ЛЕКЦИЯ 8

- Модуль 3
- 31.01.2024
- Сериализация данных
- Двоичная и JSON-сериализация

ЦЕЛИ ЛЕКЦИИ

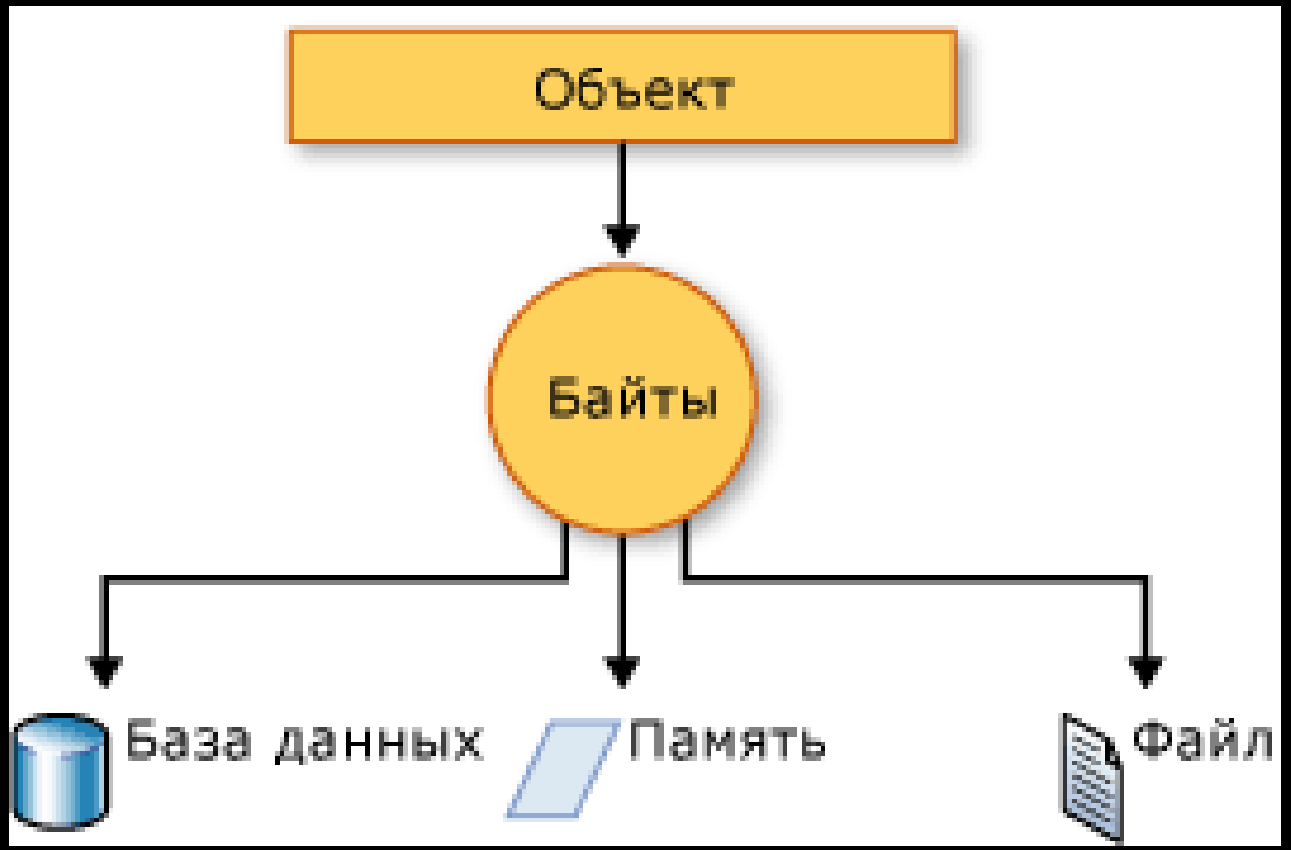
- Изучить понятие сериализации
- Разобраться с двоичной и JSON-сериализацией
- Посмотреть на приложение атрибутирования и рефлексии к процессам сериализации

ПОНЯТИЕ СЕРИАЛИЗАЦИИ

- **Сериализация** – процесс преобразования структуры данных в последовательность байтов (или XML-узлов / JSON / etc.).
- **Десериализация** – восстановление структуры данных из последовательности байтов (или XML-узлов / JSON / etc.).

- Disclaimer: в исходных кодах данной презентации отсутствует код обработки исключений (исключительно для удобочитаемости и отсутствия излишних нагромождений). Однако, это не означает, что исключения не надо обрабатывать…
  - SerializationException (как и многое другое) может выпасть при сериализации и десереализации

ЗАЧЕМ НУЖНА СЕРИАЛИЗАЦИЯ?



МЕХАНИЗМЫ СЕРИАЛИЗАЦИИ .NET

- контракты данных
- двоичная
- SOAP-сериализация
- XML-сериализация
- IXmlSerializable
- JSON-сериализация

КОНТРАКТЫ ДАННЫХ

- **Контракт данных** - формальное соглашение между службой и клиентом, абстрактно описывающее данные, обмен которыми происходит
  - Это значит, что для взаимодействия клиент и служба не обязаны совместно использовать одни и те же типы, достаточно совместно использовать одни и те же контракты данных
  - Контракт данных для каждого параметра и возвращаемого типа четко определяет, какие данные сериализуются (превращаются в XML/JSON/etc…) для обмена

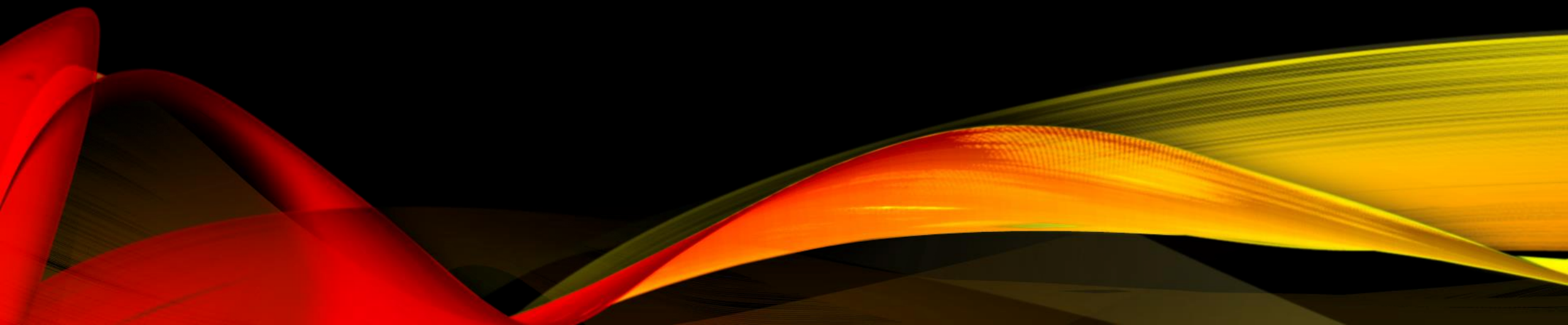
- Появились в .NET как часть Windows Communication Foundation (WCF)

«ШАГИ» СЕРИАЛИЗАЦИИ (НА ПРИМЕРЕ ХРАНЕНИЯ В ФАЙЛЕ)

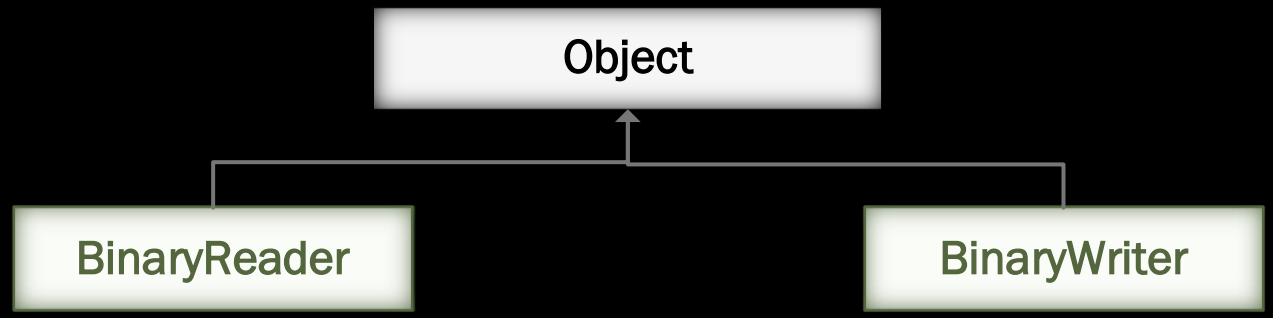
1. Создать объект класса
2. Создать байтовый поток и связать его с потоком для записи (например, FileStream)
3. Создать объект сериализации, называемый форматером
4. Используя метод Serialize() / WriteObject() объекта-форматера сохранить в потоке представление объекта
5. Закрыть поток

Примечание: в качестве целевого хранилища можно вместо файла использовать память (MemoryStream), сеть (NetworkStream) и т.д.

АДАПТЕРЫ ДВОИЧНОГО ПОТОКА



КЛАССЫ-АДАПТЕРЫ ДЛЯ ЧТЕНИЯ И ЗАПИСИ БИНАРНЫХ ФАЙЛОВ



КОНСТРУКТОРЫ КЛАССОВ BINARYWRITER И BINARYREADER

- BinaryWriter (Stream поток)
- BinaryWriter (Stream поток, Encoding кодировка)
- BinaryWriter (Stream поток, Encoding кодировка, Bool открыт)

- BinaryReader (Stream поток)
- BinaryReader (Stream поток, Encoding кодировка)
- BinaryReader (Stream поток, Encoding кодировка, Bool открыт)

СРЕДСТВА ЗАПИСИ BINARYWRITER

- public virtual void Write(byte[ ] буфер)
- public virtual void Write(byte[ ] буфер, int индекс- начало, int счетчик)
- public virtual void Write(char[] буфер, int индекс- начало, int счетчик)
- protected void Write7BitEncodedInt(int value)
- public virtual void Write(. . .)

Класс BinaryWriter реализует интерфейс IDisposable

ЧЛЕНЫ КЛАССА BINARYWRITER

Методы:

- `public virtual void Close()` – Закрывает бинарный и базовый потоки
- `public void Dispose()` – Освобождает ресурсы
- `public virtual void Flush()` – Очищает буферы
- `public virtual long Seek(int смещение, SeekOrigin точка_отсчета)` – устанавливает позицию записи

- `BaseStream` – свойство класса `BinaryWriter`

BINARYREADER

Конструкторы:

- `BinaryReader (Stream поток)`
- `BinaryReader (Stream поток, Encoding кодировка)`
- `BinaryReader (Stream поток, Encoding кодировка, Bool открыт)`
- открыт == leaveOpen

`BaseStream` – свойство

Методы:

- `Close()`
- `PeekChar()` – “подсмотреть” символ (возвращает следующий доступный для чтения символ, не перемещая позицию байта или символа вперед).
- `Dispose()`

ЧТЕНИЕ ИЗ BINARYREADER

- `int Read()` – чтение отдельного символа
- `void Read(byte[] буфер, int индекс-начало, int счётчик)`
- `void Read(char[] буфер, int индекс-начало, int счётчик)`

- `int Read7BitEncodedInt()` – читает упакованное целое число
- Если целое число будет помещаться в семь бит, целое число займет только один байт. (ожидается, что целое число записали через `BinaryWriter.Write7BitEncodedInt()`)

Исключение IOException

ЧТЕНИЕ ИЗ BINARYREADER ЗНАЧЕНИЙ БАЗОВЫХ ТИПОВ

- bool ReadBoolean()
- byte ReadByte()
- byte [] ReadBytes(Int32)
- char ReadChar()
- char [] ReadChars(Int32)
- decimal ReadDecimal()
- double ReadDouble()
- short ReadInt16()
- int ReadInt32()
- long ReadInt64()
- sbyte ReadSByte()
- float ReadSingle()
- string ReadString()
- ushort ReadUInt16()
- uint ReadUInt32()
- ulong ReadUInt64()

ПРИМЕР. ЗАПИСЬ И ЧТЕНИЕ БИНАРНОГО ФАЙЛА

```
struct Discovery {
    public string Name { get; set; }
    public int Date { get; set; }
}
```

Данные из структуры «Изобретение» пишем в бинарный файл

```
static class BinFileOp {
    public static void WriteData(string path)
    { … }
    public static void ReadData(string path)
    { … }
}
```

Для чтения и записи создадим статический класс с двумя статическими методами

```
BinFileOp.WriteData(@"..\..\..\data.bin");
Console.OutputEncoding = Encoding.UTF8;
Console.WriteLine();
BinFileOp.ReadData(@"..\..\..\data.bin");
```

Тестовый код

https://replit.com/@olgamaksimenkova/BinFileExample

ПРИМЕР. ЗАПИСЬ И ЧТЕНИЕ БИНАРНОГО ФАЙЛА

```
public static void WriteData(string path)
    {
        Discovery[] discoveries =  {
    new Discovery { Name = "Радиоприемник", Date = 1895 },
    new Discovery { Name = "Мазер", Date = 1954 } ,
    new Discovery { Name = "Парашют", Date = 1911 },
    new Discovery { Name = "Гальванопластика", Date = 1840 },
    new Discovery { Name = "Коллайдер", Date = 1960 },
    new Discovery { Name = "Иконоскоп", Date = 1929 }
  };
        using (FileStream fs = File.Create(path))
        using (BinaryWriter bw = new BinaryWriter(fs))
            foreach (Discovery dis in discoveries)
            {
                bw.Write(dis.Name);
                bw.Write(dis.Date);
            }
    }
```

ПРИМЕР. ЗАПИСЬ И ЧТЕНИЕ БИНАРНОГО ФАЙЛА

```
public static void ReadData(string path)
    {
        using (FileStream fs = new FileStream(path, FileMode.Open))
        using (BinaryReader br = new BinaryReader(fs))
        {
            while (true)
                try
                {
                    string name = br.ReadString();
                    int date = br.ReadInt32();
                    Console.WriteLine("Name={0}, Date={1}", name, date);
                }
                catch (EndOfStreamException) { break; }
        }

    }
```

ПРИМЕР С BINARYWRITER

```
using System;
using System.IO;
class Program
{
    static void Main()
    {
        // Запись целых в двоичный поток.
        BinaryWriter fOut = new BinaryWriter(new FileStream("t.dat", FileMode.Create));
        for (int i = 0; i <= 10; i += 2)
        {
            fOut.Write(i);
        }
        fOut.Close();
    }
}
```

ПРИМЕР С BINARYREADER

```
FileStream f = new FileStream("t.dat", FileMode.Open);
BinaryReader fIn = new BinaryReader(f);
long n = f.Length / 4; // Определяем количество целых 4байтовых в потоке.
int a;
for (int i = 0; i < n; i++)
{
    a = fIn.ReadInt32();
    Console.Write(a + " ");
}
fIn.Close();
f.Close();
```

Вывод:
0 2 4 6 8 10

ПРИМЕР С BINARYREADER И ПОЗИЦИОНИРОВАНИЕМ

```
FileStream f = new FileStream("t.dat", FileMode.Open);
BinaryReader fIn = new BinaryReader(f);
long n = f.Length / 4; // Определяем количество целых 4байтовых в потоке.
int a;
for (int i = 0; i < n; i+=4)
{
    f.Seek(i, SeekOrigin.Begin ); // Позиционирование при чтении бинарного потока.
    a = fIn.ReadInt32();
    Console.Write(a + " ");
}
fIn.Close();
f.Close();
```

Вывод:
0 2

ПРИМЕР. ЗАПИСЬ ЦЕЛЫХ В СЖАТОМ ФОРМАТЕ

```
public class MyBinaryWriter : BinaryWriter
{
    public MyBinaryWriter(Stream stream) : base(stream) { }
    public new void Write7BitEncodedInt(int i)
    {
        base.Write7BitEncodedInt(i);
    }
}
public class MyBinaryReader : BinaryReader
{
    public MyBinaryReader(Stream stream) : base(stream) { }
    public new int Read7BitEncodedInt()
    {
        return base.Read7BitEncodedInt();
    }
}
```

protected void Write7BitEncodedInt(int value) – записывает 32-битное целое в сжатом формате
protected int Read7BitEncodedInt()

ПРИМЕР. ЗАПИСЬ ЦЕЛЫХ В СЖАТОМ ФОРМАТЕ

```
MemoryStream stream = new MemoryStream();
MyBinaryWriter writer = new MyBinaryWriter(stream);
writer.Write7BitEncodedInt(127);
Console.WriteLine("BaseStream.Length = " + stream.Length);
writer.Write7BitEncodedInt(127);
Console.WriteLine("BaseStream.Length = " + stream.Length);
writer.Write7BitEncodedInt(256);
Console.WriteLine("BaseStream.Length = " + stream.Length);
writer.Write7BitEncodedInt(4096);
Console.WriteLine("BaseStream.Length = " + stream.Length);
writer.Write7BitEncodedInt(-4096);
Console.WriteLine("BaseStream.Length = " + stream.Length);
```

```
stream.Position = 0; // Чтение из битового потока с начала.
MyBinaryReader reader = new MyBinaryReader(stream);
Console.WriteLine(reader.Read7BitEncodedInt());
Console.WriteLine(reader.Read7BitEncodedInt());
Console.WriteLine(reader.Read7BitEncodedInt());
Console.WriteLine(reader.Read7BitEncodedInt());
Console.WriteLine(reader.Read7BitEncodedInt());
```

КЛАСС, ПРИГОДНЫЙ ДЛЯ СЕРИАЛИЗАЦИИ

Применение атрибута, указывающего о возможности сериализации объекта типа

```
[Serializable]
public class Point
{
    public double X { get; set; }
    public double Y { get; set; }
    public Point(double x, double y) => (X, Y) = (x, y);
    public Point() : this(0.0, 0.0) { }
    public double Distance(Point point) =>
        Math.Sqrt((X - point.X) * (X - point.X) + (Y - point.Y) * (Y - point.Y));
    public double Distance(double x, double y) =>
        Math.Sqrt((X - x) * (X - x) + (Y - y) * (Y - y));
    public override string ToString() => $"x={X:f2}; y={Y:f2}";
}
```

ДВОИЧНАЯ СЕРИАЛИЗАЦИЯ ОБЪЕКТА

1. Пространство имен двоичной сериализации:
   - `using System.Runtime.Serialization.Formatters.Binary;`
2. Создание двоичного форматера – объекта класса:
   - `BinaryFormatter bf = new BinaryFormatter();`
3. Создание байтового потока (и файла):
   - `FileStream fs = new FileStream("Point.bin", FileMode.Create)`
4. Собственно сериализация – обращение к методу:
   - `bf.Serialize(байтовый_поток, сериализуемый_объект);`
5. Здесь байтовый_поток – это `fs` или ссылка на другой источник

ДВОИЧНАЯ СЕРИАЛИЗАЦИЯ ОДНОГО ОБЪЕКТА

```
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;
```

```
 Point point = new Point(1,1);
    BinaryFormatter bf = new BinaryFormatter();
    using (FileStream fs = new FileStream("Point.bin", FileMode.Create))
    {
        bf.Serialize(fs, point);
    }
}
```

Сериализует / десериализует объект и полный граф связанных объектов в двоичном формате

Результат сериализации в файле Point.bin:

```
????яяяя??????????JBinarySerialization, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=null???BinarySerialization.Point???<X>k__BackingField<Y>k__Ba
ckingField??????????p???????p?
```

ДВОИЧНАЯ ДЕСЕРИАЛИЗАЦИЯ ОБЪЕКТА

1. Пространство имен двоичной десериализации:

- `using System.Runtime.Serialization.Formatters.Binary;`

2. Создать ссылку с типом десериализуемого объекта:

- `Point newPoint = null;`

3. Создать объект сериализации (форматер):

- `BinaryFormatter binformatter = new BinaryFormatter();`

4. Создать байтовый поток и связать его с файлом (FileStream):

- `FileStream fs = new FileStream("Point.bin", FileMode.Open)`

5. Выполнить десериализацию методом Deserialize(поток):

- `newPoint = (Point)bf.Deserialize(fs);`

ПРИМЕР ДВОИЧНОЙ ДЕСЕРИАЛИЗАЦИИ

```
Point newPoint = null;
using(FileStream fs = new FileStream("Point.bin", FileMode.Open))
{
    newPoint = (Point)bf.Deserialize(fs);
    Console.WriteLine(newPoint.ToString());
}
```

Результат выполнения десериализации и печати:
x=1,00; y=1,00

ДВОИЧНАЯ СЕРИАЛИЗАЦИЯ МАССИВА ОБЪЕКТОВ

```
// Сериализация массива объектов.
Point[] points = { new Point(1,1), new Point(3,2), new Point(0,1)};
BinaryFormatter bf = new BinaryFormatter();

using(FileStream fs = new FileStream("Points.bin", FileMode.Create))
{
    bf.Serialize(fs, points);
}
```

Результат сериализации в файле Points.bin

```
    ????яяяя??????????JBinarySerialization, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=null ?????????BinarySerialization.Point???  ??? ???
    ??????BinarySerialization.Point???<X>k__BackingField<Y>k__BackingField??????
????p???????p????????????@???????@????????????????????p?
```

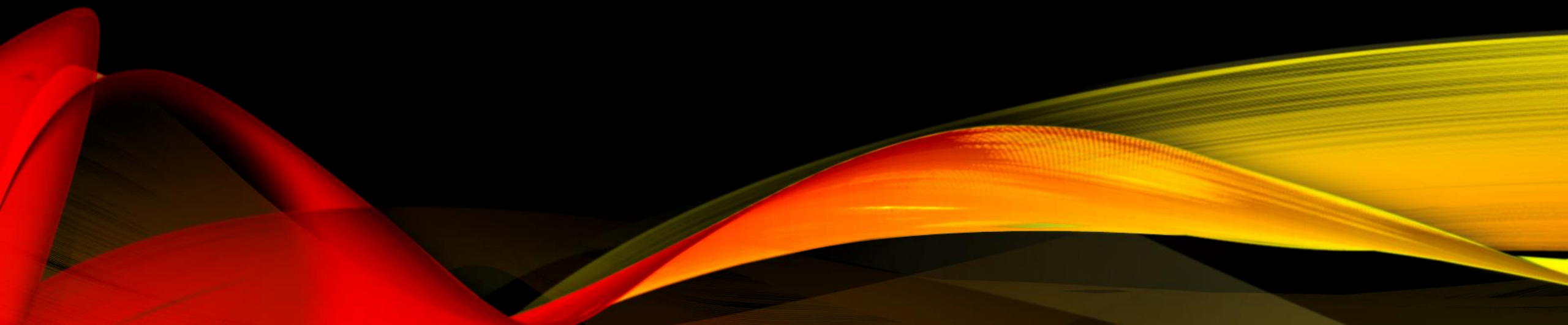
ДВОИЧНАЯ ДЕСЕРИАЛИЗАЦИЯ МАССИВА ОБЪЕКТОВ

```
Point[] restored = null;
using (FileStream fs = new FileStream("Points.bin", FileMode.Open))
{
    restored = (Point[])bf.Deserialize(fs);
    Console.WriteLine("Восстановленные объекты:");
    foreach(Point p in restored)
    {
        Console.WriteLine(p);
    }
}
```

Вывод:

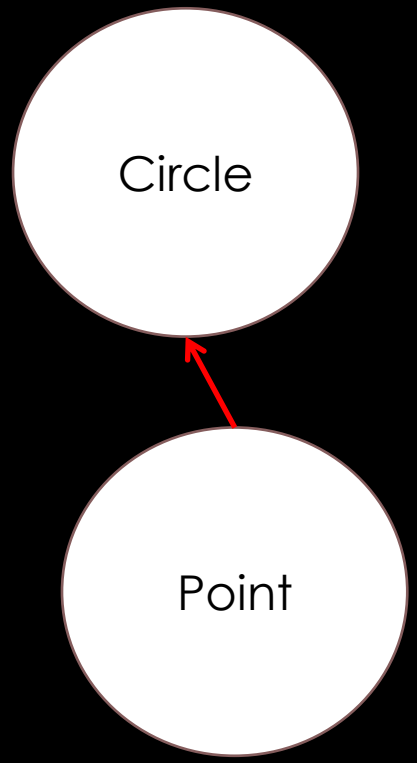
```
Восстановленные объекты:
x=1,00; y=1,00
x=3,00; y=2,00
x=0,00; y=1,00
```

ГРАФ СЕРИАЛИЗАЦИИ



ГРАФ ОБЪЕКТОВ (АГРЕГАЦИЯ)

- **Граф объектов** – участвующий в процедуре сериализации набор взаимосвязанных объектов



КЛАССЫ В ОТНОШЕНИИ КОМПОЗИЦИИ

```
[Serializable]
public class Point
{
    public double X { get; set; }
    public double Y { get; set; }
    public Point(double a, double b) => (X,Y) = (a,b);
    // Расстояние между точками.
    public double Distance(Point ps)
    {
        double dx = X - ps.X;
        double dy = Y - ps.Y;
        return Math.Sqrt(dx * dx + dy * dy);
    }
}   // class Point
```

```
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;
```

```
[Serializable] // Каждый класс сериализуем отдельно.
public class Circle
{
    public Point Center { get; set; }    // Центр круга.
    double rad;                 // Радиус круга.
    public Circle(double xc, double yc, double rad)
    {
        Center = new Point(xc, yc); // Композиция классов.
        this.rad = rad;
    }   // Circle( )
    public override string ToString() =>
        $"xc={Center.X:g5}\tyc={Center.X:g5},\tRad={rad:g5}" ;

    public double Rad { get => rad; } // Радиус круга.
}   // class Circle
```

ДВОИЧНАЯ СЕРИАЛИЗАЦИЯ АГРЕГИРОВАННЫХ ОБЪЕКТОВ

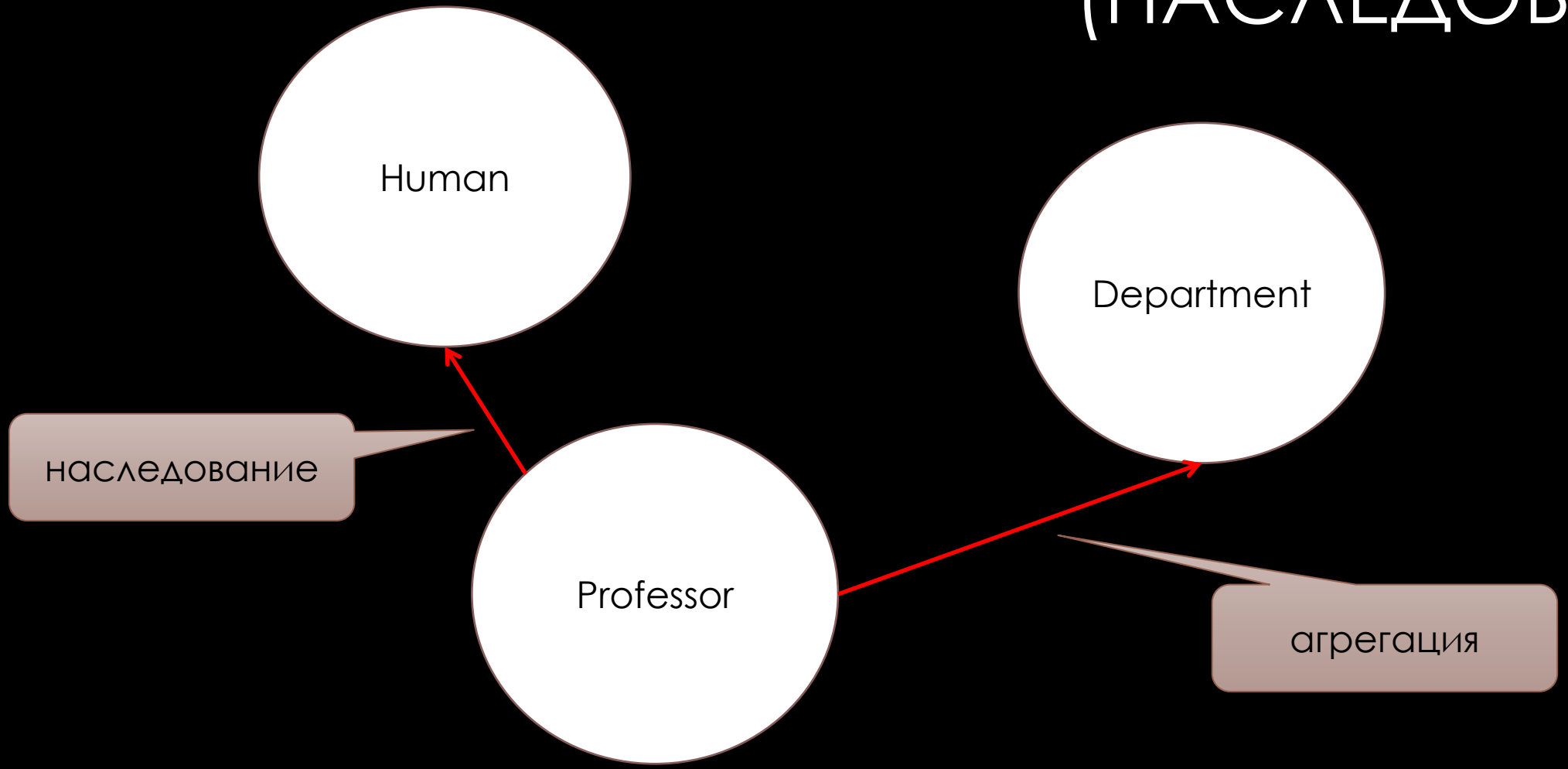
```
Circle cir = new Circle(1, 1, 1);
BinaryFormatter binformatter = new BinaryFormatter();
using (FileStream fs = new FileStream("Circle.bin", FileMode.Create))
{   // Выполнение сериализации:
    binformatter.Serialize(fs, cir);
}
```

```
Circle inCir = null;
using (FileStream fs1 = new FileStream("Circle.bin", FileMode.Open))
{
    inCir = (Circle)binformatter.Deserialize(fs1);
    Console.WriteLine("Restored object::");
    Console.WriteLine(inCir);
}
```

Вывод:

```
Restored object::
xc=1    yc=1,   Rad=1
```

ГРАФ ОБЪЕКТОВ (НАСЛЕДОВАНИЕ)



ДВОИЧНАЯ СЕРИАЛИЗАЦИЯ УНАСЛЕДОВАННЫХ ОБЪЕКТОВ

```
[Serializable]
public class Human {
    public string Name { get; set; }
    public int Age { get; set; }
    public Human() { }
    public Human(string name, int age)     {
        Name = name;
        Age = age;
    }
    public override string ToString() => $"Name={Name},Age={Age}";
}
```

```
[Serializable]
public struct Department {
    public string Name { get; set; }
    public override string ToString() => $"Department={Name}";
}
```

```
[Serializable]
public class Professor : Human  {
    public Department department { get; set; }
    public Professor() { }
    public Professor(string name, int age, Department department) : base(name,age)  {
        this.department = department;
    }
    public override string ToString() => base.ToString() + "," + department;
}
```

ДВОИЧНАЯ СЕРИАЛИЗАЦИЯ УНАСЛЕДОВАННЫХ ОБЪЕКТОВ

```
Department dep = new Department();
dep.Name = "Software Engineering";
Professor prof = new Professor("Black", 46, dep);
BinaryFormatter binformatter = new BinaryFormatter();
using (FileStream fs = new FileStream("Deps.bin", FileMode.Create))
{   // Выполнение сериализации:
    binformatter.Serialize(fs, prof);
}

Professor inProf = null;
using (FileStream fs1 = new FileStream("Deps.bin", FileMode.Open))
{
    inProf = (Professor)binformatter.Deserialize(fs1);
    Console.WriteLine("Restored object::");
    Console.WriteLine(inProf);
}
```

Вывод:

```
Restored object::
Name=Black,Age=46,Department=Software Engineering
```

АТРИБУТЫ ДЛЯ ДВОИЧНОЙ СЕРИАЛИЗАЦИИ

Двоичная сериализация:

- Атрибуты
- Реализация интерфейса `ISerializable`
- `[Serializable]` – в объявлении типа
- `[NonSerialized]` – в объявлении игнорируемых полей

Атрибуты для методов:

- `[OnSerializing]` – перед сериализацией
- `[OnSerialized]` – после сериализации

- `[OnDeserializing]` – перед десериализацией
- `[OnDeserialized]` – после десериализации

АТРИБУТЫ ДВОИЧНОЙ СЕРИАЛИЗАЦИИ. ПРИМЕР

```
[Serializable]
class Person    {  // Версия 1
        public string name;
}

[Serializable]
class Person    {  //  Версия 2
       public string name;
       [OptionalField (VersionAdded = 2)]
       public DateTime DateOfBirth;
}
```

Только для полей (см. название)

КОНФИГУРИРОВАНИЕ КЛАССОВ

- Атрибут `Serializable` **не наследуется**
- Атрибут `NonSerialized` **наследуется**

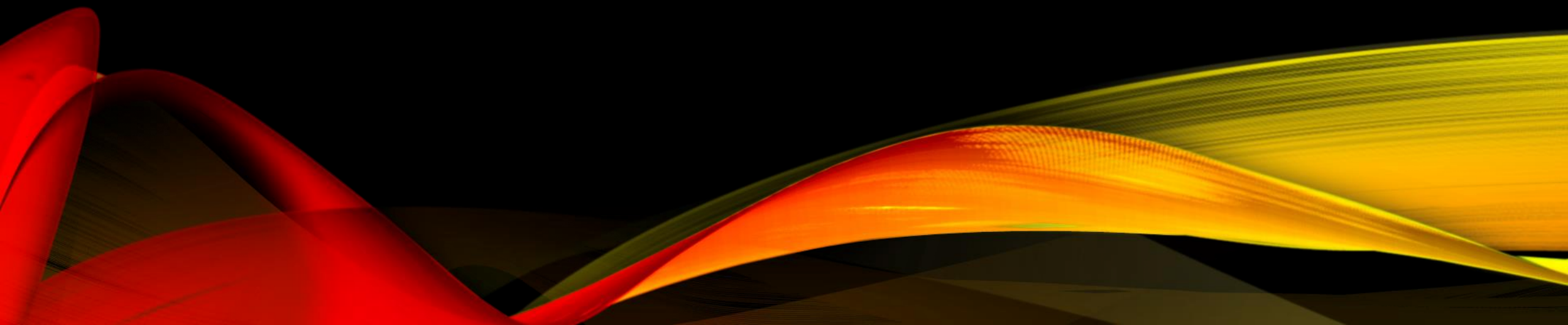
[Serializable]

```
[Serializable]
class Student
```

[NonSerialized]

```
[NonSerialized]
private string Surname;
```

JSON-СЕРИАЛИЗАЦИЯ



JSON. ПРИМЕР ОБЪЕКТА С МАССИВОМ

```
{
  "employeeId": 1234567,
  "name": "Ivanov Ivan",
  "hireDate": "2022-01-10",
  "location": "Moscow, RU",
  "hasDrivingLicence": false,
  "childrenAges": [ 3, 9 ]
}
```

JSON. КОГДА ИСПОЛЬЗОВАТЬ?

- Передача данных на сервер (от сервера), особенно в web-приложениях
- Выполнение асинхронных AJAX-вызовов без перезагрузки страниц в web-приложениях
- Работа с базами данных (особенно документно-ориентированными)
- Сохранение данных в локальном хранилище (сериализация)

АТРИБУТЫ КОНТРАКТОВ ДАННЫХ

- `[DataContract]` – для сериализуемых типов;
- `[DataMember]` – для членов сериализуемых типов;
- `[EnumMember]` – для членов перечислений;

JSON-СЕРИАЛИЗАЦИЯ ЧЕРЕЗ КОНТРАКТЫ ДАННЫХ

Сборка:
- `System.Runtime.Serialization.dll`

Пространство имен:
- `using System.Runtime.Serialization.Json;`

Класс определяет объект-форматер:
- `DataContractJsonSerializer`

Методы форматера:
- `WriteObject()` - сериализация
- `ReadObject()` - десериализация

JSON-СЕРИАЛИЗАЦИЯ И НАСЛЕДОВАНИЕ

```
[DataContract, KnownTypeAttribute(typeof(Professor))]
public class Human
{
    [DataMember]
    public string Name { get; set; }
    [DataMember]
    public int Age { get; set; }
    public Human() { }
    public Human(string name, int age) => (Name, Age) = (name, age);
    public override string ToString() => $"Name={Name},Age={Age}";
}
```

Устанавливаем тип, который должен распознаваться сериализатором

```
[DataContract]
public class Professor : Human
{
    public Professor() { }
    public Professor(string name, int age) : base(name, age) { }

    public override string ToString() => base.ToString();
}
```

JSON-СЕРИАЛИЗАЦИЯ И НАСЛЕДОВАНИЕ

```
DataContractJsonSerializer serializer =
    new DataContractJsonSerializer(typeof(Professor));
// Сериализация в файл.
using (FileStream fs = new FileStream("doc.json", FileMode.Create))
    serializer.WriteObject(fs, new Professor("Ivanov", 68));

// Десериализация.
Professor prof;
DataContractJsonSerializer deser = new
DataContractJsonSerializer(typeof(Professor));
using (FileStream fs = new FileStream("doc.json ", FileMode.Open))
    prof = deser.ReadObject(fs) as Professor;
Console.WriteLine(prof);
```

Вывод:
Name=Ivanov,Age=68

Содержимое doc.json, прекрасно просматривается из IDE Visual Studio

```
{"Age":68,"Name":"Ivanov"}
```

ССЫЛКИ

- https://docs.microsoft.com/ru-ru/dotnet/framework/wcf/feature-details/types-supported-by-the-data-contract-serializer
- Editing JSON with Visual Studio Code (https://code.visualstudio.com/docs/languages/json)
- Введение в NuGet (https://learn.microsoft.com/ru-ru/nuget/what-is-nuget)
- Сериализация в .NET (https://learn.microsoft.com/ru-ru/dotnet/standard/serialization/)
- System.Text.Json Пространство имен (https://learn.microsoft.com/ru-ru/dotnet/api/system.text.json?view=net-8.0)
- Обобщённое копирование связных графов объектов в C# и нюансы их реализации (https://habr.com/ru/articles/332516/)