Projektowanie Graficznych Interfejsów Użytkownika

Robert Szmurło

System.Windows.Forms

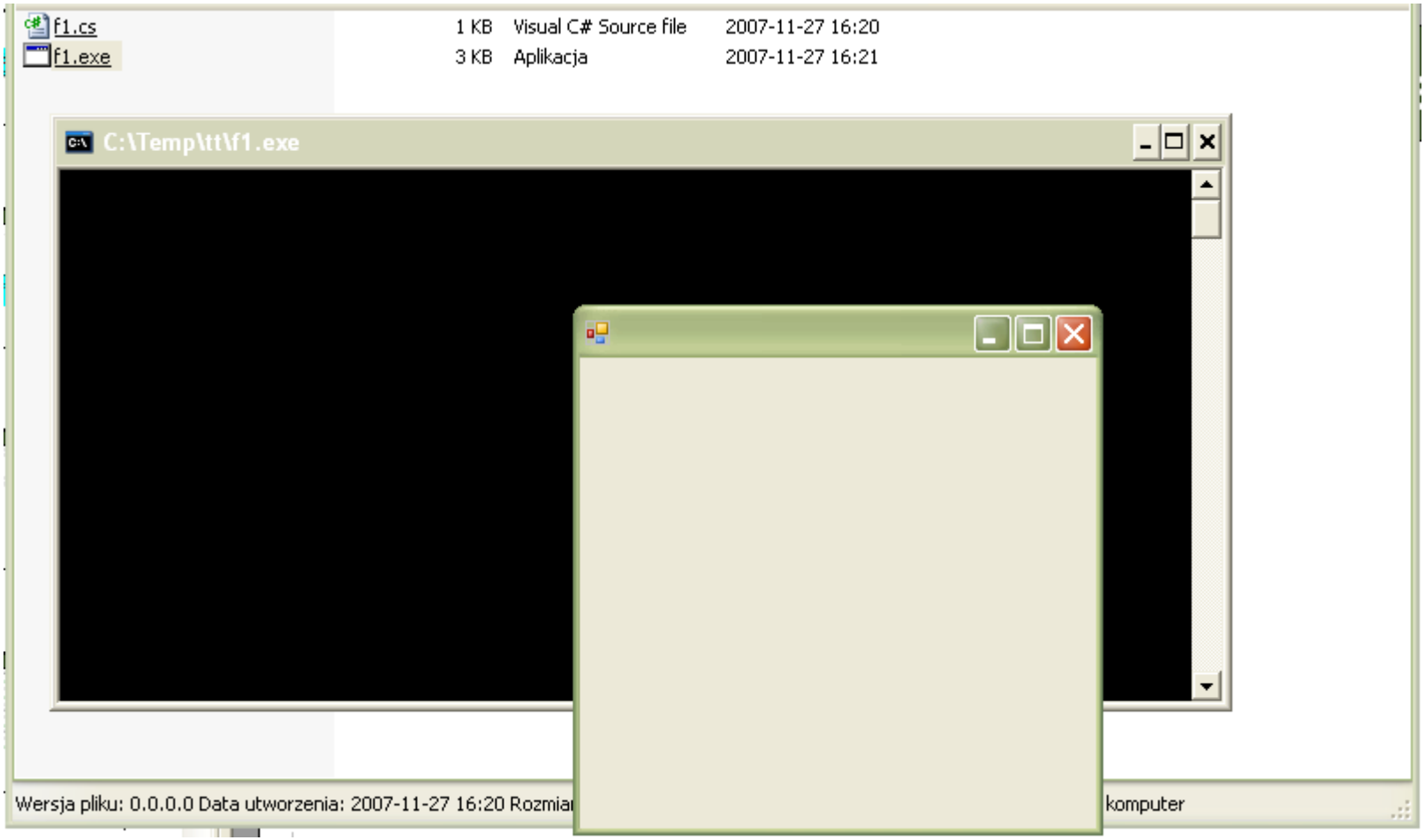
- System.Windows.Forms
  - **Core infrastructure** – podstawowe operacje związane obsługą programu .NET opartego na Forms. (Form, Application, etc.)
  - **Controls** – typy do tworzenia bogatego interfejsu użytkownika (rich Uis: Button, MenuStrip, ProgressBar, DataGridView, etc.), konfigurowalne w trakcie projektowania oraz widoczne po uruchomieniu
  - **Components** – typy, które nie dziedziczą z klasy Control ale udostępniają pewne elementy wizualne. Nie widoczne po uruchomieniu.
  - **Common dialog boxes** – podstawowe okna dialogowe

Manualne tworzenie okna głównego

```
using System;
using System.Windows.Forms;
namespace MyWindowsApp
{
    public class MainWindow : Form
    {
        // Run this application and identify the main window.
        static void Main(string[] args)
        {
            Application.Run(new MainWindow());
        }
    }
}
```

Manualna kompilacja 1

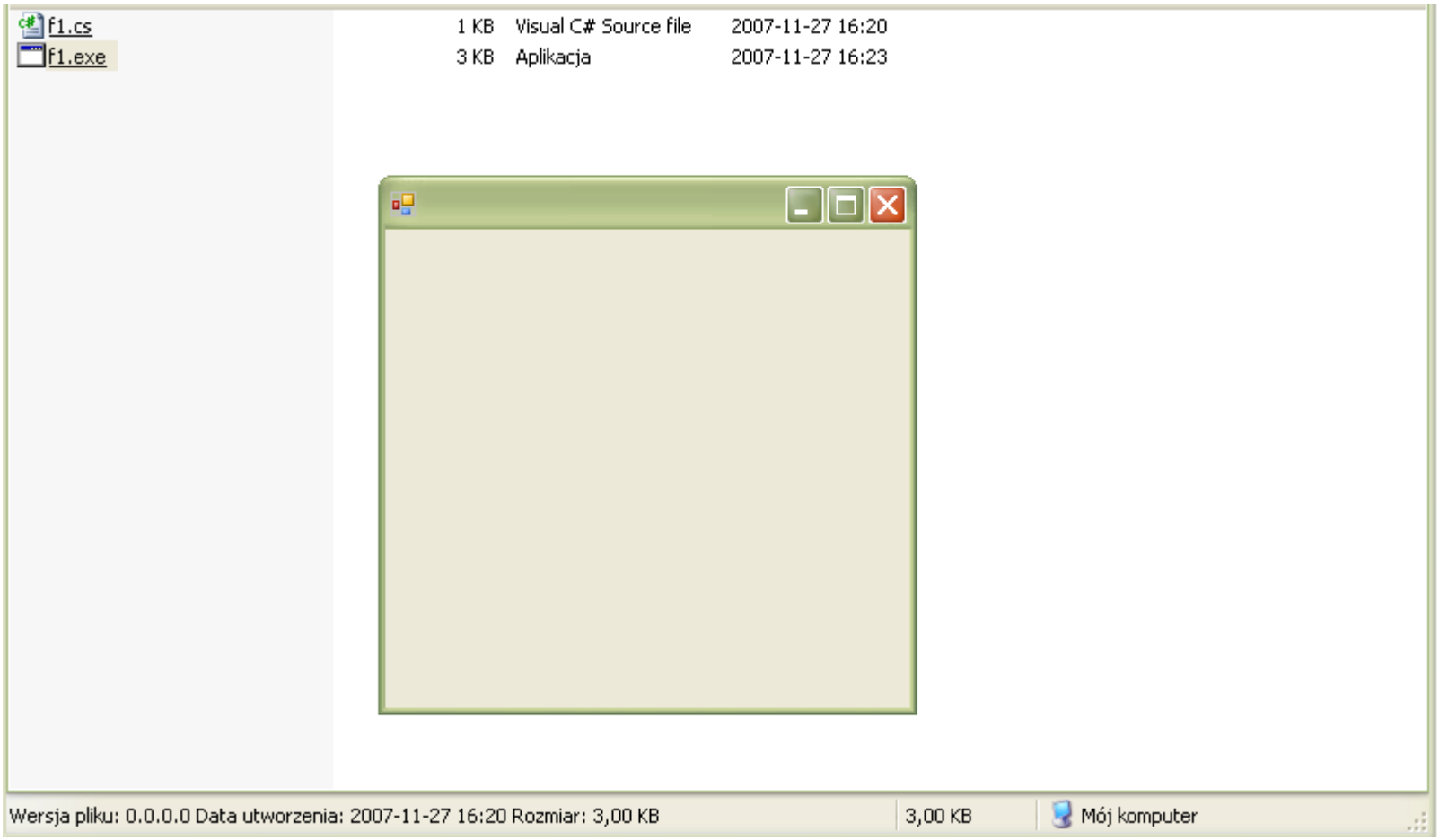
- csc f1.cs



- Niepotrzebne okno linii komend.

Manualna kompilacja 2

- csc /target:winexe f1.cs



- Już nie ma okna linii komend.

Zadania klasy Application

- Wybrane zadania klasy Application
  - **DoEvents()** - podczas długich operacji umożliwia wykonanie komunikatów oczekujących w kolejce
  - **Exit()** - kończy działanie aplikacji
  - **EnableVisualStyles()** - włącza obsługę styli Windows XP (musi być wywołane przed Application.Run()
- Wybrane zdarzenia:
  - **ApplicationExit** – wywoływane tuż przed zamknięciem programu
  - **Idle** – wywoływane po wykonaniu wszystkich zadań w kolejce, tuż przed przejściem w stan oczekiwania
  - **ThreadExit** – wywoływane w tuż przed zakończeniem wątku w aplikacji

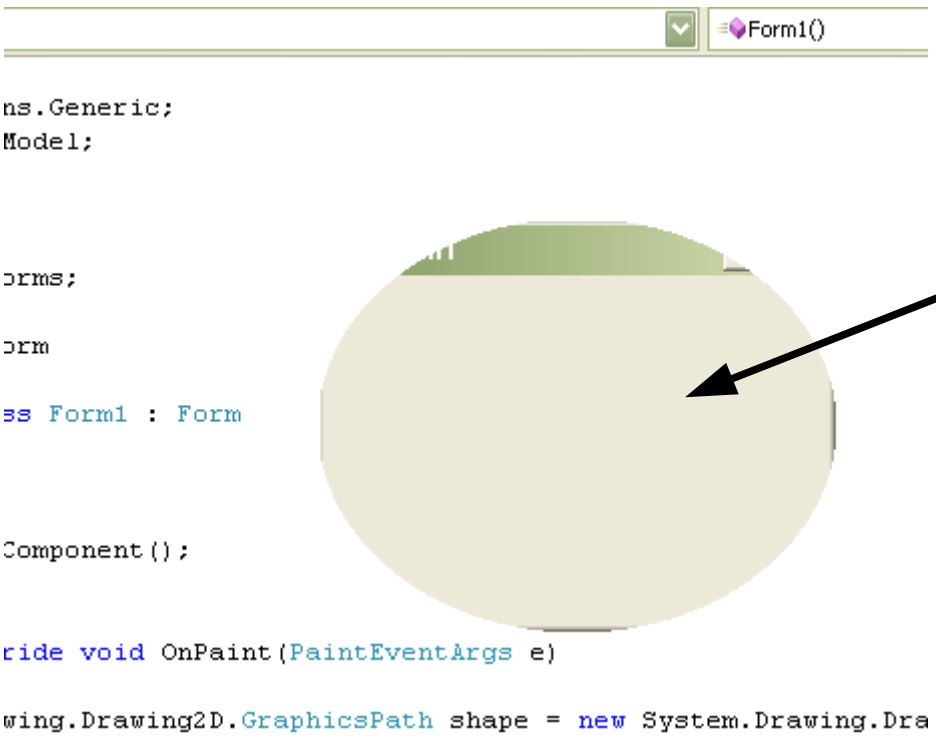
Anatomia formularza

- Klasy bazowe w łańcuchu dziedziczenia klasy Form:
  - **System.Object**
  - **System.MarshalByRefObject** – przy zdalnym wywołaniu typ jest wywoływany przez referencję (a nie kopię)
  - **System.ComponentModel.Component** – wparcie dla edycji właściwości w trakcie projektowania (design time editing)
  - **System.Windows.Forms.Control** – podstawowe elementy interfejsu użytkownika
  - **System.Windows.Forms.ScrollableControl** – wsparcie pasków przesuwania
  - **System.Windows.Forms.ContainerControl** – zarządzanie 'focusem' w kontrolkach, które mogą posiadać wewnątrz siebie inne kontrolki,
  - **System.Windows.Forms.Form** – dowolne okno (MDI Child, dialog, Formularz)

Ciekawostka

- Jak szybko można tworzyć formularze o własnych kształtach..

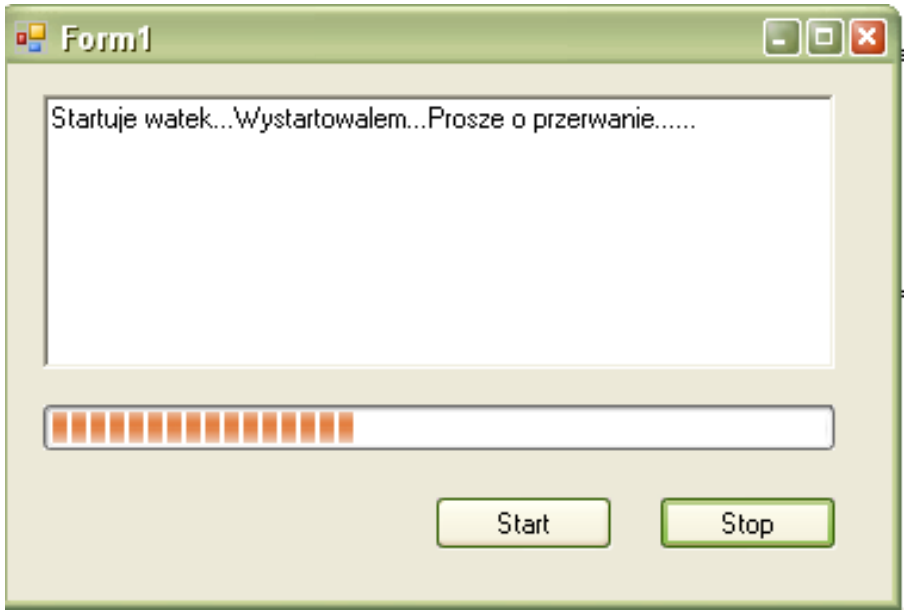
```
protected override void OnPaint(PaintEventArgs e)
{
    System.Drawing.Drawing2D.GraphicsPath shape = new System.Drawing.Drawing2D.GraphicsPath();
    shape.AddEllipse(0, 0, this.Width, this.Height);
    this.Region = new System.Drawing.Region(shape);

}
```



To nie jest fotomontaż :-)

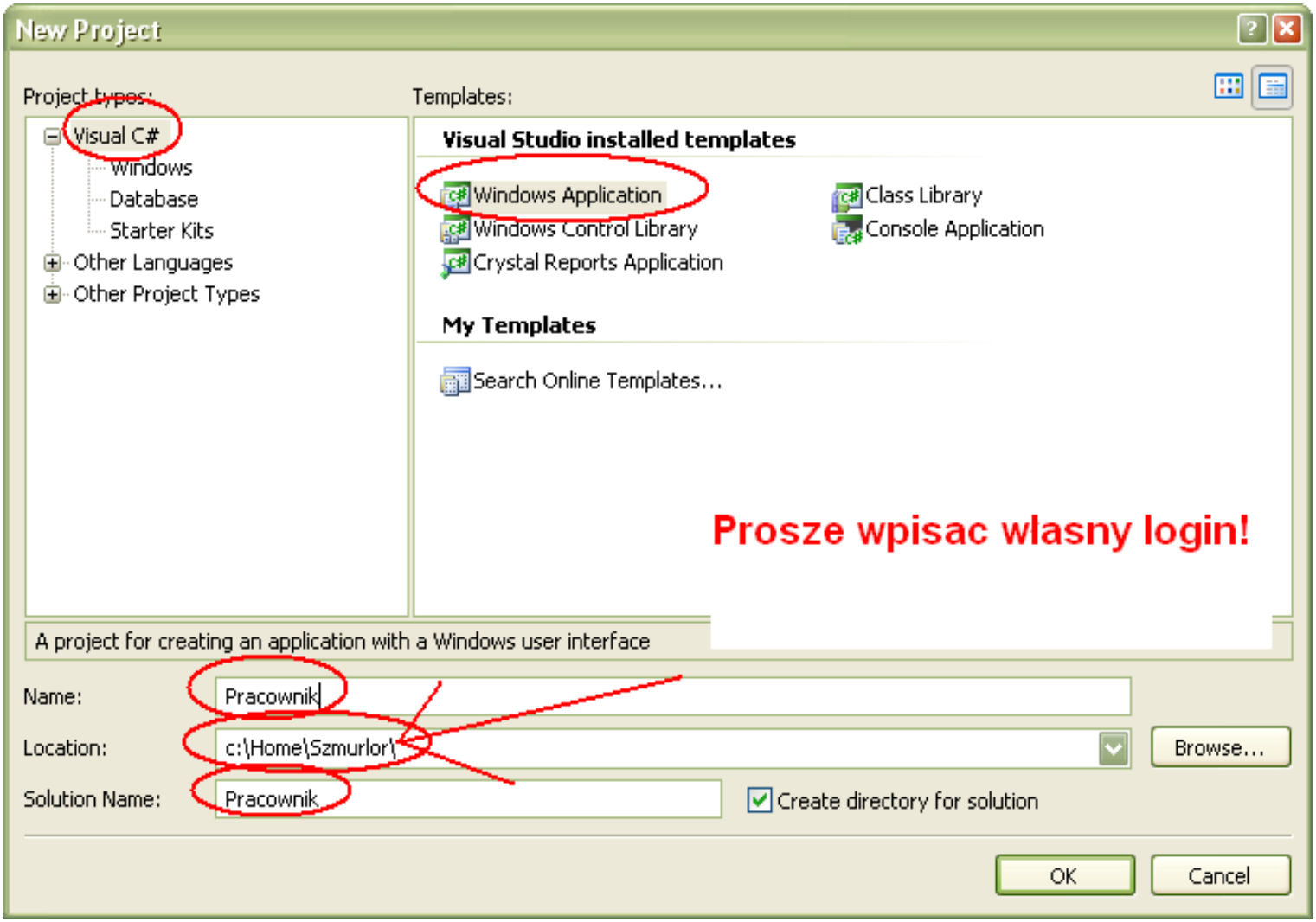
- Przykładowe wykorzystanie wątków dla podniesienia użyteczności aplikacji wykorzystując Wątki i Timery.
- Można pokazać kilka rozwiązań:
  - Oparte na Timerze i funkcji, która jest przez niego wywoływana (bez wywłaszczania)
  - Oparte na Wątku, który co jakiś czas powiadamia obserwatora o postępie prac.
  - Oprate na Wątku, który uaktualnia swój własny licznik, a obserwator samodzielnie sprawdza jego stan – korzystne z punktu widzenia wątkowego. Jedno z najprostszych rozwiązań.
  - Oparte na wątku i wzorcu obserwatora (delegacje).

- Zapewnienie kontroli użytkownika nad aplikacją przy długotrwałych operacjach.
  - a. wykorzystamy dodatkowy wątek, który będzie wykonywał pracę w tle,
  - b. wykorzystamy również Timer jako narzędzie pozwalające obserwować postęp prac.

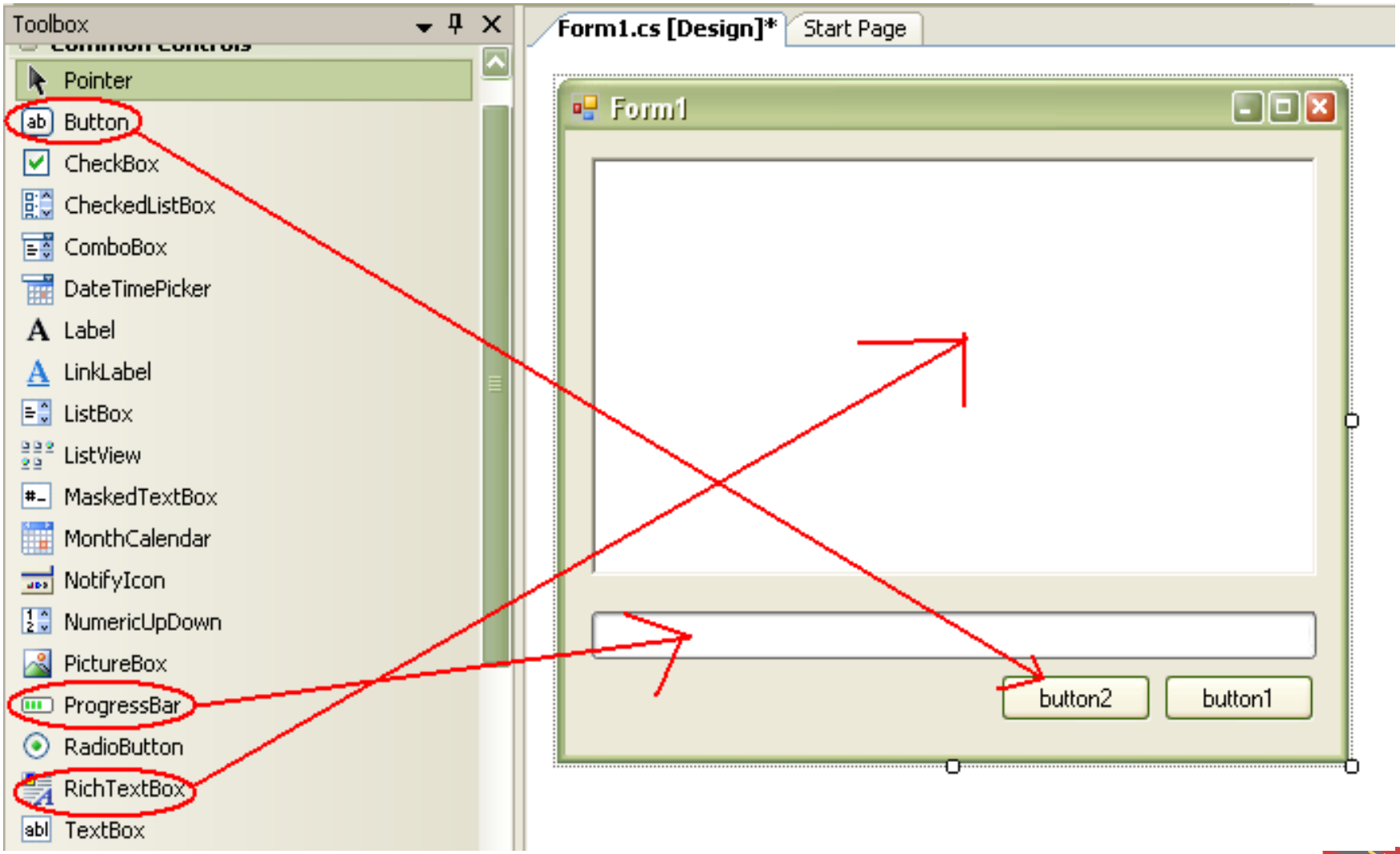


  - Nasz pracownik zakładamy, że jest bezrobotny i nic konkretnego nie musi robić, poza zużywanie czasu :-)

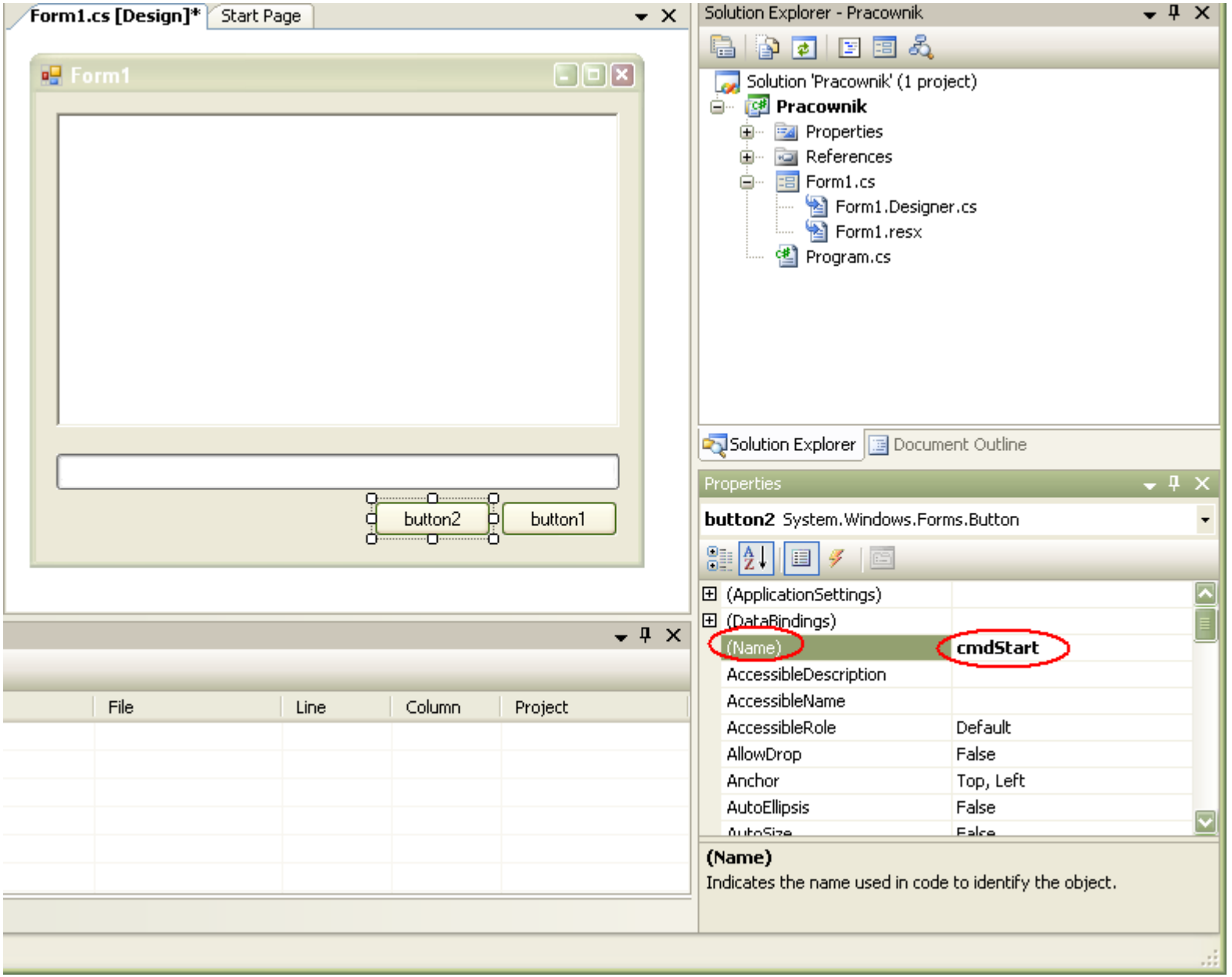
Tworzymy nowy projekt



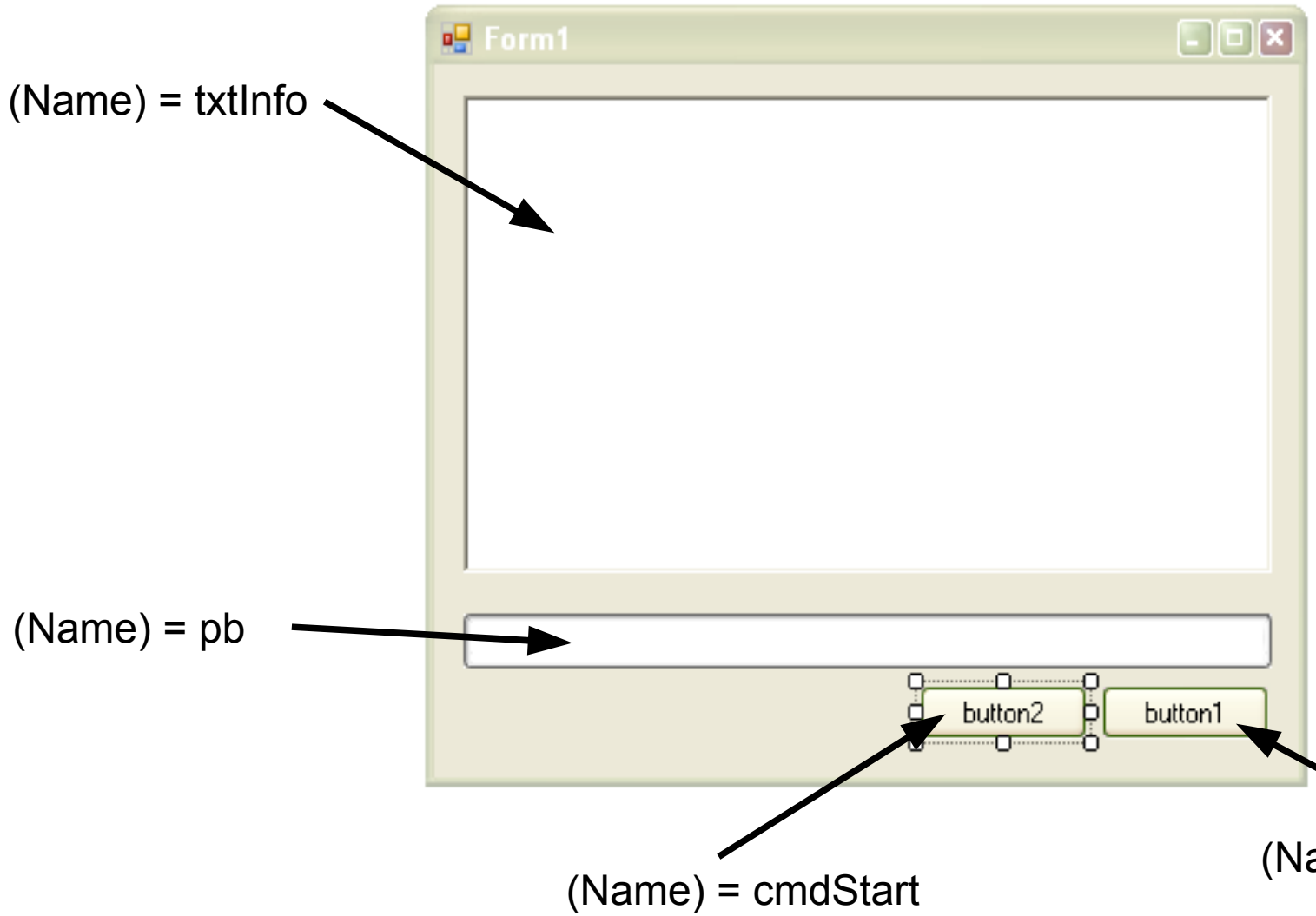
Tworzymy układ GUI



Nazywamy Kontrolki



Ujednolicamy Nazewnictwo



Implementujemy...

- Procedura obsługi start:

```
private void cmdStart_Click(object sender, EventArgs e)
{
    prac = new Pracownik();
    Thread thr = new Thread(prac.doRoboty);
    timer1.Start();

    txtInfo.AppendText("Startuje watek...");
    thr.Start();
    txtInfo.AppendText("Wystartowalem...");

}
```

Zatrzymaj pracownika...

- Zatrzymaj pracownika...

```
private void cmdStop_Click(object sender, EventArgs e)
{
    timer1.Stop();
    txtInfo.AppendText("Prosze o przerwanie...");
    prac.przerwa();
    txtInfo.AppendText("...");
}
```

Klasa pracownika

```
public class Pracownik
{
    public volatile int ileZrobilem = 0;

    public void doRoboty()
    {
        _przerwij = false;
        while (!_przerwij)
        {
            ileZrobilem++;
            Console.WriteLine("Robie..." + ileZrobilem.ToString());
            Thread.Sleep(1000);
        }
    }

    public void przerwa()
    {
        _przerwij = true;
    }

    private volatile bool _przerwij;
}
```

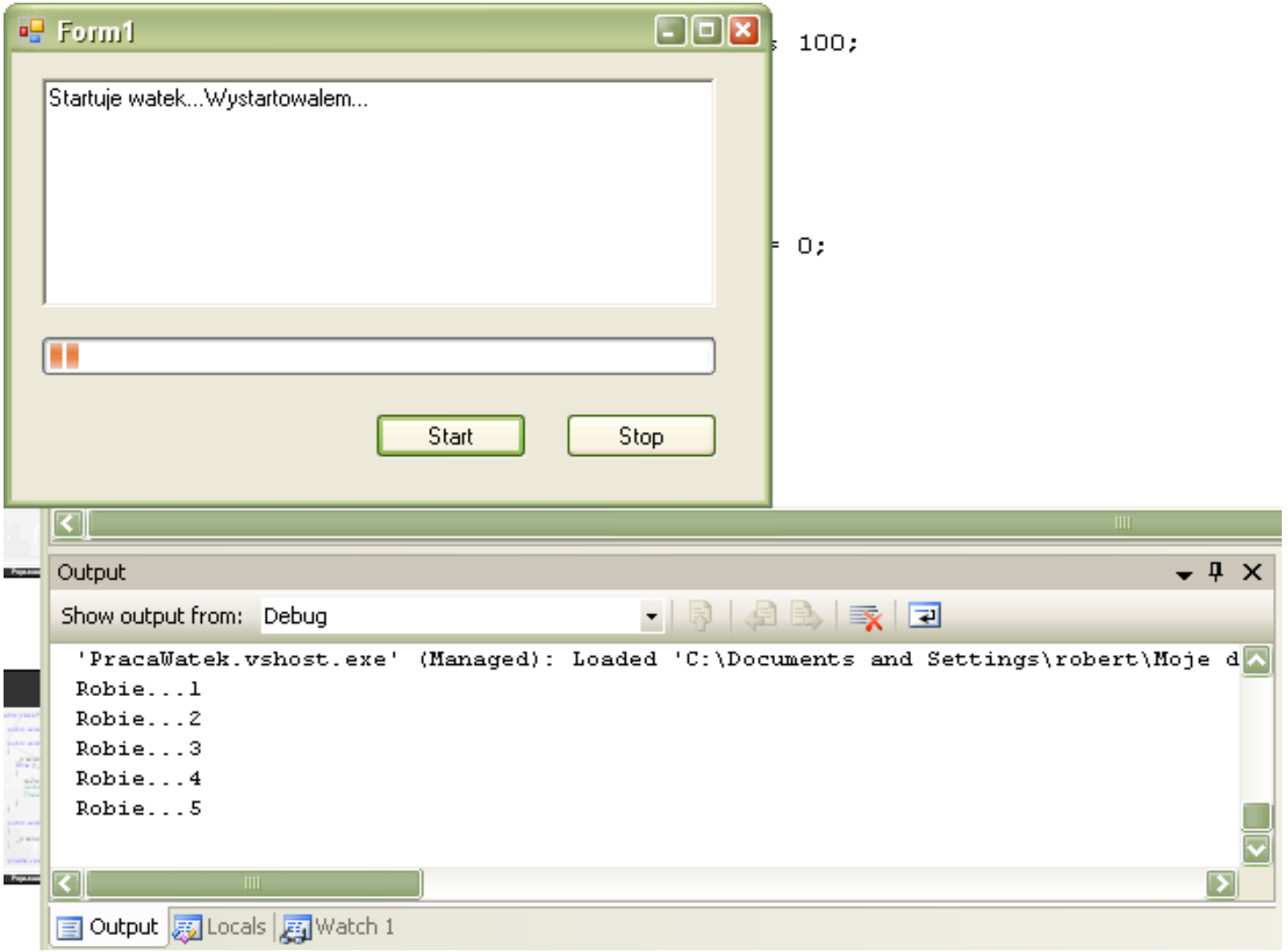
Kompletny kod.

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

namespace WindowsApplication1
{
   public partial class Form1 : Form
   {
      Pracownik prac;

      public Form1()
      {
         InitializeComponent();
      }

      private void cmdStart_Click(object sender, EventArgs e)
      {
         prac = new Pracownik();
         Thread thr = new Thread(prac.doRoboty);
         timer1.Start();

         txtInfo.AppendText("Startuje watek...");
         thr.Start();
         txtInfo.AppendText("Wystartowalem...");

      }

      private void cmdStop_Click(object sender, EventArgs e)
      {
         timer1.Stop();
         txtInfo.AppendText("Prosze o przerwanie...");
         prac.przerwa();
         txtInfo.AppendText("...");
      }
```

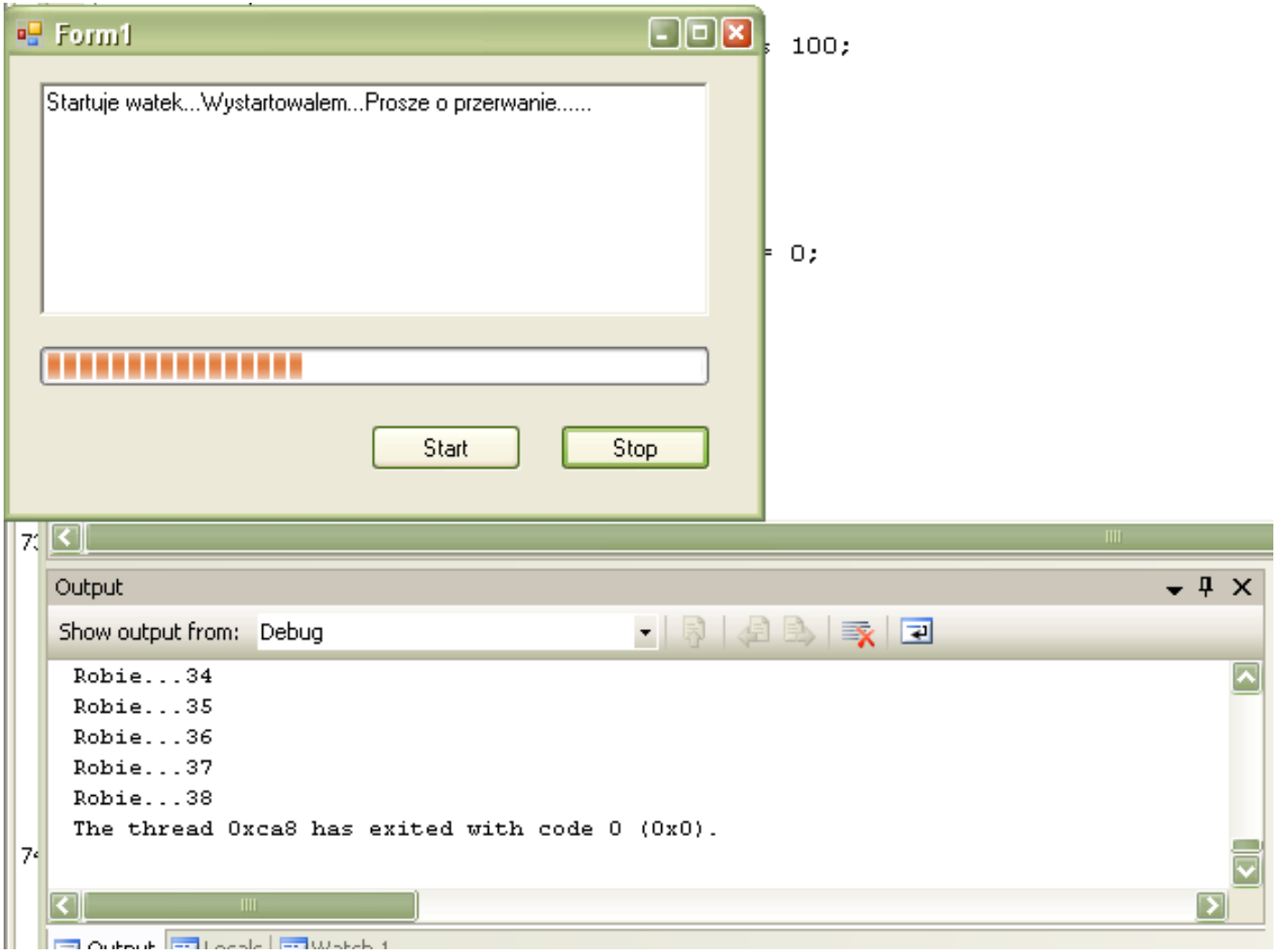
dalszy ciąg...

```
      private void timer1_Tick(object sender, EventArgs e)
      {
         pb.Value = prac.ileZrobilem % 100;
      }
   }

   public class Pracownik
   {
      public volatile int ileZrobilem = 0;

      public void doRoboty()
      {
         _przerwij = false;
         while (!_przerwij)
         {
            ileZrobilem++;
            Console.WriteLine("Robie..." + ileZrobilem.ToString());
            Thread.Sleep(1000);
         }
      }

      public void przerwa()
      {
         _przerwij = true;
      }

      private volatile bool _przerwij;
   }
}
```

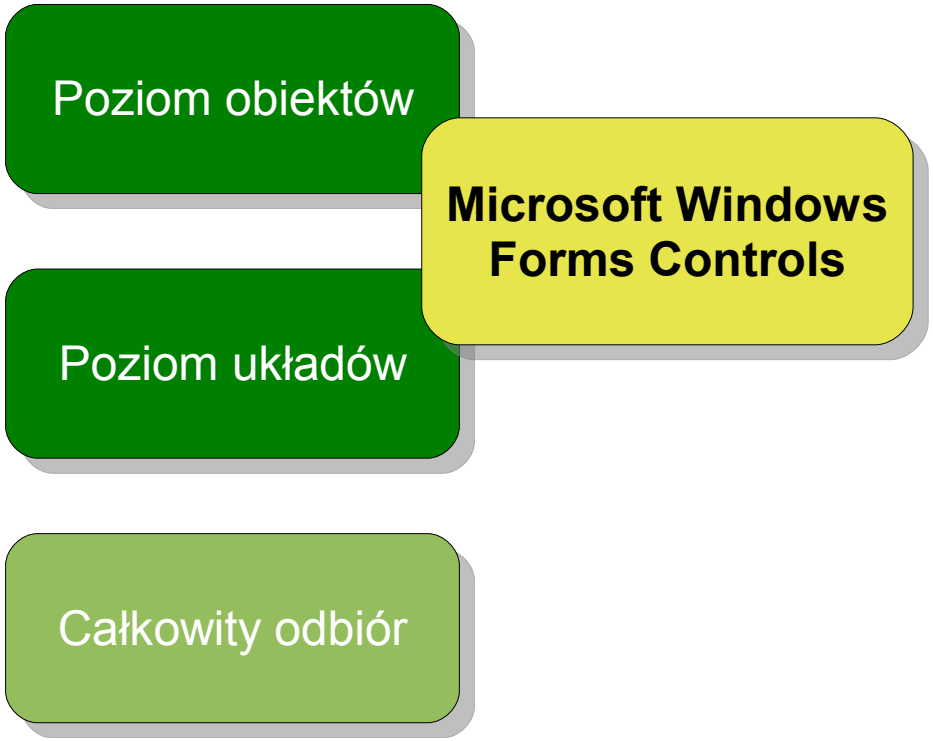
Po uruchomieniu



Po zakończeniu



Projekt wizualny

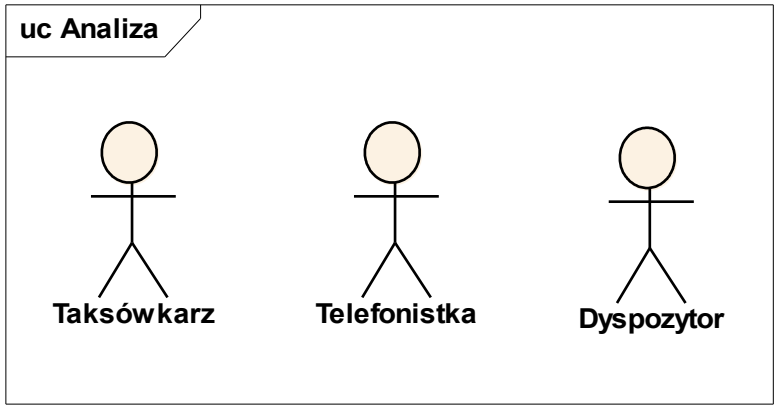


Microsoft Windows Forms Controls jako narzędzie do tworzenia widoków

Studium przypadku „korporacja taksówkowa”

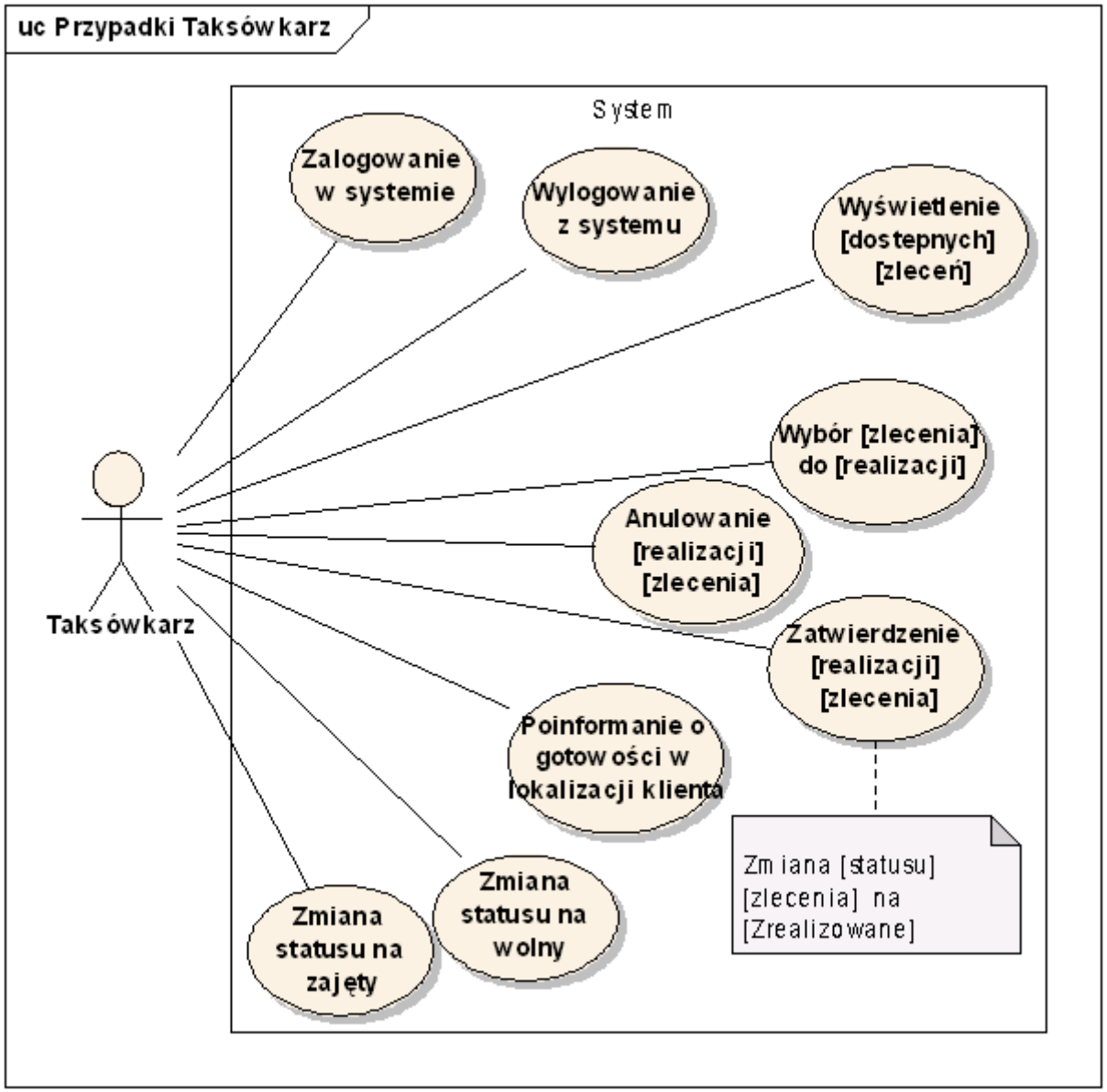
- Aktorzy: telefonistka, dyspozytor, taksówkarz
- Cele użytkownika < przypadki
- Obiekty użytkownika
- Widoki abstrakcyjne obiektów
- Układy abstrakcyjne
- Stworzymy projekty wizualne dla widoków obiektów (tylko dla przykładowych)
- Stworzymy układy projektów wizualnych wykorzystując technologię Microsoft Windows Windows Forms Controls
- Na początku trochę omówimy technologię Microsoft Windows Forms Controls

Aktorzy

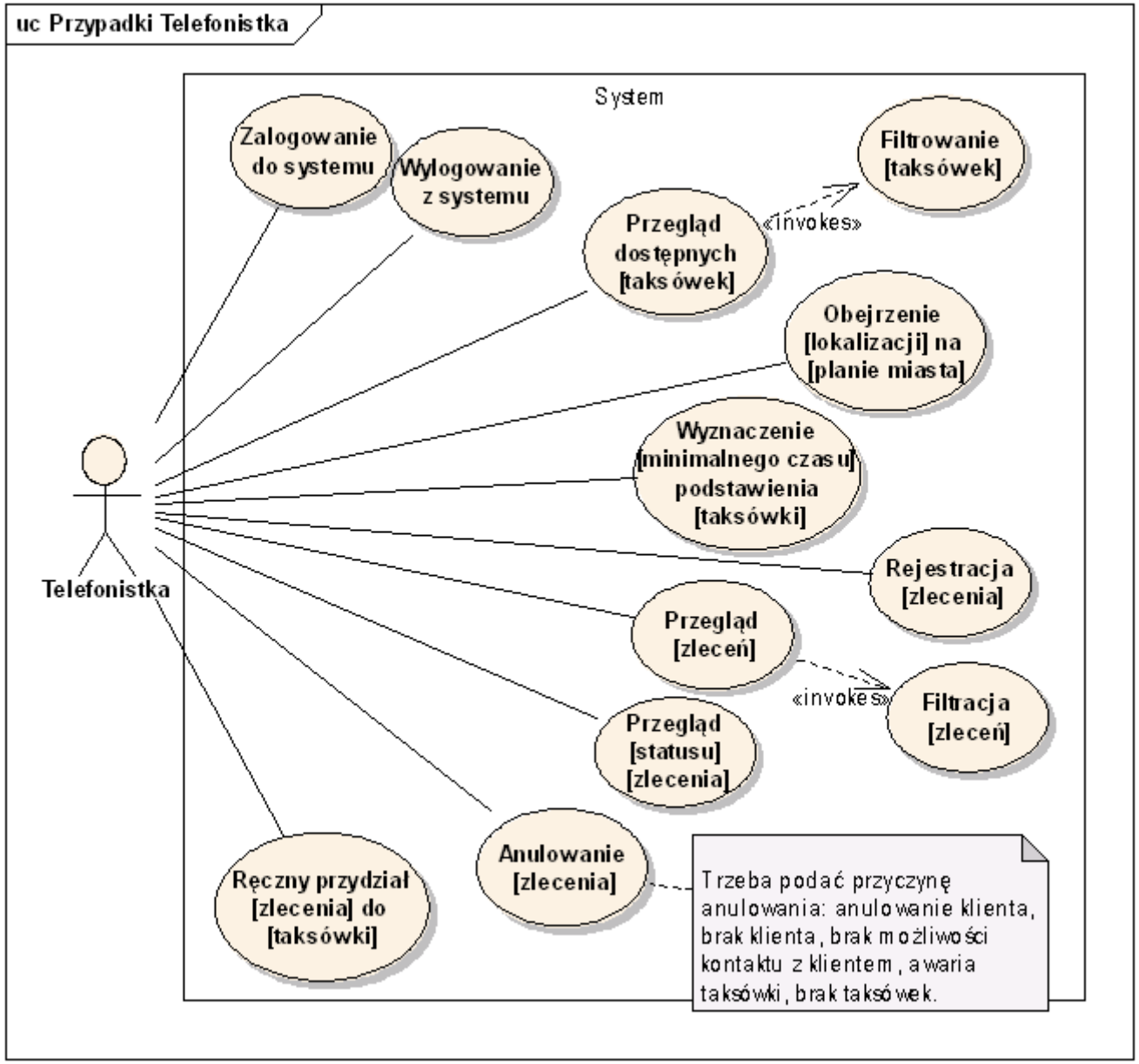


Przypadki użycia: Taksówkarz

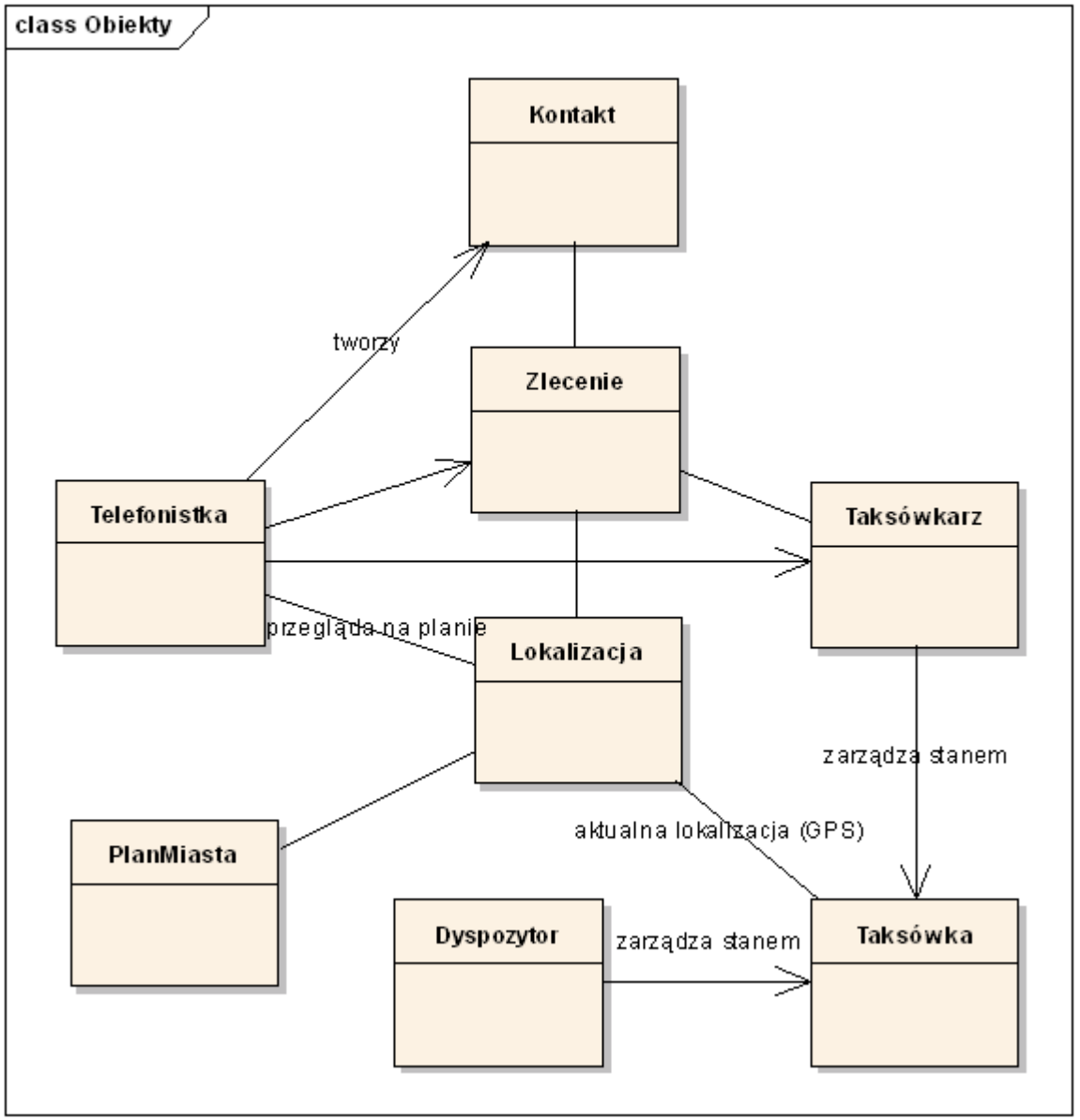
- Przypadki użycia systemu Taksówkarza



Przypadki użycia: Telefonistka

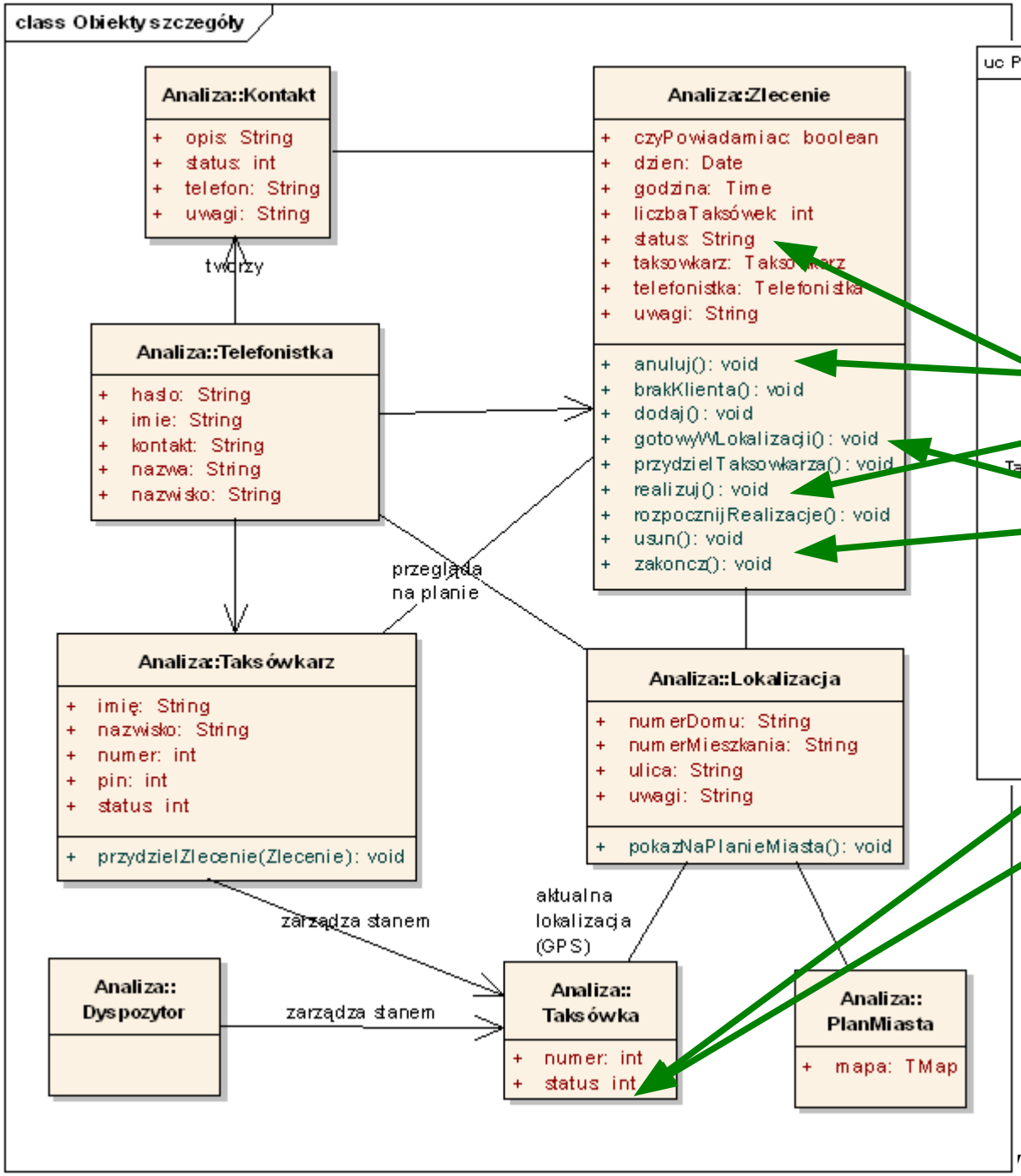


Obiekty użytkownika



Przechodzimy do projektu abstrakcyjnego

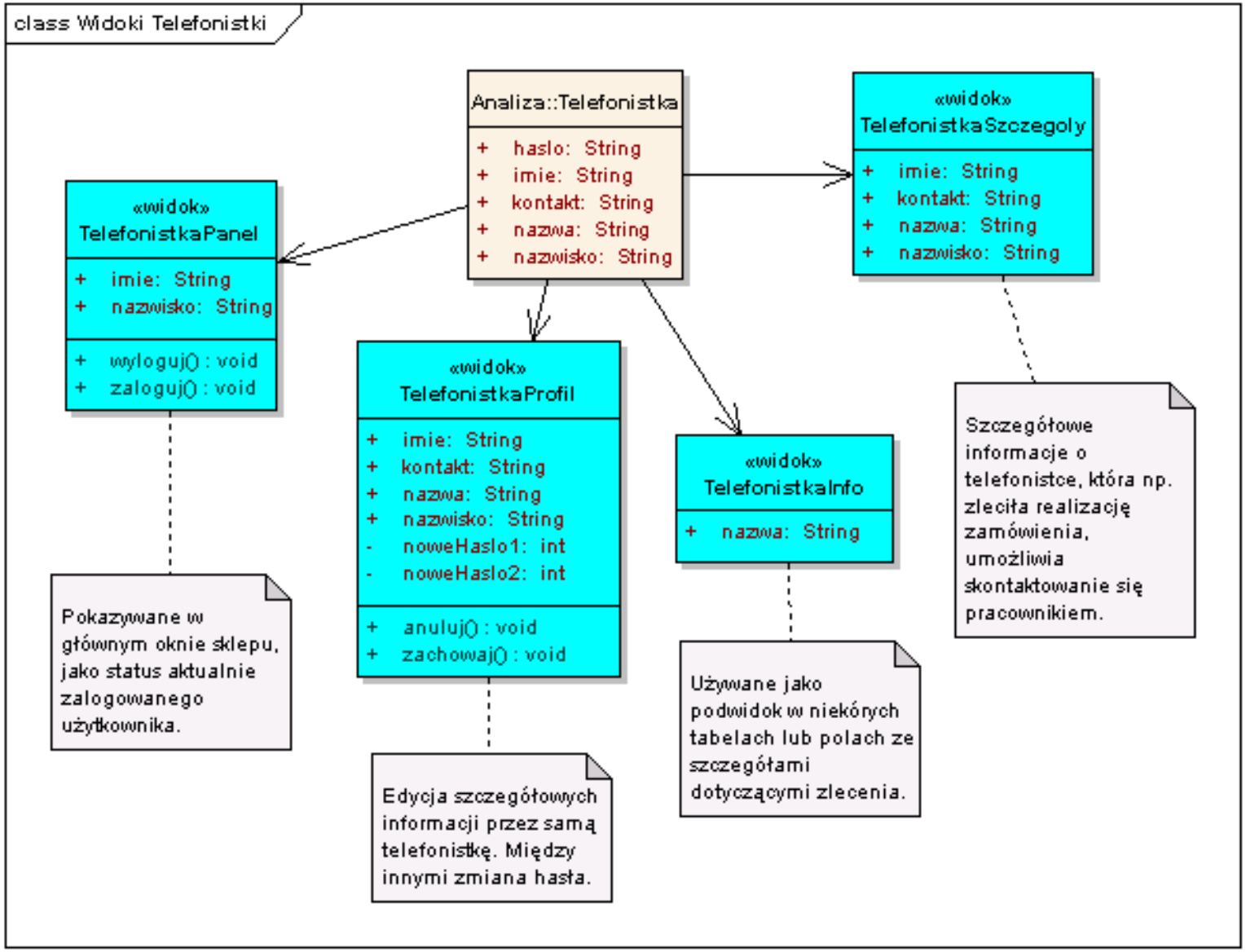
Szczegółowy projekt obiektów



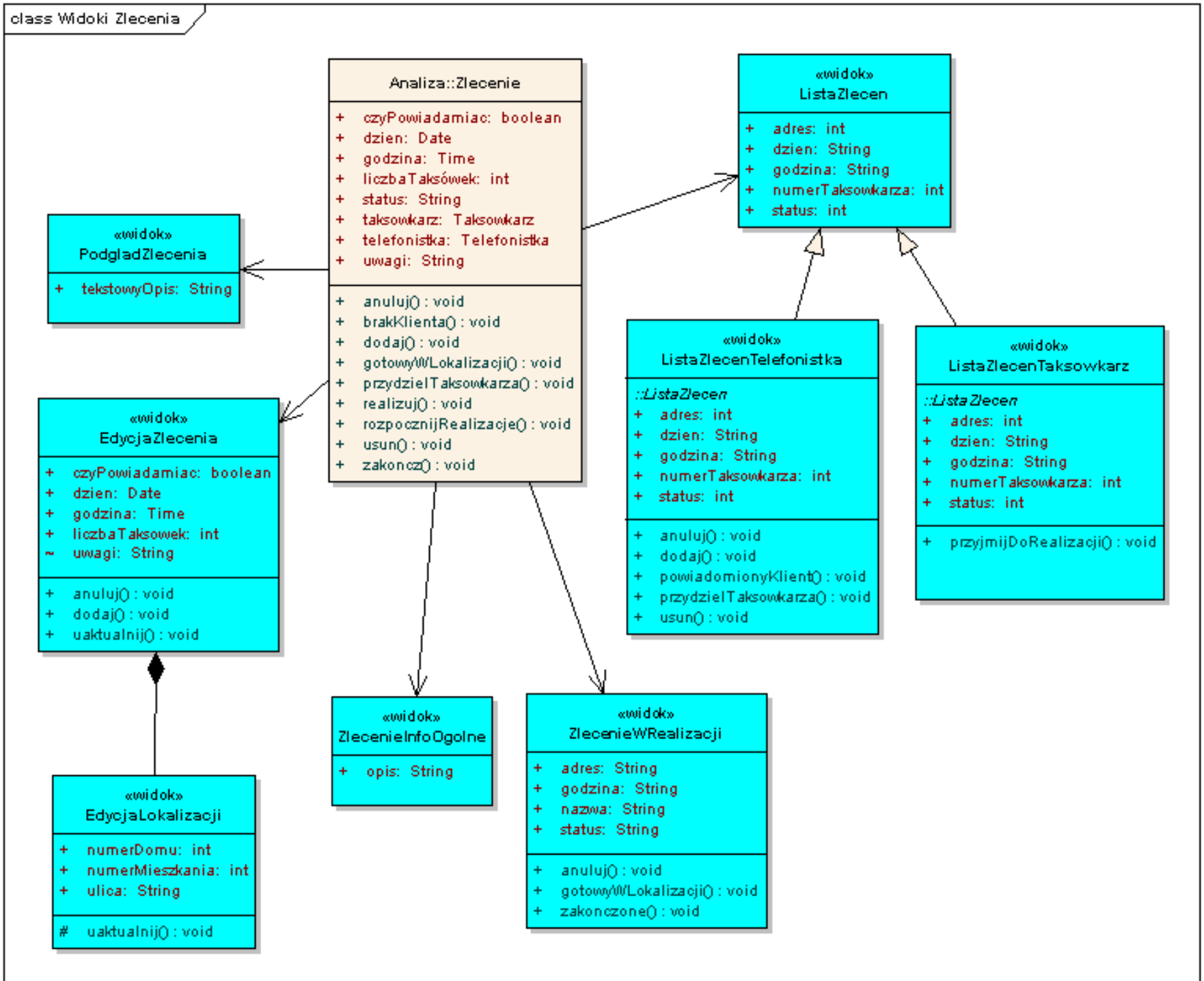
Projekty widoków abstrakcyjnych

- Na początku zdefiniujemy widoki abstrakcyjne na poziomie obiektów.
  - Staramy się przewidzieć jakie widoki danych obiektów będą nam potrzebne.

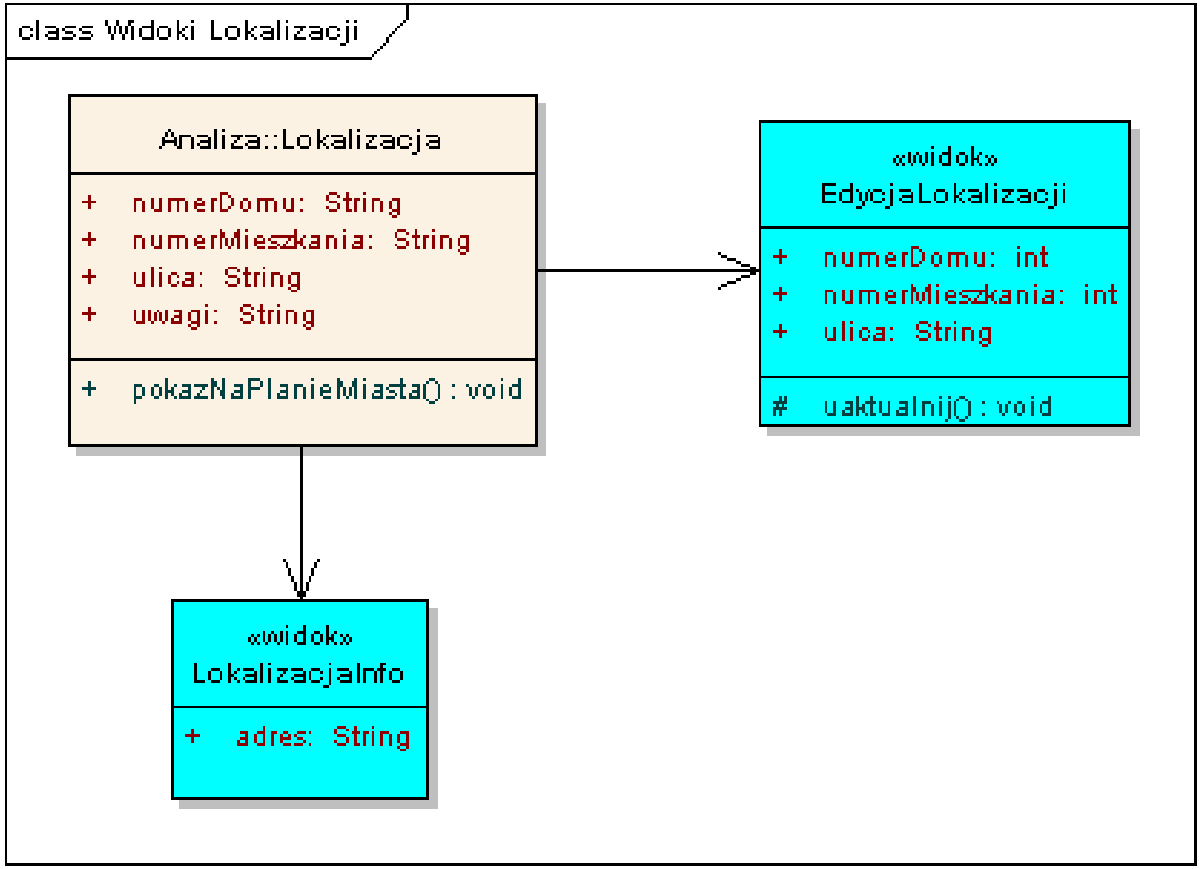
Widoki: Telefonistki



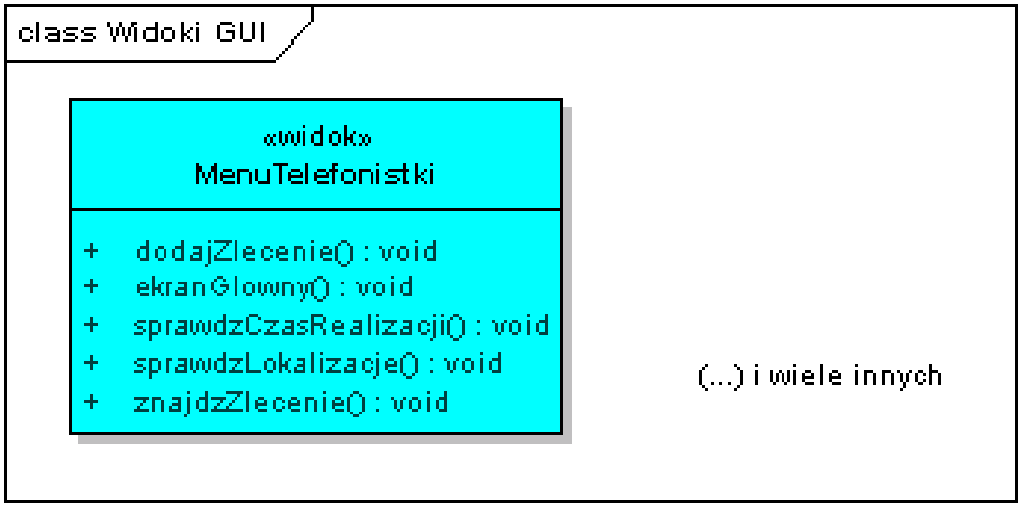
Widoki abstrakcyjne: Zlecenie



Widoki abstrakcyjne: Lokalizacja

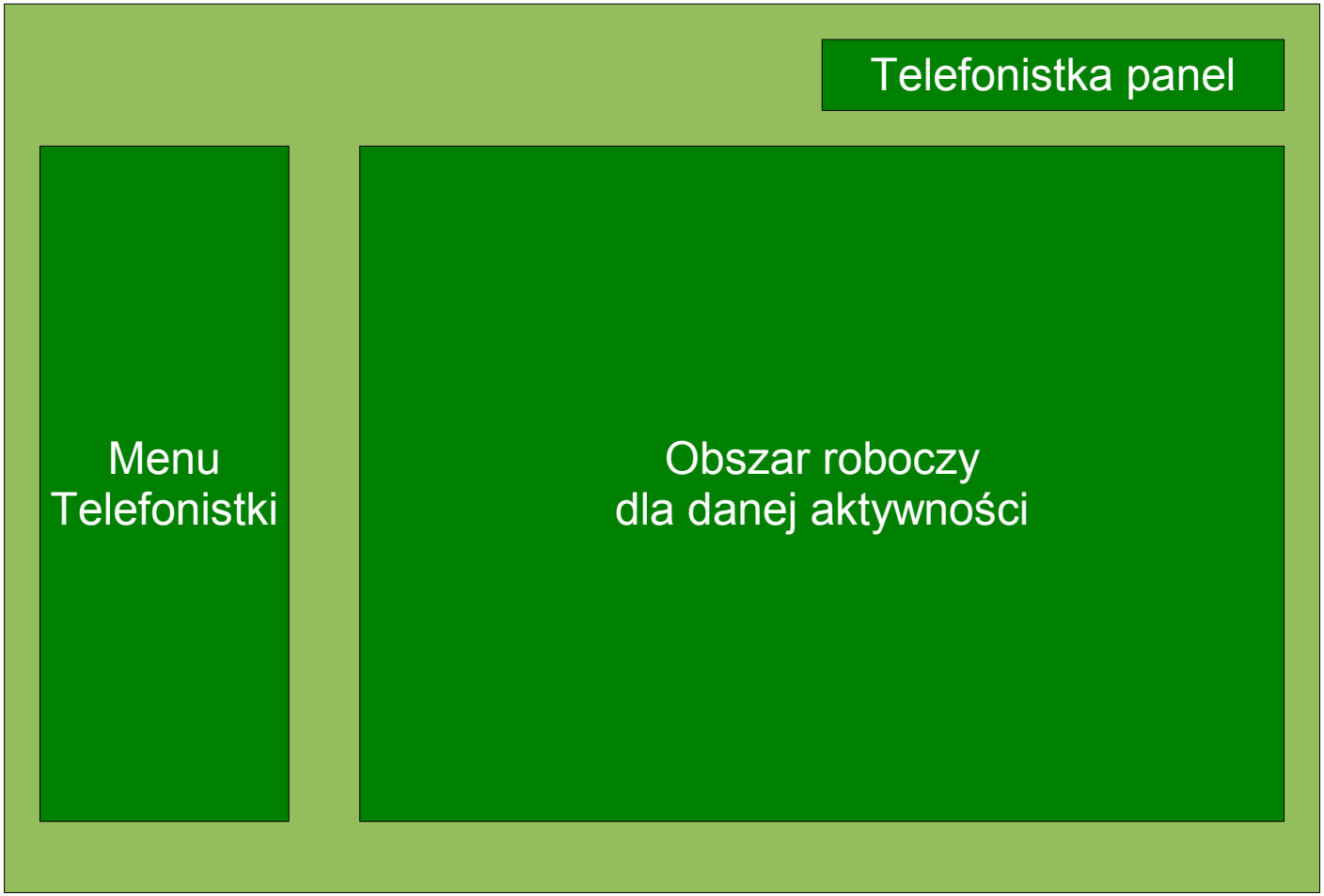


Widoki związane z GUI

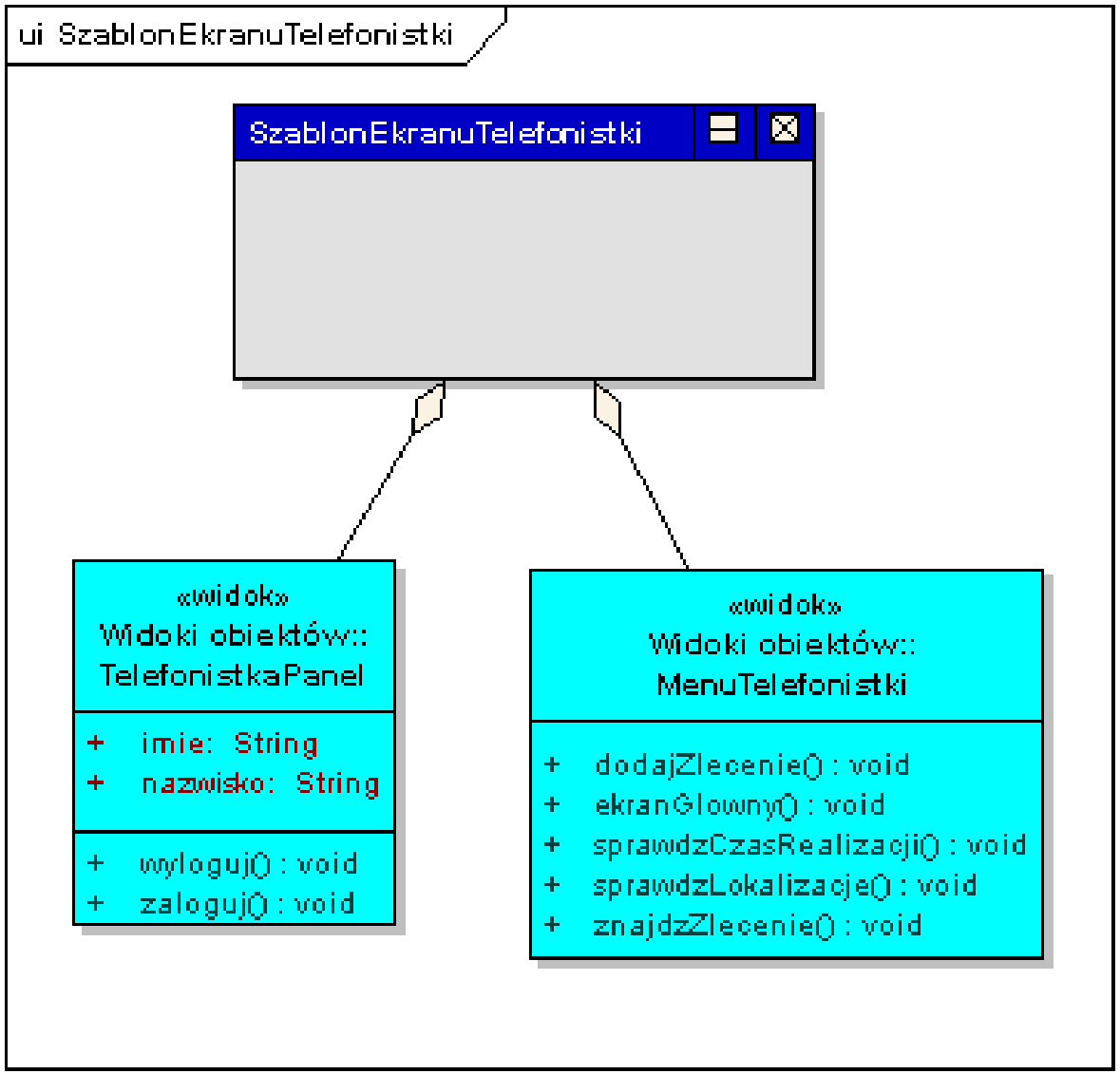


Układy abstrakcyjne

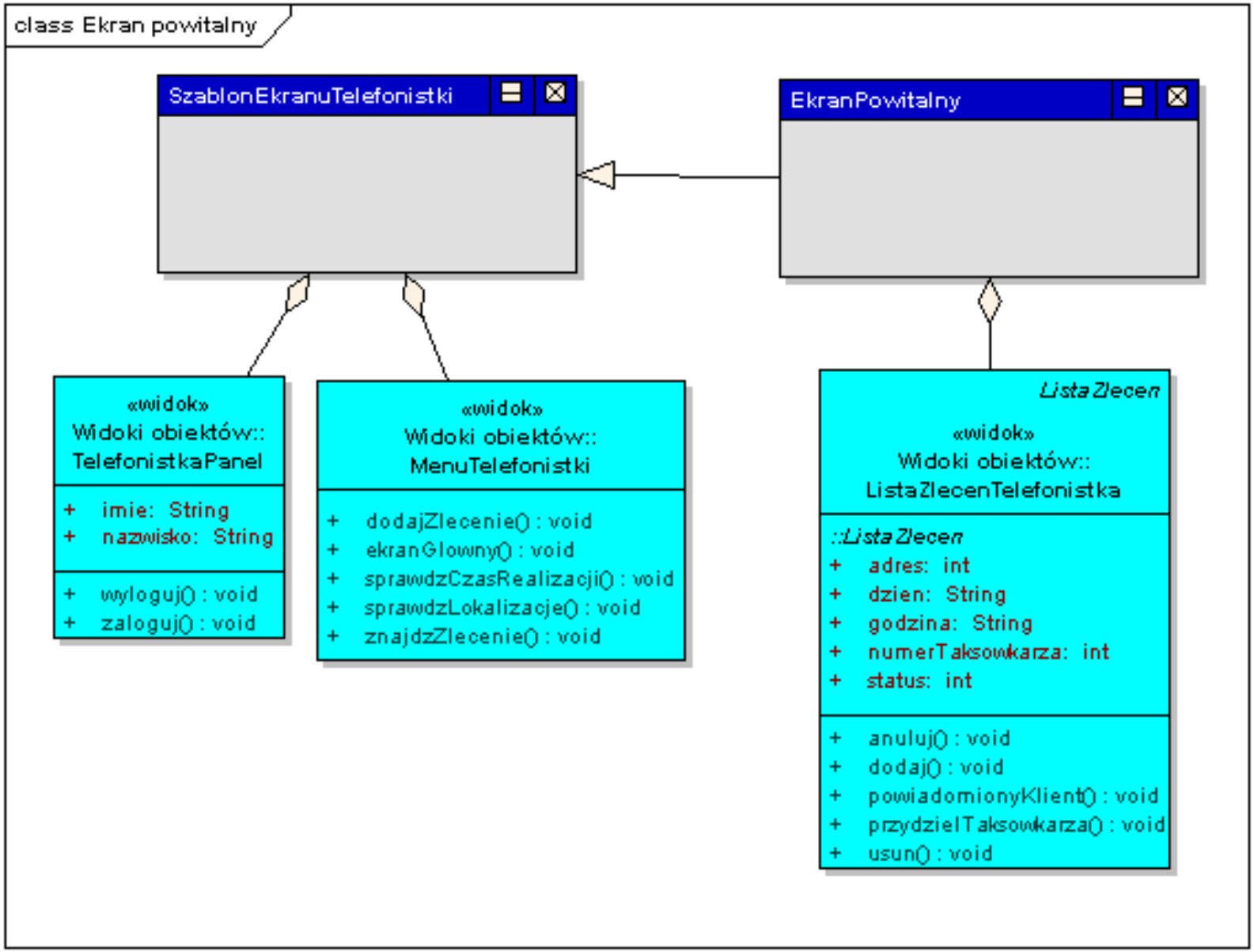
- Definiujemy co musi znajdować się na poszczególnych ekranach (najlepiej związanych z przypadkami użycia, czyli najkorzystniej z scenariuszami)
- Szablon:



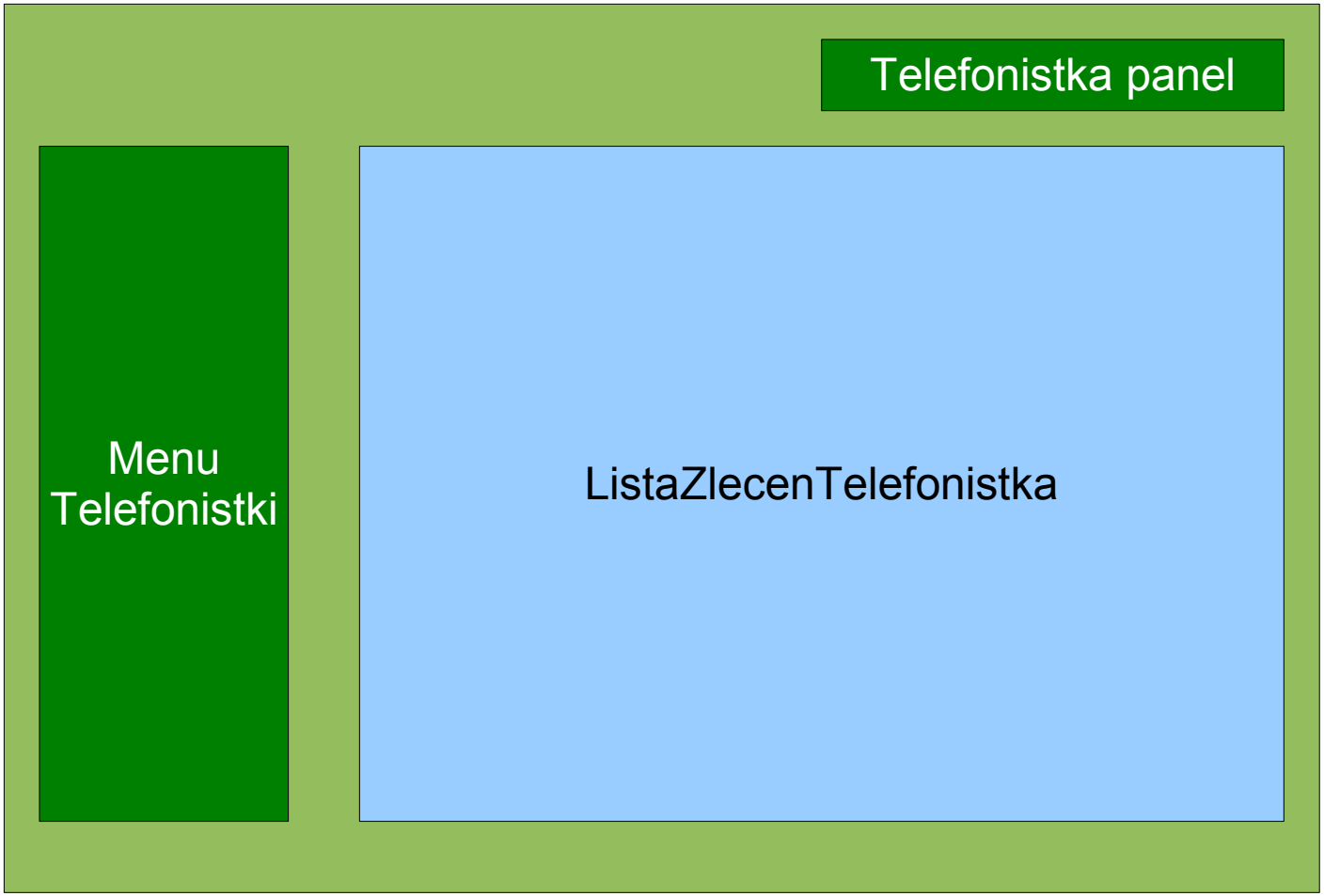
Szablon ekranu telefonistki



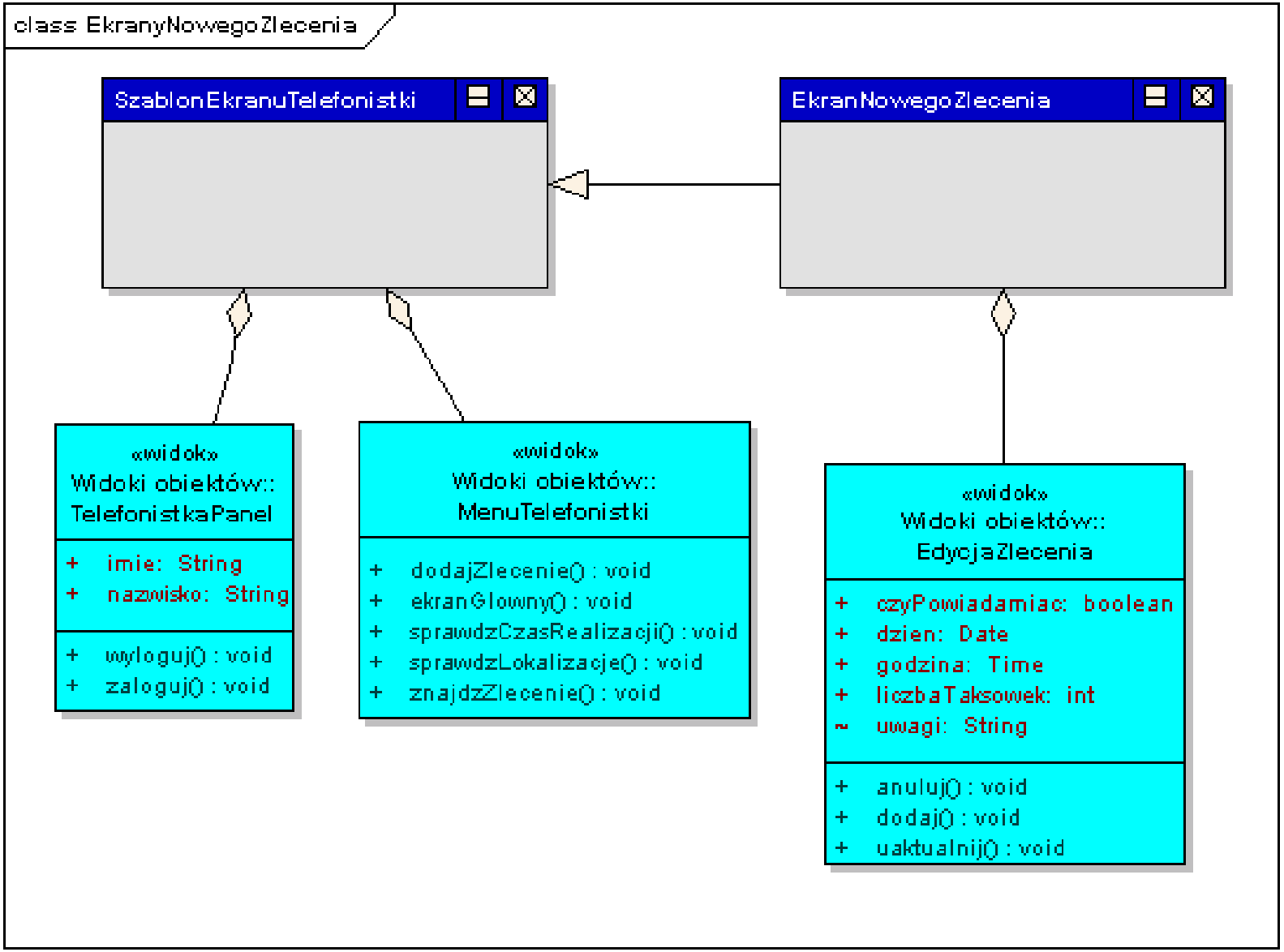
Ekran powitalny Telefonistki



Ekran powitalny Telefonistki



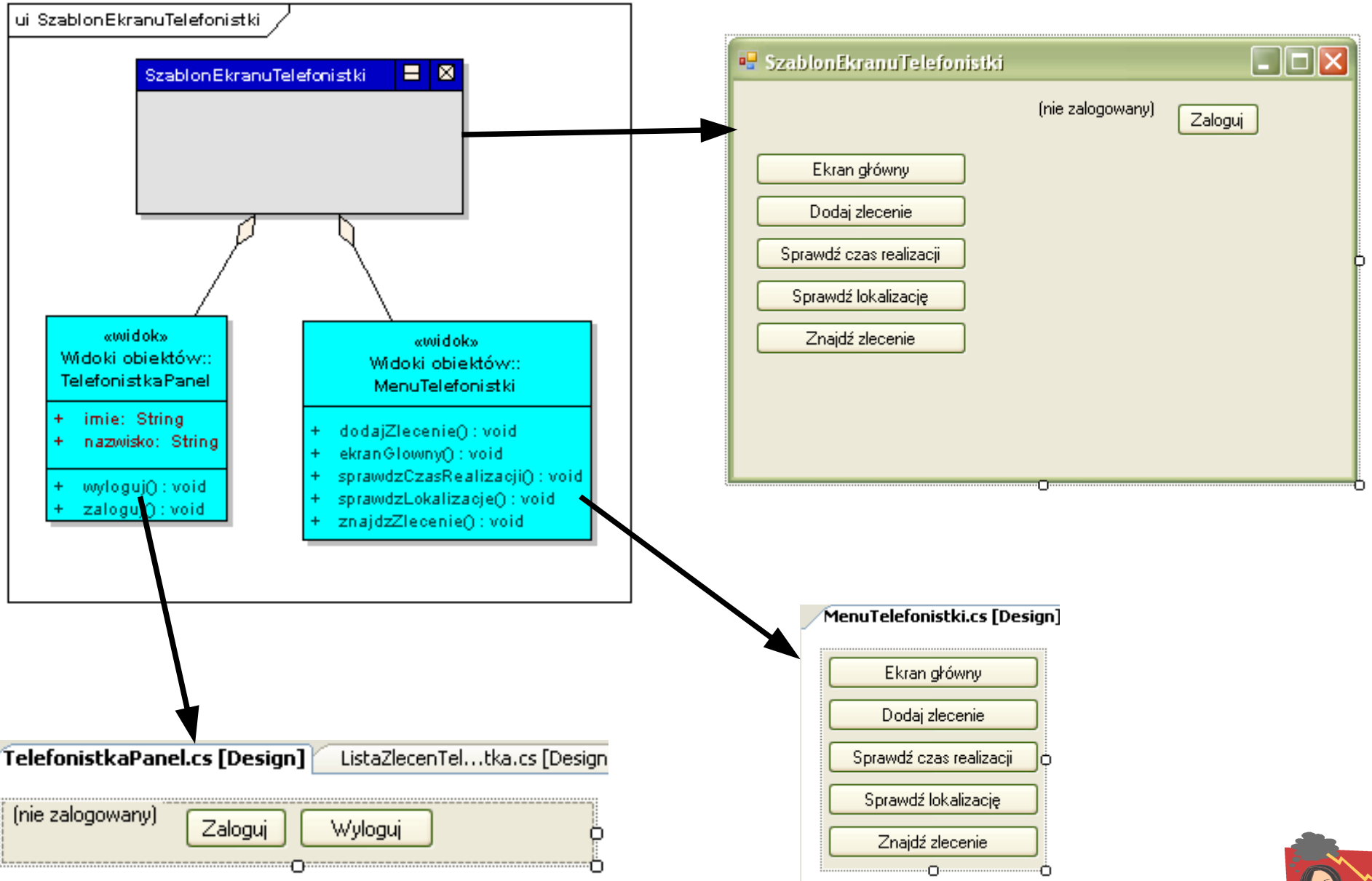
Edycja zlecenia przez Telefonistkę



Projekty widoków wizualnych

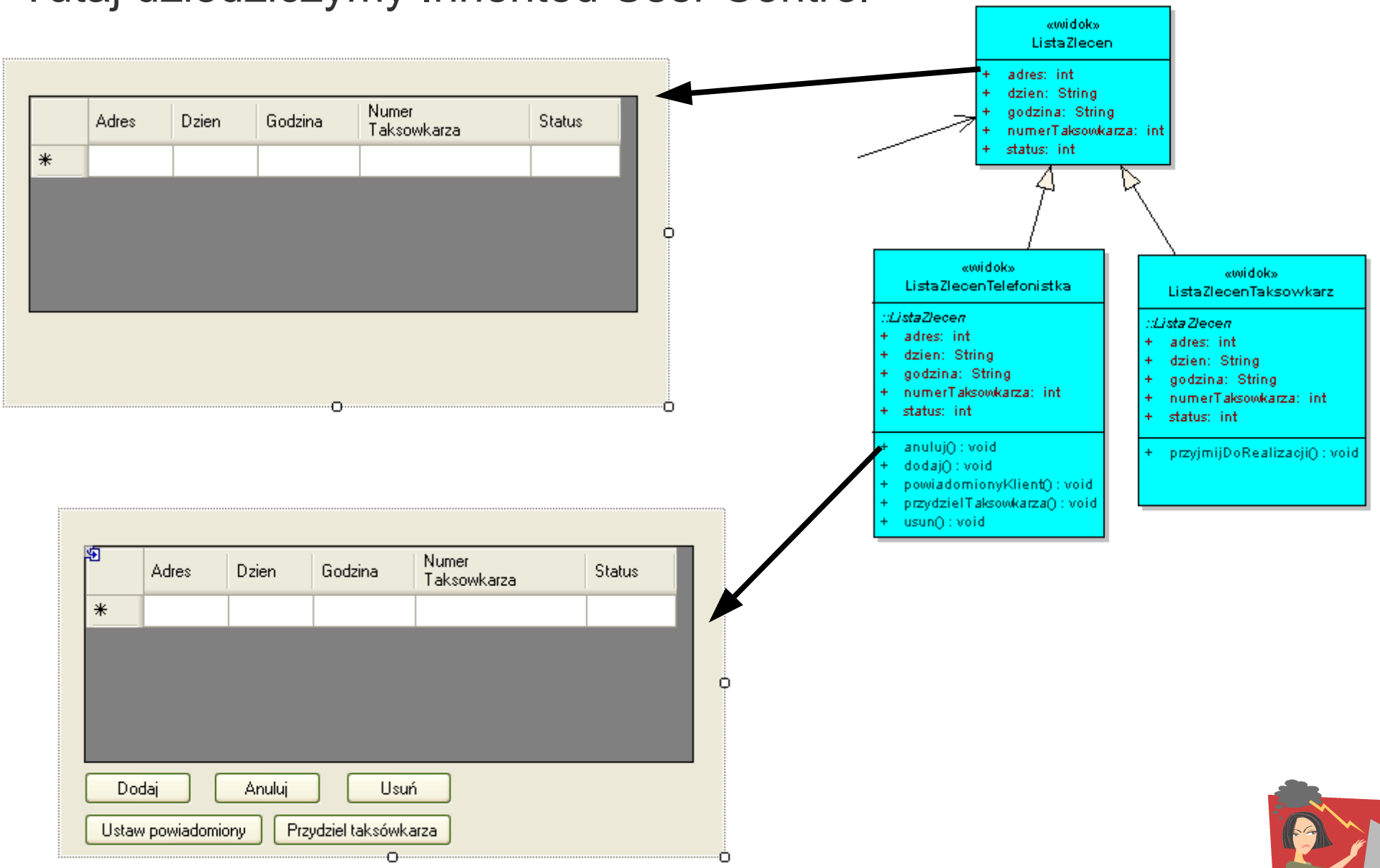
- Zrealizujemy wykorzystując technologię Microsoft Windows Forms Controls.

SzablonEkranuTelefonistki



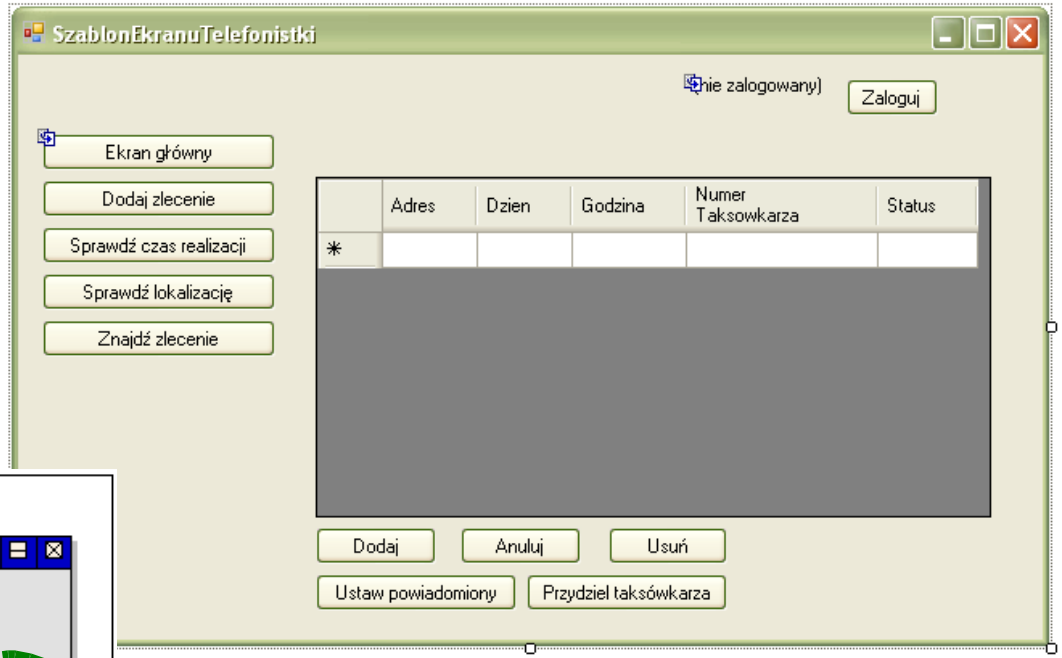
Dziedziczenie widoku ListaZleceń

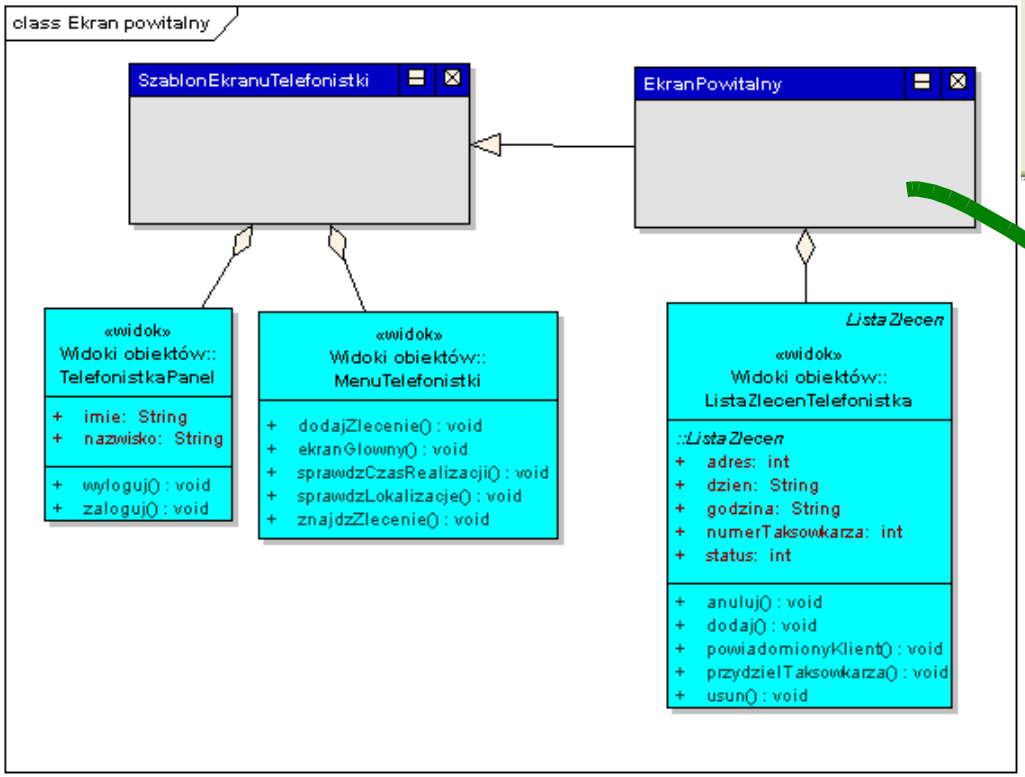
- Tutaj dziedziczymy Inherited User Control



EkranPowitalny

- Tutaj dziedziczymy szablon ekranu: Inherited Form





Interakcja

- Dziękuję za uwagę.

- Chcemy być coraz lepsi!
- Jeżeli coś cię zainteresowało napisz e-maila:
  - robert@iem.pw.edu.pl
- Jeżeli coś cię bardzo znudziło napisz e-maila:
  - robert@iem.pw.edu.pl
- Jeżeli zauważyłeś błąd napisz e-maila:
  - robert@iem.pw.edu.pl