ZAAWANSOWANE PROGRAMOWANIE
LAB05: WYKRESY

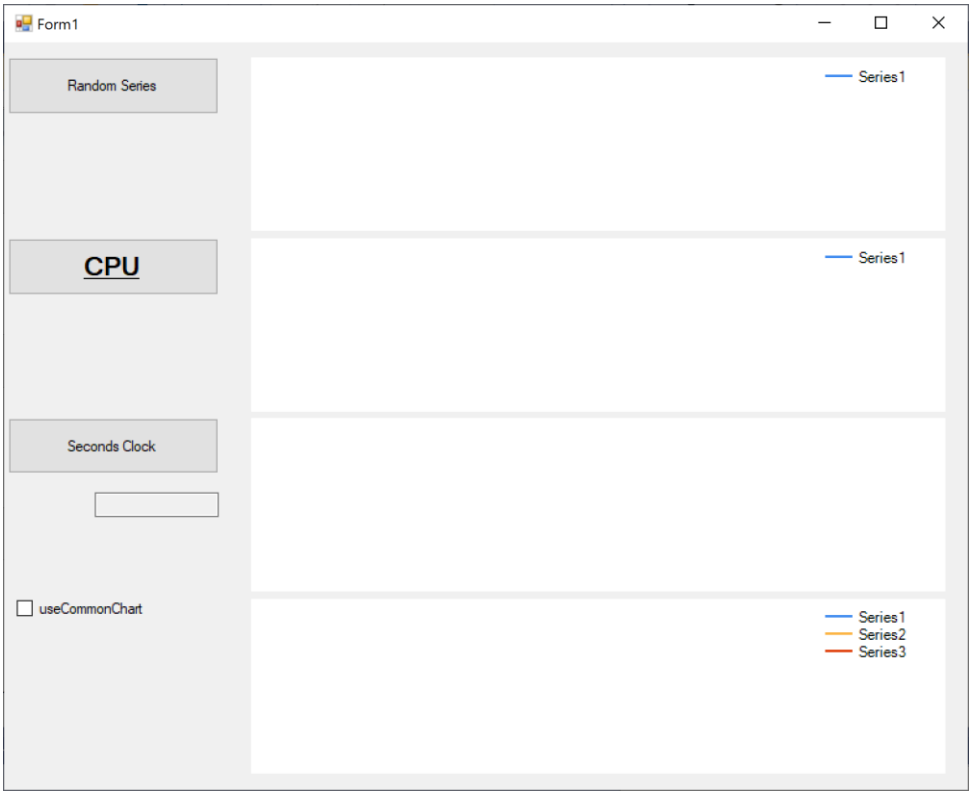
Data: 5.11.2023

LABORATORIUM

Projekt dostępny na stronie pod adresem:

http://www.cs.put.poznan.pl/mradom/teaching/laboratories/ZaawProg/WindowsFormsApp_Charts.7z

Program stanowi prezentację podstawowych możliwości wyświetlania danych na wykresach, przede wszystkich uwzględniając przesyłanie do nich danych z innych wątków. Okno wygląda następująco:



Dolny **CheckBox** o nazwie **useCommonChart** odpowiada za to, że przyciski poza wyświetlaniem danych na swoim wykresie (na prawo od przycisku każdy), będą też aktualizować stan wykresu dolnego, zawierającego wszystkie trzy serie danych.

Dodatkowa uwaga odnośnie serii danych (Series) – tutaj w całym programie używamy Spline (`s.ChartType = System.Windows.Forms.DataVisualization.Charting.SeriesChartType.Spline`) do którego dodajemy wartości np. poprzez `chartRandom.Series["Series1"].Points.AddY(value);`

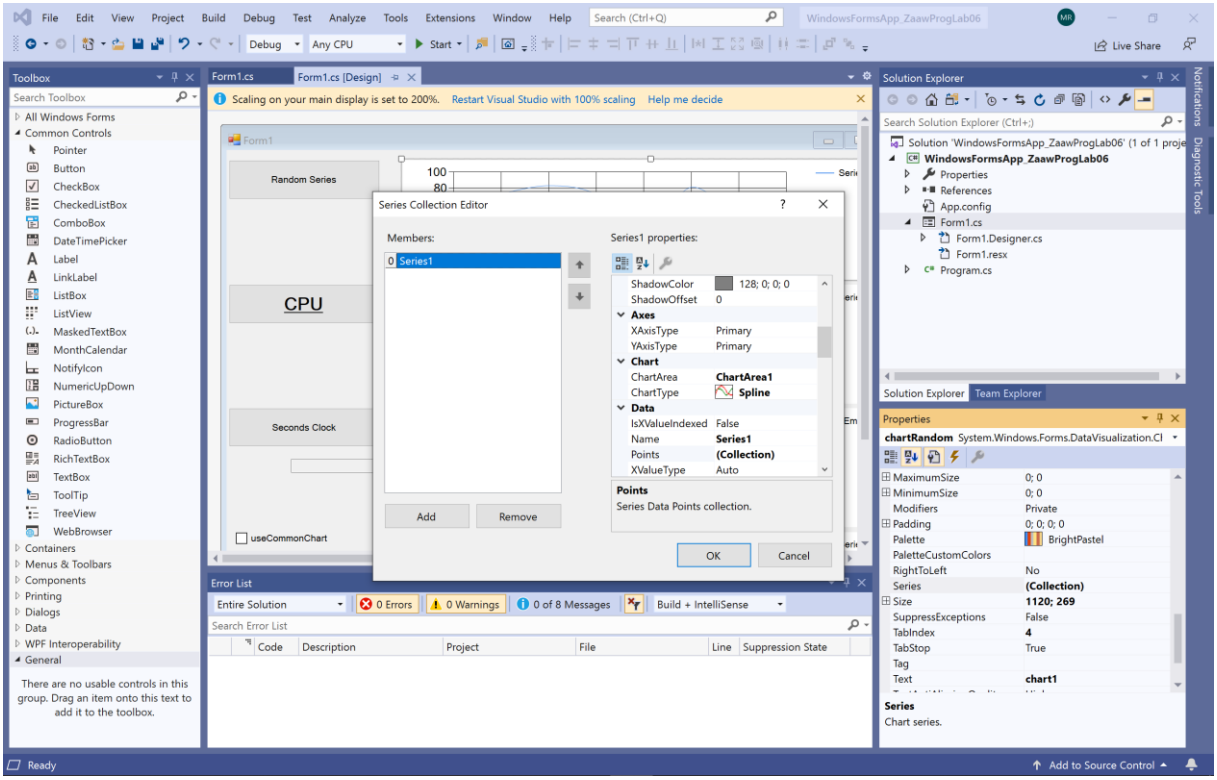
. Należy pamiętać, że jeśli zmienimy typ wykresu (np. na paskowy, kołowy, itd.) to zmieniają się także metody dodawania danych (np. niekoniecznie jest to wtedy Points.AddY(…)).

PRZYCISK **RANDOMSERIES** TO NAJPROSTSZY PRZYKŁAD PRZESYŁANIA DANYCH NA WYKRES.

Przesyła on dane z tego samego wątku okna do elementu tego okna. Kod jest następujący:

```
Random gen = new Random();
for (int i = 0; i < 100; i++) {
     int value = gen.Next(100);
     Thread.Sleep(50);
     chartRandom.Series["Series1"].Points.AddY(value);
     chartRandom.Update();
     if(checkBox1.Checked) {
          chartWspolny.Series["Series2"].Points.AddY(value);
          chartWspolny.Update();
     }
     Application.DoEvents();
 }
```

W pętli na 100 kroków generowana jest losowa wartość od 0 do 99 i posyłana do odpowiedniego obiektu **Series** – nazwa np. „**Series1**” odwołuje się do istniejącej już w ramach wykresu – obiektu **Chart** o nazwie **chartRandom** (górny wykres). Ręczne tworzenie serii danych umożliwia kliknięcie właściwości Series – pojawi się okno dodawania serii, tak jak np. miało to miejsce przy dodawaniu nowych zakładek do obiektu **TabControl** wcześniej na zajęciach.



Ponieważ kod przycisku **RandomSeries** nakazuje w pętli zasypianie wątkowi głównego (bo w jego ramach działa) co krok na 50ms (1/20 sekundy), w pętli musiała się też znaleźć instrukcja:

```
Application.DoEvents();
```

Ponieważ bez niej ciężko byłoby się „doklikać” np. do innych przycisków, dopóki nie skończyłoby się działanie pętli przycisku **RandomSeries**.

Ogólnie jednak mówiąc, jest to przykład negatywny – jeżeli w programie będzie istnieć inny wątek (np. obliczeń heurystyki), to komunikacja z wykresem musi się odbywać inaczej.

DRUGI PRZYCISK BĘDZIE NA DRUGIM WYKRESIE WYŚWIETLAĆ STOPIEŃ UŻYCIA PROCESORA.

Kod tego przycisku:

```
        private void button1_Click(object sender, EventArgs e) {
            cpuThread = new Thread(new ThreadStart(this.getPerformanceCounters));
            cpuThread.IsBackground = true;
            cpuThread.Start();
        }
```

Tworzy nowy wątek, do którego „wkłada” metodę `getPerformanceCounters`. Jej kod:

```
private void getPerformanceCounters() {
     var cpuPerfCounter = new PerformanceCounter("Processor Information", "% Proce
ssor Time", "_Total");
     while (true) {
        cpuArray[cpuArray.Length- 1] = Math.Round(cpuPerfCounter.NextValue(), 0);
        Array.Copy(cpuArray, 1, cpuArray, 0, cpuArray.Length - 1);
        if (cpuChart.IsHandleCreated) {
             this.Invoke((MethodInvoker)delegate { UpdateCpuChart(); });
             Application.DoEvents();
        } else {
                    //......
        }
        Thread.Sleep(500);
    }
}
```

To jak dokładnie jest pobierany stan procesora jest tutaj najmniej istotne. Najważniejsza jest część:

```
if (cpuChart.IsHandleCreated) {
        this.Invoke((MethodInvoker)delegate { UpdateCpuChart(); });
```

Jak widać, komunikacja wątku utworzonego przez przycisk button1_click (ten od stanu CPU) odbywa się poprzez delegata, który „dostaje” do srodka metodę UpdateCpuChart. To metoda lokalna Form1, czyli taka, wewnątrz której mogą być instrukcji normalnie odwołujące się do obiektu Chart (wykresu).

```
        private void UpdateCpuChart() {
            cpuChart.Series["Series1"].Points.Clear();
            chartWspolny.Series["Series3"].Points.Clear();
            for (int i = 0; i < cpuArray.Length - 1; ++i) {
                cpuChart.Series["Series1"].Points.AddY(cpuArray[i]);
                if (checkBox1.Checked)
                    chartWspolny.Series["Series3"].Points.AddY(cpuArray[i]);
            }
        }
```

Innymi słowy: nowy wątek utworzony przyciskiem button1, tworzy delegata, do którego wchodzi metoda **UpdateCpuChart**. Ten wątek za każdym razem, gdy chce coś wysłać na wykres, wysyła to przez delegata, który posyła dane do metody **UpdateCpuChart**, która normalnie zapełnia wykres. Bezpośrednie umieszczenie

instrukcji z metody **UpdateCpuChart** w kodzie `getPerformanceCounters` skończy się wyjątkiem nieprawidłowej komunikacji między wątkami.

OSTATNI PRZYCISK INICJALIZUJE OBIEKT „STOPERA" TO MIERZENIA CZASU.

```
if (!token) {
   token = true;
   System.Windows.Forms.DataVisualization.Charting.Series s = chartSeconds.Series.Add(
"SeriaTimer");
   s.ChartType = System.Windows.Forms.DataVisualization.Charting.SeriesChartType.Spline;

   TimerCallback timeCB = new TimerCallback(doSmth);
   // Establish timer settings.
   t = new System.Threading.Timer(
      timeCB, // The TimerCallback delegate object.
      "SeriaTimer", // Any info to pass into the called method (null for no info).
      0, // Amount of time to wait before starting (in milliseconds).
    500); // Interval of time between calls (in milliseconds).
}
```

Istotne jest tutaj odwołanie do metody `doSmth` :

```
private void doSmth(object state) {
     int v = DateTime.Now.Second;
     string name = (string)state; //rozpakowanie "MojaSeria" z obiektu state

     chartSeconds.BeginInvoke((Action)(() => {
        chartSeconds.Series[name].Points.AddY(v);
        textBox1.Text = "" + v;
        if(checkBox1.Checked)
             chartWspolny.Series["Series1"].Points.AddY(v);
        //Application.DoEvents();
     }));
 }
```

Tutaj również komunikacja jest poprzez delegata tworzonego przez obiekt `Action` (patrz: wykład o delegatach i wielowątkowości).

Jak się można pod koniec domyślić, zaznaczenie **useCommonChart** spowoduje, że przyciski będą posyłać swoje dane na dolny wspólny wykres. Każdy z nich wypełnia swoją serię danych – wykres dolny ma stworzone 3 serie co można zobaczyć w jego właściwości **Series** w oknie właściwości VS.