladuj_MouseMove(object sender, MouseEventArgs e)
{
    panel_left.Height = button_zaladuj.Height;
    panel_left.Top = button_zaladuj.Top;
    label_zaladuj.Visible = true;
    label_wyczysc.Visible = false;
    label_wykres.Visible = false;
    label_zapisz.Visible = false;
}

0 references
private void button_wyswietl_MouseMove(object sender, MouseEventArgs e)
{
    panel_left.Height = button_wyswietl.Height;
    panel_left.Top = button_wyswietl.Top;
    label_zaladuj.Visible = false;
    label_wyczysc.Visible = false;
    label_wykres.Visible = true;
    label_zapisz.Visible = false;
}
```

Dla każdego przycisku zaimplementowana jest funkcja do podświetlania przycisku na którego użytkownik najedzie kursorem.

```
public partial class Wykres : Form
{
    1 reference
    public static Legend legend;
    6 references | 6 references
    public static Series s1, s2;
    6 references
    List<double> probki = new List<double>();
    2 references
    public static string tytul;
    5 references
    public static ChartArea chartArea;
    1 reference
    public Wykres(List<double> data, string text)
    {
        InitializeComponent();
        tytul = text;
        probki = data;
        panel_left.Height = Seria.Height;
        panel_left.Top = Seria.Top;

        chart1.ChartAreas[0].AxisX.ScaleView.Zoomable = true;
        chart1.ChartAreas[0].AxisY.ScaleView.Zoomable = true;

    }
    0 references
    private void Wykres_Load(object sender, EventArgs e)
    {
        legend = chart1.Legends["Legend1"];
        s1 = chart1.Series["Odczyt Temperatur"];
        s2 = chart1.Series["Różnica Temperatur"];
        Title mainTitle = chart1.Titles["Wykres Pomiarów"];
        mainTitle.Text = tytul;
        chartArea = chart1.ChartAreas["ChartArea1"];
        for (int x = 0; x < probki.Count; x++)
            s1.Points.AddXY(x + 1, probki[x]);
        for (int x2 = 1; x2 < probki.Count; x2++)
            s2.Points.AddXY(x2, probki[x2] - probki[x2 - 1]);
    }
```

Jest to początek klasy Wykres, gdzie początkowo deklarowane są potrzebne elementy, a później w metodzie Wykres_Load częściowo wykorzystywane są do określania wartości i nazw na wykresie.

```
private void Legenda_CheckedChanged(object sender, EventArgs e)
{
    if (Legenda.Checked == true)
    {
        chart1.Legends["Legend1"].Enabled = true;
        //s1.Enabled
    }
    else
    {
        chart1.Legends["Legend1"].Enabled = false;
    }
}
0 references
private void Seria_CheckedChanged(object sender, EventArgs e)
{
    if (Seria.Checked == true)
    {
        chart1.Series["Odczyt Temperatur"].Enabled = true;
    }
    else
    {
        chart1.Series["Odczyt Temperatur"].Enabled = false;
    }
}
0 references
private void Seria2_CheckedChanged(object sender, EventArgs e)
{
    if (Seria2.Checked == true)
    {
        chart1.Series["Różnica Temperatur"].Enabled = true;
    }
    else
    {
        chart1.Series["Różnica Temperatur"].Enabled = false;
    }
}
0 references
private void WykresCheck_CheckedChanged(object sender, EventArgs e)
{
    if (WykresCheck.Checked == true)
    {
        s1.ChartType = SeriesChartType.Column;
    }
    else
    {
        s1.ChartType = SeriesChartType.Spline;
    }
}
```

Metody odpowiadające za włączanie/wyłączanie lub pokazywanie/ukrywanie odpowiednich elementów na wykresie.

```
private void Legenda_MouseMove(object sender, MouseEventArgs e)
{
    panel_left.Height = Legenda.Height;
    panel_left.Top = Legenda.Top;
}
0 references
private void Seria_MouseMove(object sender, MouseEventArgs e)
{
    panel_left.Height = Seria.Height;
    panel_left.Top = Seria.Top;
}
0 references
private void Siatka_MouseMove(object sender, MouseEventArgs e)
{
    panel_left.Height = Siatka.Height;
    panel_left.Top = Siatka.Top;
}
0 references
private void WykresCheck_MouseMove(object sender, MouseEventArgs e)
{
    panel_left.Height = WykresCheck.Height;
    panel_left.Top = WykresCheck.Top;
}
0 references
private void kolorSerii_MouseMove(object sender, MouseEventArgs e)
{
    panel_left.Height = kolorSerii.Height;
    panel_left.Top = kolorSerii.Top;
}
0 references
private void Seria2_MouseMove(object sender, MouseEventArgs e)
{
    panel_left.Height = Seria2.Height;
    panel_left.Top = Seria2.Top;
}
0 references
private void WykresRoznicaCheck_MouseMove(object sender, MouseEventArgs e)
{
    panel_left.Height = WykresRoznicaCheck.Height;
    panel_left.Top = WykresRoznicaCheck.Top;
}
```

Podobnie jak w klasie Form1, są to metody odpowiadające za podświetlanie przycisku (poprzez pokazanie odpowiedniego elementu label przy przyciskach). Jest to element stricte wizualny.

```
private void chart1_MouseWheel(object sender, MouseEventArgs e)
{
    var chart = (Chart)sender;
    var xAxis = chart.ChartAreas[0].AxisX;
    var yAxis = chart.ChartAreas[0].AxisY;

    try
    {
        if (e.Delta < 0) // Scrolled down.
        {
            xAxis.ScaleView.ZoomReset();
            yAxis.ScaleView.ZoomReset();
        }
        else if (e.Delta > 0) // Scrolled up.
        {
            var xMin = xAxis.ScaleView.ViewMinimum;
            var xMax = xAxis.ScaleView.ViewMaximum;
            var yMin = yAxis.ScaleView.ViewMinimum;
            var yMax = yAxis.ScaleView.ViewMaximum;

            var posXStart = xAxis.PixelPositionToValue(e.Location.X) - (xMax - xMin) / 4;
            var posXFinish = xAxis.PixelPositionToValue(e.Location.X) + (xMax - xMin) / 4;
            var posYStart = yAxis.PixelPositionToValue(e.Location.Y) - (yMax - yMin) / 4;
            var posYFinish = yAxis.PixelPositionToValue(e.Location.Y) + (yMax - yMin) / 4;

            xAxis.ScaleView.Zoom(posXStart, posXFinish);
            yAxis.ScaleView.Zoom(posYStart, posYFinish);
        }
    }
    catch { }
}
```

Metoda odpowiadająca za możliwość przybliżania wykresu w aplikacji. Charakteryzuje się tym, że przybliżany jest jedynie wykres, a nie całe okno. Niestety funkcja ta wymaga dopracowania, gdyż możliwość przybliżania jest nieco nieudolna i mało praktyczna, niemniej spełnia swoje zadanie - wykres zostaje przybliżony i odpowiednio przeskalowany.