3 Работа с файлами. Классы File и FileInfo

Класс File – предоставляет набор статических методов для работы с файлом и получения информации о файле.

Общая форма вызова методов:

```
File.ИмяМетода("имя файла", параметры метода)
```

Первым параметром передается имя файла, например: "C:\\Temp\\1.txt" или @"C:\Temp\1.txt".

Класс FileInfo – предоставляет набор методов экземпляра для работы с файлом и свойств для получения информации о файле.

Для использования обязательно создать объект класса:

```
FileInfo имяОбъекта = new FileInfo("имя файла");
имяОбъекта.ИмяМетода(параметры метода) или имяОбъекта.ИмяСвойства
```

Имя файла указано при создании объекта, поэтому в параметрах метода не указывается

Примеры методов и свойств классов File и FileInfo и их вызовов

Действие	Применение **File**	Применение **FileInfo**

Предварительное действие для работы примеров	Создать строковую переменную **fileName** для хранения имени файла:
	`string fileName = @"C:\Temp\1.txt";`
		Создать переменную **file** типа FileInfo:
		`string fileName = @"C:\Temp\1.txt";`
		`FileInfo file = new FileInfo(fileName);`

Создание файла	File.**Create(**fileName**)**;	file.**Create()**;
Удаление файла	File.**Delete(**fileName**)**;	file.**Delete()**;
Копирование файла	File.**Copy(**filename, `"новое имя"`**)**;	file.**CopyTo(**`"новое имя"`**)**;
Перемещение файла	File.**Move(**fileName**,** `"новое имя"`**)**;	file.**MoveTo(**`"новое имя"`**)**;

Проверка существования файла	File.**Exists(**fileName**)**;
	`if (File.Exists(fileName))`
	`{`
	`  // работа с существующим файлом`
	`}`
		Свойство **Exists**
		`if (file.Exists)`
		`{`
		`  // работа с существующим файлом`
		`}`

Свойства переменной типа FileInfo для получения информации о файле:

- **Name** – имя файла (без пути)
- **Extension** – расширение файла
- **FullName** – полное имя файла (путь+имя файла)
- **DirectoryName** – путь к файлу (без имени)
- **Length** – размер файла

4 Чтение и запись данных в текстовые файлы

Способ №1. Применение методов класса File

Класс File содержит следующие статические методы для чтения и записи данных в файл:

- **File.ReadДанные(**имя файла**)** – чтение данных из файла в переменную
- **File.WriteДанные(**имя файла, записываемые данные**)** – запись данных из приложения в файл
- **File.AppendДанные(**имя файла, записываемые данные**)** – дозапись данных из приложения в конец файла

Последним параметром у методов можно указать кодировку (например: **Encoding.ASCII**)

Данные, указываемые в названиях методов ReadДанные, WriteДанные, AppendДанные:

- **AllBytes** – массив байт. Пример: `File.WriteAllBytes(path, new byte[]{ 1, 2, 3});`
- **AllLines** – массив строк. Пример: `File.AppendAllLines(fileName, new string[]{ "hello", "world"});`
- **AllText** – строка. Пример: `string text = File.ReadAllText(fileName);`

Способ №2. Применение классов StreamWriter и StreamReader

Класс StreamWriter – предоставляет методы для записи данных в текстовый файл:
- **Write(значение)** – метод записи в файл значения без перевода на новую строку
- **WriteLine(значение)** – метод записи в файл значения с переводом на новую строку

Класс StreamReader – предоставляет методы для чтения данных из текстового файла:
- **Read()** – метод чтения из файла одного или нескольких символов
- **ReadLine()** – метод чтения из файла строки символов (с текущей позиции до перевода строки)
- **ReadToEnd()** – метод чтения из файла всех символов с текущей позиции до конца файла

Классы StreamWriter и StreamReader требуют создания объекта. в конструкторах указывается имя файла или файловый поток **FileStream** и можно указать кодировку, разрешение на дозапись (чтобы данные в файле не удалялись при записи).

В примерах объекты **создаются в операторе using** для того, чтобы не писать обработчик ошибок try-finally, не вызывать методы открытия и закрытия файловых потоков, не освобожать вручную ресурсы.

Пример 3. Запись 10 строк и текущей даты в файл:
```
string fileName = @"C:\Temp\1.txt";
// создание файлового потока на запись данных в текстовый файл
using (StreamWriter writer = new StreamWriter(fileName))
{
    for (int i = 1; i <= 10; i++)
    {
        // записываем значения переменной I в файл построчно
        writer.WriteLine($"строка {i}");
    }
    // запись текущей даты в конец файла
    writer.WriteLine(DateTime.Now);
}
```

Пример 4. Чтение данных из файла в метку fileLabel:
```
string fileName = @"C:\Temp\1.txt";
fileLabel.Text = "";
string result = "";
// создание файлового потока на чтение данных из текстового файла
using (StreamReader reader = new StreamReader(fileName))
{
    // читаем построчно, пока есть строки для чтения
    // каждая записывается в строковую переменную result
    while ((result = reader.ReadLine()) != null)
    {
        // запись данных из переменной result в метку fileLabel
        fileLabel.Text += result + Environment.NewLine;
    }
}
```

5 Чтение и запись бинарных (двоичных) файлов

Двоичный (бинарный) файл — последовательность произвольных байтов. Расширение: .dat, .bin, etc.

Класс BinaryWriter – предоставляет метод для записи двоичных данных в файл:

- **Write(значение)** – метод записи в файл значения определенного типа (из стандартных)

Класс BinaryReader – предоставляет методы для чтения двоичных данных из файла:

- **PeekChar()** – метод, возвращающий следующий доступный для чтения символ, не перемещая позицию байта или символа вперед. Применяется для проверки, что есть данные для чтения.
- **ReadТип()** – метод чтения из файла значения или массива значений указанного типа (из стандартных).

Примеры методов чтения данных из двоичного файла:

- ReadBoolean(): считывает значение bool и перемещает указатель на один байт
- ReadChar(): считывает значение char, то есть один символ, и перемещает указатель на столько байтов, сколько занимает символ в текущей кодировке
- ReadInt32(): считывает значение int и перемещает указатель на 4 байта
- ReadString(): считывает значение string. Каждая строка предваряется значением длины строки, которое представляет 7-битное целое число

Классы BinaryWriter и BinaryReader требуют создания объекта. в конструкторах указывается файловый поток **FileStream** и можно указать кодировку. Файловый поток можно создать, используя File.Open(…):

`File.Open(имя файла, FileMode.ВариантОткрытия)`

Варианты открытия файла:

- FileMode.Append – открытие на дозапись
- FileMode.Open – открытие существующего файла
- FileMode.Create – создание файла, существующий перезапишется
- FileMode.CreateNew – создание файла, если такой не существует
- FileMode.OpenOrCreate – открытие существующего или создание нового файла

В примерах объекты **создаются в операторе using** для того, чтобы не писать обработчик ошибок try-finally, не вызывать методы открытия и закрытия файловых потоков.

Пример 5. Запись в двоичный файл строки и числа:

```
using (BinaryWriter writer = new BinaryWriter(File.Open(fileName, FileMode.OpenOrCreate)))
{
    // запись данных в двоичный файл
    writer.Write("admin"); // запись имени пользователя (строка)
    writer.Write(18);      // запись возраста пользователя (число)
}
```

Пример 6. Чтение из двоичного файла строки и числа:

```
using (BinaryReader reader = new BinaryReader(File.Open(fileName, FileMode.Open)))
{
    // чтение данных из двоичного файла, пока далее есть символ для чтения
    while (reader.PeekChar() > -1)
    {
        string login = reader.ReadString();      // чтение строки
        int age = reader.ReadInt32();            // чтение числа
    }
}
```