Prática 7
(Persistência de Dados)

Data de Entrega: 27/11/2017

Valor Trabalho: 3 pontos

Leia antes de começar:

1. A prática deve ser feita individualmente, pode conversar com o colega, mas cada um faz o seu.

2. Todos exercícios devem ser entregues resolvidos até a próxima aula prática.

3. A prática deve ser postada no iLang. O arquivo final deve ter todos os arquivos de código-fonte dos exercícios compactados em formato **ZIP** com os nomes do(a) aluno(a) no seguinte formato.

   Exemplo: **Pratica7-JoaoDeDeus.zip**

4. Caso seja entregue atrasada, o valor da prática será reduzido de acordo com o número de dias de atraso (logo o aluno tem até 24hs para postar o trabalho sem ser penalizado) segundo a seguinte regra:

$$valor\ trabalho = \frac{valor\ trabalho}{numero\ dias\ atraso * 2}$$

5. Trabalhos idênticos serão considerados cópias, logo, suas avaliações serão anuladas.

6. Todos os arquivos devem ter um comentário de cabeçalho contendo:

```
/* Nome do Aluno:  Fulano de Tal
 * RA:
 * Nome do Programa:
 * Data:
 */
```

UNIDADE 1: ESCREVENDO E LENDO EM ARQUIVOS

Texto na integra de: https://www.caelum.com.br/apostila-csharp-orientacao-objetos/system-io/

PARTE 1: LENDO UM ARQUIVO

A entrada de dados no C# funciona em duas etapas. Na primeira etapa, temos uma classe abstrata que representa uma sequência de bytes na qual podemos realizar operações de leitura e escrita. Essa classe abstrata é chamada de Stream.

Como o Stream é uma classe abstrata, não podemos usá-la diretamente, precisamos de uma implementação para essa classe. No caso de leitura ou escrita em arquivos, utilizamos um tipo de Stream chamado FileStream, que pode ser obtido através do método estático Open da classe File. Quando utilizamos o Open, devemos passar o nome do arquivo que será aberto e devemos informá-lo o que queremos fazer com o arquivo (ler ou escrever).

Para abrirmos o arquivo entrada.txt para leitura, utilizamos o código a seguir:

```
Stream entrada = File.Open("entrada.txt", FileMode.Open);
```

Agora que temos o Stream, podemos ler seu próximo byte utilizando o método ReadByte.

```
byte b = entrada.ReadByte();
```

Porém, trabalhar com bytes não é fácil. Para facilitar a leitura de Streams, o C# nos oferece uma classe chamada StreamReader, responsável por ler caracteres ou strings de um Stream. O StreamReader precisa saber qual é a Stream que será lida, portanto passaremos essa informação através de seu construtor:

```
StreamReader leitor = new StreamReader(entrada);
```

Para ler uma linha do arquivo, utilizamos o método ReadLine do

StreamReader:

```
string linha = leitor.ReadLine();
```

Enquanto o arquivo não terminar, o método ReadLine() devolve um valor diferente de nulo, portanto, podemos ler todas as linhas de um arquivo com o seguinte código:

```
string linha = leitor.ReadLine();
while(linha != null)
{
    MessageBox.Show(linha);
    linha = leitor.ReadLine();
}
```

Assim que terminamos de trabalhar com o arquivo, devemos sempre lembrar de fechar o Stream e o StreamReader:

```
leitor.Close();

entrada.Close();
```

O código completo para ler de um arquivo fica da seguinte forma:

```
Stream entrada = File.Open("entrada.txt", FileMode.Open);
StreamReader leitor = new StreamReader(entrada);
string linha = leitor.ReadLine();
while(linha != null)
{
    MessageBox.Show(linha);
    linha = leitor.ReadLine();
}
leitor.Close();
entrada.Close();
```

Porém, o arquivo pode não existir e, nesse caso, o C# lança a FileNotFoundException. Devemos, portanto, verificar se o arquivo existe antes de abri-lo para leitura. Podemos verificar se um arquivo existe utilizando o método Exists da classe File:

```
if(File.Exists("entrada.txt"))
{
    // Aqui temos certeza que o arquivo existe
}
```

O código da leitura com a verificação fica assim:

```
if(File.Exists("entrada.txt"))
{
    Stream entrada = File.Open("entrada.txt", FileMode.Open);
    StreamReader leitor = new StreamReader(entrada);
    string linha = leitor.ReadLine();
    while(linha != null)
    {
        MessageBox.Show(linha);
        linha = leitor.ReadLine();
    }
    leitor.Close();
    entrada.Close();
}
```

PARTE 2: ESCREVENDO EM UM ARQUIVO

Assim como a leitura, a escrita também acontece em duas etapas. Na primeira etapa, trabalhamos novamente escrevendo bytes para a saída. Para isso utilizaremos novamente a classe abstrata Stream.

Para escrevermos em um arquivo, precisamos primeiro abri-lo em modo de escrita utilizando o método Open do File passando o modo FileMode.Create:

```
Stream saida = File.Open(“saida.txt”, FileMode.Create);
```

Porém, não queremos trabalhar com Bytes, então utilizaremos uma classe especializada em escrever em um Stream chamada StreamWriter.

```
StreamWriter escritor = new StreamWriter(saida);
```

Podemos escrever uma linha com o StreamWriter utilizando o método WriteLine:

```
escritor.WriteLine("minha mensagem");
```

Depois que terminamos de utilizar o arquivo, precisamos fechar todos os recursos:

```
escritor.Close();

saida.Close();
```

O código completo para escrever no arquivo fica da seguinte forma:

```
Stream saida = File.Open("saida.txt", FileMode.Create);
StreamWriter escritor = new StreamWriter(saida);
escritor.WriteLine("minha mensagem");
escritor.Close();
saida.Close();
```

Repare que, por usarmos uma classe abstrata, podemos então trocar facilmente a classe concreta por outra. Por exemplo, poderíamos ler de um Socket, ou de uma porta serial, e o código seria o mesmo: basta a classe ser filha de Stream. Repare que o uso de classes abstratas e polimorfismo nos possibilita ler/escrever em diferentes lugares com o mesmo código. Veja que a própria Microsoft fez bom uso de orientação a objetos para facilitar a vida dos desenvolvedores.

PARTE 3: SEPARANDO AS PARTES DE UMA STRING USANDO O MÉTODO SPLIT

Para separar as partes de uma String podemos utilizar o método Split da própria classe String. Esse método permite que se divida um texto com base em um caractere separador como, por exemplo, o caractere ";":

```
String value = "This is a short string.";

Char delimitador = ';';

String[] substrings = value.Split(delimitador);
```

Conforme o exemplo acima, deve-se informar qual a String a ser separada (value) e qual o caractere separador (delimitador).

Além disso pode-se, através da propriedade *Length*, recuperar a quantidade de palavras encontradas à paritr da divisão da string e recuperar, uma a uma as palavras separadas através da sua posição no array.

```
int count = substrings.Length;
for (int i = 0; i < count; i++)
{
     String palavra = substrings[i];
    MessageBox.Show(palavra);
}
```

Atividades

1. **ESCRITA SIMPLES:** Faça um programa em Java que escreva em um arquivo chamado "Exercicio1.txt" o texto "Primeiro Texto Escrito".

2. **ESCRITA SEQUENCIAL:** Faça um programa em Java que escreva em um arquivo chamado "Exercicio2.txt" uma sequência de textos sequenciais de um até 100 sendo que cada texto é uma linha no arquivo como no exemplo abaixo.

   Texto 1
   Texto 2
   Texto 3
   Texto 4
   Texto 5
   ...
   Texto 100

3. **ESCRITA COM ENTRADA DE DADOS:** Faça um programa em Java que leia o nome, o sobrenome e a idade digitados pelo usuário e os escreva-os e uma linha do arquivo separados pelo caracter ";". Como no exemplo abaixo:

   Flavio;Calado;33;

4. **CRIAÇÂO DE CLASSE:** crie uma classe que represente uma conta bancária com número da conta, agência e saldo. Faça um programa leia os dados de uma conta bancária digitados pelo usuário e armazene-os em um objeto da classe conta.

5. **ESCRITA COM ENTRADA DE DADOS HETEROGENEOS:** Faça um programa leia os dados de uma conta bancária digitada pelo usuário, armazene-os em um objeto da classe conta e os escreva-os e uma linha do arquivo separados pelo caracter ";".

6. **ESCRITA COM ENTRADA DE DADOS HETEROGENEOS SEQUENCIAIS:** Faça um programa leia os dados de várias contas bancárias digitadas pelo usuário, armazene-os em um objeto da classe conta e os escreva cada conta uma linha do arquivo com seus campos separados pelo caracter ";".

7. **LEITURA SEGUENCIAL:** Faça um programa em Java que leia todo o arquivo do Exercício 2 desta lista do início ao fim e imprima-o na tela

8. **SEPARAÇAO DE PALAVRAS:** Faça um programa que leia uma frase digitada pelo usuário e imprima, na tela, cada palavra em uma linha.

9. **LEITURA POR PALAVRA:** Faça um programa que leia o arquivo do Exercício 5 e imprima cada campo na tela separado pelo caractere " - ". Exemplo:
   Linha no arquivo: "345030;2309;100.00"
   Impresso na tela: 345030 - 2309 - 100.00

10. **LEITURA SELETIVA:** Faça um programa que o usuário digite o número de uma conta bancária, o programa percorra o arquivo do exercício 6 e imprima somente a linha que tiver o número da conta digitada.