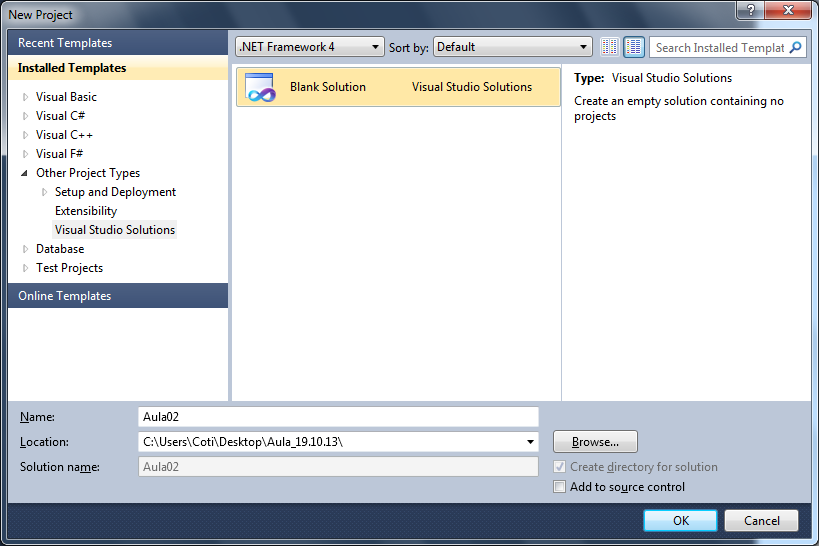
**Criação da Solution**

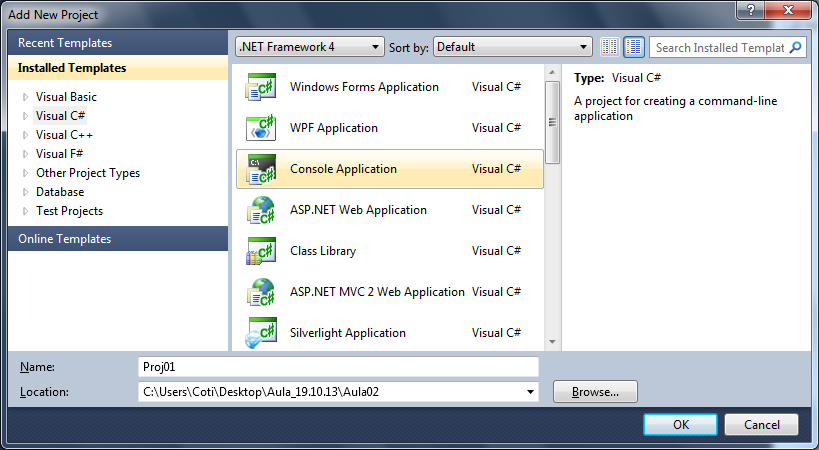
Ambiente de trabalho



**Criação do Projeto**

Linguagem: C#

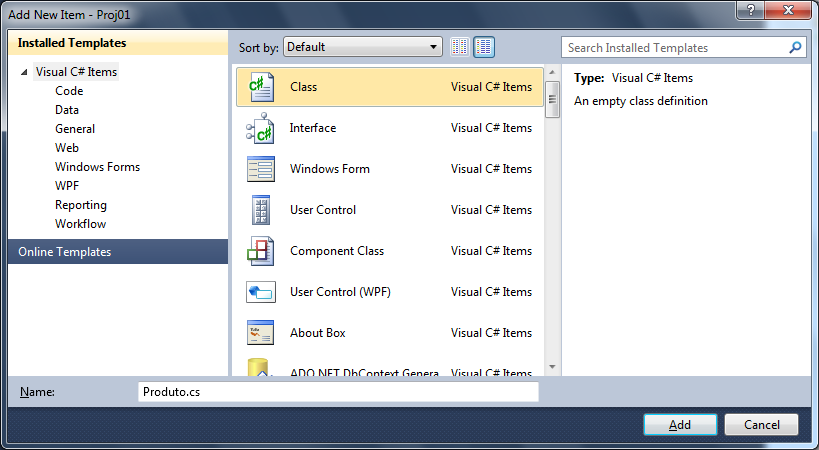
Projeto: Console Application (Prompt de comando 🡪 DOS)



Modelagem de Classes

Classes voltadas para representação de entidades de um sistema:

* Produto
  + Atributos privados
  + Construtores
    - Default (vazio)
    - Entrada de dados
  + Propriedades (Encapsulamento)



using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

//localização da Classe dentro do projeto

namespace Proj01.Model

{

//declaração da Classe

public class Produto

{

//Atributos (dados)

//private -> acesso somente é permitido dentro da própria Classe

private int idProduto;

private string nome;

private double preco;

//[ctor] + 2x[tab]

public Produto()

{

//Construtor default (vazio)

}

//Construtor com entrada de parâmetros

//Sobrecarga de Métodos (Overloading)

public Produto(int idProduto, string nome, double preco)

{

this.idProduto = idProduto;

this.nome = nome;

this.preco = preco;

}

//Encapsulamento

public int IdProduto

{

set { idProduto = value; } //entrada

get { return idProduto; } //saída

}

public string Nome

{

set { nome = value; }

get { return nome; }

}

public double Preco

{

set { preco = value; }

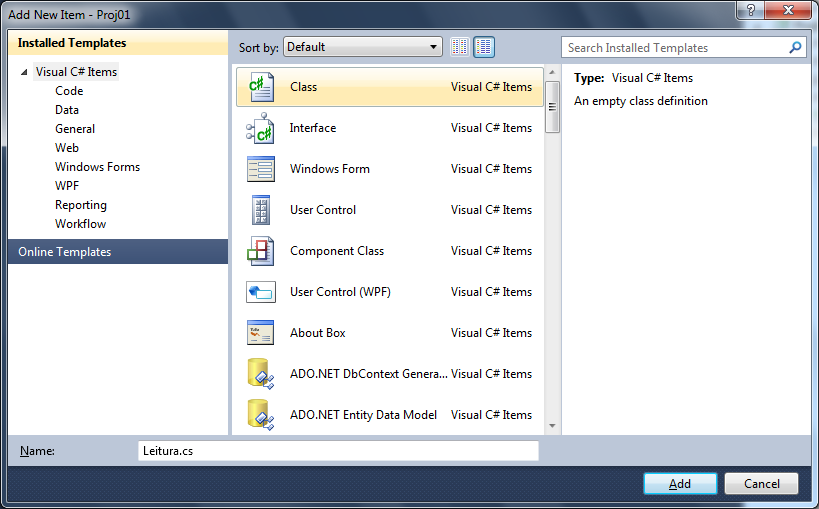
get { return preco; }

}

}

}

Classe para realizar leitura   
de dados para Produto...



using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Proj01.Input

{

public class Leitura

{

//Método para ler o id de um produto informado pelo usuário

public int LerIdProduto()

{

Console.Write("Informe o Id do Produto.....: ");

int idproduto = Convert.ToInt32(Console.ReadLine());

return idproduto;

}

//Método para ler o nome de um produto informado

//pelo usuário no prompt do DOS

public string LerNome()

{

//Impressão do prompt do DOS [cw] + 2x[tab]

Console.Write("Informe o Nome do Produto...: ");

string nome = Console.ReadLine(); //Leitura

//retornar a variável

return nome;

}

//Método para ler o preço de um produto informado pelo usuário

public double LerPreco()

{

Console.Write("Informe o Preco do Produto..: ");

double preco = Convert.ToDouble(Console.ReadLine());

return preco;

}

}

}

Testando...

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using Proj01.Input; //importando o conteudo do namespace

using Proj01.Model; //importando o conteudo do namespace

namespace Proj01

{

class Program

{

static void Main(string[] args)

{

//Objeto para a Classe Produto

Produto p = new Produto();

Leitura l = new Leitura();

p.IdProduto = l.LerIdProduto();

p.Nome = l.LerNome();

p.Preco = l.LerPreco();

//Impressão

Console.WriteLine("\nDados do Produto:");

Console.WriteLine("\tCódigo....: " + p.IdProduto);

Console.WriteLine("\tNome......: " + p.Nome);

Console.WriteLine("\tPreco.....: " + p.Preco);

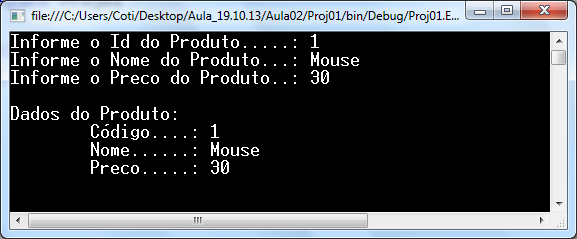
Console.ReadKey(); //Pausar

}

}

}

Resultado...



Erros em tempo de compilação

São aqueles que impedem que o programa seja compilado pela IDE. Geralmente, erros de sintaxe, lógica, etc... escrita de código como um todo.

Erros em tempo de execução

São aqueles evidenciados durante a execução do programa e que podem ser tratados por programação. Estes erros geram exceções uma aplicação.

try / catch

Tentativa e captura da exceção

Toda exceção tratada em C# pode ser representada pela Classe Exception (System)

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Proj01.Input

{

public class Leitura

{

//Método para ler o id de um produto informado pelo usuário

public int LerIdProduto()

{

try //tentativa

{

Console.Write("Informe o Id do Produto.....: ");

int idproduto = Convert.ToInt32(Console.ReadLine());

return idproduto;

}

catch(Exception e) //captura da exceção

{

Console.WriteLine("Erro: " + e.Message);

return LerIdProduto(); //Recursividade

}

}

//Método para ler o nome de um produto informado

//pelo usuário no prompt do DOS

public string LerNome()

{

try

{

//Impressão do prompt do DOS [cw] + 2x[tab]

Console.Write("Informe o Nome do Produto...: ");

string nome = Console.ReadLine(); //Leitura

//retornar a variável

return nome;

}

catch(Exception e)

{

Console.WriteLine("Erro: " + e.Message);

return LerNome(); //Recursividade

}

}

//Método para ler o preço de um produto informado pelo usuário

public double LerPreco()

{

try

{

Console.Write("Informe o Preco do Produto..: ");

double preco = Convert.ToDouble(Console.ReadLine());

return preco;

}

catch(Exception e)

{

Console.WriteLine("Erro: " + e.Message);

return LerPreco(); //Recursividade

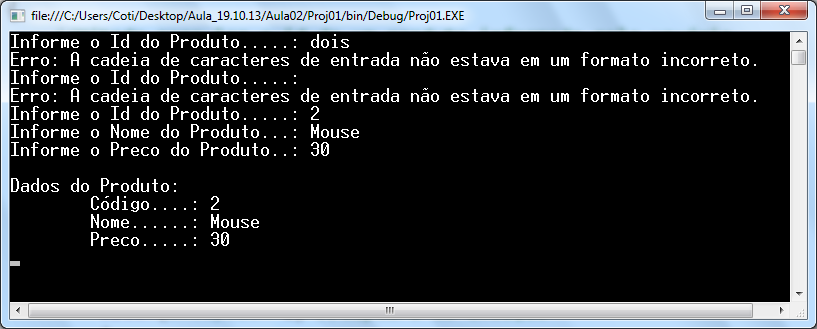
}

}

}

}

Executando...



Criando regras de Validação

* IdProduto deverá ser um numero inteiro maior que zero.
* Nome deverá ser do tipo texto contendo:
  + Letras e números apenas
  + Espaços
  + 3 a 30 caracteres
* Preço deverá ser num numero decimal maior que zero e menor que 10000.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Proj01.Input

{

public class Leitura

{

//Método para ler o id de um produto informado pelo usuário

public int LerIdProduto()

{

try //tentativa

{

Console.Write("Informe o Id do Produto.....: ");

int idproduto = Convert.ToInt32(Console.ReadLine());

if (idproduto > 0) //regra de validação

{

return idproduto;

}

else //gera um fluxo de exceção baseado em uma regra de validação

{

//gerar uma exceção (redirecionar para o catch)

**throw new Exception("Id Produto deve ser**

**maior que zero.");**

}

}

catch(Exception e) //captura da exceção

{

Console.WriteLine("Erro: " + e.Message);

return LerIdProduto(); //Recursividade

}

}

//Método para ler o nome de um produto informado

//pelo usuário no prompt do DOS

public string LerNome()

{

try

{

//Impressão do prompt do DOS [cw] + 2x[tab]

Console.Write("Informe o Nome do Produto...: ");

string nome = Console.ReadLine(); //Leitura

//retornar a variável

return nome;

}

catch(Exception e)

{

Console.WriteLine("Erro: " + e.Message);

return LerNome(); //Recursividade

}

}

//Método para ler o preço de um produto informado pelo usuário

public double LerPreco()

{

try

{

Console.Write("Informe o Preco do Produto..: ");

double preco = Convert.ToDouble(Console.ReadLine());

if(preco > 0 && preco <= 10000)

{

return preco;

}

else

{

**throw new Exception("Preco deve ser**

**maior que zero e até 10000.");**

}

}

catch(Exception e)

{

Console.WriteLine("Erro: " + e.Message);

return LerPreco(); //Recursividade

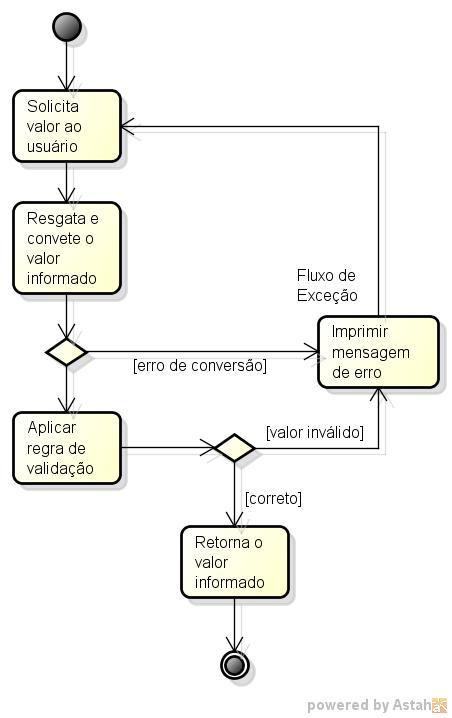
}

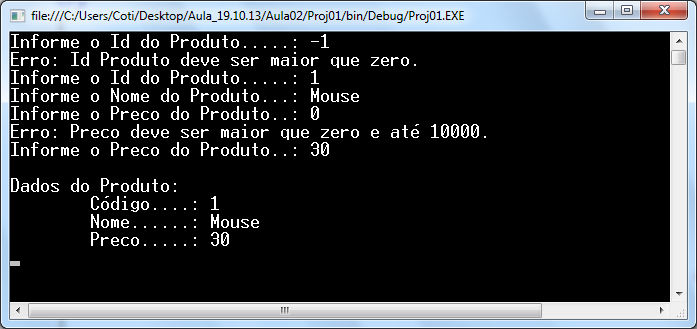
}

}

}

Fluxo em Diagrama   
de Atividade





Expressões Regulares (Regex)

São uma forma de verificar o conteúdo de uma cadeia de caracteres armazenado dentro de uma string.

* Nome deverá ser do tipo texto contendo:
  + Letras e números apenas
  + Espaços
  + 3 a 30 caracteres

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Text.RegularExpressions; //Regex

namespace Proj01.Input

{

public class Leitura

{

//Método para ler o id de um produto informado pelo usuário

public int LerIdProduto()

{

try //tentativa

{

Console.Write("Informe o Id do Produto.....: ");

int idproduto = Convert.ToInt32(Console.ReadLine());

if (idproduto > 0) //regra de validação

{

return idproduto;

}

else //gera um fluxo de exceção baseado em uma regra de validação

{

//gerar uma exceção (redirecionar para o catch)

throw new Exception("Id Produto deve ser maior que zero.");

}

}

catch(Exception e) //captura da exceção

{

Console.WriteLine("Erro: " + e.Message);

return LerIdProduto(); //Recursividade

}

}

//Método para ler o nome de um produto informado

//pelo usuário no prompt do DOS

public string LerNome()

{

try

{

//Impressão do prompt do DOS [cw] + 2x[tab]

Console.Write("Informe o Nome do Produto...: ");

string nome = Console.ReadLine(); //Leitura

//Criar a expressão regular

**Regex r = new Regex("^[A-Za-zÀ-Üà-ü**

**0-9\\s]{3,30}$");**

if(r.IsMatch(nome)) //passou na expressão (regex)

{

return nome;

}

else

{

throw new Exception("Nome Inválido.");

}

}

catch(Exception e)

{

Console.WriteLine("Erro: " + e.Message);

return LerNome(); //Recursividade

}

}

//Método para ler o preço de um produto informado pelo usuário

public double LerPreco()

{

try

{

Console.Write("Informe o Preco do Produto..: ");

double preco = Convert.ToDouble(Console.ReadLine());

if(preco > 0 && preco <= 10000)

{

return preco;

}

else

{

throw new Exception("Preco deve ser maior que

zero e até 10000.");

}

}

catch(Exception e)

{

Console.WriteLine("Erro: " + e.Message);

return LerPreco(); //Recursividade

}

}

}

}

Entendendo a expressão...

^[A-Za-zÀ-Üà-ü0-9\\s]{3,30}$

^ Marca o inicio da expressão

$ Marca o final da expressão

[] Define os caracteres permitidos

{} Define o tamanho máximo e/ou mínimo

A-Z Somente letras do alfabeto em caixa-alta

a-z Somente letras do alfabeto em caixa-baixa

À-Ü Somente caracteres acentuados em caixa-alta

à-ü Somente caracteres acentuados em caixa-baixa

0-9 Digitos numéricos

\\s Espaços em branco.

.Trim()

Limpa os espaços no inicio e no final de uma string

.TrimStart()

Limpa os espaços somente no inicio de uma string

.TrimEnd()

Limpa os espaços somente no final de uma string

public string LerNome()

{

try

{

//Impressão do prompt do DOS [cw] + 2x[tab]

Console.Write("Informe o Nome do Produto...: ");

**string nome = Console.ReadLine().Trim(); //Leitura**

//Criar a expressão regular

Regex r = new Regex("^[A-Za-zÀ-Üà-ü0-9\\s]{3,30}$");

if(r.IsMatch(nome)) //passou na expressão (regex)

{

return nome;

}

else

{

throw new Exception("Nome Inválido.");

}

}

catch(Exception e)

{

Console.WriteLine("Erro: " + e.Message);

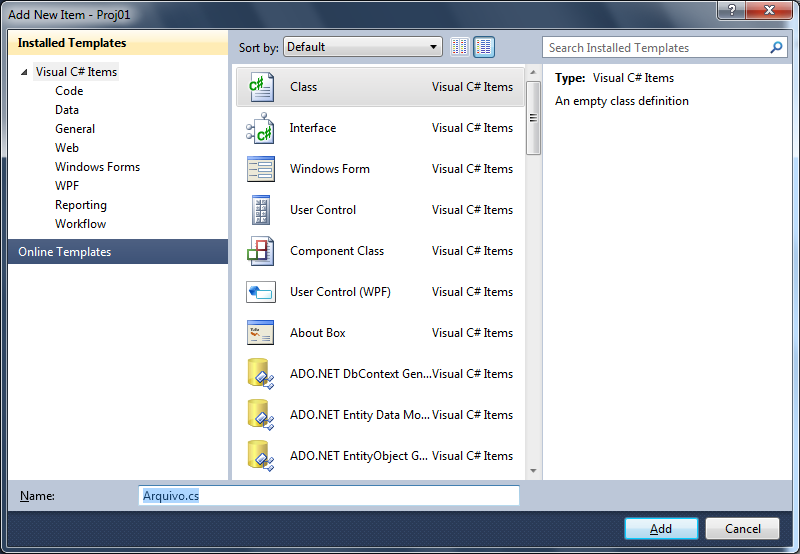
return LerNome(); //Recursividade

}

}

Manipulação de Arquivos

Pode ser feita com o objetivo de armazenar informações. Em programação, o nome dado a classes e rotinas voltadas para armazenamento de dados é chamado de Persistência de dados



using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

**using System.IO; //input / output**

**using Proj01.Model; //importando o namespace**

namespace Proj01.Persistence

{

/// <summary>

/// Classe para manipulação de dados em arquivos.

/// </summary>

public class Arquivo

{

/// <summary>

/// Método para gravar os dados de um Produto em arquivo Word

/// </summary>

/// <param name="p">Objeto da Classe Produto contendo os dados</param>

**public void GravarWord(Produto p)**

{

try //[try] + 2x[tab]

{

//Abrindo o arquivo

**StreamWriter sw = new StreamWriter("d://aula//produtos.doc");**

//Escrevendo no arquivo

sw.WriteLine("Código..: " + p.IdProduto);

sw.WriteLine("Produto.: " + p.Nome);

sw.WriteLine("Preço...: " + p.Preco);

sw.WriteLine("...");

//Fechar o arquivo

sw.Close();

Console.WriteLine("Dados gravados com sucesso.");

}

catch (Exception e)

{

//Imprimir mensagem de erro

Console.WriteLine("Erro: " + e.Message);

}

}

}

}

Podemos criar hierarquias de tratamento de exceção, utilizando tipos de exceções mais específicas.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.IO; //input / output

using Proj01.Model; //importando o namespace

namespace Proj01.Persistence

{

/// <summary>

/// Classe para manipulação de dados em arquivos.

/// </summary>

public class Arquivo

{

/// <summary>

/// Método para gravar os dados de um Produto em arquivo Word

/// </summary>

/// <param name="p">Objeto da Classe Produto contendo os dados</param>

public void GravarWord(Produto p)

{

try //[try] + 2x[tab]

{

//Abrindo o arquivo

StreamWriter sw = new StreamWriter("d://aula//produtos.doc");

//Escrevendo no arquivo

sw.WriteLine("Código..: " + p.IdProduto);

sw.WriteLine("Produto.: " + p.Nome);

sw.WriteLine("Preço...: " + p.Preco);

sw.WriteLine("...");

//Fechar o arquivo

sw.Close();

Console.WriteLine("Dados gravados com sucesso.");

}

**catch(UnauthorizedAccessException e)**

{

Console.WriteLine("Acesso Negado: " + e.Message);

}

**catch(DirectoryNotFoundException e)**

**//diretorio não existente**

{

Console.WriteLine("Caminho do Arquivo não existe: " + e.Message);

}

**catch (Exception e)**

{

//Imprimir mensagem de erro

Console.WriteLine("Erro: " + e.Message);

}

}

}

}

\*\* Importante: Sempre a hierarquia de exceções deverá ser do erro mais especifico para o erro mais abrangente.

Testando...

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using Proj01.Input; //importando o conteudo do namespace

using Proj01.Model; //importando o conteudo do namespace

using Proj01.Persistence; //importando o conteudo do namespace

namespace Proj01

{

class Program

{

static void Main(string[] args)

{

//Objeto para a Classe Produto

Produto p = new Produto();

Leitura l = new Leitura();

p.IdProduto = l.LerIdProduto();

p.Nome = l.LerNome();

p.Preco = l.LerPreco();

//Impressão

Console.WriteLine("\nDados do Produto:");

Console.WriteLine("\tCódigo....: " + p.IdProduto);

Console.WriteLine("\tNome......: " + p.Nome);

Console.WriteLine("\tPreco.....: " + p.Preco);

**Arquivo a = new Arquivo();**

**a.GravarWord(p);**

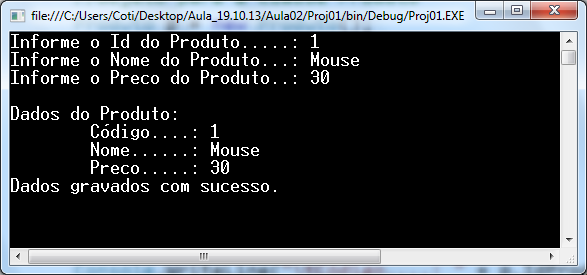
Console.ReadKey(); //Pausar

}

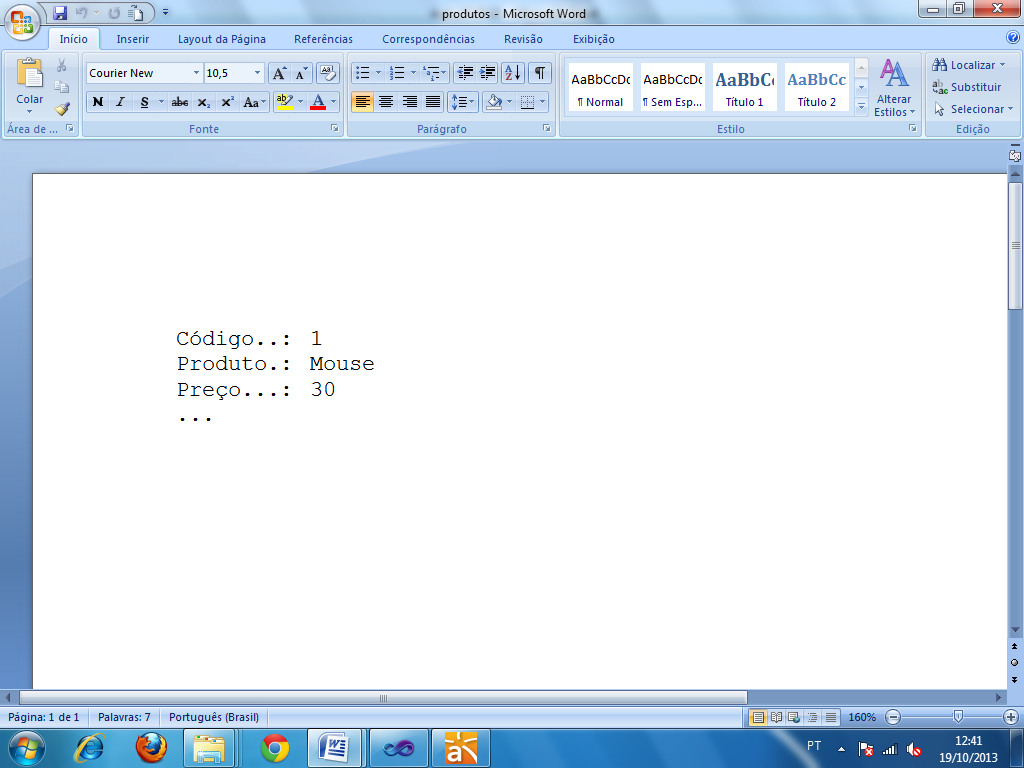
}

}

Executando...



Arquivo gravado...



flag para o arquivo nunca ser sobrescrito, mas sim, ter o conteúdo adicionado... (modo append)

**StreamWriter sw = new StreamWriter**

**("d://aula//produtos.doc", true);**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.IO; //input / output

using Proj01.Model; //importando o namespace

namespace Proj01.Persistence

{

/// <summary>

/// Classe para manipulação de dados em arquivos.

/// </summary>

public class Arquivo

{

/// <summary>

/// Método para gravar os dados de um Produto em arquivo Word

/// </summary>

/// <param name="p">Objeto da Classe Produto contendo os dados</param>

public void GravarWord(Produto p)

{

try //[try] + 2x[tab]

{

//Abrindo o arquivo

StreamWriter sw = new StreamWriter

("d://aula//produtos.doc", **true**);

//Escrevendo no arquivo

sw.WriteLine("Código..: " + p.IdProduto);

sw.WriteLine("Produto.: " + p.Nome);

sw.WriteLine("Preço...: " + p.Preco);

sw.WriteLine("...");

//Fechar o arquivo

sw.Close();

Console.WriteLine("Dados gravados com sucesso.");

}

catch(UnauthorizedAccessException e)

{

Console.WriteLine("Acesso Negado: " + e.Message);

}

catch(DirectoryNotFoundException e) //diretorio não existente

{

Console.WriteLine("Caminho do Arquivo não existe: " + e.Message);

}

catch (Exception e) //captura exceções genericas

{

//Imprimir mensagem de erro

Console.WriteLine("Erro: " + e.Message);

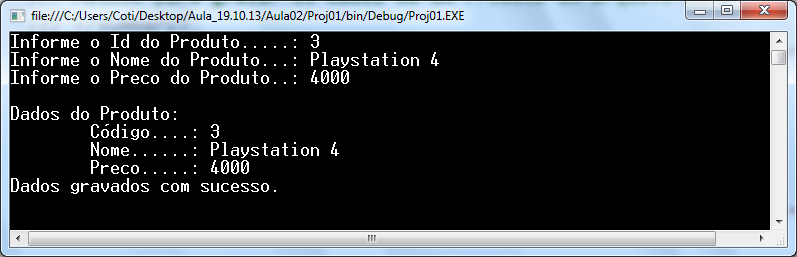
}

}

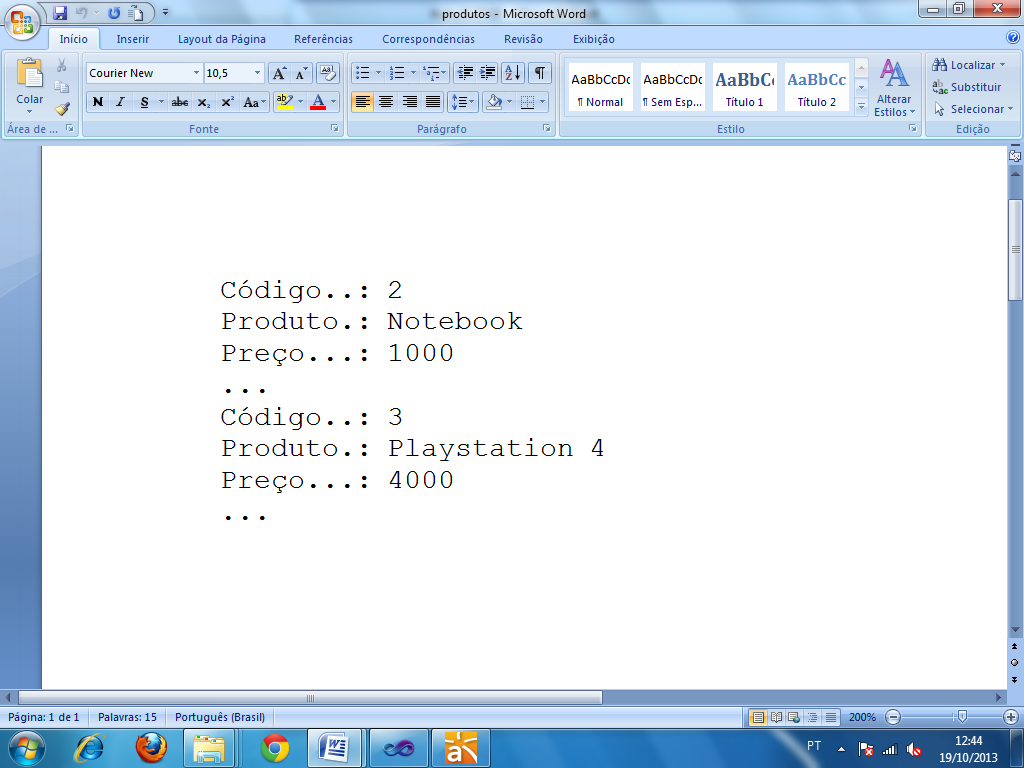
}

}

Executando...



Arquivo:



Programa para ler e imprimir no

DOS o conteúdo do arquivo...

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.IO; //input / output

using Proj01.Model; //importando o namespace

namespace Proj01.Persistence

{

/// <summary>

/// Classe para manipulação de dados em arquivos.

/// </summary>

public class Arquivo

{

/// <summary>

/// Atributo para definir o caminho do arquivo Word

/// </summary>

private const string path = "d://aula//produtos.doc";

/// <summary>

/// Método para gravar os dados de um Produto em arquivo Word

/// </summary>

/// <param name="p">Objeto da Classe Produto contendo os dados</param>

public void GravarWord(Produto p)

{

try //[try] + 2x[tab]

{

//Abrindo o arquivo

StreamWriter sw = new StreamWriter(path, true);

//Escrevendo no arquivo

sw.WriteLine("Código..: " + p.IdProduto);

sw.WriteLine("Produto.: " + p.Nome);

sw.WriteLine("Preço...: " + p.Preco);

sw.WriteLine("...");

//Fechar o arquivo

sw.Close();

Console.WriteLine("Dados gravados com sucesso.");

}

catch(UnauthorizedAccessException e)

{

Console.WriteLine("Acesso Negado: " + e.Message);

}

catch(DirectoryNotFoundException e) //diretorio não existente

{

Console.WriteLine("Caminho do Arquivo não existe: " + e.Message);

}

catch (Exception e) //captura exceções genericas

{

//Imprimir mensagem de erro

Console.WriteLine("Erro: " + e.Message);

}

}

**/// <summary>**

**/// Método para ler e imprimir no prompt o conteúdo do arquivo Word**

**/// </summary>**

**public void LerWord()**

**{**

**try**

**{**

**//Classe para leitura do arquivo**

**StreamReader sr = new StreamReader(path);**

**//Percorrer cada linha do arquivo até o seu final**

**while( ! sr.EndOfStream ) //enquanto não é o final do arquivo**

**{**

**//Imprimir cada linha do arquivo**

**Console.WriteLine(sr.ReadLine());**

**}**

**}**

**catch(Exception e)**

**{**

**Console.WriteLine("Erro: " + e.Message);**

**}**

**}**

}

}

Testando...

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using Proj01.Input; //importando o conteudo do namespace

using Proj01.Model; //importando o conteudo do namespace

using Proj01.Persistence; //importando o conteudo do namespace

namespace Proj01

{

class Program

{

static void Main(string[] args)

{

//Objeto para a Classe Produto

Produto p = new Produto();

Leitura l = new Leitura();

p.IdProduto = l.LerIdProduto();

p.Nome = l.LerNome();

p.Preco = l.LerPreco();

//Impressão

Console.WriteLine("\nDados do Produto:");

Console.WriteLine("\tCódigo....: " + p.IdProduto);

Console.WriteLine("\tNome......: " + p.Nome);

Console.WriteLine("\tPreco.....: " + p.Preco);

**Arquivo a = new Arquivo();**

**a.GravarWord(p);**

**a.LerWord();**

Console.ReadKey(); //Pausar

}

}

}

