МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«Санкт–Петербургский национальный исследовательский университет информационных технологий, механики и оптики»

Факультет информационных технологий и программирования

ЛАБОРАТОРНАЯ РАБОТА № 3

Выполнил студент:

Ершов Михаил Николаевич
группа: M3206

Санкт-Петербург, 2019 г.

Лабораторная работа №3

Создать инструмент для обработки конфигурационного INI файла.

Описать и реализовать необходимые классы, которые позволят производить обработку конфигурационного файла, который представляет собой текстовый файл, разделенный на СЕКЦИИ, которые содержат пары ИМЯ, ЗНАЧЕНИЕ.

Пример файла:

```
[COMMON]
StatisterTimeMs = 5000
LogNCMD = 1 ; Logging ncmd proto
LogXML = 0 ; Logging XML proto
DiskCachePath = /sata/panorama ; Path for file cache
OpenMPThreadsCount = 2

[ADC_DEV]
BufferLenSeconds = 0.65 ; Buffer length for ADC data in GPU memory, seconds.
SampleRate = 120000000.0 ; Sample rate of ADC.
Driver = libusb ; cypress / libusb / random / fileIQS

[NCMD]
EnableChannelControl = 1 ; Use or not CHG / CHGEXT commands
SampleRate = 900000.0 ; ANOTHER Sample Rate.
TidPacketVersionForTidControlCommand = 2

; TidPacket versions
; 0 – no packets
; 1 – header: data size, tid
; 2 – header: data size, tid, timestamp

[LEGACY_XML]
ListenTcpPort = 1976

[DEBUG]
PlentySockMaxQSize = 126
```

Все имена параметров и секций – это строки без пробелов, состоящие из символов латинского алфавита, цифр и знаков нижнего подчеркивания.

Имена секций заключены в квадратные скобки, без пробелов.

Значения параметров отделены от имен параметров знаком = (равенство

Значения параметров могут быть одним из типов:

- целочисленным,
- вещественным,
- строковым: без пробелов, но в отличие от имени параметра может содержать также символ «точка».

Файл может содержать комментарии. Комментарием считается всё, что находится после знака «точка с запятой». Комментарии, как и сам знак «точка с запятой» должны быть проигнорированы.

Должны быть реализованы методы «получить значение определенного типа с таким-то именем из такой-то секции» (например, получить целое ListenTcpPort из секции LEGACY_XML)

Должны быть обработаны ошибки:

- Ошибка файловой подсистемы (например, если файл не найден)
- Ошибка формата файла (если файл имеет неверный формат)
- Неверный тип параметра (ошибка при приведении типа)
- Заданной пары СЕКЦИЯ ПАРАМЕТР нет в конфигурационном файл

и другие, при необходимости.

Для реализации предлагается использовать Generics, коллекции, исключения.

Если у вас есть вопросы по заданию, или вы хотите как-то его изменить (что-то туда добавить, или упростить, или ещё что-то), обратитесь к преподавателю

1 Program.cs

```
using System;

namespace lab3 {
    class Program {
        static void Main(string[] args) {
            HandlerFileINI handlerFileINI = new HandlerFileINI("test.in");
            //handlerFileINI.PrintAll();
            var res = handlerFileINI.GetValueDouble("ADC_DEV", "BufferLenSeconds
                ");
            Console.WriteLine($"{res}");
        }
    }
}
```

2 Exeptions.cs

```
public class SectionNotFound : System.Exception {
    public SectionNotFound() {}
}

public class ErrorReadFile : System.Exception {
    public ErrorReadFile() {}
}

public class InvalidTypeFile : System.Exception {
    public InvalidTypeFile() {}
}

public class AnotherType : System.Exception {
    public AnotherType() {}
}

public class KeyNotFound : System.Exception {
    public KeyNotFound() {}
}
```

3 HandlerFileINI.cs

```
using System;
using System.IO;
using System.Collections.Generic;
using System.Linq;
using System.Globalization;

class HandlerFileINI {
    public string fileName = "";
    public List<Section> Sections = new List<Section>();

    public HandlerFileINI(string pathFile) {
        if (Path.GetExtension(pathFile) != ".ini") {
            throw new InvalidTypeFile();
        }
        else {
            try {
                using(StreamReader sr = new StreamReader(pathFile)) {
                    string line;
                    while ((line = sr.ReadLine()) != null) {
                        line = line.Trim();
                        if (line[0] == '[') {
                            line = line.Substring(1, line.Length - 2);
                            Sections.Add(new Section(line));
                        } else if (line != "" && line[0] != ';') {
                            string[] data = line.Split(new [] {"=", " ", ";"},
                                StringSplitOptions.RemoveEmptyEntries);
                            if (Sections.Count > 0 && data.Length >= 2) {
                                Sections[Sections.Count - 1].AddData(data[0],
                                    data[1]);
                            }
                        }
                    }
                }
            } catch {
                throw new ErrorReadFile();
            }
        }
    }

    private string GetValue(string sectionName, string key) {
        Section mySection = null;
        try {
            mySection = Sections.Where(section => section.Name == sectionName).
                ToList()[0];
        } catch {
            throw new SectionNotFound();
        }
        if (mySection != null) {
            return mySection.Find(key);
        }
        return "";
    }

    public Int32 GetValueINT(string sectionName, string key) {
```

```
        string res = GetValue(sectionName, key);
        try {
            return Convert.ToInt32(res);
        } catch {
            throw new AnotherType();
        }
    }

    public Double GetValueDouble(string sectionName, string key) {
        string res = GetValue(sectionName, key);
        try {
            return Convert.ToDouble(res, CultureInfo.InvariantCulture);
        } catch {
            throw new AnotherType();
        }
    }

    public string GetValueString(string sectionName, string key) {
        string res = GetValue(sectionName, key);
        return res;
    }

    public void PrintAll() {
        foreach(var item in Sections) {
            item.PrintSection();
        }
    }
}
```

4 Section.cs

```
using System;
using System.Collections.Generic;

class Section {
    public string Name = "";
    private Dictionary<string, string> Data = new Dictionary<string, string>();

    public Section(string name) {
        Name = name;
    }

    public Section(string name, Dictionary<string, string> data) {
        Name = name;
        Data = data;
    }

    public void AddData(Dictionary<string, string> data) {
        foreach(var item in data) {
            Data.Add(item.Key, item.Value);
        }
    }

    public void AddData(string key, string value) {
        Data.Add(key, value);
    }

    public string Find(string key) {
        try {
            return Data[key];
        } catch {
            throw new KeyNotFound();
        }
    }

    public void PrintSection() {
        Console.WriteLine();
        Console.WriteLine("--------------------");
        Console.WriteLine($"Print section: {Name}");
        Console.WriteLine("Data:");
        foreach(var item in Data) {
            Console.WriteLine($"\t{item.Key} : {item.Value}");
        }
        Console.WriteLine("--------------------");
        Console.WriteLine();
    }
}
```