Стриймове, файлове и директории

Видове файлове, използване на стриймове и манипулиране на файлове

SoftUni Foundation

Проект "Отворено учебно съдържание по програмиране и ИТ", СофтУни Фондация

https://github.com/BG-IT-Edu

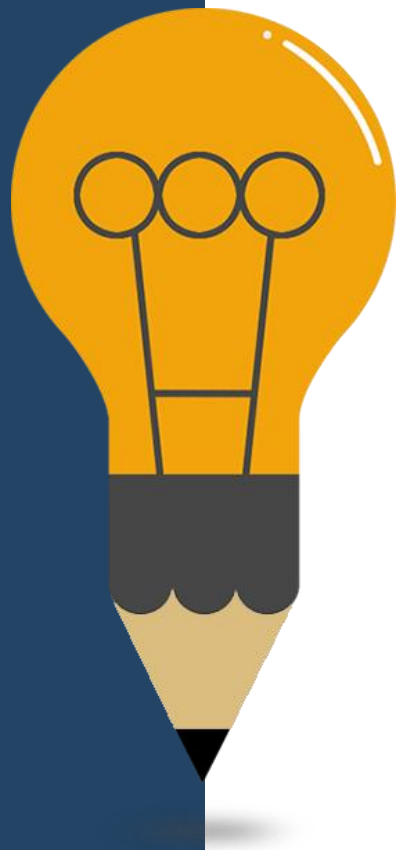
Курс "Структури от данни и алгоритми"

Софтуерни и хардуерни науки

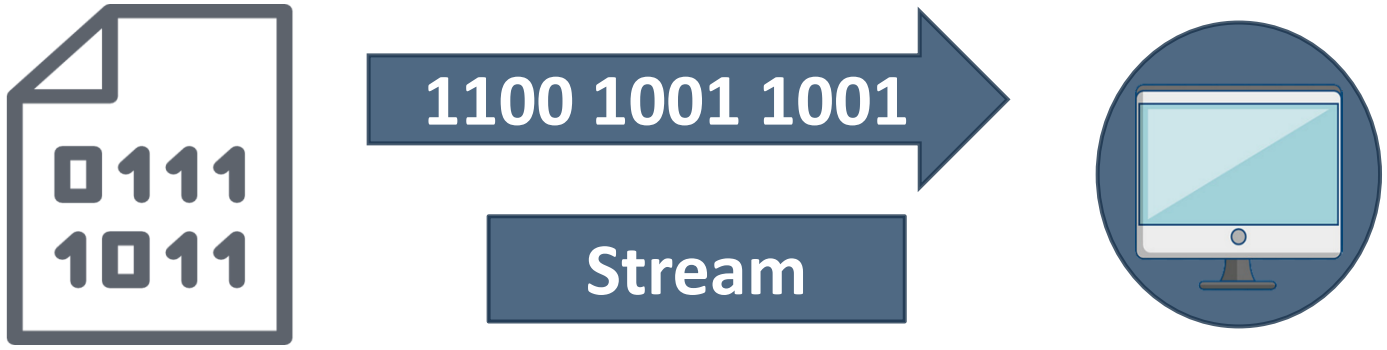
Съдържание

Какво е стриймване?

Какво е стриймване?



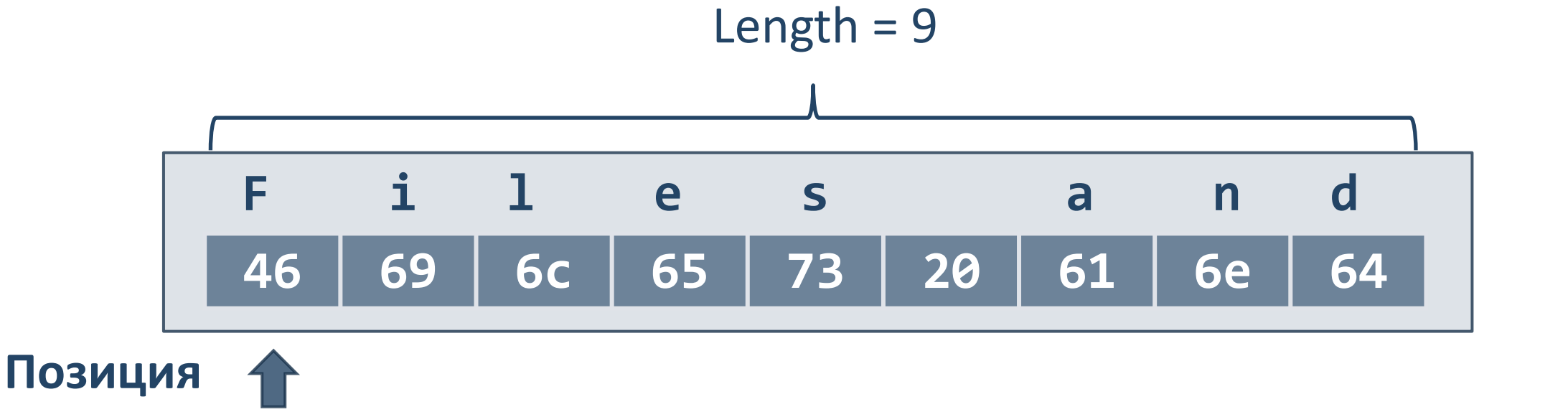
- Стриймването се използва за **пренасяне на информация**
- Създаваме стрийм, за да:
  - **Четем** информация
  - **Пишем** информация



Основи на стриймването

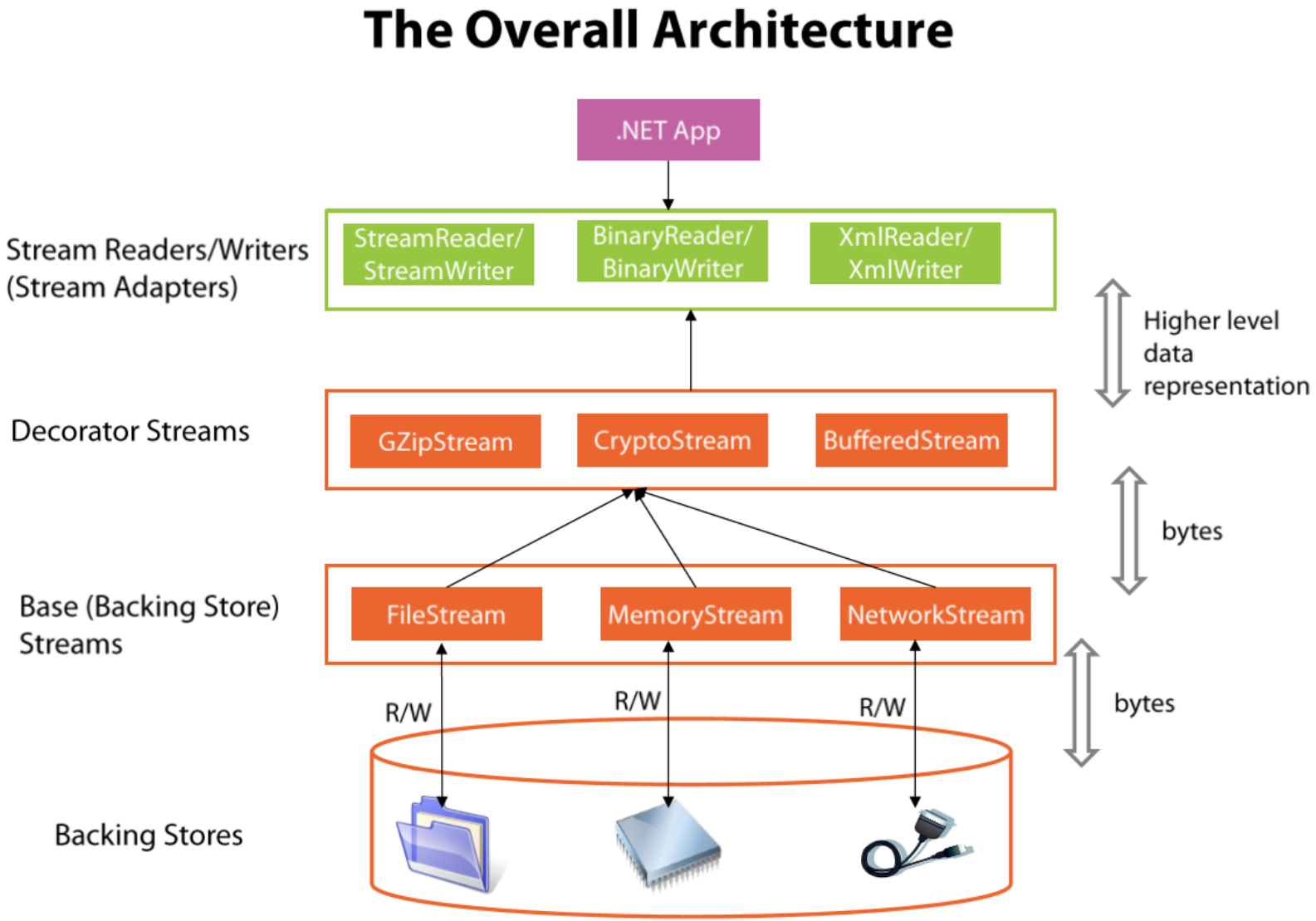
- Стриймване означава **пренасяне** (четене и писане) **на информация**
- Стриймването е **редица от битове**
- Има различни видове стриймове за различни типове информация:
  - достъп до **файл**, достъп до **мрежа** и други
- Стриймовете се **отварят преди** да се използват и се **затварят след** тяхната работа

Стриймове и буфери – Примери



- **Позицията** е текущия индекс от стрийма
- **Буферът** съдържа **n** бита на стрийм от текущия индекс

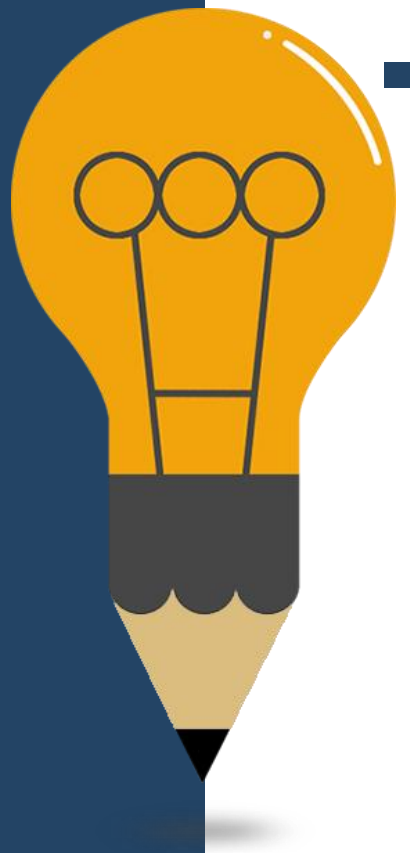
Типове стрийм в .NET



Работа със стриймове

Четене и писане в C#

Използване на StreamReader



- **`StreamReader`** в C# чете **`текст`** от а файл / стрийм
- Използването на **`using(…)`** затваря правилно стрийма накрая

```
var reader = new StreamReader(fileName);
using (reader)
{
  // Използвайте четенето тук, примерно:
  // string line = reader.ReadLine();
}
```

Задачи: Четни редове (1)

- В ресурсите е предоставен файл **`text.txt`**
- Създайте **VS конзолен проект** и **копирайте** текстовия файл там
- **Прочетете съдържанието** на файла
- Отпечатайте **четните редове** на конзолата
- Броенето на редовете се извършва по индекс (започваме от **0**)

Задачи: Четни редове (2)

```
-I was quick to judge him, but it wasn't his fault.
-Is this some kind of joke?! Is it?
-Quick, hide here. It is safer.
```

```
-I was quick to judge him, but it wasn't his fault.
-Quick, hide here. It is safer.
```

Задачи: Четни редове (3)

- **Важно**: Използвайте следната структура за решението

```
namespace EvenLines
{
    public class EvenLines
    {
        static void Main()
        {
            string inputFilePath = @"..\..\..\text.txt";
            Console.WriteLine(ProcessLines(inputFilePath));
        }
        public static string ProcessLines(string inputFilePath)
        { }
    }
}
```

TODO: РЕШЕНИЕ НА ЗАДАЧАТА

SoftUni Foundation

Използване на StreamWriter

- **StreamWriter** в C# ни позволява да **записваме текст** във файл

```
StreamWriter writer = new StreamWriter(filePath);
using (writer)
{
  // Пишем съдържание тук, примерно:
  writer.WriteLine("Hello");
}
```

- Ако указаният файл все още **не съществува**, той ще бъде **създаден**, след което ще се запише в него **съдържанието**

Задача: Нечетни редове (1)

- Прочетете съдържанието от файла **`input.txt`** (предоставен е в ресурсите)
- Запишете съдържанието на **нечетните редове** във файла **`output.txt`**
- Броенето на редовете **започва от 0**

Задача: Нечетни редове (2)

- Пример:

```
Two households, both alike in dignity,
In fair Verona, where we lay our scene,
From ancient grudge break to new mutiny,
Where civil blood makes civil hands unclean.
```

```
In fair Verona, where we lay our scene,
Where civil blood makes civil hands unclean.
```

Задача: Нечетни редове (3)

- **Важно**: Използвайте следната структура за решението

```
namespace OddLines
{
    public class OddLines
    {
        static void Main()
        {
                Console.WriteLine(ExtractOddLines(@"..\..\..\input.txt",
                                                  @"..\..\..\output.txt"));
        }
        public static void ExtractOddLines(string inputFilePath,
                                           string outputFilePath) { }
    }
}
```

Решение: Нечетни редове

```
StreamReader reader = new StreamReader("input.txt");
using (reader) {
  int counter = 0;
  string line = reader.ReadLine();
  StreamWriter writer = new StreamWriter("output.txt");
  using (writer) {
    while (line != null)
      if (counter % 2 == 1)
        writer.WriteLine(line);
    counter++;
    line = reader.ReadLine();
  }
}
```

Задача: Номерирани редове

- Прочетете файла **input.txt**
- Добавете **номер на реда** за всеки ред на файла
- Запишете го в **output.txt**

```
Two households, both alike in dignity,
In fair Verona, where we lay our scene,
From ancient grudge break to new mutiny,
Where civil blood makes civil hands unclean.
```

```
1. Two households, both alike in dignity,
2. In fair Verona, where we lay our scene,
3. From ancient grudge break to new mutiny,
4. Where civil blood makes civil hands unclean.
```

Решение: Номерирани редове

```
using (var reader = new StreamReader("input.txt"))
{
  string line = reader.ReadLine();
  int counter = 1;
  using (var writer = new StreamWriter("output.txt"))
    while (line != null)
    {
      writer.WriteLine($"{counter}. {line}");
      line = reader.ReadLine();
      counter++;
    }
}
```

Класът File в .NET

Четене на текстов файл

- `File.ReadAllText()` → `низ` - чете текст наведнъж

```
using System.IO;
…
string text = File.ReadAllText("file.txt");
```

- `File.ReadAllLines()` → `низ[]` - чете текста по редове

```
using System.IO;
…
string[] lines = File.ReadAllLines("file.txt");
```

Писане на текстов файл

- Пише **низ** към текстов файл:

```
File.WriteAllText("output.txt", "Files are fun :)");
```

- Пише **редица** от низове в текстов файл, разделени с ред:

```
string[] names = { "peter", "irina", "george", "maria" };
File.WriteAllLines("output.txt", names);
```

- **Добавя** допълнителен текст към съществуващ файл:

```
File.AppendAllText("output.txt", "\nMore text\n");
```

Четене/Писане на двоичен файл

- Писане на **byte[]** в текстов файл:

```
using System.IO;
…
byte[] bytesToWrite = { 0, 183, 255 };
File.WriteAllBytes("output.txt", bytesToWrite);
```

- Четене на двоичен файл с **byte[]**:

```
using System.IO;
…
byte[] bytesRead = File.ReadAllBytes("binaryFile.txt");
```

Класът Directory в .NET

Основни операции на директории

- **Създаване** на директория (с всичките ѝ поддиректории по посочения път), освен ако вече **не съществуват**:

```
Directory.CreateDirectory("TestFolder");
```

- **Изтриване** на директория (със съдържание):

```
Directory.Delete("TestFolder", true);
```

- **Преместване** на файл от една директория в друга:

```
Directory.Move("Test", "New Folder");
```

Списък на съдържанието на директорията

- **`GetFiles()`** – Връща имената на файловете (включително техния път) в определена директория

```
string[] filesInDir =
  Directory.GetFiles("TestFolder");
```

- **`GetDirectories()`** – връща имената на подпапките (включително техните пътища) в определена директория

```
string[] subDirs =
  Directory.GetDirectories("TestFolder");
```

Задача: Изчисление на размера на папка

- Дадена ви е папка с името **TestFolder**
- Изчислете **размера на всички файлове в нея** (включително и подпапките)
- Отпечатайте резултата във файла "**output.txt**" в мегабайти

output.txt
5.16173839569092

Решение: Изчисление на размера на папка

```
double sum = 0;

DirectoryInfo dir = new DirectoryInfo("TestFolder");
FileInfo[] infos = dir.GetFiles("*", SearchOption.AllDirectories);

foreach (FileInfo fileInfo in infos)
{
    sum += fileInfo.Length;
}

sum = sum / 1024;

File.WriteAllText("output.txt", sum.ToString());
```

Получаваме **всички файлове** от дадената папка и нейните **подпапки**

Бинарна сериализация

Бинарна сериализация

- **Бинарната сериализация** е процес на преобразуване на обекти в поток от байтове, който може бъде:
  - Съхраняван
  - Предаван по мрежа
- **Запазва** данните в компактен и ефективен формат
  - Позволява **трансфер** и **съхранение** на данни между **различни системи**

Предимства и недостатъци на бинарната сериализация

- Предимства
  - **По-бърза** от текстовите формати
    - Няма нужда от **преобразуване на данните** в символни низове и обратно

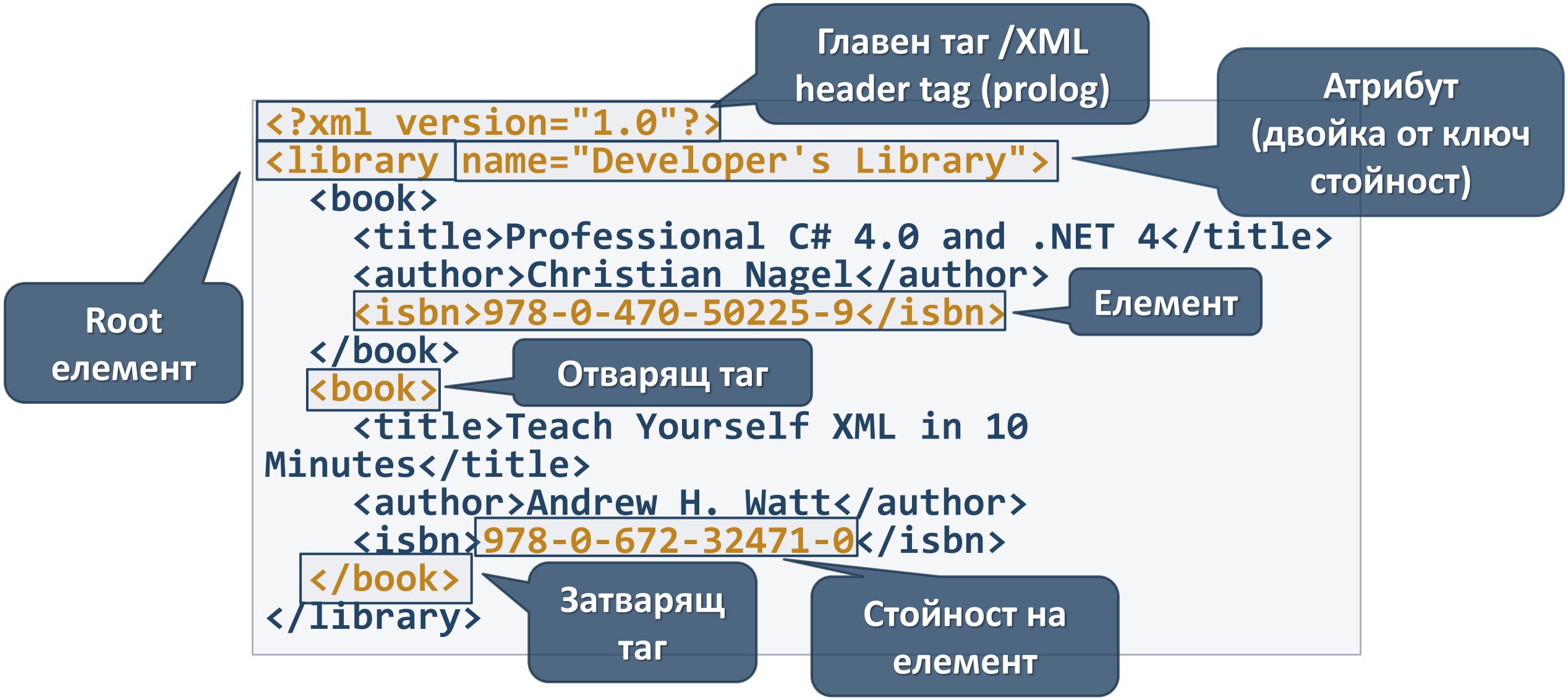
- Недостатъци
  - При използване на **бинарна сериализация** има вероятност от **злоупотреби**
  - За повече информация: https://aka.ms/binaryformatter

Какво е XML?

Какво е XML?

- E**X**tensible **M**arkup **L**anguage е:
  - **Универсална нотация** (формат / език на данни) за описване на **структурирани данни**
    - Използва текст с етикети
  - Проектиран за съхраняване и **пренос на данни**
  - Данните се съхраняват заедно с техните **мета-данни**

XML – Пример



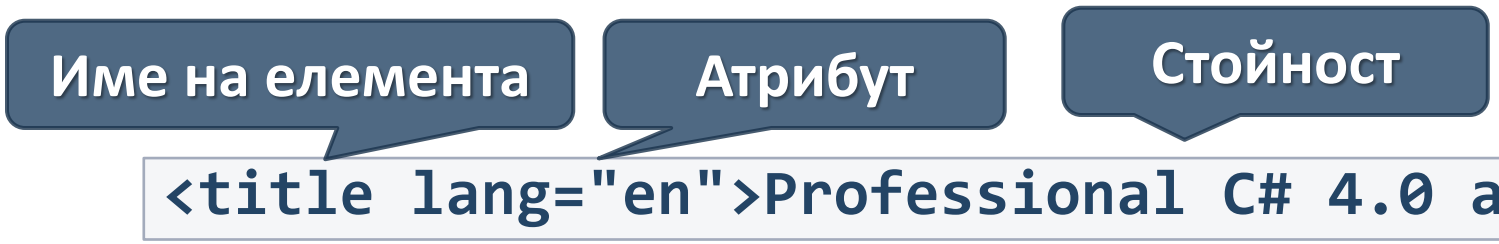

```
<?xml version="1.0"?>
<library name="Developer's Library">
  <book>
    <title>Professional C# 4.0 and .NET 4</title>
    <author>Christian Nagel</author>
    <isbn>978-0-470-50225-9</isbn>
  </book>
  <book>
    <title>Teach Yourself XML in 10
Minutes</title>
    <author>Andrew H. Watt</author>
    <isbn>978-0-672-32471-0</isbn>
  </book>
</library>
```

XML синтаксис

- **Хедър** – дефинира версията и кодирането на знаци

```
<?xml version="1.0" encoding="UTF-8"?>
```

- **Елементи** – дефиницията на структурата
- **Атрибути** – мета данните на елемента
- **Стойности** – реалните данни в елемента



- **Root** елемент – задължително е да има **само един**

Сериализиране на XML във файлове

- За да запишете XML документ във файл с настройки по подразбиране:

```
xmlDoc.Save("myBooks.xml");
```

- За да деактивирате автоматичното подравняване:

```
xmlDoc.Save("myBooks.xml",
SaveOptions.DisableFormatting);
```

- Сериализиране на **всякакъв обект** във файл:

```
var serializer = new XmlSerializer(typeof(ProductDTO));
using (var writer = new StreamWriter("myProduct.xml");)
{
  serializer.Serialize(writer, product);
}
```

Десериализиране на XML от низов XML

- Десериализиране на обект от низов XML

```
var serializer = new XmlSerializer(typeof(OrderDto[]), new
XmlRootAttribute("Orders"));

var deserializedOrders = (OrderDto[])serializer.Deserialize(new
StringReader(xmlString));
```

- Конкретизиране на **стойността на атрибута**

```
var attr = new XmlRootAttribute("Orders");
var serializer = new XmlSerializer(typeof(OrderDto[]), attr);

var deserializedOrders =
  (OrderDto[])serializer.Deserialize(new
StringReader(xmlString));
```

Какво е JSON?

JSON формат

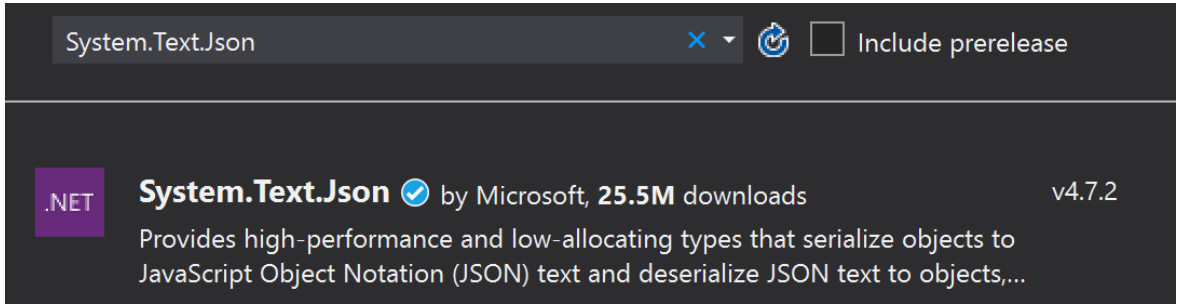
- **JSON** (**J**ava**S**cript **O**bject **N**otation) е
  - Четим за човека и машината
  - Базиран на обекти от **JavaScript**
  - Независим за разработващата среда и езика

```
{
    "firstName": "Pesho",
    "courses": ["C#", "JS", "ASP.NET"]
    "age": 23,
    "hasDriverLicense": true,
    "date": "2012-04-23T18:25:43.511Z",
    // ...
}
```

- JSON данните се състоят от двойки:
  - **Ключ** (низ)
  - **Стойност** (низ, число, масив, друга двойка ключ-стойност)

Вградена поддържка на JSON

- .NET има вградена поддръжка за JSON чрез **System.Text.Json** в NuGet Package



  - Поддържа сериализация и десериализация на обекти

- Трябва да включите във вашия проект **namespace-овете** :

```
using System.Text.Json;
using System.Text.Json.Serialization;
```

Сериализация на JSON (1)

- Сериализоторът System.Text.Json може да чете и да пише JSON

```
class WeatherForecast
{
    public DateTime Date { get; set; } = DateTime.Now;
    public int TemperatureC { get; set; } = 30;
    public string Summary { get; set; } = "Hot summer day";
}

static void Main()
{
    WeatherForecast forecast = new WeatherForecast();
    string weatherInfo = JsonSerializer.Serialize(forecast);
    Console.WriteLine(weatherInfo);
}
```

Сериализация на JSON (2)

- Създаване на JSON файлове

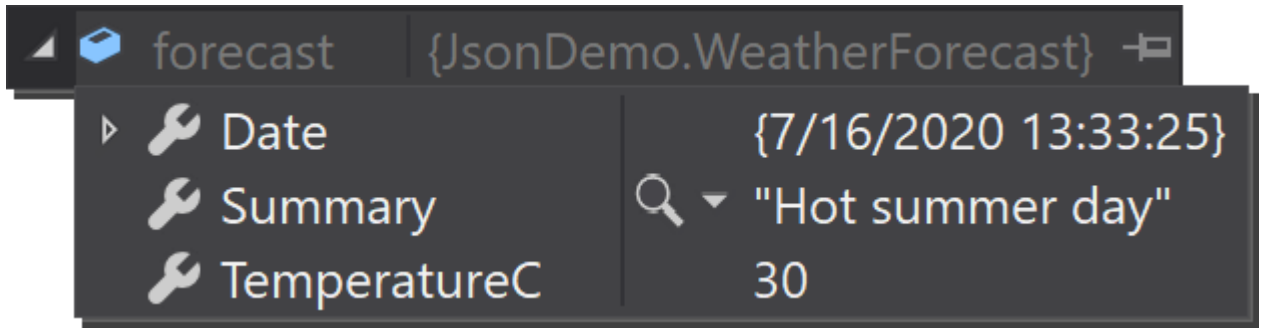
```
static void Main()
{
    WeatherForecast forecast = new WeatherForecast();
    string weatherInfo = JsonSerializer.Serialize(forecast);
    File.WriteAllText(file, weatherInfo);
}
```

```
{"Date":"2020-07-16T13:33:25","TemperatureC":30,"Summary":"Hot summer day"}
```

Десериализация на JSON

- За да десериализираме от файл, може да го прочетем и да използваме метода **Deserialize<>()**

```
static void Main()
{
    string jsonString = File.ReadAllText(file);
    WeatherForecast forecast =
      JsonSerializer.Deserialize<WeatherForecast>(jsonString);
}
```



Какво научихме днес?

SoftUni Foundation

- **Стрийм ==** подредена редица от битове
- Използвайте **StreamReader** / **StreamWriter** за текстови данни
- Използвайте **FileStream** да четете / пишете двоичен файл
- Използвайте класа **File** да четете / пишете файл наведнъж
- Използвайте класа **Directory** за работа с директории
- **Сериализация** – бинарна, XML, JSON