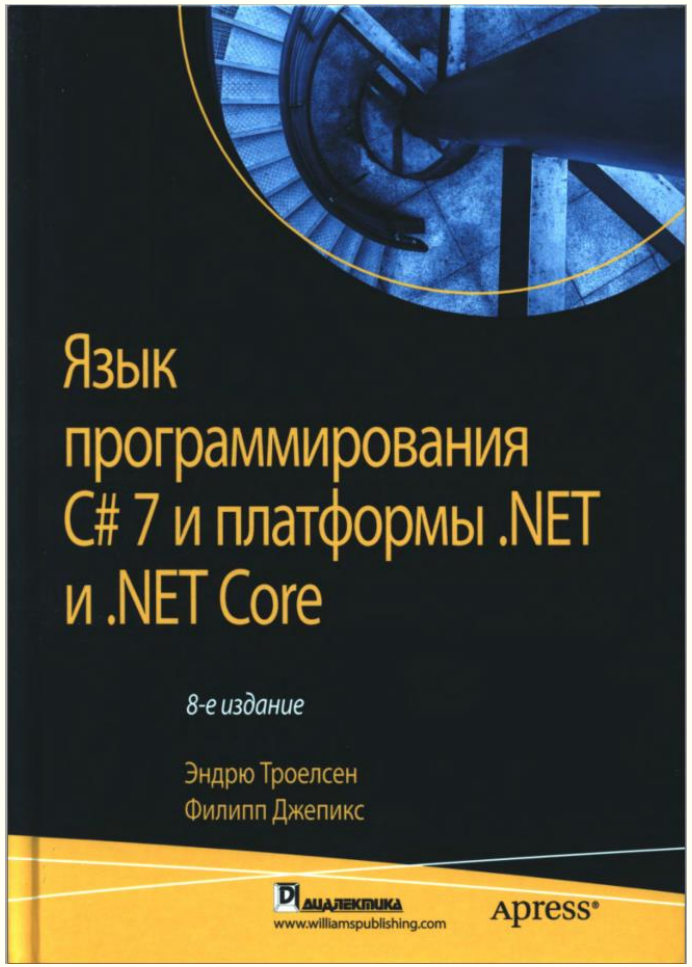
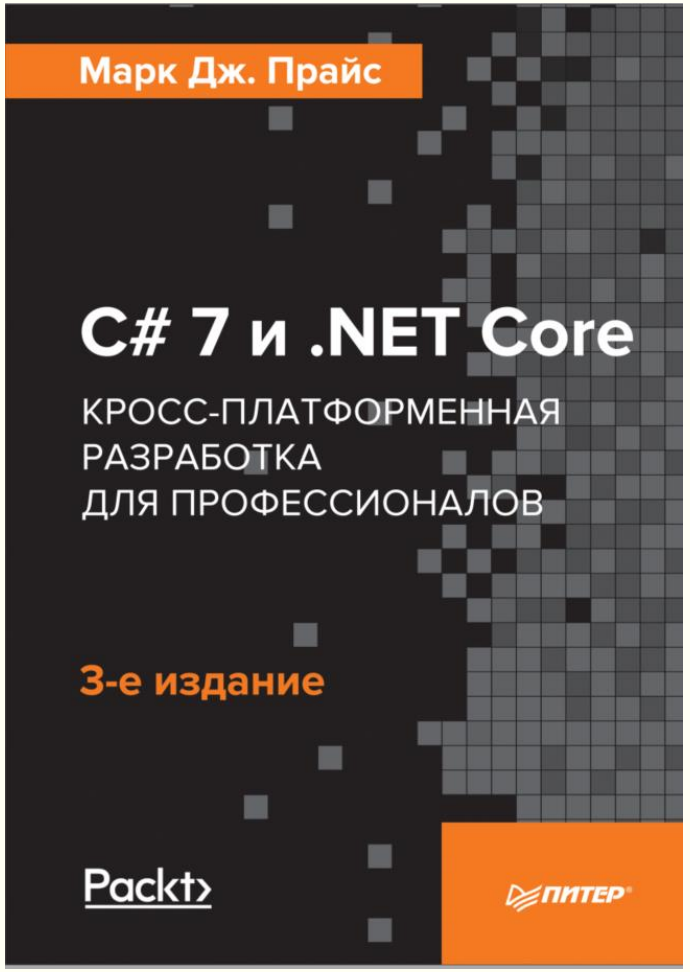
СЕРІАЛІЗАЦІЯ ОБ’ЄКТІВ

Питання 10.3.

Література


Язык программирования C# 7 и платформы .NET и .NET Core
8-е издание
Эндрю Троелсен
Филипп Джепикс
Диалектика
www.williamspublishing.com
apress


Марк Дж. Прайс
C# 7 и .NET Core
КРОСС-ПЛАТФОРМЕННАЯ РАЗРАБОТКА ДЛЯ ПРОФЕССИОНАЛОВ
3-е издание
Packt
ПИТЕР

Серіалізація об’єктів

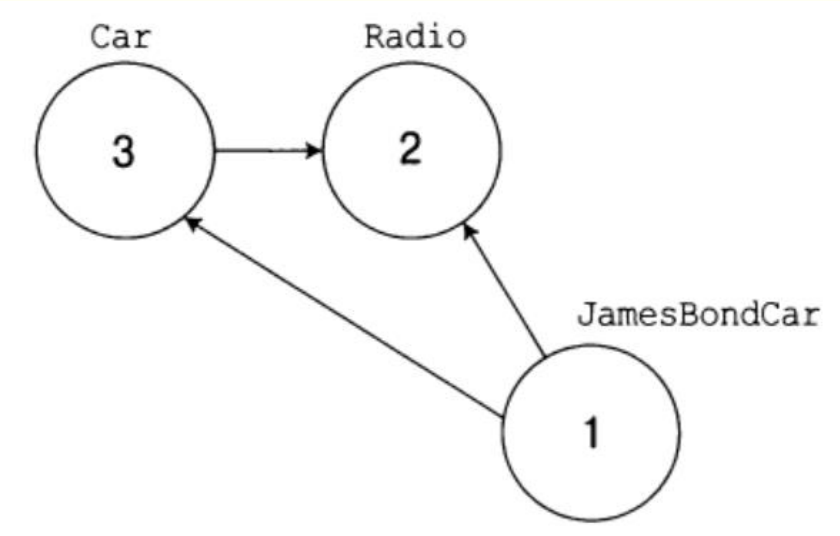
- Термін «серіалізація» описує процес збереження (та, можливо, передачі) стану об’єкта в потоці (наприклад, файловому потоці чи потоці в пам’яті).
  - Послідовність даних, що зберігаються, містить всю інформацію, необхідну для реконструкції (десеріалізації) стану об’єкта з метою подальшого використання.
- Часто збереження даних за допомогою служб серіалізації утворює код меншого об’єму, ніж застосування класів для зчитування/запису з простору імен System.IO.
  - Нехай потрібно створити настільний додаток з графічним інтерфейсом користувача, який повинен надавати можливість зберігати користувацькі налаштування (колір вікон, розмір шрифта тощо).
  - Визначимо клас UserPrefs та інкапсулюємо в ньому близько 20 полів даних.
  - При застосуванні System.IO.BinaryWriter довелось би вручну зберігати кожне поле об’єкта UserPrefs, а при завантаженні даних з файлу назад у пам’ять – використовувати System.IO.BinaryReader та вручну читати кожне значення для реконструкції об’єкта UserPrefs.
- Значно зекономить час атрибут [Serializable]

Серіалізація об’єктів

```
[Serializable]
public class UserPrefs
{
  public string WindowColor;
  public int FontSize;
}
```

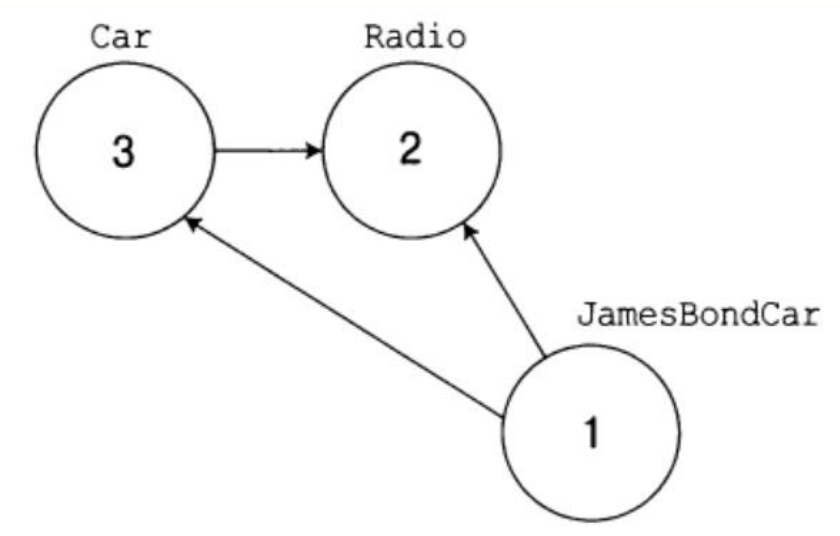
```
static void Main(string[] args)
{
  UserPrefs userData = new UserPrefs();
  userData.WindowColor = "Yellow";
  userData.FontSize = 50;

  // BinaryFormatter сохраняет данные в двоичном формате.
  // Чтобы получить доступ к BinaryFormatter, понадобится
  // импортировать System.Runtime.Serialization.Formatters.Binary.
  BinaryFormatter binFormat = new BinaryFormatter();

  // Сохранить объект в локальном файле.
  using(Stream fStream = new FileStream("user.dat",
    FileMode.Create, FileAccess.Write, FileShare.None))
  {
    binFormat.Serialize(fStream, userData);
  }
  Console.ReadLine();
}
```

Серіалізація об’єктів



- Коли об’єкт зберігається в потоці, всі пов’язані з ним дані (дані базового класу та об’єкти, які в ньому містяться) також автоматично серіалізуються.
  - Набір взаємопов’язаних об’єктів представляється за допомогою ***графа об’єктів***.
  - Служби серіалізації .NET також дозволяють зберігати граф об’єктів у різних форматах.
  - У попередньому прикладі застосовувався тип BinaryFormatter, тому стан об’єкта UserPrefs зберігається в компактному двійковому форматі.
- Граф об’єктів також можна зберегти в форматі SOAP або XML, застосовуючи інші типи форматерів.
  - Ці формати корисні для гарантування можливості передачі збережених об’єктів між різними ОС, мовами та архітектурами.
- Коли об’єкт серіалізується, середовище CLR враховує всі зв’язані об’єкти, щоб гарантувати коректне зберігання даних.
  - Графи об’єктів надають простий способ документування того, як між собою пов’язані елементи з набору.

Роль графів об’єктів



- Зауважте, що графи об’єктів не позначають відношення “являється” та “має” з ООП.
  - Стрілки в графі об’єктів можна читати як “вимагає” або “залежить від”.
- Кожний об’єкт у графі отримує унікальне числове значення.
  - Ці числа довільні та не мають значення для зовнішнього світу.
  - Після цього граф об’єктів може записувати набір залежностей для кожного об’єкта.
- Для прикладу розглянемо набір класів, які моделюють автомобілі.
  - Формула для представлення відношення записується наближено так:
  - [Car 3, ref 2], [Radio 2], [JamesBondCar 1, ref 3, ref 2]
  - Базовий клас Car “має” клас Radio.
  - Клас JamesBondCar розширяє базовий тип Car.

Конфігурування об’єктів для серіалізації

```
[Serializable]
public class Radio
{
  public bool hasTweeters;
  public bool hasSubWoofers;
  public double[] stationPresets;
  [NonSerialized]
  public string radioID = "XF-552RR6";
}
```

```
[Serializable]
public class Car
{
  public Radio theRadio = new Radio();
  public bool isHatchBack;
}
```

```
[Serializable]
public class JamesBondCar : Car
{
  public bool canFly;
  public bool canSubmerge;
}
```

- Щоб зробити об’єкт доступним для служб серіалізації .NET, потрібно тільки декорувати кожний зв’язаний клас (або структуру) атрибутом [Serializable].
  - Якщо деякий тип має члени-дані, які не повинні серіалізуватись, їх помічають атрибутом [NonSerialized].
- Поля даних у класах визначені як public для спрощення.
  - Слід віддавати перевагу закритим даним, представленим відкритими властивостями.
  - Для простоти в цих типах не визначались спеціальні конструктори, тому всі неініціалізовані поля даних отримають очікувані стандартні значення.
- Типи BinaryFormatter або SoapFormatter запрограмовані для серіалізації всіх серіалізованих полів, незалежно від того, представлені вони відкритими полями, закритими полями чи відкритими властивостями.

Ситуація суттєво змінюється, якщо використоується тип XmlSerializer

```
[Serializable]
public class Person
{
  // Открытое поле.
  public bool isAlive = true;
  // Закрытое поле.
  private int personAge = 21;
  // Открытое свойство/закрытые данные.
  private string fName = string.Empty;
  public string FirstName
  {
    get { return fName; }
    set { fName = value; }
  }
}
```

- Він серіалізує тільки відкриті поля даних або властивості.
  - При обробці цього типу за допомогою BinaryFormatter або SoapFormatter виявиться, що поля isAlive, personAge та fName зберігаються в вибраному потоці.
  - Проте XmlSerializer не збереже значення personAge, оскільки ця частина закритих даних не інкапсульована у відкритій властивості.
- Після конфігурування типів для участі в схемі серіалізації .NET наступний крок – вибір формату (двійкового, SOAP або XML) для зберігання стану об'єктів. Відповідні класи:
  - BinaryFormatter
  - SoapFormatter
  - XmlSerializer

Вибір форматера серіалізації

- ***Тип BinaryFormatter*** серіалізує стан об'єкта в потік, використовуючи двійковий формат.
  - Визначений у просторі імен System.Runtime.Serialization.Formatters.Binary зі збірки mscorlib.dll.
  - using System.Runtime.Serialization.Formatters.Binary;
  - Двійковий формат серіалізації може бути небезпечним.
- ***Тип SoapFormatter*** зберігає стан об'єкта у вигляді SOAP-повідомлення (стандартный XML-формат для передачі та прийому повідомлень від веб-служб).
  - Визначений у просторі імен System.Runtime.Serialization.Formatters.Soap в окремій збірці.
  - Для форматування графа об'єктів у SOAP-повідомлення спочатку потрібно додати посилання (Add Reference) на System.Runtime.Serialization.Formatters.Soap.dll, а потім вказати директиву using:
  - using System.Runtime.Serialization.Formatters.Soap;
- Для зберігання дерева об'єктів у документі XML передбачено ***тип XmlSerializer***.
  - Для використання потрібно вказати директиву using для простору імен System.Xml.Serialization та додати посилання на збірку System.Xml.dll.
  - Шаблони проектів Visual Studio автоматично посилаються на System.Xml.dll, тому достатньо:
  - using System.Xml.Serialization;

Інтерфейси IFormatter і IRemotingFormatter

```
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);
}
```

```
public interface IRemotingFormatter : IFormatter
{
  object Deserialize(Stream serializationStream, HeaderHandler handler);
  void Serialize(Stream serializationStream, object graph, Header[] headers);
}
```

```
static void SerializeObjectGraph(IFormatter itfFormat,
                                 Stream destStream, object graph)
{
  itfFormat.Serialize(destStream, graph);
}
```

- Всі форматери безпосередньо успадковані від System.Object, тому вони НЕ розділяють спільний набір членів від деякого базового класу серіалізації.
  - Проте типи BinaryFormatter і SoapFormatter підтримують спільні члени через реалізацію інтерфейсів IFormatter та IRemotingFormatter.
- В інтерфейсі System.Runtime.Serialization.IFormatter визначені основні методи Serialize() і Deserialize(), які виконують чорнову роботу з переміщення графів об'єктів у визначений потік і назад.
  - Також в IFormatter визначено кілька властивостей, які використовуються "за кулісами" реалізуючим типом
- Хоч взаємодіяти з цими інтерфейсами в більшості випадків не потрібно, поліморфізм на базі інтерфейсів дозволяє підставляти екземпляри BinaryFormatter або SoapFormatter там, де очікується IFormatter.

Точність типів серед форматерів

- Очевидна відмінність між форматтерами – спосіб збереження графу об’єктів у потоці.
  - Коли використовується тип BinaryFormatter, він зберігає не лише дані полів об’єктів з графа, але й повністю задану назву кожного типу й повну назву збірки, в якій він визначається (назва, версія, маркер відкритого ключа і культура).
  - Тому BinaryFormatter доречний при передаванні об’єктів за значенням (повне копіювання) між межами машин для використання в .NET-додатках.
- Форматер SoapFormatter зберігає трасування збірок-джерел за рахунок використання простору імен XML.
  - Наприклад, для вище згаданого класу Person відкриваючий елемент Person буде доповнений згенерованим параметром xlmns.

```
<a1:Person id="ref-1" xmlns:a1=
  "http://schemas.microsoft.com/clr/nsassem/SimpleSerialize/MyApp%2C%20
  Version%3D1.0.0.0%2C%20Culture%3Dneutral%2C%20PublicKeyToken%3Dnull">
  <isAlive>true</isAlive>
  <personAge>21</personAge>
  <fName id="ref-3">Mel</fName>
</a1:Person>
```

Точність типів серед форматерів

- Проте XmlSerializer не записує для типу його повну назву або збірку, в якій цей тип визначено.
  - Причина: відкрита природа представлення даних XML. Приклад для Person:

```
<?xml version="1.0"?>
<Person xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <isAlive>true</isAlive>
  <PersonAge>21</PersonAge>
  <FirstName>Frank</FirstName>
</Person>
```

- Для кросплатформності дотримуватись повної точності типів не варто, оскільки неможливо розраховувати, що всі можливі адресати зможуть зрозуміти специфічні для .NET типи даних.
  - SoapFormatter і XmlSerializer доречні, коли потрібно гарантувати якомога ширше розповсюдження дерева об'єктів.

Серіалізація об’єктів за допомогою BinaryFormatter

```
// Не забудьте импортировать пространства имен
// System.Runtime.Serialization.Formatters.Binary и System.IO!
static void Main(string[] args)
{
  Console.WriteLine("***** Fun with Object Serialization *****\n");
  // Создать JamesBondCar и установить состояние.
  JamesBondCar jbc = new JamesBondCar();
  jbc.canFly = true;
  jbc.canSubmerge = false;
  jbc.theRadio.stationPresets = new double[]{89.3, 105.1, 97.1};
  jbc.theRadio.hasTweeters = true;
  // Сохранить объект в указанном файле в двоичном формате.
  SaveAsBinaryFormat(jbc, "CarData.dat");
  Console.ReadLine();
}
```

Ключові методи BinaryFormatter:

- ***Serialize()*** – зберігає граф об’єктів у вказаний потік як послідовність байтів;
- ***Deserialize()*** перетворює збережену послідовність байтів у граф об’єктів.

- Нехай після інстанціювання JamesBondCar та модифікації деяких даних потрібно зберегти цей екземпляр у файлі *.dat.
  - Почнемо зі створення файлу * .dat за допомогою екземпляру типу FileStream.
  - Потім створимо екземпляр BinaryFormatter та передамо йому FileStream і граф об’єктів для зберігання.

Метод SaveAsBinaryFormat()

```
static void SaveAsBinaryFormat(object objGraph, string fileName)
{
  // Сохранить объект в файл CarData.dat в двоичном виде.
  BinaryFormatter binFormat = new BinaryFormatter();

  using(Stream fStream = new FileStream(fileName,
     FileMode.Create, FileAccess.Write, FileShare.None))
  {
    binFormat.Serialize(fStream, objGraph);
  }
  Console.WriteLine("=> Saved car in binary format!");
}
```

- Метод BinaryFormatter.Serialize() відповідає за побудову графа об'єктів і передачу послідовності байтів у деякий об'єкт породженого від Stream типу.
  - Тут потік представляє фізичний файл, проте серіалізувати об'єкти можна в будь-який породжений від Stream тип, зокрема, область памяті або мережевий потік.

CarData.dat | CarCollection.xml | CarData.soap

```
00000000  00 01 00 00 00 FF FF FF  FF 01 00 00 00 00 00 00  ................
00000010  00 0C 02 00 00 00 46 53  69 6D 70 6C 65 53 65 72  ......FSimpleSer
00000020  69 61 6C 69 7A 65 2C 20  56 65 72 73 69 6F 6E 3D  ialize, Version=
00000030  31 2E 30 2E 30 2E 30 2C  20 43 75 6C 74 75 72 65  1.0.0.0, Culture
00000040  3D 6E 65 75 74 72 61 6C  2C 20 50 75 62 6C 69 63  =neutral, Public
00000050  4B 65 79 54 6F 6B 65 6E  3D 6E 75 6C 6C 05 01 00  KeyToken=null...
00000060  00 00 1C 53 69 6D 70 6C  65 53 65 72 69 61 6C 69  ...SimpleSeriali
00000070  7A 65 2E 4A 61 6D 65 73  42 6F 6E 64 43 61 72 04  ze.JamesBondCar.
00000080  00 00 00 06 63 61 6E 46  6C 79 0B 63 61 6E 53 75  ....canFly.canSu
00000090  62 6D 65 72 67 65 08 74  68 65 52 61 64 69 6F 0B  bmerge.theRadio.
000000a0  69 73 48 61 74 63 68 42  61 63 6B 00 00 04 00 01  isHatchBack.....
000000b0  01 15 53 69 6D 70 6C 65  53 65 72 69 61 6C 69 7A  ..SimpleSerializ
000000c0  65 2E 52 61 64 69 6F 02  00 00 00 01 02 00 00 00  e.Radio.........
000000d0  01 00 09 03 00 00 00 00  05 03 00 00 00 15 53 69  ..............Si
000000e0  6D 70 6C 65 53 65 72 69  61 6C 69 7A 65 2E 52 61  mpleSerialize.Ra
000000f0  64 69 6F 03 00 00 00 0B  68 61 73 54 77 65 65 74  dio.....hasTweet
00000100  65 72 73 0D 68 61 73 53  75 62 57 6F 6F 66 65 72  ers.hasSubWoofer
00000110  73 0E 73 74 61 74 69 6F  6E 50 72 65 73 65 74 73  s.stationPresets
00000120  00 00 07 01 01 06 02 00  00 00 01 00 09 04 00 00  ................
00000130  00 0F 04 00 00 00 03 00  00 00 06 33 33 33 33 33  ...........33333
00000140  53 56 40 66 66 66 66 66  46 5A 40 66 66 66 66 66  SV@fffffFZ@fffff
00000150  46 58 40 0B                                       FX@.
```

Десеріалізація об’єктів за допомогою BinaryFormatter

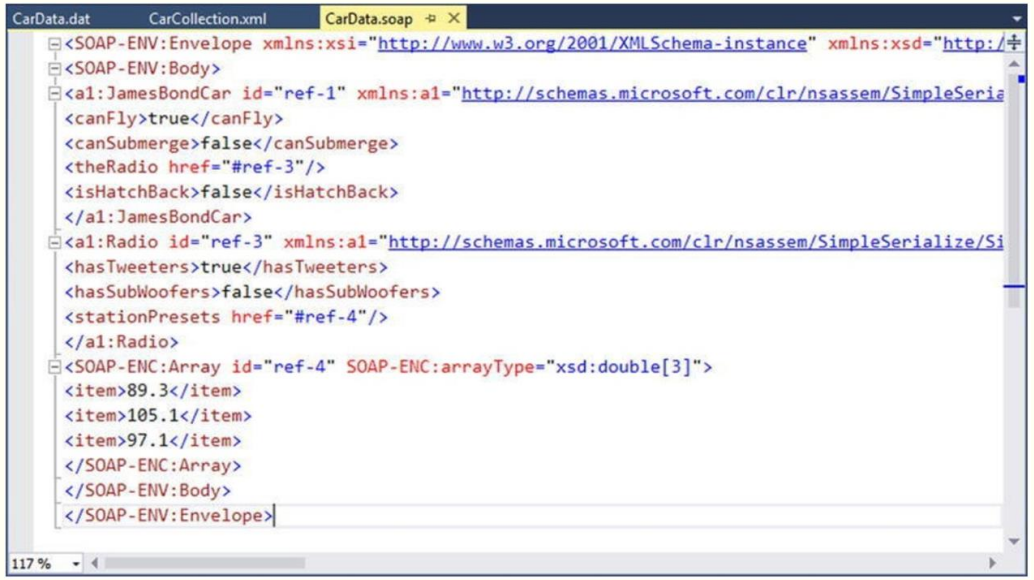
```
static void LoadFromBinaryFile(string fileName)
{
  BinaryFormatter binFormat = new BinaryFormatter();
  // Прочитать JamesBondCar из двоичного файла.
  using(Stream fStream = File.OpenRead(fileName))
  {
    JamesBondCar carFromDisk =
      (JamesBondCar)binFormat.Deserialize(fStream);
    Console.WriteLine("Can this car fly? : {0}", carFromDisk.canFly);
  }
}
```

- Після відкриття файлу CarData.dat можна викликати метод Deserialize() класу BinaryFormatter.
  - Метод Deserialize() повертає об’єкт типу System.Object, тому потрібно застосувати зведення типів.
  - При виклику Deserialize() йому передається породжений від Stream тип, який надає розташування збереженого графа об’єктів.

Серіалізація об’єктів за допомогою SoapFormatter

- Протокол SOAP (Simple Object Access Protocol — простий протокол доступу до об’єктів) визначає стандартний процес виклику методів у незалежній від платформи та ОС манері.
  - У файлі знаходяться XML-елементи, які описують значення стану поточного об’єкта JamesBondCar, а також відношення між об’єктами в графі, представлені за допомогою лексем #ref

```
// Не забудьте импортировать пространства имен
// System.Runtime.Serialization.Formatters.Soap
// и установить ссылку на System.Runtime.Serialization.Formatters.Soap.dll!
static void SaveAsSoapFormat (object objGraph, string fileName)
{
  // Сохранить объект в файле CarData.soap в формате SOAP.
  SoapFormatter soapFormat = new SoapFormatter();
  using(Stream fStream = new FileStream(fileName,
    FileMode.Create, FileAccess.Write, FileShare.None))
  {
    soapFormat.Serialize(fStream, objGraph);
  }
  Console.WriteLine("=> Saved car in SOAP format!");
}
```



Серіалізація об’єктів за допомогою XmlSerializer

- Цей форматер може застосовуватись для збереження відкритого стану заданого об’єкта в вигляді чистої XML-розмітки, на відміну від даних XML всередині SOAP-повідомлення.
  - Клас XmlSerializer вимагає, щоб усі серіалізовані типи в графі об’єктів підтримували стандартний конструктор, інакше під час виконання згенерується виняток InvalidOperationException.
  - Ключова відмінність: тип XmlSerializer вимагає вказування інформації щодо типу (класу), який потрібно серіалізувати.

```
static void SaveAsXmlFormat(object objGraph, string fileName)
{
  // Сохранить объект в файле CarData.xml в формате XML.
  XmlSerializer xmlFormat = new XmlSerializer(typeof(JamesBondCar),
    new Type[] { typeof(Radio), typeof(Car) });
  using(Stream fStream = new FileStream(fileName,
    FileMode.Create, FileAccess.Write, FileShare.None))
  {
    xmlFormat.Serialize(fStream, objGraph);
  }
  Console.WriteLine("=> Saved car in XML format!");
}
```

```
<?xml version="1.0"?>
<JamesBondCar xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <theRadio>
    <hasTweeters>true</hasTweeters>
    <hasSubWoofers>false</hasSubWoofers>
    <stationPresets>
      <double>89.3</double>
      <double>105.1</double>
      <double>97.1</double>
    </stationPresets>
    <radioID>XF-552RR6</radioID>
  </theRadio>
  <isHatchBack>false</isHatchBack>
  <canFly>true</canFly>
  <canSubmerge>false</canSubmerge>
</JamesBondCar>
```

Керування генерацією даних XML

- Дійсні XML-документи відповідають узгодженим правилам форматування, які зазвичай визначені в схемі XML або файлі визначення типу документу (Document-type Definition — DTD).
  - За умовчанням клас XmlSerializer серіалізує всі відкриті поля/властивості як елементи XML, а не як атрибути XML.
  - Для керування генерацією результуючого документу XML за допомогою класу XmlSerializer, необхідно декорувати типи будь-якою кількістю додаткових атрибутів з простору імен System.Xml.Serialization.
- У простому прикладі показано поточне представлення даних полів JamesBondCar в XML

```
<?xml version="1.0" encoding="utf-8"?>
<JamesBondCar xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xmlns:xsd="http://www.w3.org/2001/XMLSchema">
...
  <canFly>true</canFly>
  <canSubmerge>false</canSubmerge>
</JamesBondCar>
```

Керування генерацією даних XML

- Якщо потрібно вказати спеціальний простір імен XML, який уточнює JamesBondCar і кодує значення canFly і canSubmerge у вигляді атрибутів XML, модифікуємо визначення класу JamesBondCar:

```
[Serializable, XmlRoot(Namespace = "http://www.MyCompany.com")]
public class JamesBondCar : Car
{
  [XmlAttribute]
  public bool canFly;
  [XmlAttribute]
  public bool canSubmerge;
}
```

- Це видає показаний нижче документ XML:

```
<?xml version="1.0"""?>
<JamesBondCar xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  canFly="true" canSubmerge="false"
  xmlns="http://www.MyCompany.com">
...
</JamesBondCar>
```

Серіалізація колекцій об’єктів

- Метод Serialize() інтерфейсу IFormatter не надає способу вказати довільну кількість об’єктів у якості вводу (допускається тільки один об’єкт System.Object).
  - Також метод Deserialize() повертає одиночний об’єкт System.Object (те ж обмеження стосується XmlSerializer):

```
public interface IFormatter
{
  ...
  object Deserialize(Stream serializationStream);
  void Serialize(Stream serializationStream, object graph);
}
```

- Якщо передати об’єкт, який відмічений атрибутом [Serializable] та містить в собі інші об’єкти [Serializable], то за допомогою єдиного виклику даного методу буде зберігатись весь набір об’єктів.
  - На щастя, більшість типів з просторів імен System.Collections і System.Collections.Generic вже відмічені атрибутом [Serializable].
  - Таким чином, щоб зберегти множину об’єктів, просто додайте її в контейнер, на зразок масиву, ArrayList або List<T>), і серіалізуйте даний об’єкт в вибраний потік.

Керування генерацією даних XML

- Нехай клас JamesBondCar доповнено конструктором, який приймає 2 аргументи, для установки кількох фрагментів даних стану (зверніть увагу, що необхідно додавати стандартний конструктор, як того вимагає XmlSerializer):

```
[Serializable, XmlRoot(Namespace = "http://www.MyCompany.com")]
public class JamesBondCar : Car
{
  public JamesBondCar(bool skyWorthy, bool seaWorthy)
  {
    canFly = skyWorthy;
    canSubmerge = seaWorthy;
  }
  // XmlSerializer требует стандартного конструктора!
  public JamesBondCar(){}
...
}
```

```
static void SaveListOfCars()
{
  // Сохранить список List<T> объектов JamesBondCar.
  List<JamesBondCar> myCars = new List<JamesBondCar>();
  myCars.Add(new JamesBondCar(true, true));
  myCars.Add(new JamesBondCar(true, false));
  myCars.Add(new JamesBondCar(false, true));
  myCars.Add(new JamesBondCar(false, false));
  using(Stream fStream = new FileStream("CarCollection.xml",
    FileMode.Create, FileAccess.Write, FileShare.None))
  {
    XmlSerializer xmlFormat = new XmlSerializer(typeof(List<JamesBondCar>));
    xmlFormat.Serialize(fStream, myCars);
  }
  Console.WriteLine("=> Saved list of cars!");
}
```

Керування генерацією даних XML

- Оскільки використовується XmlSerializer, необхідно вказати інформацію про тип для кожного з підоб'єктів всередині кореневого об'єкта (тут – List<JamesBondCar>).
  - Якби застосовувались типи BinaryFormatter або SoapFormatter, то логіка була б ще простіша.

```
static void SaveListOfCarsAsBinary()
{
  // Сохранить объект ArrayList (myCars) в двоичном виде.
  List<JamesBondCar> myCars = new List<JamesBondCar>();

  BinaryFormatter binFormat = new BinaryFormatter();
  using(Stream fStream = new FileStream("AllMyCars.dat",
    FileMode.Create, FileAccess.Write, FileShare.None))
  {
    binFormat.Serialize(fStream, myCars);
  }
  Console.WriteLine("=> Saved list of cars in binary!");
}
```

Налаштування процесів серіалізації SOAP і двійкової серіалізації

- У більшості випадків стандартна схема серіалізації, яка постачається платформою .NET, доречна: потрібно лише застосувати атрибут [Serializable] до пов'язаних типів і передати дерево об'єктів обраному форматеру для обробки.
- Проте інколи може знадобитись втручання в процес конструювання дерева і процес серіалізації.
  - Наприклад, може існувати бізнес-правило: «всі поля даних повинні зберігатись у певному форматі», або необхідно додати доповнюючі дані в потік, які напряму не відображаються на поля об'єкта, що зберігається (наприклад, часові мітки або унікальні ідентифікатори).
  - Простір System.Runtime.Serialization.
- Коли BinaryFormatter серіалізує граф об'єктів, він відповідає за передачу наступної інформації в указаний потік:
  - повністю задану назву об'єкта в графі (наприклад, MyApp.JamesBondCar);
  - назву збірки, яка визначає граф об'єктів (наприклад, MyApp.exe);
  - екземпляр класу SerializationInfo, який містить всі дані стану, що підтримуються членами графа об'єктів.

Поглиблений погляд на серіалізацію об’єктів

- У ході десеріалізації BinaryFormatter використовує ту ж інформацію для побудови ідентичної копії об’єкта с применением данных, извлеченных из потока-источника.
  - Процес, який виконує SoapFormatter, дуже схожий.
  - Для максимальної мобільності об’єкта форматер XmlSerializer не зберігає повністю задану назву типу або збірки, в якій він міститься. Цей тип може зберігати тільки відкриті дані.
- Крім переміщення необхідних даних у потік і назад, форматери також аналізують члени графа об’єктів на предмет перелічених нижче частин інфраструктури.
  - ***Перевірка відмітки об’єкта атрибутом [Serializable].*** Якщо об’єкт не відмічений, генерується виняток SerializationException.
  - Якщо об’єкт відмечений атрибутом [Serializable], виконується ***перевірка, чи реалізує об’єкт інтерфейс ISerializable***. Якщо так, на цьому об’єкті викликається метод GetObjectData().
  - Якщо об’єкт не реалізує інтерфейс ISerializable, використовується стандартний процес серіалізації, який обробляє всі поля, не відмічені як [NonSerialized].
  - На додачу до визначення підтримки типом інтерфейсу ISerializable, форматери також відповідають за дослідження типів на предмет підтримки членів, оснащених атрибутами [OnSerializing], [OnSerialized], [OnDeserializing] або [OnDeserialized].

Налаштування серіалізації за допомогою атрибутів

```
[Serializable]
class MoreData
{
  private string dataItemOne = "First data block";
  private string dataItemTwo= "More data";

  [OnSerializing]
  private void OnSerializing(StreamingContext context)
  {
    // Вызывается во время процесса сериализации.
    dataItemOne = dataItemOne.ToUpper();
    dataItemTwo = dataItemTwo.ToUpper();
  }

  [OnDeserialized]
  private void OnDeserialized(StreamingContext context)
  {
    // Вызывается по завершении процесса десериализации.
    dataItemOne = dataItemOne.ToLower();
    dataItemTwo = dataItemTwo.ToLower();
  }
}
```

- Переважно налаштування серіалізації відбувається за допомогою атрибутів [OnSerializing], [OnSerialized], [OnDeserializing] та [OnDeserialized].
  - Код компактніший у порівнянні з реалізацією інтерфейсу ISerializable, оскільки не потрібно вручну взаємодіяти з параметром SerializationInfo.
  - Замість цього можна напряму модифікувати дані стану, коли форматер працює з типом.
- У випадку застосування цих атрибутів метод повинен визначатись так, щоб приймати параметр StreamingContext і не повертати нічого.
  - Застосовувати кожний з атрибутів серіалізації не обов'язково, можна просто втручатись у ті потрібні стадії процесу серіалізації.
  - Для ілюстрації наведемо новий тип [Serializable], який має ті ж вимоги, що й StringData, проте покладається на використання атрибутів [OnSerializing] та [OnDeserialized].

Налаштування серіалізації за допомогою атрибутів

```
static void Main(string[] args)
{
  Console.WriteLine("***** Fun with Custom Serialization *****");
  // Вспомните, что этот тип реализует ISerializable.
  StringData myData = new StringData();
  // Сохранить в локальный файл в формате SOAP.
  SoapFormatter soapFormat = new SoapFormatter();
  using(Stream fStream = new FileStream("MyData.soap",
    FileMode.Create, FileAccess.Write, FileShare.None))
  {
    soapFormat.Serialize(fStream, myData);
  }
  Console.ReadLine();
}
```

- Виконавши серіалізацію цього типу, виявиться, що дані зберігаються в верхньому регістрі, а десеріалізуються — в нижньому.
  - При наповненні об’єкта типу SerializationInfo всередині методу GetObjectData() іменувати елементи даних ідентично до назв внутрішніх змінних-членів типу не обов’язково.
  - Це корисно, якщо потрібно відв’язати дані типу від формату зберігання.
  - Проте отримувати значення в спеціальному захищеному конструкторі необхідно з указуванням тих же імен, що були призначені всередині GetObjectData().
  - Для демонстрації спеціалізованої серіалізації припустимо, що екземпляр MyStringData зберігається із застосуванням SoapFormatter (оновіть відповідним чином посилання на збірки та директиви using).

Переглядаючи отриманий файл *.soap, видно, що рядкові поля дійсно збережені у верхньому регістрі

```
<SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:clr="http://schemas.microsoft.com/soap/encoding/clr/1.0"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>

  <a1:StringData id="ref-1" ...>
    <First_Item id="ref-3">FIRST DATA BLOCK</First_Item>
    <dataItemTwo id="ref-4">MORE DATA</dataItemTwo>
  </a1:StringData>
  </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

ДЯКУЮ ЗА УВАГУ!

Наступне питання: залік)

Доповіді

- **Серіалізація за допомогою бібліотеки System.Text.Json**
  - Підтримується з .NET Core 2.0
  - Процес міграції з Newtonsoft.Json
  - Поточний стан Newtonsoft.Json.
- **Безпекові проблеми десеріалізації**
  - Приклад для Java
  - Репозиторій з прикладом для .NET