Programowanie komputerów.
Ćwiczenia 11.

ZAPIS Z PLIKU I ODCZYT Z PLIKU, MECHANIZM REFLEKSJI, SERIALIZACJA

Przed przystąpieniem do zadań należy przeanalizować treść wykładu na temat **serializacji i deserializacji danych** oraz z rozdziałem 6. podręcznika *„Programowanie komputerów w zadaniach. Język C#”* (zadania 6.1-6.5).

Zadanie 1.

Utwórz klasę **`Pracownik`** z czterema polami prywatnymi zdefiniowanymi na podstawie zawartości pliku *pracownicy.csv.* W klasie tej zaimplementuj metodę obliczającą pensję jako iloczyn liczby godzin i stawki oraz przesłonięcie metody **`ToString`**, która zwróci łańcuch znaków opisujący pracownika (wraz z obliczoną pensją). Ponadto zdefiniuj klasę **`Pracownicy`**, która będzie definiowała listę pracowników (listę obiektów klasy Pracownik). W klasie tej zdefiniuj trzy metody.

Pierwsza do załadowania listy obiektów na postawie danych odczytanych z pliku *pracownicy.csv* (wykorzystaj metodę `Split` z klasy **`String`**).

Druga do wyświetlania informacji o pracownikach (przykładowy wynik poniżej).

Pracownicy

Imię	Nazwisko	Godziny	Stawka	Pensja
Jan	Kowalski	30	170	5100
Tomasz	Adamski	28	168	4704
Adam	Nowak	25	150	3750
Anna	Sosna	30	180	5400
Mirek	Tokarski	29	175	5075

Trzecia metoda ma zapisać informacje o pracownikach (razem z pensją) do pliku *pracownicyPensja.csv*. Opis danych na kolejnym slajdzie.

Wskazówki:

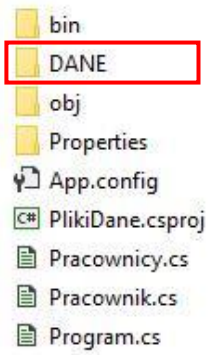
- Przykładowa zawartość metody **`Main`**

```
static void Main(string[] args)
{
    string sciezka1 = @"..\..\DANE\pracownicy.csv";
    string sciezka2 = @"..\..\DANE\pracownicyPensja.csv";

    Pracownicy obj = new Pracownicy();
    obj.Wczytaj(sciezka1);
    obj.PokazListe();
    obj.Zapisz(sciezka2);

    Console.ReadKey();
}
```

- W folderze projektu utwórz folder DANE i wgraj tam plik *pracownicy.csv*



Zadanie 2.

a)

Napisz program, który automatycznie wygeneruje kolekcje 1000 obiektów klasy Pogoda i zapisze do pliku w formacie CSV. Klasa ta powinna posiadać takie pola jak czas pomiaru, temperatura, prędkość wiatru i jego kierunek, wilgotność oraz zachmurzenie (duże, średnie, małe, brak). Zarówno kierunek wiatru jak i zachmurzenie opisz przy pomocy typu wyliczeniowego. Czas pomiaru – należy wpisać czas począwszy od wybranej daty co godzinę, a wartości dla pozostałych pól należy wygenerować losowo.

b)

Do programu dodaj możliwość odczytania danych z pliku CSV przy pomocy klasy **`StreamReader`**. Odczytane dane z pliku i umieszczone w kolekcji **`List<Pogoda>`** należy wyświetlić na ekranie (w postaci tabelarycznej), a pod wykazem wypisać podsumowania dla dwóch parametrów pogody: ile było pomiarów z brakiem chmur w badanym okresie oraz ile było pomiarów z wilgotnością powyżej 80%.

Zadanie 3.

a)

Wykonaj program podobny do zadania 6.3 z podręcznika *Programowanie komputerów w zadaniach. Język C#* (tamten program zapisuje wybrane dane o metodach klasy **`System.Math`** do pliku CSV). Nowy program ma wpisać do pliku CSV dane o metodach klasy **`System.IO.Stream`** (przy pomocy mechanizmu refleksji). W tym programie mają być zapisywane do pliku następujące dane o metodach: nazwa metody, czy jest to metoda wirtualna, czy jest to metoda abstrakcyjna, typ zwracanej wartości oraz łańcuch znakowy złożony z listy argumentów tej metody (np. dla metody **`Seek`**, byłby to łańcuch: *long offset, SeekOrigin origin*). Podobnie jak w zadaniu 6.3 zdefiniuj klasę **`Metoda`** do opisania metod według podanych danych.

Wskazówki:

- Na tej stronie:

https://docs.microsoft.com/en-us/dotnet/api/system.reflection.methodinfo?view=netframework-4.8

znajdziesz wszystkie potrzebne właściwości i metodę do pobrania argumentów:

- Wpisując w Visual Studio **`System.IO.Stream`** (z kursorem na słowie **`Stream`**) i klikając klawisz F12 uzyskasz szybki podgląd składowych klasy **`Stream`** – może się przydać do testowania programu.

b)

Wykonaj program podobny do zadania 6.4 z podręcznika *Programowanie komputerów w zadaniach. Język C#* i odczytaj z pliku CSV dane o metodach z klasy **`System.IO.Stream`** zapisane w programie z poprzedniego zadania. Odczytane dane należy wpisać do listy, posortować według nazwy i liczby argumentów wejściowych, a następnie wyświetlić na ekranie w postaci tabelarycznej.

Przy czym w ostatniej kolumnie, zamiast pełnego łańcucha z listą argumentów metody, ma być wyświetlona tylko ich liczba (np. dla metody **`Seek`** będzie to wartość 2).

Dokument XML

Zapoznaj się z podstawowymi zasadami budowy dokumentów XML, np. na stronie: http://webmaster.helion.pl/starocie/xml/pierwszy_dokument.htm

Dla danych:

PersonId	FirstName	LastName
1	Adam	Kowalski
2	Jan	Kowalski

- zapis w elementach potomnych (zagnieżdżonych):

```
<?xml version="1.0" encoding="iso-8859-2"?>
<ArrayOfPersons>
   <Person>
      <PersonId>1</PersonId>
      <FirstName>Adam</FirstName>
      <LastName>Kowalski</LastName>
   </Person>
  <Person>
    <PersonId>2</PersonId>
    <FirstName>Jan</FirstName>
    <LastName>Kowalski</LastName>
  </Person>
</ArrayOfPersons>
```

- zapis w elementach opisanych znacznikiem *Person*:

```
<?xml version="1.0" encoding="iso-8859-2"?>
<ArrayOfPersons>
   <Person PersonId="1" FirstName="Adam" LastName="Kowalski"/>
   <Person PersonId="2" FirstName="Jan" LastName="Kowalski"/>
</ArrayOfPersons>
```

Podczas serializacji i deserializacji korzysta się często z kolekcji, wówczas dokument xml powinien mieć *root* różny od elementu dla całej kolekcji, np.:

```
<?xml version="1.0"?>
<Root> <-- root
  <Cars> <-- element (ale nie root), zaczynający kolekcję...
    <Car>  <-- element (jako pojedyncza pozycja kolekcji)...
      <Name>Fiat</Name>
      <Year>2005</Year>
    </Car>
```

```
    <Car>
      <Name>Opel</Name>
      <Year>2010</Year>
    </Car>
  </Cars>
</Root>)
```

Serializacja kolekcji do XML

Przeanalizuj program z zadania 6.5. z podręcznika *Programowanie komputerów w zadaniach. Język C#*, który wykonuje serializację i deserializację kolekcji danych.

```
// test zapisu(serializacji) obiektów do XML
Towar[] tabTowary =
        { new Towar("Długopis", 4.5, 10),
          new Towar("Ołówek", 2.5, 25),
          new Towar("Blok rysunkowy", 3.0, 15) };
ZbiorTowarow obj = new ZbiorTowarow(tabTowary);
ZapisOdczytXML.Zapisz(plik, obj);
```

```
<?xml version="1.0"?>
<KolekcjaTowarow xmlns:xsi="http://w
  <Towary>
    <Towar>
      <Nazwa>Długopis</Nazwa>
      <Cena>4.5</Cena>
      <Ilosc>10</Ilosc>
    </Towar>
    <Towar>
      <Nazwa>Ołówek</Nazwa>
      <Cena>2.5</Cena>
      <Ilosc>25</Ilosc>
    </Towar>
    <Towar>
      <Nazwa>Blok rysunkowy</Nazwa>
      <Cena>3</Cena>
      <Ilosc>15</Ilosc>
    </Towar>
  </Towary>
</KolekcjaTowarow>
```

Zadanie 4.

a)

Wykonaj program dokonujący serializacji i deserializacji XML kolekcji danych. Program ma być podobny do zadania 6.5. z podręcznika *Programowanie komputerów w zadaniach. Język C#*, przy czym zamiast danych o klasie **`Towar`**, program ma serializować i deserializować te same dane, jakie były wykorzystane w zadaniu 3 – tj. dane o metodach z klasy **`System.IO.Stream`**.

b)

Zmodyfikuj program, tak, aby podczas serializacji zapisać argumenty metod z klasy **`Stream`** w osobnych elementach XML, a nie jako łańcuch w jednym elemencie. Zapis ten należy odpowiednio uwzględnić w deserializacji i w prezentacji danych na ekranie (wyświetlając dla każdej metody, w osobnych wierszach, jej argumenty).

Fragment wynikowego pliku XML dla 3 metod. Pierwsza z nich nie ma żadnych argumentów, druga ma dwa, a trzecia jeden:

```xml
<Metoda>
  <Nazwa>ReadByte</Nazwa>
  <CzyWirtualna>true</CzyWirtualna>
  <CzyAbstrakcyjna>false</CzyAbstrakcyjna>
  <TypZwracany>System.Int32</TypZwracany>
</Metoda>
<Metoda>
  <Nazwa>Seek</Nazwa>
  <CzyWirtualna>false</CzyWirtualna>
  <CzyAbstrakcyjna>true</CzyAbstrakcyjna>
  <TypZwracany>System.Int64</TypZwracany>
  <Argumenty>
    <Argument>long offset</Argument>
    <Argument>SeekOrigin origin</Argument>
  </Argumenty>
</Metoda>
<Metoda>
  <Nazwa>SetLength</Nazwa>
  <CzyWirtualna>false</CzyWirtualna>
  <CzyAbstrakcyjna>true</CzyAbstrakcyjna>
  <TypZwracany>System.Void</TypZwracany>
  <Argumenty>
    <Argument>long value</Argument>
  </Argumenty>
```

Zadanie 5.

a)

Napisz program, który wykonuje serializację i deserializację XML dla kolekcji danych z klasy **`Rachunek`** zawierającej dane: numer rachunku, datę zakupu oraz pozycje rachunku (**`List<Towar>`**). Na rachunku może być jedna lub więcej pozycji z towarami. Klasa **`Towar`** powinna posiadać takie dane jak: nazwa towaru, ilość, cena brutto.

Fragment wynikowego pliku XML dla jednego rachunku z kilkoma towarami:

```xml
<Rachunki>
  <Rachunek>
    <listaTowarow>
      <Towar>
        <Nazwa>Długopis</Nazwa>
        <Cena>4.5</Cena>
        <Ilosc>10</Ilosc>
      </Towar>
      <Towar>
        <Nazwa>Ołówek</Nazwa>
        <Cena>2.5</Cena>
        <Ilosc>25</Ilosc>
      </Towar>
      <Towar>
        <Nazwa>Blok rysunkowy</Nazwa>
        <Cena>3</Cena>
        <Ilosc>15</Ilosc>
      </Towar>
    </listaTowarow>
    <Numer>101/2019</Numer>
    <Data>2020-04-28T12:48:16.463806+02:00</Data>
  </Rachunek>
```

b)

Zmodyfikuj program korzystając z interfejsu **`IXmlSerializable`** zmień sposób serializacji klasy **`Towar`** tak aby podczas serializacji dopisać jeszcze jeden element do pliku XML – **`Wartosc`**, który należy obliczyć jako iloczyn ceny i ilości. Podczas deserializacji należy zignorować ten dodatkowy element.

Fragment wynikowego pliku XML dla jednego rachunku z kilkoma towarami:

```
<Rachunki>
  <Rachunek>
    <listaTowarow>
      <Towar>
        <Nazwa>Długopis</Nazwa>
        <Cena>4.5</Cena>
        <Ilosc>10</Ilosc>
        <Wartosc>45</Wartosc>
      </Towar>
      <Towar>
        <Nazwa>Ołówek</Nazwa>
        <Cena>2.5</Cena>
        <Ilosc>25</Ilosc>
        <Wartosc>62.5</Wartosc>
      </Towar>
      <Towar>
        <Nazwa>Blok rysunkowy</Nazwa>
        <Cena>3</Cena>
        <Ilosc>15</Ilosc>
        <Wartosc>45</Wartosc>
      </Towar>
    </listaTowarow>
    <Numer>101/2019</Numer>
    <Data>2020-04-28T12:57:13.6852007+02:00</Data>
  </Rachunek>
```