**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П.О.СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

Дисциплина «Объектно-ориентированное проектирование и программирование»

ОТЧЕТ

Лабораторная работа № 1

Работа с файловой системой. Потоки ввода/вывода

Выполнил студент

Группы ИТД-21

Чайдаков И.М.

Принял преподаватель

Башаримов Ю. С.

Гомель 2025

Работа с файловой системой. Потоки ввода/вывода

**Цель работы:** изучить основы XML, получить навыки работы с XML документами средствами .NET. Освоить средства валидации XML документов с применением XSD.

**Задание:** Разработать структуру XML документа и соответствующую ему XSD схему.

В XSD обязательно задавать типы данных, которые принимают узлы, и указывать пределы допустимых значений. В случае если документ не соответствует схеме, выводить соответствующее сообщение. Обосновать выбранный вид схемы XML.

Выделенные курсивом данные оформить как перечисление.

Разработать библиотеку классов для работы с данными в формате XML согласно варианту. Для работы с XML использовать DOM или SAX parser (XML-сериализацию или LINQ2XML использовать нельзя!)

Разработать WPF-приложение, поддерживающее работу с данными в формате XML (чтение, отображение, редактирование и сохранение) согласно варианту.

Приложение должно быть простым в использовании и включать в себя полную обработку исключений.

При написании и оформлении кода обязательно руководствоваться Code Convention, принципами ООП, SOLID и использовать элементы авто документирования с генерацией соответствующих файлов.

Написать unit-тесты и исходные файлы (не менее 30 записей) для тестирования разработанных библиотечных классов, тестирование должно покрывать более 80% библиотечного кода.

**Ход выполнения задания:**

**1.** Создаем класс DomToolKid для работы с xml-документом (см. Приложение А).

**2.** Описываем модульные тесты для верификации класса (см. Приложение B).

**3.** Создаем графический интерфейс для пользовательского доступа к функциям класса.

**4.** Запустим исполняемый файл и увидим перед собой созданный интерфейс с набором функций (Рисунок 1).

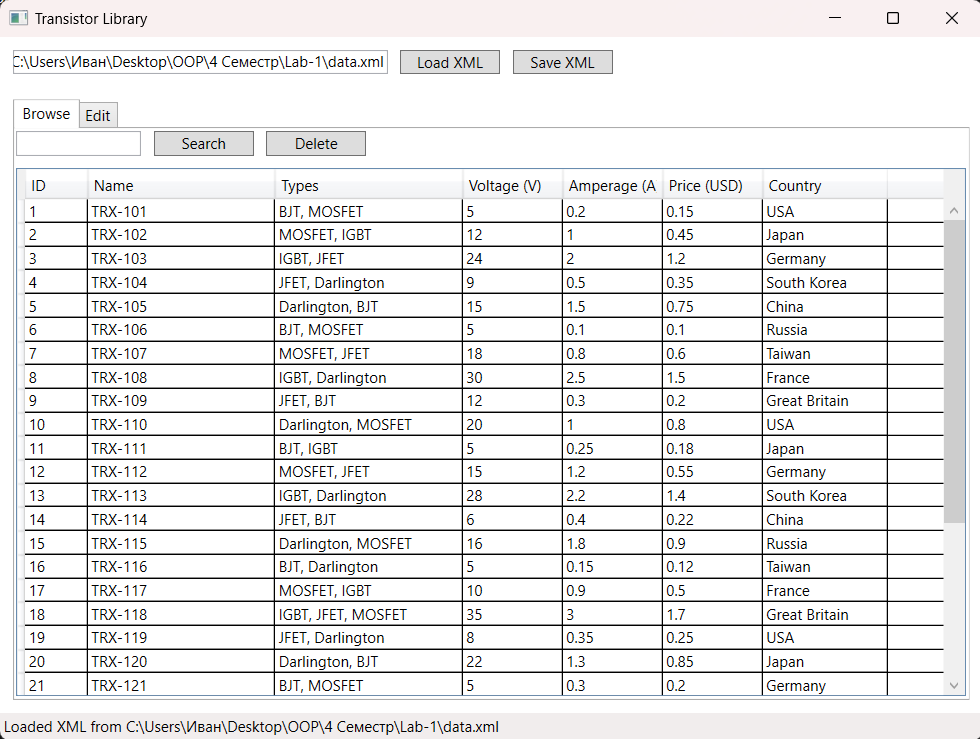


Рисунок 1 — Интерфейс программы.

**5.** Запустим тестовое приложение и убедимся что созданные нами классы работают корректно (рисунок 2).

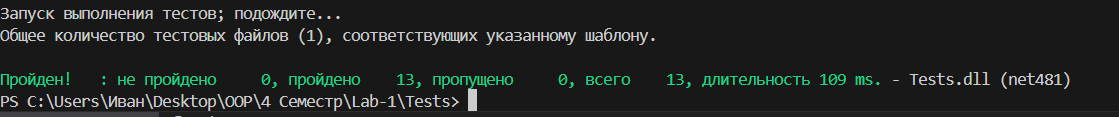


Рисунок 2 — Просмотр тестов

**Вывод:** XML документы позволяют работать с большими объемами данных укомплектованных в логических таблицах. Инструменты DOM позволяют интегрировать данные таблиц в приложения для последующей обработки.

**ПРИЛОЖЕНИЕ А**

**DomToolKid.cs**

namespace Library;

using System.Xml;

using System.Globalization;

public class DomToolkid

{

    public XmlDocument \_doc { get; set; }

    public string \_path { get; set; }

    public DomToolkid(string xmlPath)

    {

        \_doc = new XmlDocument();

        \_path = xmlPath;

        if (File.Exists(\_path))

        {

            \_doc.Load(\_path);

        }

    }

    public void saveDoc()

    {

        \_doc.Save(\_path);

    }

    public Transistor nodeShow(int id)

    {

        List<TransistorType> typeList = new List<TransistorType>();

        string transistorId = id.ToString("D3");

        XmlNode node = \_doc.SelectSingleNode($"/root/transistor[@id='{transistorId}']");

        if (node != null)

        {

            XmlNode typeNodes = node.SelectSingleNode("types");

            foreach(XmlNode typeNode in typeNodes.ChildNodes)

            {

                if(!Enum.TryParse(typeNode.InnerText, out TransistorType type))

                {

                    Console.WriteLine("type reading error");

                }

                typeList.Add(type);

            }

            string nodeName = node.SelectSingleNode("name").InnerText;

            if(float.TryParse(node.SelectSingleNode("voltage").InnerText, NumberStyles.Float, CultureInfo.InvariantCulture, out float nodeVoltage))

            {

                Console.WriteLine("voltage readed");

            }

            else

            {

                Console.WriteLine("voltage reading failed");

            }

            if(float.TryParse(node.SelectSingleNode("amperage").InnerText, NumberStyles.Float, CultureInfo.InvariantCulture, out float nodeAmperage))

            {

                Console.WriteLine("amperage readed");

            }

            else

            {

                Console.WriteLine("amperage reading failed");

            }

            if(float.TryParse(node.SelectSingleNode("price").InnerText, NumberStyles.Float, CultureInfo.InvariantCulture, out float nodePrice))

            {

                Console.WriteLine("price readed");

            }

            else

            {

                Console.WriteLine("voltage reading failed");

            }

            string nodeCountry = node.SelectSingleNode("country").InnerText;

            Transistor trn = new Transistor(id, nodeName, typeList, nodeVoltage, nodeAmperage, nodePrice, nodeCountry);

            Console.WriteLine("transistor readed\n");

            return trn;

        }

        else

        {

            Console.WriteLine("nodeShow error.");

            return null;

        }

    }

    public void nodeCreate(Transistor trn)

    {

        XmlElement transistor = \_doc.CreateElement("transistor");

        string transistorId = trn.\_id.ToString("D3");

        transistor.SetAttribute("id", transistorId);

        XmlElement name = \_doc.CreateElement("name");

        name.InnerText = trn.\_name;

        transistor.AppendChild(name);

        XmlElement types = \_doc.CreateElement("types");

        foreach(TransistorType type in trn.\_types)

        {

            XmlElement xtype = \_doc.CreateElement("type");

            string stType = type.ToString();

            xtype.InnerText = stType;

            types.AppendChild(xtype);

        }

        transistor.AppendChild(types);

        transistor.AppendChild(CreateMeasurementElement(\_doc, "voltage", trn.\_voltage, "V"));

        transistor.AppendChild(CreateMeasurementElement(\_doc, "amperage", trn.\_amperage, "A"));

        transistor.AppendChild(CreateMeasurementElement(\_doc, "price", trn.\_price, "USD"));

        XmlElement country = \_doc.CreateElement("country");

        country.InnerText = trn.\_name;

        transistor.AppendChild(country);

        \_doc.DocumentElement.AppendChild(transistor);

        saveDoc();

        Console.WriteLine("new element created.");

    }

    public void nodeRemove(int id)

    {

        string transistorId = id.ToString("D3");

        XmlNode node = \_doc.SelectSingleNode($"/root/transistor[@id='{transistorId}']");

        if(node != null)

        {

            XmlNode parent = node.ParentNode;

            parent.RemoveChild(node);

            saveDoc();

            Console.WriteLine("element removed");

        }

        else

        {

            Console.WriteLine("Element not found. Remove data error");

        }

    }

    public void nodeUpdate(int id, Transistor trn)

    {

        string transistorId = id.ToString("D3");

        XmlNode node = \_doc.SelectSingleNode($"/root/transistor[@id='{transistorId}']");

        if(node != null)

        {

            node.Attributes["id"].Value = trn.\_id.ToString("D3");

            XmlNode nameNode = node.SelectSingleNode("name");

            if(nameNode != null) nameNode.InnerText = trn.\_name;

            XmlNode typesNode = node.SelectSingleNode("types");

            typesNode.RemoveAll();

            foreach(TransistorType type in trn.\_types)

            {

                XmlElement xtype = \_doc.CreateElement("type");

                xtype.InnerText = type.ToString();

                typesNode.AppendChild(xtype);

            }

            XmlNode voltageNode = node.SelectSingleNode("voltage");

            if(voltageNode != null) voltageNode.InnerText = trn.\_voltage.ToString();

            XmlNode amperageNode = node.SelectSingleNode("amperage");

            if(amperageNode != null) amperageNode.InnerText = trn.\_amperage.ToString();

            XmlNode priceNode = node.SelectSingleNode("price");

            if(priceNode != null) priceNode.InnerText = trn.\_price.ToString();

            XmlNode countryNode = node.SelectSingleNode("country");

            if(countryNode != null) countryNode.InnerText = trn.\_country;

            saveDoc();

            Console.WriteLine("element updated");

        }

        else

        {

            Console.WriteLine("Update error. transistor not found");

        }

    }

    public XmlElement CreateMeasurementElement(XmlDocument doc, string tagName, float value, string currency)

    {

        XmlElement element = doc.CreateElement(tagName);

        element.SetAttribute("currency", currency);

        element.InnerText = value.ToString();

        return element;

    }

}

**ПРИЛОЖЕНИЕ B**

**Модульные тесты**

using System;

using System.IO;

using System.Linq;

using Xunit;

using Library;

using System.Globalization;

namespace Library.Tests

{

    public class DomToolkidTests : IDisposable

    {

        private readonly string \_testFilePath = "test\_transistors.xml";

        private readonly DomToolkid \_domToolkid;

        public DomToolkidTests()

        {

            // Создаем тестовый XML файл

            File.WriteAllText(\_testFilePath, "<root></root>");

            \_domToolkid = new DomToolkid(\_testFilePath);

        }

        public void Dispose()

        {

            // Удаляем тестовый файл после каждого теста

            if (File.Exists(\_testFilePath))

            {

                File.Delete(\_testFilePath);

            }

        }

        [Fact]

        public void Constructor\_ShouldCreateDocument\_WhenFileExists()

        {

            Assert.NotNull(\_domToolkid.\_doc);

            Assert.Equal(\_testFilePath, \_domToolkid.\_path);

        }

        [Fact]

        public void Constructor\_ShouldCreateNewDocument\_WhenFileNotExists()

        {

            var nonExistentPath = "nonexistent.xml";

            var toolkid = new DomToolkid(nonExistentPath);

            Assert.NotNull(toolkid.\_doc);

            File.Delete(nonExistentPath);

        }

        [Fact]

        public void NodeCreate\_ShouldAddNewTransistor()

        {

            var transistor = CreateTestTransistor(1);

            \_domToolkid.nodeCreate(transistor);

            var loaded = \_domToolkid.nodeShow(1);

            Assert.Equal(transistor.\_id, loaded.\_id);

        }

        [Fact]

        public void NodeShow\_ShouldReturnNull\_ForNonExistentId()

        {

            var result = \_domToolkid.nodeShow(999);

            Assert.Null(result);

        }

        [Fact]

        public void NodeRemove\_ShouldDeleteTransistor()

        {

            var transistor = CreateTestTransistor(1);

            \_domToolkid.nodeCreate(transistor);

            \_domToolkid.nodeRemove(1);

            var result = \_domToolkid.nodeShow(1);

            Assert.Null(result);

        }

        [Fact]

        public void NodeRemove\_ShouldNotThrow\_ForNonExistentId()

        {

            var exception = Record.Exception(() => \_domToolkid.nodeRemove(999));

            Assert.Null(exception);

        }

        [Fact]

        public void NodeUpdate\_ShouldModifyExistingTransistor()

        {

            var original = CreateTestTransistor(1);

            \_domToolkid.nodeCreate(original);

            var updated = new Transistor(

                1,

                "UpdatedName",

                new List<TransistorType> { TransistorType.MOSFET, TransistorType.IGBT },

                12.5f,

                0.8f,

                2.99f,

                "Germany");

            \_domToolkid.nodeUpdate(1, updated);

            var result = \_domToolkid.nodeShow(1);

            Assert.Equal("UpdatedName", result.\_name);

            Assert.Contains(TransistorType.MOSFET, result.\_types);

        }

        [Fact]

        public void SaveDoc\_ShouldPersistChangesToFile()

        {

            var transistor = CreateTestTransistor(1);

            \_domToolkid.nodeCreate(transistor);

            \_domToolkid.saveDoc();

            var fileContent = File.ReadAllText(\_testFilePath);

            Assert.Contains("<transistor id=\"001\">", fileContent);

        }

        [Fact]

        public void NodeCreate\_ShouldGenerateCorrectIdFormat()

        {

            var transistor = CreateTestTransistor(1);

            \_domToolkid.nodeCreate(transistor);

            var fileContent = File.ReadAllText(\_testFilePath);

            Assert.Contains("id=\"001\"", fileContent);

        }

        [Fact]

        public void MultipleNodeCreates\_ShouldWorkCorrectly()

        {

            for (int i = 1; i <= 5; i++)

            {

                \_domToolkid.nodeCreate(CreateTestTransistor(i));

            }

            for (int i = 1; i <= 5; i++)

            {

                var result = \_domToolkid.nodeShow(i);

                Assert.Equal(i, result.\_id);

            }

        }

        [Fact]

        public void NodeCreate\_ShouldHandleEmptyTypesList()

        {

            var transistor = new Transistor(

                1, "Test",

                new List<TransistorType>(),

                12.5f, 0.8f, 2.99f, "USA");

            var exception = Record.Exception(() => \_domToolkid.nodeCreate(transistor));

            Assert.Null(exception);

        }

        [Fact]

        public void NodeUpdate\_ShouldChangeId\_WhenNewIdProvided()

        {

            var original = CreateTestTransistor(1);

            \_domToolkid.nodeCreate(original);

            var updated = CreateTestTransistor(2);

            \_domToolkid.nodeUpdate(1, updated);

            Assert.Null(\_domToolkid.nodeShow(1));

            Assert.NotNull(\_domToolkid.nodeShow(2));

        }

        [Fact]

        public void NodeCreate\_ShouldThrow\_ForNullTransistor()

        {

            Assert.Throws<NullReferenceException>(() => \_domToolkid.nodeCreate(null));

        }

        private Transistor CreateTestTransistor(int id)

        {

            return new Transistor(

                id,

                "TestTransistor",

                new List<TransistorType> { TransistorType.BJT, TransistorType.MOSFET },

                12.0f,

                0.5f,

                1.99f,

                "USA");

        }

    }

}