TRABALHANDO COM ARQUIVOS

Prof. Alexandre Krohn

INTRODUÇÃO

Toda entrada e saída de dados em .NET envolve a utilização de ***Streams***

Stream é uma abstração de um dispositivo serial

- Arquivo de disco
- Canal de rede
- Local de Memória
- ...qualquer dispositivo que permita leitura, escrita ou ambos de forma serial (sequencialmente)

SYSTEM.IO

Contém as classes para manipulação de arquivos e diretórios

Contém os tipos que habilitam a escrita e leitura de *streams* de arquivos e dados

Para utilizar as funções de manipulação de arquivos, adicione :

```
using System.IO;
```

Ao início do seu código fonte.

SYSTEM.IO

Classe	Descrição
Directory	Expõe Estático métodos para Criar, mover e navegar através de pastas e subpastas. Esta classe não pode ser herdada.
DirectoryInfo	Expõe como exemplo métodos de instância para criar, mover, e enumerar através de pastas e subpastas. Esta classe não pode ser herdada.
DriveInfo	Fornece acesso às informações em uma unidade.
File	Fornece métodos estáticos para a criação, cópia, exclusão, movimentação e a Abrindo de arquivos e ajuda na criação de FileStream objetos.
FileInfo	Fornece propriedades e métodos de instância para criar, copiar, excluir, mover, e abrir arquivos, e ajuda na criação de objetos de FileStream. Esta classe não pode ser herdada.
BinaryReader	Lê tipos de dados primitivos como valores binários em uma codificação específica.
BinaryWriter	Grava tipos primitivos em binário em um fluxo e suporta cadeias de caracteres de escrita em uma codificação específica.

SYSTEM.IO

Classe	Descrição
Stream	Fornece uma forma genérica de ver uma sequência de bytes.
StreamReader	Implementa um TextReader que lê caracteres de um fluxo de bytes em uma codificação específica.
StreamWriter	Implementa TextWriter para escrever caracteres em um fluxo em uma determinada codificação.
TextReader	Representa um leitor pode ler uma série sequencial de caracteres.
TextWriter	Representa um gravador que possa escrever uma série sequencial de caracteres. Essa classe é abstrato.
DirectoryNotFoundException	A exceção que é lançada quando parte de um arquivo ou diretório não foi encontrado.
FileNotFoundException	A exceção que é lançada quando uma tentativa de acessar um arquivo que não existe no disco falha.
IOException	A exceção que é lançada quando ocorre um erro de e/S.

CLASSES FILE E DIRECTORY

Estas classes são utilitárias pois disponibilizam diversos métodos para manipular arquivos e diretórios

- Copiar, mover, renomear, criar, abrir, excluir e adicionar no final (append)

Os métodos são estáticos, isto é, não precisa instanciar um objeto para utilizá-los

- `File.Copy(“arquivo-origem”, “arquivo-destino”);`

CLASSE FILE

Método	Descrição
AppendAllText()	Acrescenta a cadeia de caracteres especificada no arquivo, criando o arquivo se ele ainda não existir.
AppendText()	Cria StreamWriter que anexa o texto codificado UTF-8 a um arquivo existente, ou um novo arquivo se o arquivo especificado não existe.
Copy()	Copia um arquivo existente para um novo arquivo. Substitua um arquivo de mesmo nome não é permitido.
Create()	Cria ou substitui um arquivo no caminho especificado.
Delete()	exclui o arquivo especificado.
Exists()	determina se o arquivo especificado existe.
GetCreationTime()	Retorna a data e hora de criação de arquivo ou diretório especificado.
GetLastAccessTime()	Retorna a data e hora o arquivo ou diretório especificado foi acessado pela última vez.
GetLastWriteTime()	Retorna a data e hora o arquivo ou diretório especificado eram último gravados.
Move()	Move o arquivo especificado para um outro local, com opção de alterar o nome do mesmo.

CLASSE FILE

Verifica se arquivo existe

```
if (File.Exists(“teste.bmp”))
{
…
}
```

CLASSE DIRECTORY

Método	Descrição
CreateDirectory(String)	Cria todos os diretórios e subdiretórios no caminho especificado.
Delete(String)	Exclui um diretório vazio de um caminho especificado.
Exists	Determina se o caminho fornecido se refere a um diretório existente no disco.
GetCurrentDirectory	Obtém o diretório de trabalho corrente do aplicativo.
GetFiles(String)	Retorna os nomes dos arquivos (incluindo seus caminhos) no diretório especificado.
GetParent	Recupera o diretório pai do caminho especificado, incluindo caminhos absolutos e relativos.
Move	Mover um arquivo ou diretório e seu conteúdo para um novo local.

CLASSES FILEINFO E DIRECTORYINFO

Adicionam funcionalidades adicionais para as classes File e Directory

- Diferença – ambos devem ser instanciados
- Ambos possuem propriedades e construtores públicos
- Não podem ser herdados

```
DirectoryInfo dir = new DirectoryInfo(".");
String strDiretorio = “Diretorio Atual: “ +
 Directory.GetCurrentDirectory( ));
```

CLASSE DIRECTORYINFO

Método	Descrição
Name	Obtém o nome da instância de DirectoryInfo . (Substitui FileSystemInfo.Name.)
Parent	Obtém o diretório pai de um subdiretório especificado.
Root	Obtém a parte da raiz da pasta.
Exists	Obtém um valor indicando se o diretório existe. (Substitui FileSystemInfo.Exists.)
Extension	Obtém a cadeia de caracteres que representa a parte da extensão do arquivo. (Herdado de FileSystemInfo.)
FullName	Obtém o caminho completo do diretório ou arquivo. (Herdado de FileSystemInfo.)

STREAMS DE ARQUIVO

Existem diversas classes para manipular arquivos

- Stream, TextWriter, e TextReader

Classes Stream fornecem métodos genéricos para manipular entrada/saída

- Classe IO.Stream e derivadas – dados em nível de byte
- Classes IO.TextWriter e IO.TextReader – dados em formato texto
  - As classes StreamReader e StreamWriter são derivadas destas classes

ARQUIVOS TEXTO

Classe StreamWriter para escrever arquivos textos

- Métodos Write( ) e WriteLine( )

Classe StreamReader para ler arquivos textos

- Métodos Read( ) e ReadLine( )

Namespace System.IO

ARQUIVOS TEXTO

```
StreamWriter arqSaida = new
 StreamWriter(“nomearquivosaida.txt");


StreamReader arqEntrada = new
 StreamReader(“nomearquivoentrada.txt");
```

arqEntrada e arqSaida representam os objetos streams de arquivo

- Nomes deles são nomearquivoentrada.txt e nomearquivosaida.txt, respectivamente

ARQUIVOS TEXTO

Utilize Write( ) ou WriteLine( ) para a escrita

```
arqSaida.WriteLine(“1ª Linha do Arquivo");
```

- O WriteLine() grava o valor passado como parâmetro seguido por um terminador de linha (enter)
- Utilizar o WriteLine( ) sem parâmetro grava um terminador de linha (enter)
- Erros: stream fechada ou erro de E/S

ARQUIVOS TEXTO

Utilize Read( ) ou ReadLine( ) para a leitura

```
string str = arqEntrada.ReadLine();
```

- ReadLine() lê uma linha de caracteres da stream e retorna os dados como uma string (retorna null se o fim de arquivo é lido)
- Read() lê o próximo caractere da stream e avança a posição em um caractere (o próximo caractere da stream é representado como um objeto Int32 ou -1 se não houver mais caracteres disponíveis)
- Erros: falta de memória ou erro de E/S

Utilize Close() para fechar o arquivo

```
arqSaida.Close();
arqEntrada.Close();
```

ARQUIVOS TEXTO

```
using System;
using System.IO;

namespace LeArquivo
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            StreamReader arquivo;
            string linha;
            if (File.Exists("arquivo.txt"))
            {
                try
                {
                    arquivo = new StreamReader("arquivo.txt");
                    while ((linha = arquivo.ReadLine()) != null)
                    {
                        Console.WriteLine(linha);
                    }
                }
                catch (Exception e)
                {
                    Console.WriteLine("Erro ao abrir o arquivo de entrada");
                }
            }
        }
    }
}
```

ARQUIVOS TEXTO

Trabalhando com arquivos delimitados

```
string [] tokens;
char [] separadores = new char[] {‘;’};

while ((linha = arquivo.ReadLine()) != null)
{
   tokens = linha.Split(separadores);
   for (int i = 0; i < tokens.Length; i++)
   {
     ...
   }
}
```

ARQUIVOS BINÁRIOS

Classes

BinaryReader : leitura de arquivo binário

BinaryWriter : gravação de arquivo binário

ARQUIVOS BINÁRIOS

Classe BinaryWriter para escrever arquivos textos

- Método Write( )

Classe BinaryReader para ler arquivos textos

- Métodos ReadBoolean() , ReadByte(), ReadBytes(int count), ReadChar() , ReadChars(int count) , ReadDouble(), ReadInt32() , ReadInt64(), ReadString()

Namespace System.IO

ARQUIVOS BINÁRIOS

A classe de BinaryReader fornece diversos métodos que simplificam ler tipos de dados primitivos de um fluxo

```
BinaryReader reader = new
BinaryReader(File.Open(“dado.bin”,
FileMode.Open)));


float umFloat = reader.ReadSingle();

string umaString = reader.ReadString();

int umInt = reader.ReadInt32();

bool umBool = reader.ReadBoolean();
```

ARQUIVOS BINÁRIOS

Método	Descrição
ReadByte	Ler o byte seguir de fluxo atual e avança a posição atual de fluxo por um byte.
ReadChar	Ler o próximo caractere de fluxo atual e avança a posição atual do fluxo de acordo com **Encoding** usado e o caractere específico que está sendo lido de fluxo.
ReadDouble	Ler um valor de ponto flutuante de 8 bytes de fluxo atual e avança a posição atual de fluxo por oito bytes.
ReadInt16	Ler um inteiro com sinal de 2 bytes de fluxo atual e avança a posição atual de fluxo por dois bytes.
ReadInt32	Ler um inteiro com sinal de 4 bytes de fluxo atual e avança a posição atual de fluxo por quatro bytes.
ReadInt64	Ler um inteiro com sinal de 8 bytes de fluxo atual e avança a posição atual de fluxo por oito bytes.
ReadBoolean	Ler um valor de **Boolean** de fluxo atual e avança a posição atual de fluxo por um byte.

ARQUIVOS BINÁRIOS

A classe de BinaryWriter fornece métodos que simplificam escrever tipos de dados primitivos de um fluxo

- Método Write() é sobrecarregado

```
BinaryWriter arqSaida = new
BinaryWriter(File.Open(“dado.bin”,FileMode.Create)));

arqSaida.Write(1.250F);

arqSaida.Write(@"c:\Temp");

arqSaida.Write(10);

arqSaida.Write(true);
```

ENUM FILEMODE

Membro	Descrição
Append	Abre o arquivo, caso ele exista, e busca o final do arquivo ou cria um novo arquivo.
Create	Especifica que o sistema operacional deve criar um novo arquivo. Se o arquivo já existir, ele será substituído.
CreateNew	Especifica que o sistema operacional deve criar um novo arquivo.
Open	Especifica que o sistema operacional deve abrir um arquivo existente.
OpenOrCreate	Especifica que o sistema operacional deverá abrir um arquivo, se ele existir; caso contrário, um novo arquivo deverá ser criado.
Truncate	Especifica que o sistema operacional deve abrir um arquivo existente. Quando o arquivo é aberto, ele deve ser truncado para que seu tamanho seja de zero byte.

DECLARAÇÃO USING

A palavra-reservada ***using*** é utilizada para definir um escopo para um objeto

- O CLR automaticamente libera o recurso quando o objeto sair do escopo

- Útil quando trabalhando com arquivos ou banco de dados
  - Quando escrevendo em um arquivo, os dados não são armazenados diretamente no arquivo até que ele seja fechado
    - Falha ao fechar o arquivo resultará em um arquivo vazio

DECLARAÇÃO *USING*

Objeto StreamReader é definido e instanciado dentro utilizando um bloco

Instanciando o objeto inFile aqui faz com que ele exista somente neste bloco

É garantido que o arquivo é fechado ao sair do bloco

```
try
{
   using(StreamReader inFile = new
                              StreamReader(“nomes.txt”))
  {
    while ((inValue = inFile.ReadLine()) != null)
    {
      …
     }
   }
}
```

PROJETO

Arquivos

PROBLEMA

Um certo equipamento é instrumentado com 6 células de carga. Diversas forças são aplicadas neste equipamento, as quais são captadas pelas células de carga. Estas células de carga funcionam 24 horas por dia e devem ser monitoradas a fim de garantir que o equipamento não esteja sofrendo nenhuma força adicional que venha afetar o seu funcionamento.

Os dados das células de carga são gravados em um arquivo com a data e a hora (incluindo milissegundo) de cada leitura dos sensores.

O objetivo do programa é extrair estimar a soma das células de carga e gravar em um segundo arquivo

ARQUIVO DE DADOS

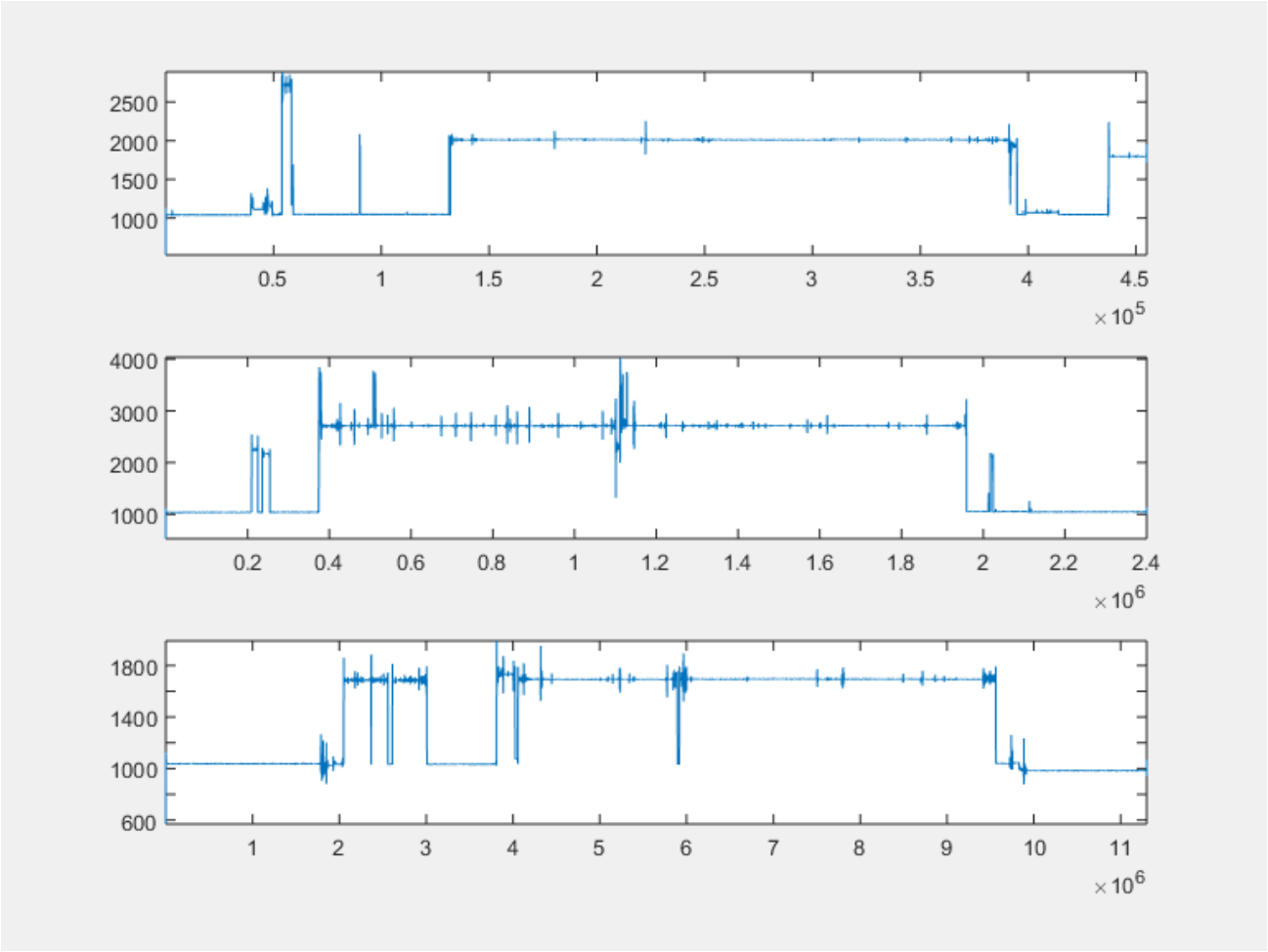
Arquivo texto

- **data**: dia da coleta no formato aaaammdd, onde aaaa é o ano com 4 dígitos, mm é o mês com 2 dígitos e dd é o dia com 2 dígitos. Por exemplo, 10/11/2008 seria representado 20081110
- **hora**: hora no formato hhMMssmmm, onde hh é a hora com 2 dígitos, MM é o minuto com 2 dígitos, ss é o segundo com 2 dígitos e mmm é o milissegundo com 3 dígitos. Por exemplo, a hora 12hs13m14s345ms seria representado como 121314345.
- **dado1**: é a força calculada na célula de carga 1
- **dado2**: é a força calculada na célula de carga 2
- **dado3**: é a força calculada na célula de carga 3
- **dado4**: é a força calculada na célula de carga 4
- **dado5**: é a força calculada na célula de carga 5
- **dado6**: é a força calculada na célula de carga 6

ARQUIVO DE DADOS

```
20090112 184715528  81.8  74.5  85.9  72.7 128.9  78.8
20090112 184715628 177.0 161.2 186.0 157.4 279.0 170.6
20090112 184715728 155.6 141.7 163.5 138.3 245.3 150.0
20090112 184715828 167.1 152.3 175.6 148.6 263.5 161.2
20090112 184715928 160.0 145.8 168.1 142.3 252.2 154.3
20090112 184716028 164.5 149.9 172.9 146.3 259.5 158.7
20090112 184716128 161.6 147.3 169.9 143.7 254.9 155.9
20090112 184716228 163.4 148.9 171.7 145.3 257.7 157.6
20090112 184716328 162.4 148.0 170.7 144.4 256.1 156.6
20090112 184716428 162.9 148.4 171.2 144.9 256.9 157.1
20090112 184716528 162.7 148.2 171.0 144.7 256.5 156.9
20090112 184716628 162.7 148.2 171.0 144.7 256.5 156.9
20090112 184716728 162.7 148.2 171.0 144.7 256.5 156.9
20090112 184716828 162.7 148.2 171.0 144.7 256.5 156.9
20090112 184716928 162.7 148.2 171.0 144.7 256.6 156.9
```

SOMA DAS CÉLULAS DE CARGA



ARQUIVOS COM PONTO DECIMAL

Ao tentar utilizar o método Parse() em string com ponto decimal em um computador configurado para vírgula decimal, o ponto decimal é ignorado

```
float dado = float.Parse("4.5");
```

No comando acima, dado recebe 45.

Para converter corretamente, deve-se utilizar o seguinte comando

```
float dado = float.Parse("4.5",
CultureInfo.InvariantCulture.NumberFormat);
```

CultureInfo está definido em System.Globalization

```
using System.Globalization;
```