APBD - Tutorial 2

pgago@pja.edu.pl

March 19, 2020

1 Task 1

In this task your goal will be to continue learning C# language and to prepare the console application, which is used for the data processing. Don't forget to create a new repository for the task 2 and to place your code there. Remember about commands commit and push. Lower you will find the application requirements. Turned out that University XYZ needs to export data and to properly prepare it for sending to the Ministry of Education. Ministry created a system which lets you import the XML file of the certain format described by the Ministry. The IT system used at University XYZ allows data export only in the CSV file. Unfortunately, the exported data contains some errors or missing data. In addition, its format does not match the format expected by the ministry. We need to create a console application that will allow the correct processing of the received CSV file and receiving the resulting file compatible with the format expected by Ministry.

1.1 Program parameters

Program should take 3 parameters:

- Path of the CSV file
- Destination path
- Data format

Below there is an example of a call: :

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

Below there is an example of a call:

- The default path of the CSV file is "data.csv"
- The default path of the CSV file is "data.csv"
- The default data type is "xml"

If the resulting file already exists in the given location - it should be overwritten. For now, the only available value for the third parameter is "xml". Expect that in the future there may be more output formats implemented in the application.

1.2 Error handling

The application should be resistant to errors. All errors should be logged in to the txt file named “log.txt ". Also remember that error messages need to be understandable to the user. In case of:

- The incorrect path - report error ArgumentException ("The path is incorrect").
- File does not exist – report error FileNotFoundException(“File does not exist”).

1.3 Input data format

An example of a data file is shown below. Each record represents a single student. Each column is separated by ",". Each student should be described by 9 columns. Here is an example of a single record in the CSV file.

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

Uwaga:

- Some records may contain errors. We omit those students who are not described by 9 data columns. We treat information about an omitted student as an error and log it into the file “log.txt "..
- If one of the students has an empty value in the column - we treat this value as missing. In this case, the student is not added to the result. Instead he is added to the file “log.txt".

In the above cases, you can define your own classes representing the error. In addition, it turned out that the data sometimes contain duplicate information about students. We must ensure that we do not add the student with the same name, surname and index number to the result twice. We always take the first student with given name, surname and index number. Every repetition of student data in source data is treated as an invalid duplicate.

2 Target format

The ministry has provided the following example of a file showing the target input data format in the form of an XML file.

```
<university
createdAt="08.03.2020"
author="Jan Kowalski">  - name and surname
        <students>
                <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>Stationary</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>Non-stationary</mode>
                        </studies>
                </student>
        </students>
        <activeStudies>
                <studies name="Computer Science" numberOfStudents="1" />
                <studies name="New Media Art" numberOfStudents="1" />
        </activeStudies>
</university>
```

3 Additional task

It turned out that customer requirements have changed. In order to limit the data sent, the Ministry prepared a new data format. This time it was decided to use the JSON format. Try adding to your code the ability to pass the "json" format call as the third parameter. How do you split your code into classes? Assuming that in the future the Ministry may introduce further data formats - how will you prepare the application in such a way that it is as flexible as possible and can be easily extended in the future?


```
{
        university: {
                createdAt: "08.03.2020",
                author: "Jan Kowalski",
                students: [
                        {
                                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 Tips

The example of reading a file (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]
        };
    }
}
```

An example of serialization using 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);
```

An example of serialization to the JSON data type using System.Text.Json.

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

Additional attributes added to the class properties. They allow you to affect the way of data serialization.

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