МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ

образовательное учреждение

высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра защиты информации

**

**ОТЧЁТ**

**по лабораторной работе № 3**

**«Стандартные Сериализация и десериализация»**

**по дисциплине: «***Программирование***»**

Выполнил:Проверил:

Студент гр. «АБ-121», «АВТФ» *доцент кафедры ЗИ*

*Новиков Втюрин Александр Романович Архипова А. Б.*

«21» декабря 2022г«\_\_\_» \_\_\_\_\_\_ 2022 г.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (подпись)

Новосибирск 2022

**Цели и задачи работы:** изучение механизмов стандартной сериализации и десериализации в языке программирования высокого уровня.

**Задание к работе:**Реализовать приложения, которые обмениваются данными структур и классов, используя бинарную (базовую), XML и JSON сериализации. Предусмотреть десериализацию. Разработка на двух ЯП (C# и другой (не питон)).

**Теория:**

Сериализация имеющихся объектов – это процесс сохранения состояния объекта в последовательность байт. Механизм, который позволят «записывать» полученный прогресс для будущей выгрузки.

Десериализация – восстановление объектов из байт, сохранение которых было произведено ранее. Процедура выгрузки «зафиксированной» информации пользователем.

**CLR** (Common language runtime) — общеязыковая исполняющая среда. Она обеспечивает интеграцию языков и позволяет объектам благодаря стандартному набору типов и метаданным), созданным на одном языке, быть «равноправными гражданами» кода, написанного на другом.  
  
Другими словами CLR этот тот самый механизм, который позволяет программе выполняться в нужном нам порядке, вызывая функции, управляя данными. И все это для разных языков (c#, VisualBasic, Fortran).

**Distributed Component Object Model (DCOM)** - программная архитектура, разработанная компанией Microsoft для распределения приложений между несколькими компьютерами в сети. Программный сервер на одной из машин может использовать DCOM для передачи сообщения (его называют удаленным вызовом процедуры) к серверу на другой машине. DCOM автоматически устанавливает соединение, передает сообщение и возвращает ответ удаленного сервера. DCOM является развитием СОМ.

**ActiveX** — это небольшие приложения, с помощью которых веб-сайты предоставляют контент (например, видео и игры). Они также позволяют взаимодействовать с таким контентом, как панели инструментов и биржевые бегущие строки, при просмотре веб-страниц. Однако эти приложения иногда могут работать неправильно или выводить нежелательный контент. В некоторых случаях эти приложения могут выполнять сбор информации, повреждать данные на компьютере, устанавливать программное обеспечение без согласия пользователя и использоваться для удаленного управления компьютером.

.**Net** (читается как «дот нет») – это кросплатформенная среда выполнения приложений. Проще говоря – это то, что позволяет запускаться нашим приложениям в системе Microsoft Windows. Кросплатформенная – означает, что созданное приложение будет работать на всех процессорах и на всех операционных системах семейства Windows (за исключением самых ранних).

**OPC** – это набор повсеместно принятых спецификаций, предоставляющих универсальный механизм обмена данными в системах контроля и управления. Аббревиатура OPC традиционно расшифровывается как OLE for Process Control.  OLE – Object Linking and Embedding (связывание и встраивание объектов).

**COM** - это метод разработки программных компонентов, небольших двоичных исполняемых файлов, которые предоставляют необходимые сервисы приложениям, операционным системам и другим компонентам. Другими словами, COM определяет стандартный механизм, с помощью которого одна часть программного обеспечения предоставляет свои сервисы другой независимо от способа их реализации.  
COM является платформно-независимой, объектно-ориентированной технологией, позволяющей создавать бинарные компоненты. Эти компоненты можно использовать как локально, так и в распределенном сетевом окружении. COM служит основой для: OLE (технология составных документов), ActiveX-объектов и элементов управления ActiveX, DCOM, COM+. На базе COM создано большинство новейших продуктов (MS Office, MTS, …) и технологий Windows.

Бинарная (базовая) сериализация и десериализация

Сериализация — это процесс хранения состояния объекта на носителе хранилища. При двоичной сериализации открытые и закрытые поля объекта и имя класса, включая сборку, содержащую класс, преобразуются в поток байтов, который затем записывается в поток данных. После десериализации объекта создается точная копия исходного объекта.

Person person1 = new Person("Tom", 29);

Person person2 = new Person("Bill", 25);

// массив для сериализации

Person[] people = new Person[] { person1, person2 };

BinaryFormatter formatter = new BinaryFormatter();

using (FileStream fs = new FileStream("people.dat", FileMode.OpenOrCreate))

{

// сериализуем весь массив people

formatter.Serialize(fs, people);

Console.WriteLine("Объект сериализован");

}

// десериализация

using (FileStream fs = new FileStream("people.dat", FileMode.OpenOrCreate))

{

Person[] deserilizePeople = (Person[])formatter.Deserialize(fs);

foreach (Person p in deserilizePeople)

{

Console.WriteLine($"Имя: {p.Name} --- Возраст: {p.Age}");

}

}

Json сериализация и десериализация

Пространство имен System.Text.Json предоставляет функциональные возможности для сериализации в нотацию объектов JavaScript (JSON) и десериализации объектов из этой нотации. *Сериализация* — это процесс преобразования состояния объекта, то есть значений его свойств в форму, которая может храниться или передаваться. Сериализованная форма не содержит никаких сведений о связанных методах объекта. *Десериализация* воссоздает объект из сериализованной формы.

using System.Text.Json;

// сохранение данных

using (FileStream fs = new FileStream("user.json", FileMode.OpenOrCreate))

{

Person tom = new Person("Tom", 37);

await JsonSerializer.SerializeAsync<Person>(fs, tom);

Console.WriteLine("Data has been saved to file");

}

// чтение данных

using (FileStream fs = new FileStream("user.json", FileMode.OpenOrCreate))

{

Person? person = await JsonSerializer.DeserializeAsync<Person>(fs);

Console.WriteLine($"Name: {person?.Name} Age: {person?.Age}");

}

class Person

{

public string Name { get;}

public int Age { get; set; }

public Person(string name, int age)

{

Name = name;

Age = age;

}

}

using System.Text.Json;

Person tom = new Person("Tom", 37);

var options = new JsonSerializerOptions

{

WriteIndented = true

};

string json = JsonSerializer.Serialize<Person>(tom, options);

Console.WriteLine(json);

Person? restoredPerson = JsonSerializer.Deserialize<Person>(json);

Console.WriteLine(restoredPerson?.Name);

class Person

{

[JsonPropertyName("firstname")]

public string Name { get;}

[JsonIgnore]

public int Age { get; set; }

public Person(string name, int age)

{

Name = name;

Age = age;

}

}

JsonPropertyName – позволяет замещать оригинальные названия свойства

JsonIgnore – позволяет исключать из сериализации определённые свойства

Xml сериализация и десериализация

Сериализация представляет собой процесс преобразования объекта в форму, подготовленную для передачи. Например, можно сериализовать объект и передать его по Интернету с использованием протокола HTTP между клиентом и сервером. И наоборот, при десериализации объект воссоздается из потока.

При XML-сериализации в поток XML сериализуются только открытые поля и значения свойств объекта. XML-сериализация не учитывает информацию о типе. Например, если имеется объект **Book**, который существует в пространстве имен **Library**, нет никакой гарантии, что он десериализуется в объект аналогичного типа.

Person[] people = new Person[]

{

new Person("Tom", 37),

new Person("Bob", 41)

};

XmlSerializer formatter = new XmlSerializer(typeof(Person[]));

// ?????????? ??????? ? ????

using (FileStream fs = new FileStream("people.xml", FileMode.OpenOrCreate))

{

formatter.Serialize(fs, people);

}

// ?????????????? ??????? ?? ?????

using (FileStream fs = new FileStream("people.xml", FileMode.OpenOrCreate))

{

Person[]? newpeople = formatter.Deserialize(fs) as Person[];

if(newpeople != null)

{

foreach (Person person in newpeople)

{

Console.WriteLine($"Name: {person.Name} --- Age: {person.Age}");

}

}

}

public class Person

{

public string Name { get; set; } = "Undefined";

public int Age { get; set; } = 1;

public Person() { }

public Person(string name, int age)

{

Name = name;

Age = age;

}

}

С#

MainWindow.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab3Sharp

{

public partial class MainWindow : Form

{

public MainWindow()

{

InitializeComponent();

Serialization\_Type.Items.AddRange(new string[] { "Binary", "XML", "JSON"});

Serialization\_Type.SelectedItem = Serialization\_Type.Items[0];

Deserialization\_Type.Items.AddRange(new string[] { "Binary", "XML", "JSON" });

Deserialization\_Type.SelectedItem = Deserialization\_Type.Items[0];

ClearInput();

}

public void ClearInput()

{

product\_info.Clear();

product\_id\_info.Clear();

warehouse\_cell\_info.Clear();

product\_quantity\_info.Clear();

}

private void Add\_product\_Click(object sender, EventArgs e)

{

Product product = new Product(product\_info.Text, product\_id\_info.Text, warehouse\_cell\_info.Text, product\_quantity\_info.Text);

Add(product);

}

private void listProduct\_SelectedIndexChanged(object sender, EventArgs e)

{

if (listProduct.SelectedItems.Count == 1)

{

Product product = (Product)listProduct.SelectedItems[0].Tag;

if (product != null)

{

product\_info.Text = product.product;

product\_id\_info.Text = product.product\_id;

warehouse\_cell\_info.Text = product.warehouse\_cell;

product\_quantity\_info.Text = Convert.ToString(product.product\_quantity);

}

}

else if (listProduct.SelectedItems.Count == 0)

{

ClearInput();

}

}

private void Serialize\_Click(object sender, EventArgs e)

{

Serialization serialization = new Serialization();

Products products = new Products();

if (Serialization\_Type.SelectedItem.ToString() == "XML")

{

foreach(ListViewItem item in listProduct.Items)

{

products.products.Add((Product)item.Tag);

}

serialization.XML\_Serialization(products);

}

if (Serialization\_Type.SelectedItem.ToString() == "Binary")

{

foreach (ListViewItem item in listProduct.Items)

{

products.products.Add((Product)item.Tag);

}

serialization.Binar\_Serialization(products);

}

if (Serialization\_Type.SelectedItem.ToString() == "JSON")

{

foreach (ListViewItem item in listProduct.Items)

{

products.products.Add((Product)item.Tag);

}

serialization.JSON\_Serialization(products);

}

}

private void Add (Product product)

{

ClearInput();

ListViewItem LVI = new ListViewItem(product.product);

LVI.Tag = product;

listProduct.Items.Add(LVI);

}

private void Deserialize\_Click(object sender, EventArgs e)

{

Deserialization deserialization = new Deserialization();

Products products = new Products();

if (Deserialization\_Type.SelectedItem.ToString() == "XML")

{

ClearInput();

products = deserialization.XML\_Deserialization();

foreach (Product product in products.products)

{

Add(product);

}

}

if (Deserialization\_Type.SelectedItem.ToString() == "Binary")

{

ClearInput();

products = deserialization.Binar\_Deserialization();

foreach (Product product in products.products)

{

Add(product);

}

}

if (Deserialization\_Type.SelectedItem.ToString() == "JSON")

{

ClearInput();

products = deserialization.JSON\_Deserialization();

foreach (Product product in products.products)

{

Add(product);

}

}

}

private void MainWindow\_Load(object sender, EventArgs e)

{

}

}

}

Product.cs

using Microsoft.VisualBasic.ApplicationServices;

using System;

using System.Collections.Generic;

using System.Configuration;

using System.IO;

using System.Linq;

using System.Runtime.Serialization.Formatters.Binary;

using System.Text;

using System.Text.Json;

using System.Threading.Tasks;

using System.Xml.Serialization;

namespace Lab3Sharp

{

[Serializable]

public class Product

{

public string product = "";

public string product\_id = "";

public string warehouse\_cell = "";

public int product\_quantity;

public Product() { }

public Product(string product, string product\_id, string warehouse\_cell, string product\_quantity)

{

this.product = product;

this.product\_id = product\_id;

this.warehouse\_cell = warehouse\_cell;

this.product\_quantity = Convert.ToInt32(product\_quantity);

}

}

[Serializable]

public class Products

{

public List <Product> products = new List <Product> ();

public Products() { }

}

public class Serialization: Products

{

public void Binar\_Serialization(Products products)

{

BinaryFormatter formatter = new BinaryFormatter();

using (FileStream fs = new FileStream("Binar\_Serialization.dat", FileMode.OpenOrCreate))

{

formatter.Serialize(fs, products);

}

MessageBox.Show("Binary serialization was successful");

}

public void XML\_Serialization(Products products)

{

XmlSerializer writer = new XmlSerializer(typeof(Products));

using (FileStream fs = new FileStream("XMLSerialize.xml", FileMode.OpenOrCreate))

{

writer.Serialize(fs, products);

}

MessageBox.Show("XML serialization was successful");

}

public void JSON\_Serialization(Products products)

{

//string jsonString = JsonSerializer.Serialize(products);

//File.WriteAllText("JSON\_Serialization.json", jsonString);

FileStream file3 = new FileStream("JSON\_Serialization.json", FileMode.OpenOrCreate);

StreamWriter stream = new StreamWriter(file3);

for (int i = 0; i < products.products.Count; i++)

{

string json1 = JsonSerializer.Serialize(products.products[i]);

stream.WriteLine(json1);

}

stream.Close();

file3.Close();

MessageBox.Show("JSON serialization was successful");

}

}

public class Deserialization: Products

{

public Products Binar\_Deserialization()

{

BinaryFormatter formatter = new BinaryFormatter();

using (FileStream fs = new FileStream("Binar\_Serialization.dat", FileMode.OpenOrCreate))

{

MessageBox.Show("Binar deserialization was successful");

return (Products)formatter.Deserialize(fs);

}

}

public Products XML\_Deserialization()

{

XmlSerializer xml = new XmlSerializer(typeof(Products));

using (FileStream fs = new FileStream("XMLSerialize.xml", FileMode.OpenOrCreate))

{

MessageBox.Show("XML deserialization was successful");

Products? products = xml.Deserialize(fs) as Products;

return products;

}

}

public Products JSON\_Deserialization()

{

Products products = new Products();

StreamReader f = new StreamReader("JSON\_Serialization.json");

while (!f.EndOfStream)

{

string s = f.ReadLine();

products.products.Add(JsonSerializer.Deserialize<Product>(s));

}

f.Close();

return products;

}

}

}

Programm.cs

namespace Lab3Sharp

{

internal static class Program

{

/// <summary>

/// The main entry point for the application.

/// </summary>

[STAThread]

static void Main()

{

// To customize application configuration such as set high DPI settings or default font,

// see https://aka.ms/applicationconfiguration.

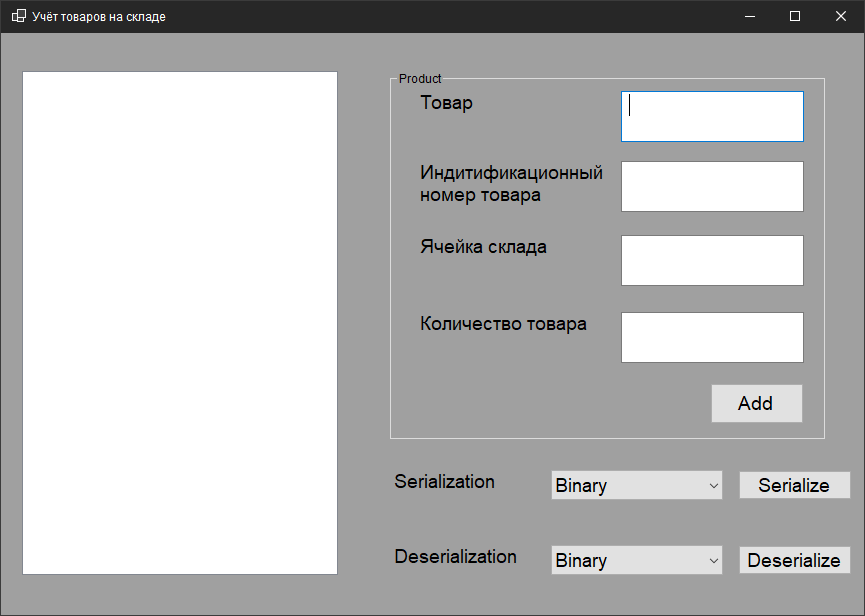
ApplicationConfiguration.Initialize();

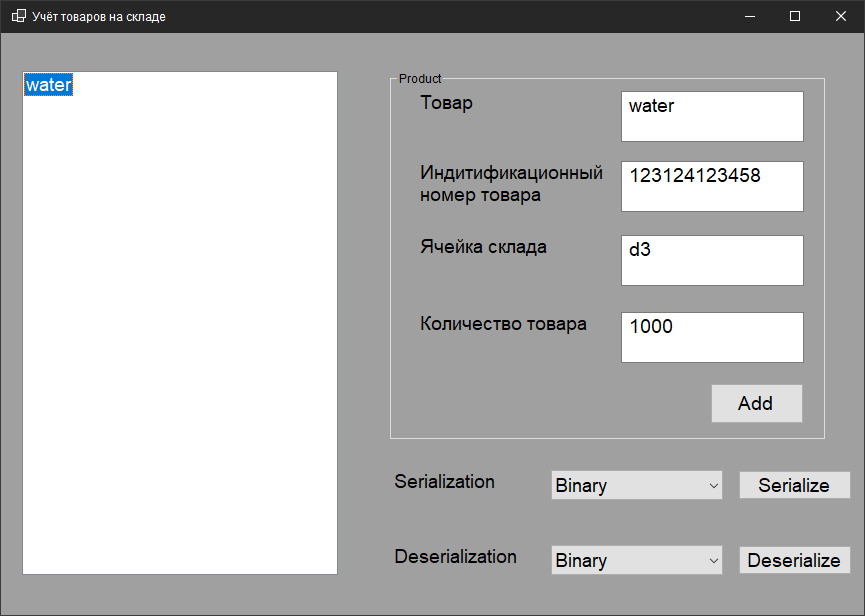
Application.Run(new MainWindow());

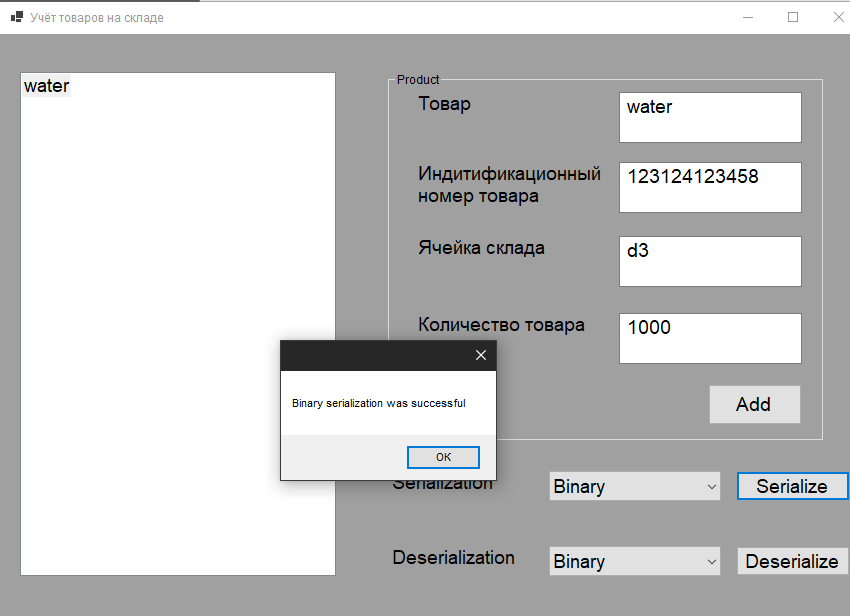
}

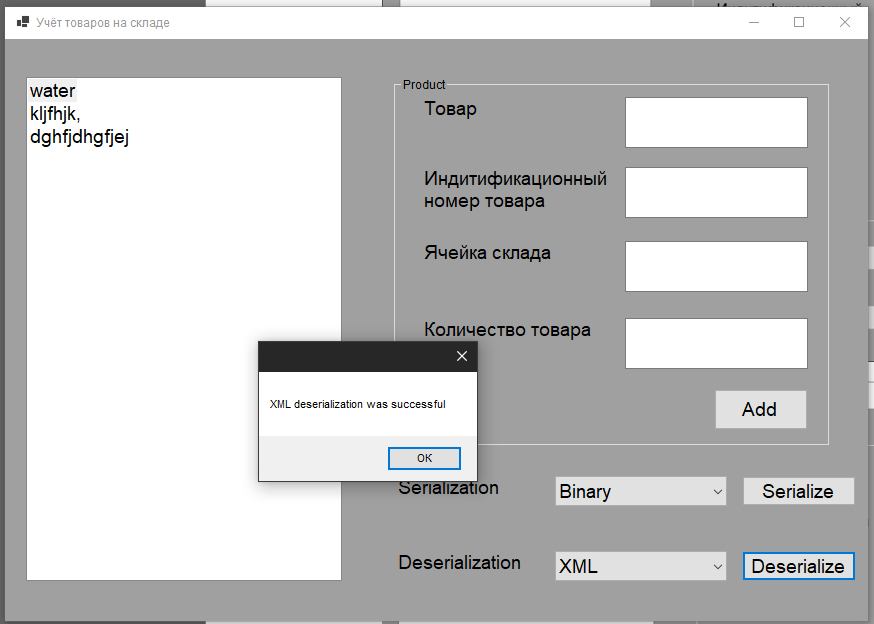
}

}









Java Script

const v8 = require('v8');

const util = require('util');

let fs = require('fs');

let fs1 = require('fs');

let fs2 = require('fs');

const xml2js = require('xml2js');

class Product {

product;

productid;

warehousecell;

productquantity;

constructor(product, product\_id, warehouse\_cell, product\_quantity) {

this.product = product;

this.productid = product\_id;

this.warehousecell = warehouse\_cell;

this.productquantity = product\_quantity;

}

}

const product = new Product("water", "89", "B3", 500);

console.log(product);

const jprod = JSON.stringify(product);

fs.writeFileSync('JsonSerialization.json', jprod);

const product1 = JSON.parse(fs.readFileSync('JsonSerialization.json', 'utf8'));

console.log(product1);

const binprod = v8.serialize(product);

fs1.writeFileSync('BinarySerialization.dat', binprod);

console.log(binprod);

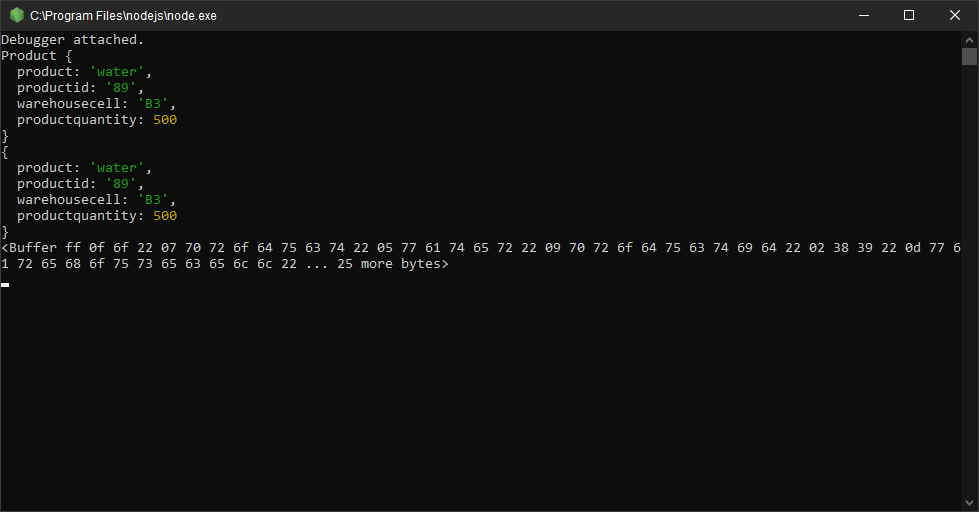
const builder = new xml2js.Builder();

const xml = parser.buildObject(product);

console.log(xml);

fs2.writeFileSync('XMLSerialization.xml', xml);

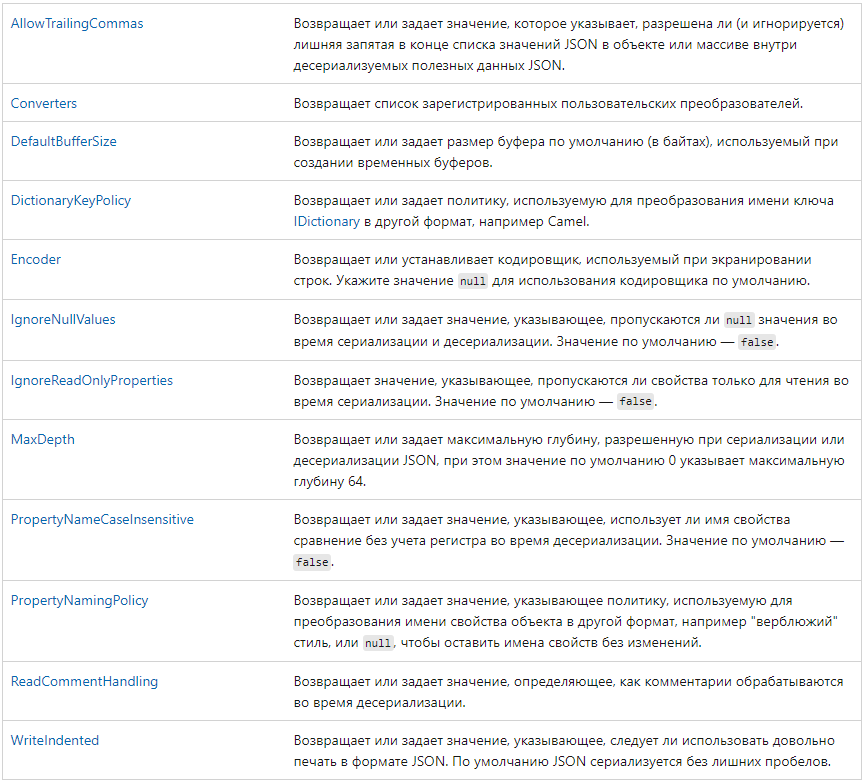
console.log(product1);



Опции сериализации:

JsonSerializerOptions – класс опций.

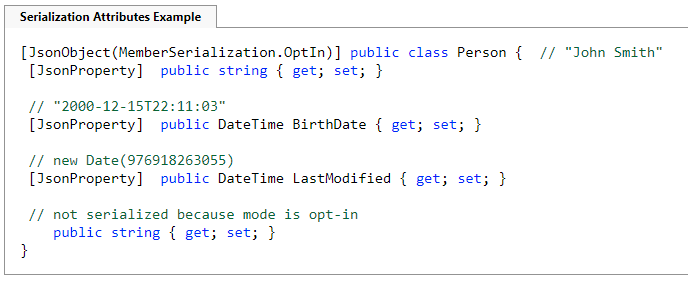
Его свойства:

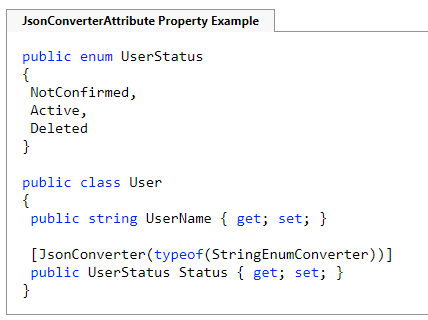


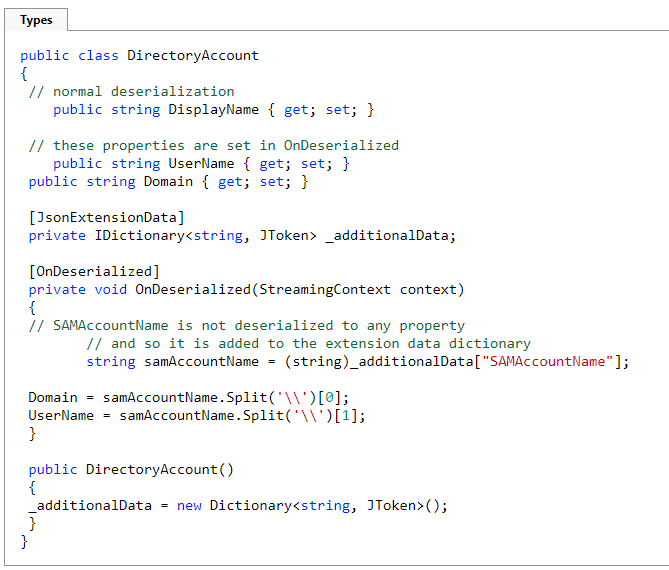
Атрибуты сериализации:

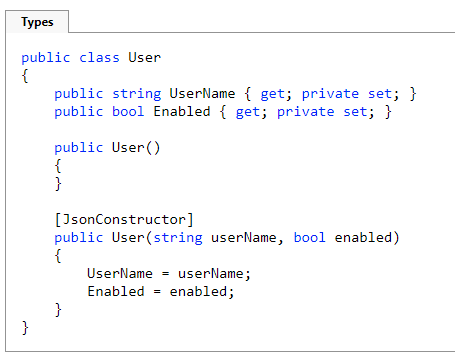
Атрибуты могут использоваться для управления тем, как работает Json.NET сериализует и десериализует объекты .NET.

* [JsonObjectAttribute](https://www.newtonsoft.com/json/help/html/T_Newtonsoft_Json_JsonObjectAttribute.htm) - Помещаются в классы для управления тем, как они должны быть сериализованы как объект JSON.
* [JsonArrayAttribute](https://www.newtonsoft.com/json/help/html/T_Newtonsoft_Json_JsonArrayAttribute.htm) - Помещается в коллекции, чтобы контролировать, как они должны быть сериализованы в виде массива JSON.
* [Атрибут JsonDictionaryAttribute](https://www.newtonsoft.com/json/help/html/T_Newtonsoft_Json_JsonDictionaryAttribute.htm) - Помещаются в словари для управления тем, как они должны быть сериализованы как объект JSON.
* [Атрибут JsonPropertyAttribute](https://www.newtonsoft.com/json/help/html/T_Newtonsoft_Json_JsonPropertyAttribute.htm) - Помещаются в поля и свойства для управления тем, как они должны быть сериализованы как свойство в объекте JSON.
* [JsonConverterAttribute](https://www.newtonsoft.com/json/help/html/T_Newtonsoft_Json_JsonConverterAttribute.htm) - Помещается либо в классы, либо в поля и свойства, чтобы указать, какой JsonConverter следует использовать во время сериализации.
* [Атрибут JsonExtensionDataAttribute](https://www.newtonsoft.com/json/help/html/T_Newtonsoft_Json_JsonExtensionDataAttribute.htm) - Помещается в поле или свойство коллекции для десериализации свойств, не имеющих соответствующего члена класса, в указанную коллекцию и записи значений во время сериализации.
* [Атрибут JsonConstructorAttribute](https://www.newtonsoft.com/json/help/html/T_Newtonsoft_Json_JsonConstructorAttribute.htm) - Помещается в конструктор, чтобы указать, что он должен использоваться для создания класса во время десериализации.









**Вывод**

В ходе выполнения лабораторной работы были реализованы приложения, которые обмениваются данными классов, используя бинарную (базовую), XML и JSON сериализации + предусмотрена десериализация. Разработка на двух ЯП (C# и JavaScript). Создано приложение с графическим дизайном, выполняющее уникальную задачу. Составлен раздел отчета с описанием теоретической базы сериализации/ десериализации, минимум 7 атрибутов стандартной сериализации, опций сериализации + продемонстрировано на примерах. Было осуществено знакомство с необходимой теоретической базой.

**Используемые источники:**

1. <https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/concepts/serialization/>
2. <https://docs.microsoft.com/ru-ru/dotnet/standard/serialization/basic-serialization>
3. <https://www.newtonsoft.com/json/help/html/SerializationAttributes.htm>
4. <https://docs.microsoft.com/ru-ru/dotnet/api/system.text.json.jsonserializeroptions?view=netcore-3.0>
5. <https://www.npmjs.com/package/xml2js>