APBD - Ćwiczenia 2

pgago@pja.edu.pl

9 marca 2020

1 Zadanie 1

W niniejszym zadaniu Waszym celem będzie dalsze zapoznanie się z językiem C# i przygotowanie aplikacji konsolowej, która służy do obróbki danych. Pamiętaj o stworzeniu nowego repozytorium dla zadania 2 i umieszczeniu swojego kodu w repozytorium kodu. Pamiętaj o komendach commit i push. Poniżej znajdziecie wymagania na aplikację.

Okazało się, że na Uniwersytecie XYZ powstała potrzeba eksportu danych i ich odpowiedniego przygotowania do przesłania do Ministerstwa Edukacji i Szkolnictwa Wyższego. Ministerstwo przygotowały system, który pozwala na zaimportowanie pliku XML w odpowiednim formacie określonym przez Ministerstwo.

System informatyczny stosowany na Uniwersytecie XYZ pozwala na eksport danych wyłącznie w pliku CSV. Niestety wyeksportowane dane zawierają pewne błędy lub brakujące dane. Ponadto ich format nie odpowiada formatowi, którego oczekuje ministerstwo.

Musimy stworzyć aplikację konsolową, która pozwoli na poprawne przetworzenie otrzymanego pliku CSV i otrzymanie pliku wynikowego zgodnego z formatem oczekiwanym przez Ministerstwo.

1.1 Parametry programu

Program powinien przyjmować trzy parametry:

- adres pliku CSV
- adres ścieżki docelowej
- format danych

Poniżej znajduje się przykład wywołania:

```
.\export_xyz "C:\Users\Jan\Desktop\csvData.csv"
"C:\Users\Jan\Desktop\wynik.xml" xml
```

W razie niepodania jednego z parametrów:

- domyślnym adresem pliku CSV jest "data.csv"
- domyślnym adresem docelowym wyniku jest żesult.xml"
- domyślnym typem danych jest "xml"

Jeśli plik wynikowy już istnieje w danej lokalizacji - powinien zostać nadpisany. Na razie jedyną dostępna wartością dla trzeciego parametru jest "xml". W przyszłości przewiduje się potrzebę implementacji innych wyjściowych formatów danych.

1.2 Obsługa błędów

Aplikacja powinna być odporna na błędy. Wszystkie wystąpienia błędów powinny być logowane do pliku txt o nazwie łog.txt". Pamiętajmy również o zrozumiałych dla użytkownika komunikatach o błędzie. W przypadku:

- Niepoprawnej ścieżki - zgłoś błąd ArgumentException("Podana ścieżka jest niepoprawna")
- Plik nie istnieje - zgłaszamy błąd FileNotFoundException("Plik nazwa nie istnieje")

1.3 Wejściowy format danych

Poniżej zaprezentowany jest przykładowy plik z danymi. Każdy wiersz reprezentuje pojedynczego studenta. Każda kolumna jest oddzielona znakiem ",". Każdy student powinien być opisywany przez 9 kolumn. Poniżej zaprezentowany jest pojedynczy wpis w pliku CSV.

```
Paweł,Nowak1,Informatyka dzienne,Dzienne,459,2000-02-12
00:00:00.000,nowak@pjwstk.edu.pl,Alina,Adam
```

Uwaga:

- Niektóre wiersze mogą zawierać błędy. Tych studentów, którzy nie są opisywani przez 9 kolumn z danymi pomijamy. Informacje o pominiętym studencie traktujemy jako błąd i logujemy do pliku łog.txt".
- Jeśli jeden ze studentów posiada w kolumnie pustą wartość (np.) - traktujemy taką wartość jako brakującą. W takim wypadku studenta nie dodajemy do zbioru wynikowego i dodajemy go do pliku łog.txt".

W powyższych przypadkach można zdefiniować własne klasy reprezentujące błąd. Ponadto okazało się, że dane zawierają czasem duplikaty informacji o studentach. Musimy zadbać o to, aby nie dodawać do wyniku dwa razy studenta o tym samym imieniu, nazwisku i numerze indeksu. Zawsze pobieramy pierwszego studenta z danym imieniem, nazwiskiem i numerem indeksu. Każde powtórzenie danych o studencie w danych źródłowych traktujemy jako niepoprawny duplikat.

2 Format docelowy

Ministerstwo udostępniło poniższy przykład pliku obrazującego docelowy format danych wejściowych w formie pliku XML.

```
<uczelnia
createdAt="08.03.2020"
author="Jan Kowalski">  - imie i nazwisko studenta
        <studenci>
                <student indexNumber="s1234">
                        <fname>Jan</fname>
                        <lname>Kowalski</lname>
                        <birthdate>03.04.1984</birthdate>
                        <email>kowalski@wp.pl</email>
                        <mothersName>Alina</mothersName>
                        <fathersName>Andrzej</fathersName>
                        <studies>
                                <name>Computer Science</name>
                                <mode>Dzienne</mode>
                        </studies>
                </student>
                <student indexNumber="s3455">
                        <fname>Anna</fname>
                        <lname>Malewska</lname>
                        <birthdate>09.07.1988</birthdate>
                        <email>kowalski@wp.pl</email>
                        <mothersName>Anna</mothersName>
                        <fathersName>Michał</fathersName>
                        <studies>
                                <name>New Media Art</name>
                                <mode>Zaoczne</mode>
                        </studies>
                </student>
        </studenci>
        <activeStudies>
                <studies name="Computer Science" numberOfStudents="1" />
                <studies name="New Media Art" numberOfStudents="1" />
        </activeStudies>
</uczelnia>
```

3 Zadania dodatkowe

Okazało się, że wymagania klienta uległy zmianie. W celu ograniczenia przesyłanych danych Ministerstwo przygotowało nowy format danych. Tym razem zdecydowano się wykorzystać format JSON. Spróbuj dodać do swojego kodu możliwość przekazania jako trzeci parametr wywołania programu formatu "json".

W jaki sposób podzielisz swój kod na klasy? Zakładając, że w przyszłości Ministerstwo może wprowadzać kolejne formaty danych - jak przygotujesz aplikację w taki sposób, aby była jak najbardziej elastyczna i dała się łatwo rozszerzać w przyszłości?

```
{
        uczelnia: {
                createdAt: "08.03.2020",
                author: "Jan Kowalski",
                studenci: [
                        {
                                indexNumber: "s1234",
                                fname: "Jan",
                                lname: "Kowalski",
                                birthdate: "02.05.1980",
                                email: "kowalski@wp.pl",
                                mothersName: "Alina",
                                fathersName: "Jan",
                                studies: {
                                        name: "Computer Science",
                                        mode: "Dzienne"
                                }
                        },
                        {
                                indexNumber: "s2432",
                                fname: "Anna",
                                lname: "Malewska",
                                birthdate: "07.10.1985",
                                email: "malewska@wp.pl",
                                mothersName: "Marta",
                                fathersName: "Marcin",
                                studies: {
                                        name: "New Media Art",
                                        mode: "Zaoczne"
                                }
                        }
```

```
                ],
                activeStudies: [
                        {
                                name: "Computer Science",
                                numberOfStudents: "1"
                        },
                        {
                                name: "New Media Art",
                                numberOfStudents: "2"
                        }
                ]
        }
}
```

3.1 Wskazówki

Przykład wczytywania pliku (System.IO).

```
using (var stream = new StreamReader(file.OpenRead()))
{
    string line = null;
    while ((line = stream.ReadLine()) != null)
    {
        string[] student = line.Split(',');
        var st = new Student
        {
            Imie = student[0],
            Nazwisko = student[1]
        };
    }
}
```

Przykład serializacji z pomocą System.XML.

```
FileStream writer = new FileStream(@"data.xml", FileMode.Create);
XmlSerializer serializer = new XmlSerializer(typeof(List<Student>), new XmlR
var list = new List<Student>();
list.Add(new Student
{
    Imie = "Jan",
    Nazwisko = "Kowalski"
});
serializer.Serialize(writer, list);
```

Przykład serializacji do typu danych JSON z pomocą System.Text.Json.

```
var list = new List<Student>()
{
    new Student{Imie="Jan", Nazwisko="Kowalski"}
};
var jsonString = JsonSerializer.Serialize(list);
File.WriteAllText("data.json", jsonString);
```

Dodatkowe atrybuty dodawane do właściwości klasy. Pozwalają wpłynąć na sposób serializacji danych.

```
[Serializable]
public class Student
{
    [XmlElement(ElementName = "InneNazwa")]
    public string Imie { get; set; }
    [XmlAttribute(AttributeName = "InnaNazwa")]
    [JsonPropertyName("LastName")]
    public string Nazwisko { get; set; }
```