Файлові операції

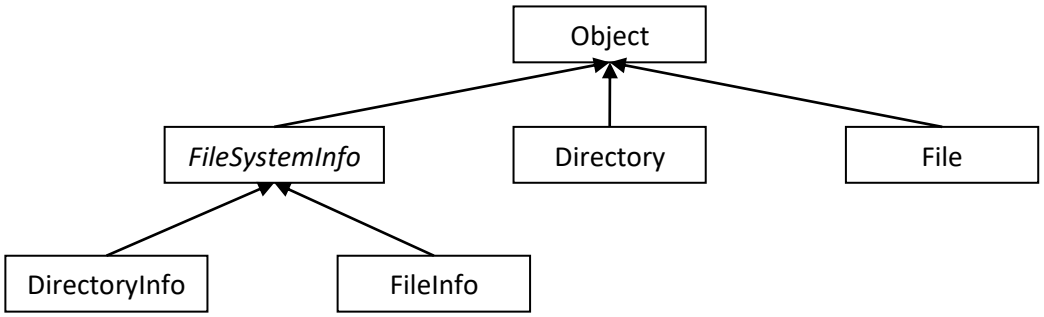
Простір імен System.IO

Неабстрактні класи введення-виведення	Опис
BinaryReader BinaryWriter	Дозволяють зберігати і завантажувати елементарні типи даних (цілі, булівські, рядкові тощо) у двійковому вигляді.
BufferedStream	Надає тимчасове сховище для потоку байтів, яке можна згодом перенести в постійне сховище.
Directory DirectoryInfo	Класи використовують для маніпулювання структурою каталогів комп'ютера. Тип Directory надає функціональність через статичні властивості й методи (створення об'єкта не потрібне). Тип DirectoryInfo надає подібні можливості через методи об'єкта, пов'язаного з конкретним каталогом.
DriveInfo	Надає детальну інформацію про диски комп'ютера.
File FileInfo	Класи дають змогу маніпулювати файлами комп'ютера: File – через статичні методи, а FileInfo – через методи екземпляра.
FileStream	Надає довільний доступ до файла з даними у вигляді потоку байтів.
FileSystemWatcher	Дає змогу відслідковувати зміни дискових файлів у певному каталозі.
MemoryStream	Потік довільного доступу до даних в пам'яті (до внутрішнього файла).
Path	Дозволяє виконувати операції над типами System.String, що містять інформацію про шлях до файла чи каталога в незалежній від платформи манері.
StreamWriter StreamReader	Використовують для запису та зчитування текстової інформації у файлі. Не підтримують довільного доступу.
StringWriter StringReader	Подібно до StreamWriter/StreamReader ці класи також працюють з текстовою інформацією, проте сховищем для даних є рядковий буфер, а не фізичний файл.

Програмна взаємодія з деревом каталогів

Класи Directory (DirectoryInfo) File (FileInfo)

Directory – для взаємодії з каталогами, File – з фізичними файлами. Ці класи надають статичні методи. Ефективно для разових дій, бо не потрібно створювати об'єкти. Повертають рядки. Класи Info роблять те саме, але через методи об'єкта. Ефективно, коли треба багато дій з одним файлом. Повертають строго типізовані об'єкти.



Властивості FileSystemInfo (спільні для каталогів і файлів):

- Attributes – можна отримати або встановити атрибути файла або каталога (системний, прихований тощо), задано значеннями переліку FileAttributes;
- CreationTime, LastAccessTime, LastWriteTime – відповідні часові атрибути файла;
- Exists – зручно перевіряти, чи файл (каталог) вже існує, чи створюємо новий;
- Name, Extension – повертає відповідно ім'я, розширення імені (тип);
- FullName – повертає повне ім'я (з повним шляхом);
- метод Delete() вилучає з файлової системи;
- метод Refresh() оновлює в об'єкті інформацію про фізичний файл, варто використовувати перед читанням атрибутів, щоб гарантувати їхню актуальність.

DirectoryInfo

Create(), CreateSubdirectory() – створює каталог (або ланцюжок підкаталогів) за правильно заданим шляхом – створює весь заданий шлях. Create, якщо об'єкт асоційовано з неіснуючим каталогом. Коли вже є каталог, у ньому можна створювати вкладені каталоги.

Delete() – вилучає каталог, якщо він порожній, інакше генерує виняток.

GetDirectories(), GetFiles() – повертають масиви DirectoryInfo, FileInfo про вкладені каталоги, файли.

MoveTo() – переміщає каталог у задане місце.

Parent – повертає надкаталог DirInfo

Root – кореневий каталог диска, на якому розташовано каталог – DirInfo.

```
// Отримати посилання на поточний робочий каталог.
DirectoryInfo currentDir = new DirectoryInfo(".");

// Посилання на конкретний каталог C:\Windows, використано рядок "як є".
DirectoryInfo windowsDir = new DirectoryInfo(@"C:\Windows");

// Посилання на каталог, якого ще нема. Створення.
DirectoryInfo newDir = new DirectoryInfo(@"C:\Docs\Testing\NewDir");
newDir.Create(); newDir.Refresh();
```

Перебір файлів

```
static void DisplayImageFiles()
{
    DirectoryInfo dir = new DirectoryInfo(@"C:\Windows\Web\Wallpaper");
    // Отримати всі файли з расширенням *.jpg.
    FileInfo[] imageFiles = dir.GetFiles("*.jpg", SearchOption.AllDirectories);
    // Скільки файлів знайдено?
    Console.WriteLine("Found {0} *.jpg files\n", imageFiles.Length);
    // Вивести інформацію про кожен файл.
    foreach (FileInfo f in imageFiles)
    {
        Console.WriteLine("File name: {0}", f.Name);    // ім'я файла
        Console.WriteLine("Full path: {0}", f.FullName);  // шлях до файла   // розмір
        Console.WriteLine("File size: {0} = {1:F2} K",
              f.Length, (double)f.Length/1024.0);
        Console.WriteLine("Creation: {0}", f.CreationTime); // час створення
        Console.WriteLine("Attributes: {0}\n", f.Attributes); // атрибути
    }
}
```

Створення підкаталога

```
      DirectoryInfo dir = new DirectoryInfo(".");
// Просто створити підкаталог
      dir.CreateSubdirectory("MyFolder");
// Створити і запам’ятати об’єкт DirectoryInfo.
      DirectoryInfo myDataFolder = dir.CreateSubdirectory(@"NewFolder\Data");
```

Використання Directory

```
        static void FunWithDirectoryType()
        {
            // Вивести список усіх дискових пристроїв комп'ютера.
            string[] drives = Directory.GetLogicalDrives();
            Console.WriteLine("Here are your drives:");
            foreach (string s in drives)
                Console.WriteLine("–> {0} ", s);
            // Вилучити створені раніше папки.
            Console.WriteLine("Press Enter to delete directories (false)");
            Console.ReadLine();
            try
            {
                // Другий параметр вказує, чи вилучати підкаталоги.
                Directory.Delete(@"C:\Docs\Testing", false);
            }
            catch (IOException e)
            {
                Console.WriteLine(e.Message);
            }
            Console.WriteLine("  once more (true) ...");
            try
            {
                // Другий параметр вказує, чи вилучати підкаталоги (і файли).
                Directory.Delete(@"C:\Docs\Testing", true);
            }
            catch (IOException e)
            {
                Console.WriteLine(e.Message);
            }
        }
```

Використання DriveInfo

```
        static void FunWithDriveInfo()
        {
            Console.WriteLine("***** Fun with DriveInfo *****\n");
            // Отримати інформацію про всі пристрої.
            DriveInfo[] myDrives = DriveInfo.GetDrives();
            // Вивести відомости про пристрої.
            foreach (DriveInfo d in myDrives)
            {
                Console.WriteLine("Name: {0}", d.Name);    // ім'я
                Console.WriteLine("Type: {0}", d.DriveType);  // тип
                // Перевірити, чи змонтовано пристрій.
                if (d.IsReady)
                {  // Вільний простір
                    Console.WriteLine("Free space: {0}", d.TotalFreeSpace);
                    Console.WriteLine("Format: {0}", d.DriveFormat); // формат
                    Console.WriteLine("Label: {0}", d.VolumeLabel);  // мітка
                }
                Console.WriteLine();
            }
        }
```

Використання FileInfo

Член класу	Опис
AppendText()	Створює об’єкт StreamWriter (описаний нижче) і дописує текст до файла
CopyTo()	Копіює наявний файл до нового файла
Create()	Створює новий файл і повертає об’єкт FileStream (описаний нижче) для взаємодії зі створеним файлом
CreateText()	Створює об’єкт StreamWriter для запису нового текстового файла
Delete()	Вилучає файл, асоційований з екземпляром FileInfo
Directory	Повертає екземпляр каталога, в якому зареєстровано асоційований файл
DirectoryName	Повертає повний шлях до батьківського каталога
Length	Повертає розмір файла
MoveTo()	Переміщує файл в нове місце, дає змогу вказати нове ім’я для файла
Name	Повертає ім’я файла
Open()	Відкриває файл зрізними правами читання/запису та спільного доступу
OpenRead()	Створює доступний тільки для читання об’єкт FileStream
OpenText()	Створює об’єкт StreamReader (описаний нижче) для читання з наявного текстового файла
OpenWrite()	Створює доступний тільки для запису об’єкт FileStream

Взаємодія з двійковими та символьними файлами

Методи отримання дескриптора файла

```
// Створити новий файл в робочому каталозі.
FileInfo f = new FileInfo(@".\Test.dat");
FileStream fs = f.Create();
// Дослідимо його можливості
Console.WriteLine("CanRead = {0}", fs.CanRead);
Console.WriteLine("CanWrite = {0}", fs.CanWrite);
Console.WriteLine("CanSeek = {0}", fs.CanSeek);
// Використати об'єкт FileStream...
string hello = "Hello, World!\n";
foreach (var c in hello) fs.WriteByte((byte)c);
Console.WriteLine("Position = {0}", fs.Position);
//fs.Write(bytes[],start,count);
// Закрити файловий потік.
fs.Close();
// Додаткові можливості: Seek(), асинхронне читання, блокування.
```

альтернативно

```
// Відкрити файл для дописування через FileInfo.Open() - точніше налаштування.
FileInfo f2 = new FileInfo(@".\Test.dat");
// Використаємо контекст using
using (FileStream fs2 = f2.Open(FileMode.Append, FileAccess.Write, FileShare.None))
{
    foreach (var c in we) fs2.WriteByte((byte)c);
    Console.WriteLine("fs2.Position = {0}", fs2.Position);
}
```

```
public enum FileMode
{ CreateNew, Create, Open, OpenOrCreate, Truncate, Append }

public enum FileAccess
{ Read, Write, ReadWrite }
```

```
public enum FileShare
{ Delete, Inheritable, None, Read, ReadWrite, Write }

// Отримати об’єкт FileStream з правами тільки для читання.
FileInfo f3 = new FileInfo(@".\Test3.dat");
using (FileStream readOnlyStream = f3.OpenRead() )
{       // Використати об’єкт FileStream...
}
// Тепер отримати об’єкт FileStream з правами тільки для запису.
FileInfo f4 = new FileInfo(@".\Test4.dat");
using (FileStream writeOnlyStream = f4.OpenWrite() )
{       // Використати об’єкт FileStream...              }
```

Метод FileInfo.OpenText()

```
// Отримати об’єкт StreamReader.
FileInfo f5 = new FileInfo(@"C:\boot.ini");
using (StreamReader sreader = f5.OpenText())
{       // Використати об’єкт StreamReader...
}
```

Методи FileInfo.CreateText() та FileInfo.AppendText()

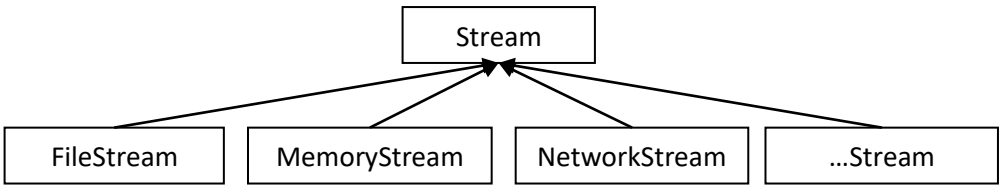
Використання File

AppendText(), Create (), CreateText (), Open (), OpenRead (), OpenWrite (), OpenText ().

Додаткові члени ReadAllBytes () – масив байтів; ReadAllLines () – масив рядків; ReadAllText () – рядок; WriteAllBytes (); WriteAllLines (); WriteAllText()

```
Console.WriteLine("***** Simple I/O with the File Type *****\n");
string[] myTasks = {"Fix bathroom sink","Call Dave","Call Mom and Dad","Play Xbox 360"};
// Записати всі дані до файла в робочому каталозі.
File.WriteAllLines(@".\tasks.txt", myTasks);
// Прочитати всі дані та вивести їх на консоль.
foreach (string task in File.ReadAllLines(@".\tasks.txt"))
{
    Console.WriteLine("TODO: {0}", task);
}
// File автоматично звільняє дескриптор файла.
```

Використання потоків



Члени абстрактного класу Stream:

CanRead, CanWrite, CanSeek, Length, Position,

Close(), Flush(), Read(), ReadByte(), Seek(), SetLength(), Write(), WriteByte().

Використання FileStream (просте втілення абстрактного класу)

Читає записує байт (масив байтів) – головний недолік – робота на низькому рівні

```
Console.WriteLine("***** Fun with FileStreams *****\n");
// Отримати об'єкт FileStream.
using (FileStream fStream = File.Open(@"..\myMessage.dat", FileMode.Create))
{
    // Закодувати рядок у вигляді масиву байтів.
    string msg = "Hello, University!";
    byte[] msgAsByteArray = Encoding.Default.GetBytes(msg);
    // Записати byte[] до файлу.
    fStream.Write(msgAsByteArray, 0, msgAsByteArray.Length);
    // Скинути внутрішній вказівник потоку.
    fStream.Position = 0;
    // Прочитати дані з файла і вивести їх на консоль.
    Console.Write("Your message as an array of bytes: ");
    byte[] bytesFromFile = new byte[msgAsByteArray.Length];
    for (int i = 0; i < msgAsByteArray.Length; i++)
    {
        bytesFromFile[i] = (byte)fStream.ReadByte();
        Console.Write(bytesFromFile[i]);
    }
    // Вивести декодоване повідомлення.
    Console.Write("\nDecoded Message: ");
    Console.WriteLine(Encoding.Default.GetString(bytesFromFile));
}
```

MemoryStream для виведення в пам’ять; NetworkStream – у мережеве з’єднання

Введення-виведення тексту

abstract class TextWriter: Close(), Flush(), NewLine, Write(), WriteLine()

class StreamWriter : TextWriter { AutoFlush } // виведення символів до файла

class StringWriter : TextWriter // виведення даних (символів) у пам’ять

```
using (StreamWriter writer = File.CreateText("reminders.txt"))
{       writer.WriteLine("Don't forget Mother's Day this year...");
        // Вставити новий (попрожній) рядок.
        writer.Write(writer.NewLine) ;
}
```

abstract class TextReader: Peek(), Read(), ReadBlock(), ReadLine(), ReadToEnd()

class StreamReader : TextReader // завантаження символів з файла

class StringReader : TextReader // завантаження символів з пам’яті

```
using (StreamReader sr = File.OpenText("reminders.txt"))
{       string input = null;
        while ((input = sr.ReadLine()) != null)
        {       Console.WriteLine (input);
        }
}
```

Безпосереднє створення

```
// Отримати StreamWriter і записати рядкові дані.
using (StreamWriter writer = new StreamWriter("reminders.txt"))
{       …       }
// Отримати StreamReader і прочитати рядкові дані з файла.
using (StreamReader sr = new StreamReader("reminders.txt"))
{       …       }
```

Введення-виведення у внутрішньому форматі

class BinaryWriter: BaseStream, Close(), Flush(), Seek(), Write()

```
// Відкрити засіб запису даних у двійковому форматі.
FileInfo f = new FileInfo("BinFile.dat");
using (BinaryWriter bw = new BinaryWriter(f.OpenWrite()))
{
        // Створити деякі дані для збереження у файлі.
        double aDouble = 1234.67;
        int anInt = 34567;
        string aString = "A, B, C";
        // Записати дані.
        bw.Write(aDouble);
        bw.Write(anInt) ;
        bw.Write(aString) ;
}
```

class BinaryReader: BaseStream, Close(), PeekChar(), Read(), ReadXXXX()

```
FileInfo f = new FileInfo("BinFile.dat");
// Читати двійкові дані з потоку.
using (BinaryReader br = new BinaryReader(f.OpenRead()))
{       // порядок читання такий, як порядок запису
        Console.WriteLine(br.ReadDouble()) ;
        Console.WriteLine(br.ReadInt32() ) ;
        Console.WriteLine(br.ReadString() ) ;
}
```

Програмне відслідковування файлів

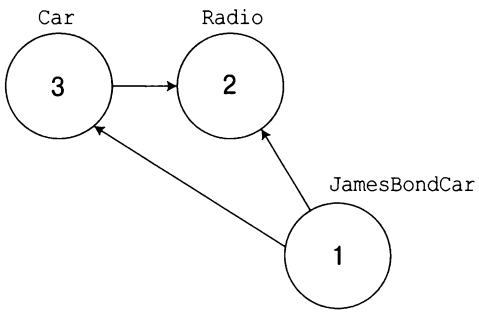
```
public enum NotifyFilters
{
       Attributes, CreationTime,
       DirectoryName, FileName,
       LastAccess, LastWrite,
       Security, Size,
}
```

```
{
    // Встановити шлях до каталога, за яким потрібно спостерігати.
    FileSystemWatcher watcher = new FileSystemWatcher();
    try
    {
        watcher.Path = @"C:\Users\Admin\Documents\Лекції\ConsoleFileDirectoryAppl";
    }
    catch (ArgumentException ex)
    {
        Console.WriteLine(ex.Message);
        return;
    }
    // Вказати цілі спостереження.
    watcher.NotifyFilter = NotifyFilters.LastAccess |
        NotifyFilters.LastWrite | NotifyFilters.FileName | NotifyFilters.DirectoryName;
    // Слідкувати за всіма текстовими файлами.
    watcher.Filter = "*.txt";
    // Додати опрацювання подій.
    watcher.Changed += new FileSystemEventHandler(OnChanged);
    watcher.Created += new FileSystemEventHandler(OnChanged);
    watcher.Deleted += new FileSystemEventHandler(OnChanged);
    watcher.Renamed += new RenamedEventHandler(OnRenamed);
    // Розпочати спостереження за каталогом.
    watcher.EnableRaisingEvents = true;
    // Очікувати команди користувача на завершення програми.
    Console.WriteLine(@"Press 'q' to quit app...");
    while (Console.Read() != 'q') ;
}
static void OnChanged(object source, FileSystemEventArgs e)
{      // Показати, що зроблено, якщо файл змінено, створено чи видалено.
     Console.WriteLine("File \"{0}\" {1}.", e.FullPath, e.ChangeType);
}
static void OnRenamed(object source, RenamedEventArgs e)
{      // Показати, що файл було перейменовано.
     Console.WriteLine("File {0} renamed to {1}", e.OldFullPath, e.FullPath);
}
```

Серіалізація об’єктів

Щоб зберегти-відновити об’єкт, можна використати двійкове введення-виведення з ручним перебором полів (у правильному порядку!). Але компактніший код отримаємо за допомогою класів серіалізації

```
[Serializable]
public class UserPrefs
{
    public string WindowColor;
    public int FontSize;
    public UserPrefs(string color, int size)
    {
        WindowColor = color;
        FontSize = size;
    }
}
    static void Save(UserPrefs a, UserPrefs b)
    {
        // BinaryFormatter зберігає дані в двійковому форматі.
        // Щоб отримати доступ до BinaryFormatter, доведеться
        // імпортувати System.Runtime.Serialization.Formatters.Binary.
        BinaryFormatter binFormat = new BinaryFormatter();
        // Зберегти об'єкт в локальному файлі.
        using (Stream fStream = new FileStream("user.dat",
                              FileMode.Create, FileAccess.Write, FileShare.None))
        {
            binFormat.Serialize(fStream, a);
            binFormat.Serialize(fStream, b);
        }
    }
    static void Load(out UserPrefs a, out UserPrefs b)
    {
        BinaryFormatter binFormat = new BinaryFormatter();
        using (Stream fStream = new FileStream("user.dat",
                              FileMode.Open, FileAccess.Read, FileShare.None))
        {
            a = (UserPrefs)binFormat.Deserialize(fStream);
            b = (UserPrefs)binFormat.Deserialize(fStream);
        }
    }
```



Граф об’єктів відображає залежності між об’єктами, зберігається у закодованому вигляді, гарантує правильну серіалізацію складних об’єктів.

[Car 3, ref 2], [Radio 2], [JamesBondCar 1, ref 3, ref 2]

У прикладі можна вказати довільний об’єкт (серіалізовний) і довільний потік (з перелічених вище)

```
[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;
}
```

Можливі формати серіалізації

BinaryFormatter, SoapFormatter (нащадки IFormatter) збережуть всі серіалізовні поля: і доступні, і закриті; XmlSerializer – тільки відкриті (або з відкритими властивостями)

• BinaryFormatter	System.Runtime.Serialization.Formatters.Binary	зберігає повні імена типів, збірок
• SoapFormatter	System.Runtime.Serialization.Formatters.Soap	сприяють поширенню на різні
• XmlSerializer	System.Xml.Serialization	платформи

Двійкове зберігання-відновлення

```
// Створити JamesBondCar і задати стан.
JamesBondCar jbc = new JamesBondCar();
jbc.canFly = true;
jbc.canSubmerge = false;
jbc.isHatchBack = true;
jbc.theRadio.stationPresets = new double[] { 89.3, 105.1, 97.1 };
jbc.theRadio.hasTweeters = true;
jbc.theRadio.hasSubWoofers = true;
Console.WriteLine(jbc);
//Зберегти об'єкт у вказаному файлі в двійковому форматі.
SaveAsBinaryFormat(jbc, "CarData.dat");
//завантажити об'єкт з указаного файла.
LoadFromBinaryFile("CarData.dat");
Console.ReadLine();

static void SaveAsBinaryFormat(JamesBondCar objGraph, string fileName)
{
    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!\n");
}
static void LoadFromBinaryFile(string fileName)
{
    BinaryFormatter binFormat = new BinaryFormatter();
    using (Stream fStream = File.OpenRead(fileName))
```

```
    {
        JamesBondCar carFromDisk = (JamesBondCar)binFormat.Deserialize(fStream);
        Console.WriteLine("Loaded car is ==>\n" + carFromDisk);
    }
}
```

Simple Object Access Protocol

```
// Не забудьте імпортувати простір імен System.Runtime.Serialization.Formatters.Soap
// і встановити посилання на System.Runtime.Serialization.Formatters.Soap.dll!
static void SaveAsSoapFormat (object objGraph, string fileName)
{       SoapFormatter soapFormat = new SoapFormatter();
        using (Stream fStream = new FileStream(fileName, FileMode.Create, FileAccess.Write, FileShare.None))
        {       soapFormat.Serialize(fStream, objGraph);
        }
}
```

XmlSerializer

```
static void SaveAsXmlFormat(object objGraph, string fileName)
{       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);
        }
}
```

Серіалізація колекцій об’єктів

```
List<JamesBondCar> myCars = new List<JamesBondCar>();
myCars.Add(new JamesBondCar(true, true)); // XmlSerializer вимагає стандартного конструктора також!
myCars.Add(new JamesBondCar(true, 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);
}

List<JamesBondCar> list = new List<JamesBondCar>();
list.Add(jbc); list.Add(jbc1);
BinaryFormatter binFormat = new BinaryFormatter();
using (Stream fStream = new FileStream("AllMyCars.dat", FileMode.Create,
                                       FileAccess.Write, FileShare.None))
{
    binFormat.Serialize(fStream, list);
}
list.Clear();
using (Stream fStream = new FileStream("AllMyCars.dat", FileMode.Open,
                                       FileAccess.Read, FileShare.None))
{
    list = (List<JamesBondCar>)binFormat.Deserialize(fStream);
}
foreach (var car in list) Console.WriteLine(car);
```

Спеціальні налаштування серіалізації можна виконати, реалізувавши інтерфейс ISerializable