

# **Сериализация времени выполнения Атрибуты**

- ▶ **Сериализация** - процесс преобразования объектов или связанных объектов в поток байт (диск, память, сеть)
- ▶ **Десериализация** - получение из потока байт сохраненного объекта

Сериализуемый тип – это тип, помеченный атрибутом [Serializable], у которого все поля имеют сериализуемый тип.  
Базовые типы платформы .NET являются сериализуемыми.

```
[Serializable]
```

```
public class Point
{
    public int x = 10;
    public int y = 20;
    public virtual int Sum()
        { return x + y; }
}
```

Свойства `Serializable` :  
не может наследоваться

```
[Serializable]
public class Point
{
    [NonSerialized]
    private int x = 10;
    private int y = 20;

    //...

    public virtual int Sum() { return x + y; }
}
```

← объект доступен для служб  
сериализации .NET,

← не будут участвовать в схеме  
сериализации (сокращает размер  
хранимых данных – например:  
фиксированные значения, случайные  
или вычисляемые значения)

► Объекты сериализуемых типов можно сохранить в  
поток в различных форматах

## ► Форматы

- бинарный
- SOAP
- xml
- JSON

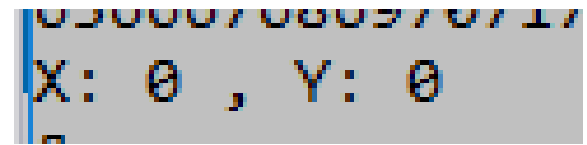
# класс **BinaryFormatter**

- ▶ функционал сериализации определен в интерфейсе **IFormatter**

```
public interface IFormatter
{
    SerializationBinder Binder { get; set; }
    StreamingContext Context { get; set; }
    ISurrogateSelector SurrogateSelector { get; set; }
    object Deserialize(Stream serializationStream);
    void Serialize(Stream serializationStream, object graph);
}
```

поток сериализации и сериализуемый объект

```
// объект для сериализации
Point center = new Point(23, 0);
// создаем объект BinaryFormatter
BinaryFormatter formatter = new BinaryFormatter();
// получаем поток, куда будем записывать сериализованный объект
using (FileStream fs = new FileStream("points.dat",
                                     FileMode.OpenOrCreate))
{
    formatter.Serialize(fs, center);
}
```



```
// десериализация
using (FileStream fs = new FileStream("points.dat",
                                     FileMode.OpenOrCreate))
{
    Point newPoint = (Point)formatter.Deserialize(fs);

    Console.WriteLine($"X: {newPoint.x} , Y: {newPoint.y}");
}
```

размещает объект в памяти и  
возвращает ссылку на него  
При этом конструктор не вызывается

# Метод Deserialize()

- размещает объект в памяти и возвращает ссылку на него
- конструкторы не вызываются

Если нужна особая инициализация объекта и восстановление несохраненных полей

[OnSerializing], [OnSerialized], [OnDeserializing], [OnDeserialized],

вызываются CLR автоматически до и после сериализации или десериализации

Метод, который обозначен атрибутом, должен принимать объект класса StreamingContext и не возвращать значений

# пример

```
[Serializable]
public class ParkIT
{
    [OnSerializing]
    private void BeforeSerialization(StreamingContext context) {
        CalculateSmt(); }

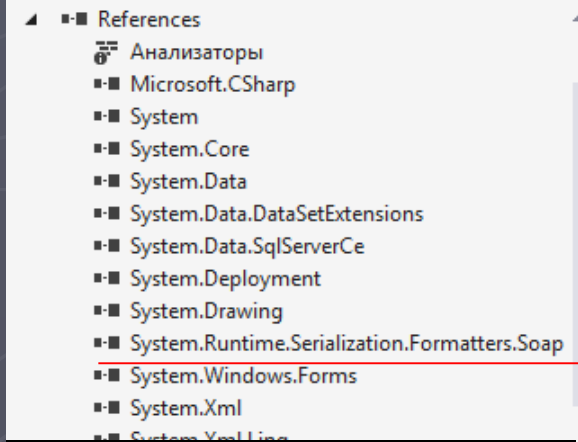
    [OnDeserialized]
    private void AfterDeserialization(StreamingContext context) {
        FindSmt();
    }
}
```

```
[Serializable]
class Student
{
    private String name;
    private int age;
    [NonSerialized]
    private String adress;
}
```

## Использовать BinaryFormatter или SoapFormatter

сериализует  
состояние объекта в  
поток, используя  
двоичный формат.

сохраняет состояние объекта в виде  
сообщения SOAP (стандартный  
XML-формат для передачи и приема  
сообщений от веб-служб).



Требует подключения



```
static class SerialBinFormatter
```

```
{
```

```
    public static void SaveBinaryFormat(object obj, string fileName)
```

```
    {
```

```
        BinaryFormatter binFormat = new BinaryFormatter();
```

```
        using (Stream fStream = new FileStream(fileName, FileMode.Create,  
                                                FileAccess.Write, FileShare.None))
```

```
        {
```

```
            binFormat.Serialize(fStream, obj);
```

```
        }
```

```
    }
```

сохраняет данные полей объектов + полное квалифицированное имя каждого типа + полное имя определяющей его сборки (имя, версия, маркер общедоступного ключа и культура).

```
Student Leha = new Student("Leha", 19, "Minsk, Sverdloava 13");
```

```
// Сохранить объект в указанном файле в двоичном формате  
SerialBinFormatter.SaveBinaryFormat(Leha, "BinData.dat");
```

```
00000000 00 01 00 00 00 FF FF FF FF 01 00 00 00 00 00 .....  
00000010 00 0C 02 00 00 00 4A D0 A0 D0 B0 D0 B1 D0 BE D1 .....J.....  
00000020 82 D0 B0 20 D1 81 20 58 4D 4C 2C 20 56 65 72 73 ... .. XML, Vers  
00000030 69 6F 6E 3D 31 2E 30 2E 30 2E 30 2C 20 43 75 6C ion=1.0.0.0, Cul  
00000040 74 75 72 65 3D 6E 65 75 74 72 61 6C 2C 20 50 75 ture=neutral, Pu  
00000050 62 6C 69 63 4B 65 79 54 6F 6B 65 6E 3D 6E 75 6C blicKeyToken=nul  
00000060 6C 05 01 00 00 00 1B D0 A0 D0 B0 D0 B1 D0 BE D1 l.....  
00000070 82 D0 B0 5F D1 81 5F 58 4D 4C 2E 53 74 75 64 65 ... .._XML.Stude  
00000080 6E 74 02 00 00 00 04 6E 61 6D 65 03 61 67 65 01 nt.....name.age.  
00000090 00 08 02 00 00 00 06 03 00 00 00 04 4C 65 68 61 .....Leha  
000000a0 13 00 00 00 0B .....
```

```
static void LoadFromBinaryFile(string fileName)
{
    BinaryFormatter binFormat = new BinaryFormatter();

    using (Stream fStream = File.OpenRead(fileName))
    {
        Student Olga =
            (Student)binFormat.Deserialize(fStream);
    }
}
```

Deserialize() - преобразует сохраненную последовательность байт в граф объектов

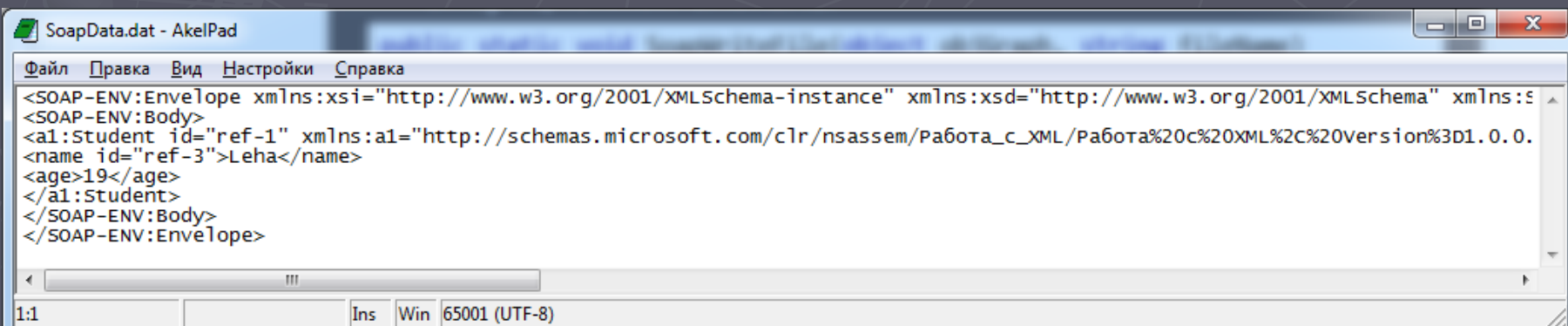
# Класс SoapFormatter

- *SOAP (Simple Object Access Protocol — простой протокол доступа к объектам)* - стандартный процесс вызова методов независимо от платформы и операционной системы

```
public static void SoapWriteFile(object objGraph, string fileName)
{
    SoapFormatter soapFormatter = new SoapFormatter();
    using (Stream fStream = new FileStream(fileName,
        FileMode.Create, FileAccess.Write, FileShare.None))
    {
        soapFormatter.Serialize(fStream, objGraph);
    }
}
```

сохраняет трассировки сборок-источников

```
SerialBinFormatter.S SoapWriteFile(Leha, "SoapData.dat");
```



# Сериализация в XML.

## XmlSerializer

### ► Ограничения

- Класс должен иметь конструктор без параметров
- Сериализации подлежат только открытые члены
- Приватные данные, не представленные свойствами, игнорируются.
- Требуется указание типа

```
[Serializable]
public class Point
{

    public int x = 10;
    public int y = 20;

    public Point()
    {
    }

    public Point(int v1, int v2)
    {
        this.x = v1;
        this.y = v2;
    }

    //...
}
```

```
// объект для сериализации
Point dot = new Point(10, 100);
// передаем в конструктор тип класса
XmlSerializer xSer = new XmlSerializer(typeof(Point));

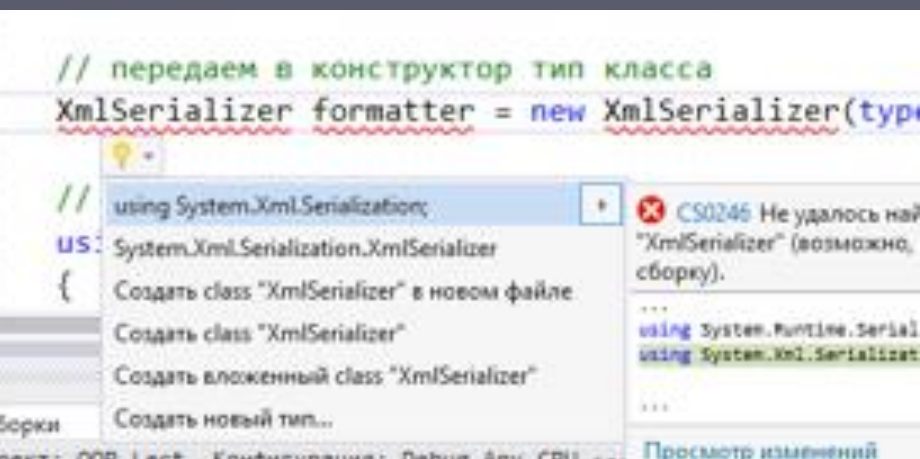
// получаем поток, куда будем записывать сериализованный объект
using (FileStream fs = new FileStream("points.xml", FileMode.OpenOrCreate)
{
    xSer.Serialize(fs, dot);
}

// десериализация
using (FileStream fs = new FileStream("points.xml", FileMode.OpenOrCreate)
{
    Point newP = xSer.Deserialize(fs) as Point;
    Console.WriteLine(newP.ToString());
}
```

X: 23 , Y: 0

X: 10 Y: 100

Для продолжения нажмите любую клавишу



Не записывает полного  
квалифицированного имени или сборки,  
в которой он определен

`<?xml version="1.0"?>`  
`<Point xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`  
`xmlns:xsd="http://www.w3.org/2001/XMLSchema">`  
 `<x>10</x>`  
 `<y>100</y>`  
`</Point>`

# ► Декорирование типов - настройка XML сериализации

управление генерацией  
резльтирующего документа XML

```
[XmlRoot]
// корневой элемент
class Book
{
    [XmlAttribute]
    // как атрибут XML
    public string id;
    [XmlElement]
    // как элемент XML с указанным именем
    public string title;
    // [XmlType]
    // атрибут предоставляет имя и пространство имен типа XML
    [XmlText]
    // стерилизовано как текст XML
    public string author;
}
```

применяется к типу и задаёт корневой элемент в XML-файле

настраивается имя и пространство имён XML-элемента



```
public class StudentBSTU
```

```
{
```

```
    [XmlAttribute("fullname")]
```

```
    public string Name { get; set; }
```

```
    [XmlIgnore]
```

```
    public double Mark { get; set; }
```

```
}
```

```
    [XmlRoot("BSTU")]
```

```
public class Group
```

```
{ [XmlArray("Isit")]
```

```
    [XmlArrayItem("studentBSTU")]
```

```
    public List<StudentBSTU> List { get; set; }
```

```
}
```

Поля и свойства, которые не должны сохраняться

настройка имени коллекции и  
имени элемента

# Сериализация контрактов данных

- ▶ Контракт данных – это тип (класс или структура), объект которого описывает информационный фрагмент (открытые поля и свойства) - один из механизмов сериализации

Основным форматом хранения контрактов данных является XML

<input checked="" type="checkbox"/>	System.Runtime.Serialization	4.0.0.0
<input checked="" type="checkbox"/>	System.Runtime.Serialization.Formatters.Soap	4.0.0.0

```
[DataContract]
public class StudentBSTU
{
    [DataMember]
    public string Name { get; set; }

    [DataMember]
    public double Mark { get; set; }
}
```

в качестве контракта данных используется обычный класс, информационный фрагмент образуют открытые поля и свойства класса  
Видимость не имеет значения

# Атрибуты контрактов

Основным форматом хранения контрактов данных является XML.

```
[DataContract(Name = "student", Namespace = "BSTU")]
public class StudentBSTU
{
    [DataMember(Name = "name", Order = 1, IsRequired = true )]
    public string Name { get; set; }

    [DataMember(Name = "value", Order = 0,
                EmitDefaultValue = false)]
    public double Mark { get; set; }
}
```

## [CollectionDataContract] - Для коллекций

[DataContract]

свойства Name и Namespace

[DataMember]

свойство Name, Order (порядок записи элементов контракта),  
IsRequired (обязательный элемент для записи),

EmitDefaultValue (нужна ли запись значения по умолчанию для  
элемента)

```
[DataContract]
```

```
[KnownType(typeof(StudentFit))]
```

```
public class Student { }
```

```
public class StudentFit : Student { }
```

планируется десериализовать в объекты  
ПОТОМКОВ СВОЕГО ТИПА

- Для выполнения сериализации контракта данных используются классы:

`DataContractSerializer` – сериализует контракт в формате XML;

`NetDataContractSerializer` – сериализует данные и тип контракта;

`DataContractJsonSerializer` – сериализует контракт в формате JSON.

```

entBSTU studentAnna = new StudentBSTU { Name = "Anna",
                                          Mark = 9.1 };

var ds = new DataContractSerializer
           (typeof (StudentBSTU));
using (Stream s = File.Create("stud.xml"))
{
    ds.WriteObject(s, studentAnna);
}
using (Stream s = File.Open("stud.xml",
                           FileMode.Open))
{
    StudentBSTU newStud =
        (StudentBSTU)ds.ReadObject(s);
}

```

```

<student xmlns="BSTU" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
<value>9.1</value><name>Anna</name></student>

```

# ► Json сериализация сложных объектов

```
[DataContract]
public class Programmer
{
    [DataMember]
    public string Name { get; set; }
    [DataMember]
    public int Age { get; set; }
    [DataMember]
    public Company Company { get; set; }
    public Programmer()
    { }
    public Programmer(string name, int age, Company comp)
    {
        Name = name;
        Age = age;
        Company = comp;
    }
}

[DataContractAttribute]
public class Company
{
    [DataMember]
    public string Name { get; set; }
    public Company() { }
    public Company(string name)
    { Name = name; }
}
```

```
using System.Runtime.Serialization.Json;
```

```
Programmer person1 = new Programmer("Anna", 21, new Company("000"));  
Programmer person2 = new Programmer("Nikita", 45, new Company("0A0"));  
Programmer[] people = new Programmer[] { person1, person2 };
```

```
DataContractJsonSerializer jsonFormatter =  
    new DataContractJsonSerializer(typeof(Programmer[]));
```

```
using (FileStream fs = new FileStream("programmers.json",  
                                     FileMode.OpenOrCreate))  
{  
    jsonFormatter.WriteObject(fs, people);  
}
```

```
using (FileStream fs = new FileStream("programmers.json",  
                                     FileMode.OpenOrCreate))  
{  
    Programmer[] newpeople =  
        (Programmer[])jsonFormatter.ReadObject(fs);  
}
```

programmers.json x

```
1 [{"Age":21,"Company":{"Name":"000"},"Name":"Anna"}, {"Age":45,"Company":{"Name":"0A0"},"Name":"Nikita"}]
```



# интерфейс ISerializable

- ▶ позволяет выполнить любые действия, связанные с формированием данных для сохранения (свой сериализатор)

```
public interface ISerializable {  
    void GetObjectData(SerializationInfo info,  
                       StreamingContext context);  
}
```

вызывается CLR автоматически при выполнении сериализации

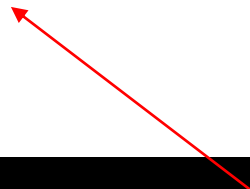
заполнение объекта `SerializationInfo` набором данных вида «ключ-значение», которые (обычно) соответствуют полям сохраняемого объекта

```
[Serializable]
public class Student : ISerializable
{
    void ISerializable.GetObjectData(SerializationInfo info,
        StreamingContext ctx) {
        info.SetType(typeof(Student));
        info.AddValue("Name", _name);
        info.AddValue("Rate", (int)(_rate * 100)); }

    private Student(SerializationInfo info, StreamingContext ctx)
    {
        _name = info.GetString("Name");
        _rate = info.GetInt32("Rate") / 100; }

    public Student() { }

    //...
}
```



должен содержать специальный private-конструктор, который будет вызывать CLR после выполнения десериализации. Конструктор должен иметь параметры типа `SerializationInfo` и `StreamingContext`.

# АТТРИБУТЫ

- ▶ атрибут (attribute) - дополнительная информация, сохраняемая в метаданных о сборке, модуле, типе, элементах типа, параметрах метода
- ▶ Наследуются от `System.Attribute`

# Создание собственного атрибута

► На основе класса

► Требования к классу:

- Класс должен прямо или косвенно наследоваться от класса `Attribute`
- Тип открытых полей, свойств и параметров конструктора класса: числовые типы (кроме `decimal`), `bool`, `char`, `string`, `object`, `System.Type`, перечисления; одномерные массивы
- Имя класса должно заканчиваться суффиксом `Attribute` (необязательно).

## ► пример

```
public class BSTUAttribute : System.Attribute
{
    public string Name { get; private set; }
    public int Version { get; set; }

    public BSTUAttribute(string name) {
        Name = name;
    }
}
```

## ► ИСПОЛЬЗОВАНИЕ

имя атрибута  
указывается без  
суффикса Attribute

именованные параметры,  
предназначенны для задания  
значения открытого поля или  
свойства

```
[BSTU ("ООР C++/C#", Version =3)]
```

```
public class Book
```

```
{
```

аргументы конструктора атрибута


```
    [BSTU("Draft")]
```

```
    public void Print() { }
```

```
}
```

## ► Настройка атрибута

Определяет цель атрибута - при возникновении неоднозначности, нужно указать перед именем атрибута префикс – assembly, module, field, event, method, param, property, return, type



```
[System.AttributeUsage(AttributeTargets.Class |  
                        AttributeTargets.Method,  
                        AllowMultiple = true)]
```

```
public class BSTUAttribute : Attribute  
{
```

## ► Настройка атрибута

определяет, может ли атрибут быть применён к программному элементу более одного раза

```
[AttributeUsage(AttributeTargets.Class |  
                AttributeTargets.Method,  
                AllowMultiple = true)]
```

```
public class BSTUAttribute : Attribute  
{
```

```
[assembly: AssemblyKeyFile("Lect.exe")]
```

```
// многократное применение атрибута
```

```
[BSTU("Serialization"), BSTU("Lecture 12")]
```

```
public class Presentation { }
```



- ▶ получение информации о применённых атрибутах
- ▶ `Attribute.GetCustomAttributes()`  
возвращает все атрибуты некоторого элемента в виде массива

```
Attribute GetCustomAttribute  
(MemberInfo element, Type attributeType)
```

элемент, у которого надо получить атрибу

тип получаемого атрибута

```
var bstuinfo = Attribute.GetCustomAttribute(typeof(Book),  
                                             typeof(BSTUAttribute));  
  
if (bstuinfo != null)  
{  
    Console.WriteLine(((BSTUAttribute)bstuinfo).Name);  
}
```

OOP C++/C#

Для пополнения

## ► Атрибуты платформы .NET

Атрибут	Цель применения	Описание
[Conditional]	Метод	Компилятор может игнорировать вызовы помеченного метода при заданном условии
[DllImport]	Метод	Импорт функций из DLL
[MTAThread]	Метод Main()	Для приложения используется модель COM Multithreaded apartment
[NonSerialized] [Serializable]		
[Obsolete]	Кроме param, assembly, module, return	Информирует, что в будущих реализациях данный элемент может отсутствовать

Атрибут	Цель применения	Описание
[STAThread]	Метод Main()	Для приложения используется модель COM Single-threaded apartment
[ThreadStatic]	Статическое поле	В каждом потоке будет использоваться собственная копия данного статического поля

```
using System.Runtime.InteropServices;
```

```
BSTU ("OOP C++/C#", Version = 3)]  
    public class Book  
    {  
        [DllImport("kernel32.dll")]  
        public static extern void GetLocalTime(SystemTime st);  
    ..  
    }
```