РАБОТА С ФАЙЛОВОЙ СИСТЕМОЙ

В пространстве имен System.IO предусмотрено четыре класса, которые предназначены для работы с файловой системой компьютера, т.е., для создания, удаления, переноса и т.д. файлов и каталогов.

Первые два типа – Directory и Fi1e реализуют свои возможности с помощью статических методов, поэтому данные классы можно использовать без создания соответствующих объектов (экземпляров классов).

Типы – DirectoryInfo и FileInfo порождены от класса FileSystemInfo и обладают схожими функциональными возможностями с Directory и Fi1e, но реализуются путем создания соответствующих экземпляров классов.

Замечание

Перед изучением данной темы на диске ***d*** *своего компьютера создайте папку EXAMPLE, структура которой соответствует Рис. 44. Будьте внимательны, имена файлов и папок должны соответствовать Рис. 44, иначе при выполнении заданий вы получите результаты, отличающиеся от того, что показано в примерах.*

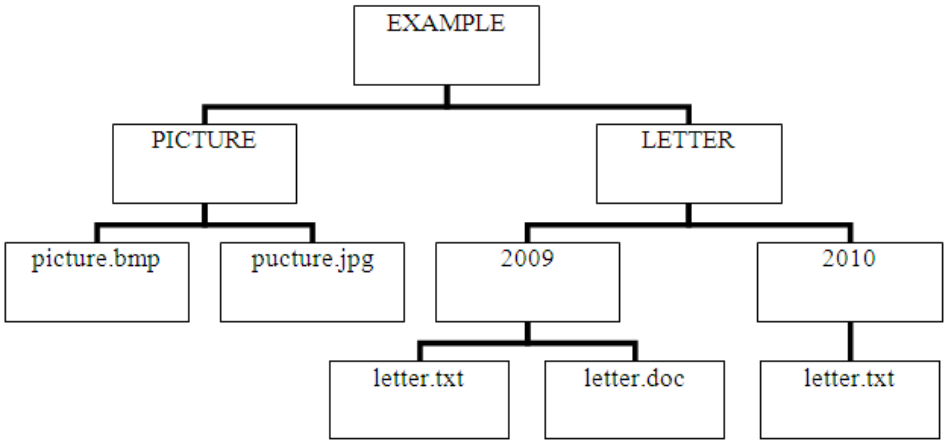

Рис. 44. Структура папки EXAMPLE

В качестве графических файлов сохраните любые рисунки. В текстовые файлы запишите любой текст.

Так как мы будем изменять структуру папки EXAMPLE и, возможно, при выполнении заданий вы будете допускать ошибки, мы рекомендуем вам создать резервную копию данной папки.

Работа с каталогами

Абстрактный класс FileSystemInfo

Значительная часть членов класса FileSystemInfo предназначена для работы с общими характеристиками файла, или каталога (метками времени, атрибутами и т.п.). Рассмотрим некоторые свойства FileSystemInfo:

Свойство	Описание
Attributes	Позволяет получить или установить атрибуты для данного объекта файловой системы. Для этого свойства используются значения и перечисления FileAttributes.

Свойство	Описание
CreationTime	Позволяет получить или установить время создания объекта файловой системы.
Exists	Может быть использовано для того, чтобы определить, существует ли данный объект файловой системы.
Extension	Позволяет получить расширение для файла.
FullName	Возвращает имя файла или каталога с указанием пути к нему в файловой системе.
LastAccessTime	Позволяет получить или установить время последнего обращения к объекту файловой системы.
LastWriteTime	Позволяет получить или установить время последнего внесения изменений в объект файловой системы.
Name	Возвращает имя указанного файла. Это свойство доступно только для чтения. Для каталогов возвращает имя последнего каталога в иерархии, если это возможно. Если нет, возвращает полный путь.

В FileSystemInfo предусмотрено и несколько методов. Например, метод Delete() позволяет удалить объект файловой системы с жесткого диска, а Refresh() – обновить информацию об объекте файловой системы.

Класс DirectoryInfo

Данный класс наследует члены класса FileSystemInfo и содержит дополнительный набор членов, которые предназначены для создания, перемещения, удаления и получения информации о каталогах и подкаталогах в файловой системе. Наиболее важные методы класса содержатся в следующей таблице:

Методы	Описание
Create() CreateSubDirectory()	Создают каталог (или подкаталог) по указанному пути в файловой системе.
Delete()	Удаляет каталог.
GetDirectories()	Позволяет получить доступ к подкаталогам текущего каталога (в виде массива объектов DirectoryInfo).
GetFiles()	Позволяет получить доступ к файлам текущего каталога (в виде массива объектов FileInfo).
MoveTo()	Перемещает каталог и все его содержимое на новый адрес в файловой системе.
Parent	Возвращает родительский каталог в иерархии файловой системы.

Работа с типом DirectoryInfo начинается с того, что мы создаем экземпляр класса, указывая при вызове конструктора в качестве параметра путь к нужному каталогу. Если мы хотим обратиться к текущему каталогу (то есть к каталогу, в котором в настоящее время

производится выполнение приложения), вместо параметра используется обозначение ".". Например:

```
// Создаем ссылку на текущий каталог
DirectoryInfo dir1 = new DirectoryInfo(".");
// создаем ссылку на каталог d:\EXAMPLE
DirectoryInfo dir2 = new DirectoryInfo(@"d:\EXAMPLE");
```

Если мы попытаемся создать объект DirectoryInfo, связав его с несуществующим каталогом, то будет сгенерировано исключение System.IO.DirectoryNotFoundException. Если же все нормально, то мы получим доступ к данному каталогу. В примере, который приведен ниже, мы создаем объект DirectoryInfo, который связан с каталогом d:\prim, и выводим информацию о данном каталоге:

```
using System;
using System.IO;

namespace Example
{
   class Program
   {
      static void Main()
      {
         DirectoryInfo dir = new DirectoryInfo(@"d:\ EXAMPLE");
         Console.WriteLine("***** {0} *****", dir.Name);
         Console.WriteLine("FullName: {0}", dir.FullName);
         Console.WriteLine("Parent: {0}", dir.Parent);
         Console.WriteLine("Creation: {0}", dir.CreationTime);
         Console.WriteLine("Attributes: {0}",
                           dir.Attributes.ToString());
         Console.WriteLine("Root: {0}", dir.Root);
      }
   }
}
```

Результат работы программы:

```
***** EXAMPLE *****
FullName: d:\ EXAMPLE
Parent:
Creation: 05.11.2008 23:05:13
Attributes: Directory
Root: d:\
```

Задание

Измените программу так, чтобы можно было получить информацию о времени последнего обращения к каталогу и о времени внесения последних изменений в каталог.

Свойство Attributes позволяет получить информацию об атрибутах объекта файловой системы. Возможные значения данного свойства приведены в следующей таблице:

Значение	**Описание**
Archive	Этот атрибут используется приложениями при проведении резервного копирования, а в некоторых случаях – при удалении старых файлов.

Значение	Описание
Compressed	Определяет, что файл является сжатым.
Directory	Определяет, что объект файловой системы является каталогом.
Encrypted	Определяет, что файл является зашифрованным.
Hidden	Определяет, что файл является скрытым (такой файл не будет выводиться при обычном просмотре каталога).
Normal	Определяет, что файл находится в обычном состоянии и для него установлены любые другие атрибуты. Этот атрибут не может использоваться с другими атрибутами.
Offline	Файл (расположенный на сервере) кэширован в хранилище off-line на клиентском компьютере. Возможно, что данные этого файла уже устарели.
Readonly	Файл доступен только для чтения.
System	Файл является системным (то есть файл является частью операционной системы, или используется исключительно операционной системой).

Через *DirectoryInfo* можно не только получать доступ к информации о текущем каталоге, но и получить доступ к информации о его подкаталогах:

```
using System;
using System.IO;
namespace Example
{
   class Program
   {
      static void Show(DirectoryInfo dir)
      {
         Console.WriteLine("***** {0} *****", dir.Name);
         Console.WriteLine("FullName: {0}", dir.FullName);
         Console.WriteLine("Parent: {0}", dir.Parent);
         Console.WriteLine("Creation: {0}", dir.CreationTime);
         Console.WriteLine("Attributes: {0}",
                           dir.Attributes.ToString());
         Console.WriteLine("Root: {0}", dir.Root);
         Console.WriteLine();
      }
      static void Main()
      {
         DirectoryInfo dir = new DirectoryInfo(@"d:\ EXAMPLE ");
         Show(dir);
         DirectoryInfo[] subDirects = dir.GetDirectories();
         Console.WriteLine("Найдено подкаталогов: {0}\n",
                           subDirects.Length);
         foreach (DirectoryInfo item in subDirects)
         {
            Show(item);
         }
      }
```

EDM Systems

```
      }
   }
```

Результат работы программы:

```
***** EXAMPLE *****
FullName: d:\ EXAMPLE
Parent:
Creation: 05.11.2008 23:05:13
Attributes: Directory
Root: d:\
Найдено подкаталогов: 2
*****LETTER*****
FullName: d:\ EXAMPLE\LETTER
Parent: EXAMPLE
Creation: 05.11.2008 23:05:13
Attributes: Directory
Root: d:\
*****PUCTURE*****
FullName: d:\ EXAMPLE\ PUCTURE
Parent: EXAMPLE
Creation: 05.11.2008 23:05:13
Attributes: Directory
Root: d:\
```

Данный пример позволяет просмотреть только подкаталоги для каталога EXAMPLE. Вспомним тот факт, что модель файловой системы можно описать с помощью АТД «дерево». В свою очередь дерево – это частный случай графа. Поэтому, чтобы просмотреть информацию обо всех подкаталогах текущего каталога, независимо от уровня вложенности, применим модификацию обхода графа в глубину.

```
using System;
using System.IO;
namespace Example
{
   class Program
   {
      static void Show(DirectoryInfo dir)
      {
         Console.WriteLine("***** {0} *****", dir.Name);
         Console.WriteLine("FullName: {0}", dir.FullName);
         Console.WriteLine("Parent: {0}", dir.Parent);
         Console.WriteLine("Creation: {0}", dir.CreationTime);
         Console.WriteLine();
         DirectoryInfo[] subDirects = dir.GetDirectories();
         if (subDirects.Length!=0)
         {
            Console.WriteLine("Найдено подкаталогов: {0}\n",
                              subDirects.Length);
            foreach (DirectoryInfo item in subDirects)
            {
               Show(item);
            }
         }
```

EPAM Systems

```
        }
        static void Main()
        {
           DirectoryInfo dir = new DirectoryInfo(@"d:\EXAMPLE");
           Show(dir);
        }
     }
  }
```

Результат работы программы:

```
***** EXAMPLE *****
FullName: d:\ EXAMPLE
Parent:
Creation: 05.11.2008 23:05:13
Найдено подкаталогов: 2
*****LETTER*****
FullName: d:\ EXAMPLE\LETTER
Parent: EXAMPLE
Creation: 05.11.2008 23:05:13
Найдено подкаталогов: 2
*****2009*****
FullName: d:\ EXAMPLE\LETTER\2009
Parent: LETTER
Creation: 17.08.2009 12:24:09
*****2010*****
FullName: d:\ EXAMPLE\LETTER\2010
Parent: LETTER
Creation: 17.08.2009 12:32:45
*****PUCTURE*****
FullName: d:\ EXAMPLE\ PUCTURE
Parent: EXAMPLE
Creation: 05.11.2008 23:05:13
```

Задание

Преобразуйте метод Show таким образом, чтобы выполнялся обход дерева каталогов в ширину.

Метод ***CreateSubdirectory***() позволяет создать в выбранном каталоге вложенный подкаталог. При этом уровень вложения может быть разным:

```
static void Main()
{
   DirectoryInfo dir = new DirectoryInfo(@"d:\EXAMPLE");
   dir.CreateSubdirectory("REFERENCE"); //создали подкаталог
   //создали вложенный подкаталог
   dir.CreateSubdirectory(@"BOOK\2008");
   Show(dir); //метод Show был рассмотрен ранее
}
```

EPAM Systems

Результат работы программы:

```
***** EXAMPLE *****
FullName: d:\ EXAMPLE
Parent:
Creation: 05.11.2008 23:05:13
Найдено подкаталогов: 4
*****BOOK*****
FullName: d:\ EXAMPLE\BOOK
Parent: EXAMPLE
Creation: 17.08.2009 13:44:12
Найдено подкаталогов: 1
*****2008*****
FullName: d:\ EXAMPLE\BOOK\2008
Parent: BOOK
Creation: 17.08.2009 13:44:12
*****LETTER*****
FullName: d:\ EXAMPLE\LETTER
Parent: EXAMPLE
Creation: 05.11.2008 23:05:13
Найдено подкаталогов: 2
*****2009*****
FullName: d:\ EXAMPLE\LETTER\2009
Parent: LETTER
Creation: 17.08.2009 12:24:09
*****2010*****
FullName: d:\ EXAMPLE\LETTER\2010
Parent: LETTER
Creation: 17.08.2009 12:32:45
*****PUCTURE*****
FullName: d:\ EXAMPLE\ PUCTURE
Parent: EXAMPLE
Creation: 05.11.2008 23:05:13
*****REFERENCE*****
FullName: d:\ EXAMPLE\ REFERENCE
Parent: EXAMPLE
Creation: 17.08.2009 13:44:11
```

Метод ***MoveTo***() позволяет переместить текущий каталог по заданному в качестве параметра пути. При этом возможно произвести переименование каталога. Например:

```
static void Main()
{
   DirectoryInfo dir = new DirectoryInfo(@"d:\EXAMPLE\REFERENCE");
   dir.MoveTo(@"d:\EXAMPLE\LETTER\REFERENCE"); //случай 1
   dir = new DirectoryInfo(@"d:\EXAMPLE\BOOK\2008");
   dir.MoveTo(@"d:\EXAMPLE\PUCTURE\2010");  //случай 2
   dir = new DirectoryInfo(@"d:\EXAMPLE");
   Show(dir);
}
```

EDAM Systems

В первом случае каталог REFERENCE перемещается по адресу d:\EXAMPLE\LETTER\REFERENCE. Так как имя перемещаемого каталога совпадает с крайним правым именем в адресе нового местоположения каталога, то переименования не происходит.

Во втором случае происходит переименование каталога 2008 (из каталога BOOK), т.к. имя перемещаемого каталога не совпадает с крайним правым именем в адресе нового местоположения d:\EXAMPLE\PUCTURE\2010.

Результат работы программы:

```
***** EXAMPLE *****
FullName: d:\ EXAMPLE
Parent:
Creation: 05.11.2008 23:05:13
Найдено подкаталогов: 3
*****BOOK*****
FullName: d:\ EXAMPLE\BOOK
Parent: EXAMPLE
Creation: 17.08.2009 13:44:12
*****LETTER*****
FullName: d:\ EXAMPLE\LETTER
Parent: EXAMPLE
Creation: 05.11.2008 23:05:13
Найдено подкаталогов: 3
*****2009*****
FullName: d:\ EXAMPLE\LETTER\2009
Parent: LETTER
Creation: 17.08.2009 12:24:09
*****2010*****
FullName: d:\ EXAMPLE\LETTER\2010
Parent: LETTER
Creation: 17.08.2009 12:32:45
*****REFERENCE*****
FullName: d:\ EXAMPLE\ LETTER\REFERENCE
Parent: LETTER
Creation: 17.08.2009 13:44:11
*****PUCTURE*****
FullName: d:\ EXAMPLE\ PUCTURE
Parent: EXAMPLE
Creation: 05.11.2008 23:05:13
Найдено подкаталогов: 1
*****2010*****
FullName: d:\ EXAMPLE\ PUCTURE \2010
Parent: PUCTURE
Creation: 17.08.2009 13:44:12
```

Задание 1

Создайте вложенный подкаталог BEST\2010 для каталога d:\EXAMPLE\PUCTURE.

Задание 2

Перенесите каталог d:\EXAMPLE\PUCTURE\BEST\2010 в каталог d:\EXAMPLE.

Задание 3

Переименуйте каталог BEST в каталог MUSIC.

Ранее мы рассмотрели метод GetDirectories, реализованный в классе DirectoryInfo, который позволял нам получить информацию обо всех подкаталогах текущего каталога. Этот же метод можно использовать для получения поиска каталогов по заданной маске, например:

```
DirectoryInfo[] subFind = dir.GetDirectories(<маска>);
```

Следует отметить, что маска – это строковый литерал, в записи которого принимают участие два специальных символа:

* – используется для обозначения последовательности символов любой длины, в том числе и пустой;

? – используется для обозначения одного любого символа.

Например:

1) маска "2*" говорит о том, что мы ищем любой объект файловой системы, имя которого начинается с символа 2;
2) маска "*.txt" указывает на то, что нас интересуют все объекты файловой системы, имя которых заканчивается последовательностью символов .txt;
3) маска "?.*" указывает на то, что нас интересуют все объекты файловой системы, имя которых начинается с любого символа, далее стоит точка, после чего следует любое количество символов.

Рассмотрим пример, который позволит нам найти все каталоги, имена которых начинаются с символа 2.

```
using System;
using System.IO;

namespace Example
{
   class Program
   {
      static void Show(DirectoryInfo dir)
      {
         Console.WriteLine("***** {0} *****", dir.Name);
         Console.WriteLine("FullName: {0}", dir.FullName);
         Console.WriteLine("Parent: {0}", dir.Parent);
         Console.WriteLine("Creation: {0}", dir.CreationTime);
         Console.WriteLine();
      }

      static void Find(DirectoryInfo dir, string mask)
      {
         DirectoryInfo[] subDirects = dir.GetDirectories();
         DirectoryInfo[] subFind = dir.GetDirectories(mask);
         foreach (DirectoryInfo item in subFind)
         {
            Show(item);
         }
```

EPAM Systems

```
            if (subDirects.Length != 0)
            {
               foreach (DirectoryInfo item in subDirects)
               {
                  Find(item, mask);
               }
            }
         }
         static void Main()
         {
            DirectoryInfo dir = new DirectoryInfo(@"d:\EXAMPLE");
            string mask = "2*";
            Find(dir, mask);
         }
      }
   }
```

Результат работы программы:

```
*****2009*****
FullName: d:\ EXAMPLE\LETTER\2009
Parent: LETTER
Creation: 17.08.2009 12:24:09
*****2010*****
FullName: d:\ EXAMPLE\LETTER\2010
Parent: LETTER
Creation: 17.08.2009 12:32:45

*****2010*****
FullName: d:\ EXAMPLE\ PUCTURE \2010
Parent: PUCTURE
Creation: 17.08.2009 13:44:12
*****2010*****
FullName: d:\ EXAMPLE\ MUSIC\2010
Parent: MUSIC
Creation: 17.08.2009 13:59:01
```

Метод ***Delete***() класса DirectoryInfo удаляет пустой каталог. Если вы попытаетесь удалить не пустой каталог, то возникнет исключение System.IO.IOException.

Для того чтобы удалить каталог вместе с вложенными каталогами и файлами, нужно обратиться к данному методу с параметром true.

Рассмотрим следующий пример:

```
static void Main()
{
   DirectoryInfo dir = new DirectoryInfo(@"d:\EXAMPLE\BOOK");
   dir.Delete(); //удалили пустой каталог
   dir = new DirectoryInfo(@"d:\EXAMPLE\MUSIC");
   dir.Delete(true); //удалили не пустой каталог
   dir = new DirectoryInfo(@"d:\EXAMPLE");
   //используем рекурсивный метод для обхода дерева каталогов
   Show(dir);
}
```

EDAM Systems

Результат работы программы:

```
***** EXAMPLE *****
FullName: d:\ EXAMPLE
Parent:
Creation: 05.11.2008 23:05:13
Найдено подкаталогов: 2
*****LETTER*****
FullName: d:\ EXAMPLE\LETTER
Parent: EXAMPLE
Creation: 05.11.2008 23:05:13
Найдено подкаталогов: 3
*****2009*****
FullName: d:\ EXAMPLE\LETTER\2009
Parent: LETTER
Creation: 17.08.2009 12:24:09
*****2010*****
FullName: d:\ EXAMPLE\LETTER\2010
Parent: LETTER
Creation: 17.08.2009 12:32:45
*****REFERENCE*****
FullName: d:\ EXAMPLE\ LETTER\REFERENCE
Parent: LETTER
Creation: 17.08.2009 13:44:11
*****PUCTURE*****
FullName: d:\ EXAMPLE\ PUCTURE
Parent: EXAMPLE
Creation: 05.11.2008 23:05:13
Найдено подкаталогов: 1
*****2010*****
FullName: d:\ EXAMPLE\ PUCTURE \2010
Parent: PUCTURE
Creation: 17.08.2009 13:44:12
```

Задание

Удалите программным путем весь каталог EXAMPLE.

Класс Directory

Работать с каталогами файловой системы компьютера можно и при помощи класса Directory, функциональные возможности которого во многом совпадают с возможностями DirectoryInfo. Единственное, о чем следует помнить – члены данного класса реализованы статически, поэтому для их использования нет необходимости создавать объект.

Замечание

Перед выполнением примеров скопируйте на диск d копию каталога EXAMPLE.

Рассмотрим пример, который позволит нам просмотреть структуру каталога EXAMPLE:

```
using System;
using System.IO;
```

EPAM Systems

```
namespace Example
{
   class Program
   {
      static void Show(string name)
      {
         Console.WriteLine("***** {0} *****", name);
         Console.WriteLine("Parent: {0}",
                           Directory.GetParent(name));
         Console.WriteLine("Creation: {0}",
                           Directory.GetCreationTime(name));
         Console.WriteLine();
         string [] subDirects = Directory.GetDirectories(name);
         if (subDirects.Length != 0)
         {
            Console.WriteLine("Найдено подкаталогов: {0}\n",
                              subDirects.Length);
            foreach (string item in subDirects)
            {
               Show(item);
            }
         }
      }
      static void Main()
      {
         string name = @"d:\EXAMPLE";
         Show(name);
      }
   }
}
```

Результат работы программы:

```
***** d:\ EXAMPLE *****
Parent: d:\
Creation: 05.11.2008 23:05:13

Найдено подкаталогов: 2

***** d:\ EXAMPLE\LETTER *****
Parent: d:\ EXAMPLE
Creation: 05.11.2008 23:05:13

Найдено подкаталогов: 2

***** d:\ EXAMPLE\LETTER\2009*****
Parent: d:\ EXAMPLE\LETTER
Creation: 17.08.2009 12:24:09

***** d:\ EXAMPLE\LETTER\2010*****
Parent: d:\ EXAMPLE\LETTER
Creation: 17.08.2009 12:32:45

***** d:\ EXAMPLE\ PUCTURE *****
Parent: d:\ EXAMPLE\
Creation: 05.11.2008 23:05:13
```

EPAM Systems

Как видно из результатов работы данного примера, при работе с классом Directory мы обращаемся к статическим методам, передавая в качестве параметра строковый литерал, который хранит полное имя каталога. Соответственно, метод GetDirectory возвращает нам строковый массив имен подкаталогов.

Следующий фрагмент программы показывает, как можно создать, перенести, переименовать, или удалить каталог с помощью статического класса Directory:

```
static void Main()
{
   //Создание каталогов
   Directory.CreateDirectory(@"d:\EXAMPLE\REFERENCE");
   Directory.CreateDirectory(@"d:\EXAMPLE\BOOK\2008");
   //Перенос каталога
   Directory.Move(@"d:\EXAMPLE\REFERENCE",
                  @"d:\EXAMPLE\LETTER\REFERENCE");
   //Перенос и переименование каталогов
   Directory.Move(@"d:\EXAMPLE\BOOK\2008",
                  @"d:\EXAMPLE\PUCTURE\2010");
   //Удаление пустого каталога
   Directory.Delete(@"d:\EXAMPLE\BOOK");
   //Удаление не пустого каталога
   Directory.Delete(@"d:\EXAMPLE\PUCTURE", true);
   string name = @"d:\EXAMPLE";
   Show(name);
}
```

Результат работы программы:

```
***** d:\ EXAMPLE *****
Parent: d:\
Creation: 05.11.2008 23:05:13
Найдено подкаталогов: 1
***** d:\ EXAMPLE\LETTER *****
Parent: d:\ EXAMPLE
Creation: 05.11.2008 23:05:13
Найдено подкаталогов: 3
***** d:\ EXAMPLE\LETTER\2009*****
Parent: d:\ EXAMPLE\LETTER
Creation: 17.08.2009 12:24:09
***** d:\ EXAMPLE\LETTER\2010*****
Parent: d:\ EXAMPLE\LETTER
Creation: 17.08.2009 12:32:45
***** d:\ EXAMPLE\ LETTER\REFERENCE *****
Parent: d:\ EXAMPLE\ LETTER
Creation: 17.08.2009 14:21:01
```

Что выбрать: Directory, или DirectoryInfo?

Как мы видели, все, что можно сделать с помощью класса Directory, можно сделать и с помощью класса DirectoryInfo. Соответственно встает вопрос: какой класс когда следует использовать?

EAM Systems

Имеет прямой смысл использовать статический класс Directory тогда, когда требуется осуществить единичное действие с каталогом. В этом случае вызов будет выполнен быстрее, поскольку .NET Framework не придется создавать новый объект с последующим вызовом соответствующего метода. Однако если приложение осуществляет несколько операций над каталогами, то более разумным представляется создать экземпляр объекта DirectoryInfo и использовать его методы. Это позволит сэкономить определенное время, поскольку объект будет заранее настроен на нужный каталог в файловой системе, в то время как статическому классу придется каждый раз осуществлять его поиск заново.

Замечание

Аналогичное правило действует при выборе между классами File и FileInfo.

Работа с файлами

Класс FileInfo

Класс FileInfo предназначен для организации доступа к физическому файлу, который находится на жестком диске компьютера, или любом другом съемном, или сетевом носителе, работа с которым осуществляется через файловую систему. Данный класс позволяет получать информацию об этом файле (например, о времени его создания, размере, атрибутах и т.п.), а также производить различные операции, например, создание файла, или его удаление. Класс FileInfo наследует члены класса FileSystemInfo и содержит дополнительный набор методов, который приведен в следующей таблице:

Метод	Описание
AppendText()	Создает объект StreamWriter для добавления текста к файлу.
CopyTo()	Копирует уже существующий файл в новый файл.
Create()	Создает новый файл и возвращает объект FileStream для взаимодействия с этим файлом.
CreateText()	Создает объект StreamWriter для записи текстовых данных в новый файл.
Delete()	Удаляет файл, которому соответствует объект FileInfo.
Directory	Возвращает каталог, в котором расположен данный файл.
DirectoryName	Возвращает полный путь к данному файлу в файловой системе.
Length	Возвращает размер файла.
MoveTo()	Перемещает файл в указанное место (этот метод позволяет одновременно переименовать данный файл).
Name	Позволяет получить имя файла.
Open()	Открывает файл с указанными правами доступа на чтение, запись, или совместное использование с другими программами.
OpenRead()	Создает объект FileStream, доступный только для чтения.
OpenText()	Создает объект StreamReader (о нем будет рассказано ниже), который позволяет считывать информацию из существующего текстового файла.

Метод	Описание
OpenWrite()	Создает объект FileStream, доступный для чтения и записи.

Как видно из таблицы, большинство методов FileInfo возвращают объекты (FileStream, StreamWriter, StreamReader и т.д.), которые позволяют различным образом взаимодействовать с файлом, например, производить чтение, или запись в него. Приемы работы с данными потоками нам уже известны.

Замечание

Перед выполнением примеров замените измененный каталог EXAMPLE на его копию.

Рассмотрим модификацию рекурсивного метода Show, который позволит нам получить полную информацию о структуре каталога EXAMPLE:

```
using System;
using System.IO;
namespace Example
{
   class Program
   {
      static void Show(DirectoryInfo dir)
      {
         Console.WriteLine("***** {0} *****", dir.Name);
         Console.WriteLine("FullName: {0}", dir.FullName);
         Console.WriteLine();
         FileInfo[] files = dir.GetFiles();
         if (files.Length != 0)
         {
            Console.WriteLine("Найдено файлов: {0}\n",
                                    files.Length);
            foreach (FileInfo item in files)
            {
               Console.WriteLine("File: {0}        Size: {1}",
                                    item.Name, item.Length);
            }
         }
         Console.WriteLine();
         DirectoryInfo[] subDirects = dir.GetDirectories();
         if (subDirects.Length != 0)
         {
            Console.WriteLine("Найдено подкаталогов: {0}",
                                   subDirects.Length);
            foreach (DirectoryInfo item in subDirects)
            {
               Show(item);
            }
         }
      }
      static void Main()
      {
         DirectoryInfo dir = new DirectoryInfo(@"d:\EXAMPLE");
            Show(dir);
```

EPAM Systems

```
        }
      }
   }
```

Результат работы программы:

```
***** EXAMPLE *****
FullName: d:\EXAMPLE
Найдено подкаталогов: 2
*****LETTER*****
FullName: d:\EXAMPLE\LETTER
Найдено подкаталогов: 2
*****2009*****
FullName: d:\EXAMPLE\LETTER\2009
Найдено файлов: 2
File: letter.doc   Size: 24064
File: letter.txt   Size: 77
*****2010*****
FullName: d:\EXAMPLE\LETTER\2010
Найдено файлов: 1
File: letter.txt   Size: 58

*****PUCTURE*****
FullName: d:\EXAMPLE\ PUCTURE
File: pucture.bmp     Size: 18454
File: pucture.jpg     Size: 11448
```

Следует отметить, что метод GetFile позволяет не просто получить коллекцию файлов в текущем каталоге, но произвести фильтрацию по заданной маске аналогично тому, как это делалось для метода GetDirectory.

Задание

Разработать метод, позволяющий осуществлять поиск файлов по заданной маске

Рассмотрим пример, позволяющий создать новый файл и записать в него некоторую информацию:

```
static void Main()
{
   //создаем новый файл  и связываем с ним строковый поток
   FileInfo f = new FileInfo(@"d:\EXAMPLE\text.txt");
   //записываем в файл данные и закрываем строковый поток,
   //при этом связь с физическим файлом для f не рвется
   using (StreamWriter fOut = new StreamWriter(f.Create()))
   {
      fOut.WriteLine("ОДИН ДВА ТРИ...");
   }
   //выводим информацию о созданном файле
   Console.WriteLine("Create file: {0}        Size: {1}",
                              f.Name, f.Length);
```

EPAM Systems

```
        using (StreamReader fIn = new StreamReader(f.OpenRead()))
        {
           Console.WriteLine("Text: {0}", fIn.ReadToEnd());
        }
     }
```

Результат работы программы:

```
Create file: text.txt Size: 27
Text: ОДИН, ДВА, ТРИ…
```

Рассмотрим пример, позволяющий копировать, переносить и переименовывать файл:

```
static void Main()
{
   FileInfo f = new FileInfo(@"d:\EXAMPLE\text.txt");
   f.CopyTo(@"d:\EXAMPLE\LETTER\text.txt");   //копируем файл
   f.MoveTo(@"d:\EXAMPLE\LETTER\2009\text.txt"); //переносим файл
   f = new FileInfo(@"d:\EXAMPLE\LETTER\2009\letter.doc");
   //копируем и переименовываем файл
   f.CopyTo(@"d:\EXAMPLE\LETTER\2010\oldLetter.doc");
   //удаяем файл
   f.Delete();
   DirectoryInfo dir = new DirectoryInfo(@"d:\EXAMPLE");
   Show(dir); //выводим информацию о структуре каталога
}
```

Результат работы программы:

```
***** EXAMPLE *****
FullName: d:\EXAMPLE
Найдено подкаталогов: 2
*****LETTER*****
FullName: d:\EXAMPLE\LETTER
Найдено файлов: 1
File: text.txt    Size: 27
Найдено подкаталогов: 2
*****2009*****
FullName: d:\EXAMPLE\LETTER\2009
Найдено файлов: 2
File: letter.txt   Size: 77
File: text.txt       Size: 27
*****2010*****
FullName: d:\EXAMPLE\LETTER\2010
Найдено файлов: 2
File: letter.txt   Size: 58
File: oldLetter.txtSize: 24064
*****PUCTURE*****
FullName: d:\EXAMPLE\ PUCTURE
File: pucture.bmp  Size: 18454
File: pucture.jpg  Size: 11448
```

ERAM Systems

Задание

Посмотрите, что произойдет, если запустить эту программу еще раз. Дайте объяснение полученным результатам.

Класс File

Доступ к физическим файлам можно получать и через статические методы класса File. Функциональные возможности данного класса во многом совпадают с возможностями класса FileInfo, но так как члены данного класса статические, то для их использования нет необходимости создавать экземпляр класса.

Задание

Самостоятельно изучите методы класса File.

Практикум №19

Выполните следующие действия программным путем.

1. В папке C:\temp создайте папки К1 и К2.
2. В папке К1:
   a) создайте файл t1.txt, в который запишите следующий текст:

   Иванов Иван Иванович, 1965 года рождения, место жительства г. Саратов

   b) создайте файл t2.txt, в который запишите следующий текст:

   Петров Сергей Федорович, 1966 года рождения, место жительства г.Энгельс
3. В папке К2 создайте файл t3.txt, в который перепишите вначале текст из файла t1.txt, а затем из t2.txt.
4. Выведите развернутую информацию о созданных файлах.
5. Файл t2.txt перенесите в папку К2.
6. Файл t1.txt скопируйте в папку К2.
6. Папку К2 переименуйте в All, а папку К1 удалите.
7. Выведите полную информацию о файлах папки All.

EPAM Systems