Laboratorium z przedmiotu Programowanie obiektowe - zestaw 09

Cel zajęć. Celem zajęć jest zapoznanie się z metodami pozwalającymi na zapis stanu obiektu do pliku i jego odczyt w języku C#.

Wprowadzenie teoretyczne. Rozważana w ramach niniejszych zajęć tematyka jest ważna, gdyż umożliwia na odtworzenie stanu obiektu np. w przypadku ponownego uruchomienia uprzednio zakończonej aplikacji. Aby ze zrozumieniem zrealizować zadania, przewidziane do wykonania w ramach zajęć laboratoryjnych, należy znać znaczenie takich zagadnień jak serializacja do pliku XML.

1. **Serializacja –** proces konwertowania obiektu lub kolekcji obiektów na format nadający się do przesyłania za pośrednictwem sieci w formie strumienia lub do zapisania w pamięci, pliku czy bazie danych. Odwrotnym procesem jest deserializacja**.**

2. **W technologii .NET zaimplementowano obsługę trzech głównych rodzajów tego procesu:**

   a. Serializację binarną – obiekty są serializowane do postaci strumienia binarnego
   b. Serializację protokołu SOAP – obiekty są serializowane do postaci kodu XML zgodnego z protokołem SOAP
   c. Serializację do formatu XML – obiekty są serializowane do postaci XML

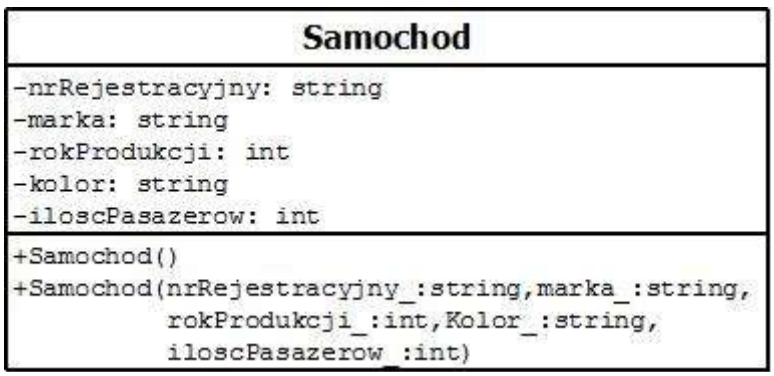
3. **Serializacja obiektu do pliku XML – wymagania, informacje:**

   a. Klasa serializowanego obiektu musi zawierać publiczny konstruktor domyślny.
   b. Serializacji podlegają jedynie publiczne pola i właściwości (jeśli pole jest prywatne lub chronione, w celu serializacji jego wartości, należy zaimplementować odpowiednią właściwość dostępową).
   c. Właściwości tylko do odczytu nie są serializowane.
   d. Serializacji mogą ulegać obiekty standardowych klas kolekcji (np. „List<T>”).

Zadanie 1. Proszę zrealizować aplikację okienkową według poniższej instrukcji:

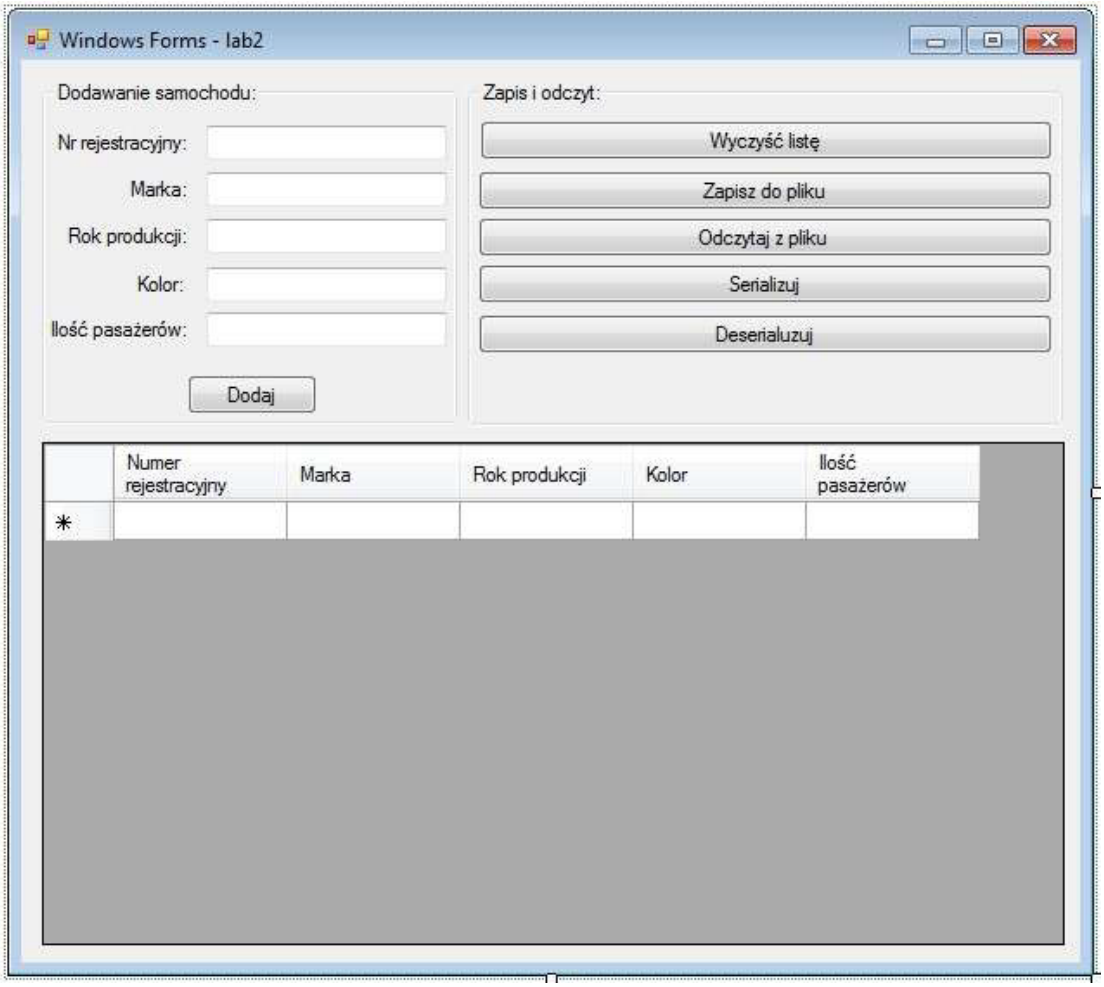
1. Proszę o utworzenie projektu okienkowego.

2. Proszę o utworzenie publicznej klasy o nazwie „Samochod” według poniższego diagramu:



3. Proszę o utworzenie prywatnego pola typu *List<Samochod>* o nazwie „samochody” w klasie *Form1*.

4. Proszę o dostosowanie okna formatki wedle rysunku zaprezentowanego poniżej:



Na formatce wykorzystano kontrolki typu *Label*, *Button*, *TextBox*, *GroupBox*, *DataGridView*.

Podpowiedź:

Zarządzanie kolumnami kontrolki typu *DataGridView* odbywa się za pomocą właściwości *Columns*.

5. Proszę o obsługę zdarzenia *Click* przycisku „Dodaj". Ma ono powodować dodanie obiektu typu *Samochod o* podanych parametrach do listy samochodów w klasie *Form1*. Informacje dotyczące utworzonego obiektu (wartości jego pól) mają być również dodawane do kontrolki typu *DataGridView*.

Podpowiedź:

Dodawanie wiersza do kontrolki typu *DataGridView* można zrealizować w następujący sposób:

```
dataGridView1.Rows.Add(nr, marka, rok, kolor, ilosc);
```

Uwaga!

Operację tę można zrealizować także w inny sposób.

Poniższa konstrukcja zwiększa liczbę wierszy o 1:

```
dataGridView1.RowCount += 1;
```

Poniższa konstrukcja umożliwia dostęp do wartości konkretnej komórki:

```
object value = dataGridView1[indexKolumny, indexWiersza].Value;
```

6. Proszę o obsługę zdarzenia *Click* przycisku „Wyczyść”. Ma ono powodować usunięcie wszystkich elementów z listy samochodów w klasie *Form1* oraz usunięcie wszystkich wierszy z kontrolki typu *DataGridView*.

Podpowiedź:

W celu usunięcia wierszy z kontrolki typu *DataGridView*, należy wywołać metodę *Clear* właściwości *Rows*.

7. Proszę o obsługę zdarzenia *Click* przycisku „Zapisz do pliku”. Ma ono powodować utworzenie pliku tekstowego „samochody.txt” w głównym katalogu programu. W pliku tym mają być zapisane parametry samochodów z listy samochodów w klasie *Form1*. Struktura pliku powinna wyglądać następująco:

[SAMOCHOD]
[nr_rejestracyjny]
SC12345
[marka]
Opel Astra
[rok_produkcji]
2000
[kolor]
Srebrny
[ilosc_pasazerow]
5
[END_SAMOCHOD]

[SAMOCHOD]
[nr_rejestracyjny]
CZB6655
[marka]
Fiat 126p
[rok_produkcji]
1990
[kolor]
Zielony
[ilosc_pasazerow]
4
[END_SAMOCHOD]

I tak dalej…

Podpowiedzi:

Należy dołączyć przestrzeń nazw *System.IO* (poprzez słowo „using”, na początku pliku).

Zapis wiersza do pliku można zrealizować w następujący sposób:

```
FileStream fs = new FileStream("./samochody.txt", FileMode.Create);
StreamWriter sw = new StreamWriter(fs);

sw.WriteLine("wiersz z tekstem...");
//...

sw.Close();
```

s

8. Proszę o obsługę zdarzenia *Click* przycisku „Odczytaj z pliku”. Ma ono powodować odczytanie z pliku tekstowego „samochody.txt”, znajdującego się w głównym katalogu programu, informacji o samochodach. Na podstawie odczytanych informacji nowe obiekty typu *Samochod* mają zostać dodane do listy samochodów w klasie *Form1*. Odczytane informacje mają zostać także dodane do kontrolki typu *DataGridView*
.
Podpowiedź:

Odczyt wszystkich wierszy z pliku (po kolei) można zrealizować w następujący sposób:

```
if (File.Exists("./samochody.txt"))
{
      FileStream fs = new FileStream("./samochody.txt",
      FileMode.Open);
      StreamReader sr = new StreamReader(fs);
      string linia;
      while ((linia = sr.ReadLine()) != null)
      {
            //zmienna linia zawiera wiersz tekstu pobrany z pliku
      }
      sr.Close();
}
```

9. Proszę o obsługę zdarzenia *Click* przycisku „Serializuj". Ma ono powodować utworzenie pliku XML „samochody.xml" w głównym katalogu programu. W pliku tym ma być zapisana zserializowana lista samochodów z klasy *Form1*.

Podpowiedzi:

Należy dołączyć przestrzeń nazw *System.Xml.Serialization*.
Należy poddać serializacji kolekcję typu *List<Samochod>*.
Przykładowa serializacja obiektu typu *Budynek*:

```
Budynek b = new Budynek();
FileStream fs = new FileStream("./budynek.xml", FileMode.Create);
XmlSerializer serializer = new XmlSerializer(typeof(Budynek));
serializer.Serialize(fs, b);
fs.Close();
```

10. Proszę o obsługę zdarzenia *Click* przycisku „Deserializuj". Ma ono powodowac deserializację listy samochodów z pliku XML „samochody.xml", znajdującego się w głównym katalogu programu. Poddana deserializacji lista ma stanowić listę samochodów w klasie *Form1*. Odczytane informacje mają zostać także dodane do kontrolki typu *DataGridView*.

Podpowiedź:

Przykładowa deserializacja obiektu typu *Budynek*:

```
if (File.Exists("./budynek.xml"))
{
      Budynek b;
      FileStream fs = new FileStream("./budynek.xml", FileMode.Open);
      XmlSerializer serializer = new XmlSerializer(typeof(Budynek));
      b = (Budynek)serializer.Deserialize(fs);
      fs.Close();
}
```

Zadanie do domu

1. Proszę o modyfikację powyższego zadania, która ma polegać na określeniu przez użytkownika ścieżki zapisywanego/odczytywanego pliku txt i xml. W tym celu proszę o wykorzystanie kontrolek typu *OpenFileDialog*, *SaveFileDialog*.

2. Proszę o obsługę sytuacji wyjątkowych, które mogą pojawić się podczas odczytu pliku txt i xml, gdy jego struktura będzie nieprawidłowa. Komunikat o błędzie ma zostać wyświetlony za pomocą klasy *MessageBox*.

3. Po poprawnie wykonanym zapisie/odczycie plików, proszę o wyświetlenie odpowiedniego komunikatu za pomocą klasy *MessageBox*.