# Erro em questão

# Uma imagem com texto, captura de ecrã, Tipo de letra Descrição gerada automaticamente

# Clinica Dentaria

## Program.cs

/\*

\*Main program Clínica Dentária

\*Autor: Eduardo Queirós, nº 23005

\*

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using ObjetosNegocio;

using RegrasNegocio;

using Dados;

namespace Clinic

{

public class Program

{

static void Main(string[] args)

{

#region Utilizadores CLinica

Pessoa p1 = new Pessoa("Tone", DescricaoPessoa.FUNCIONARIO,2, 20);

Console.WriteLine(p1);

#endregion

#region Consultas

#endregion

Console.ReadKey();

}

}

}

# Dados

## Consultas

/\*

\*Gestão das consultas

\*Autor: Eduardo Queirós, nº 23005

\*

\*/

using System;

using System.Runtime.Serialization.Formatters.Binary;

using ObjetosNegocio;

namespace Dados

{

public class Consultas

{

//dictionary de Consulta que foi criada em ObjetosNegocio

static Dictionary<int, List<Consulta>> consulta = new Dictionary<int, List<Consulta>>();

/// <summary>

/// Inicializar dictionary consulta

/// </summary>

private static void inicializConsulta()

{

consulta = new Dictionary<int, List<Consulta>>();

}

static Consultas()

{

inicializConsulta();

}

/// <summary>

/// inserir uma consulta

/// </summary>

/// <param name="c"></param>

/// <param name="dia"></param>

/// <returns></returns>

public static bool RegistaConsulta(Consultas cons, int dia)

{

if(consulta.ContainsKey(dia))

{

return true;

}

else

{

consulta.Add(dia, new List<Consulta>());

}

return false;

}

/// <summary>

/// Função para verificar se existe uma determinada consulta

/// para verificar vai comparar o id da consulta

/// </summary>

/// <param name="id"></param>

/// <returns></returns>

public static bool ExisteConsulta(int id)

{

//Ver se o dictionary foi inicializado

if (consulta == null)

{

return false;

}

foreach (var pair in consulta)

{

List<Consulta> listConsultas = pair.Value;

foreach (Consulta cons in listConsultas)

{

if (cons.idConsulta == id) {

return true;

}

}

}

return false;

}

//FAZER AS FUNÇÕES DE ler

/// <summary>

/// Guardar as consultas num ficheiro Binário

/// </summary>

/// <param name="pessoa"></param>

/// <param name="caminhoFicheiro"></param>

public static void GuardaConsultaBinario(Dictionary<int, List<Consulta>> consulta, string caminhoFicheiroConsulta)

{

using (FileStream stream = new FileStream(caminhoFicheiroConsulta, FileMode.Create))

{

//Cria um BinaryFormatter para serialização do dictionary

BinaryFormatter formatter = new BinaryFormatter();

//Serizalização do dictionary no arquivo

formatter.Serialize(stream, consulta);

}

}

/// <summary>

/// Carregar ficheiro binário das Consultas

/// </summary>

/// <param name="caminhoFicheiro"></param>

/// <returns></returns>

public static Dictionary<int, List<Consulta>> CarregarConsultaFicheiroB(string caminhoFicheiroConsulta)

{

// Verifica se o arquivo existe antes de tentar carregar

if (File.Exists(caminhoFicheiroConsulta))

{

// Cria um FileStream para ler o arquivo binário

using (FileStream stream = new FileStream(caminhoFicheiroConsulta, FileMode.Open))

{

// Cria um BinaryFormatter para desserializar o dicionário

BinaryFormatter formatter = new BinaryFormatter();

// Desserializa o dicionário do arquivo

return (Dictionary<int, List<Consulta>>)formatter.Deserialize(stream);

}

}

else

{

return new Dictionary<int, List<Consulta>>();

}

}

/// <summary>

/// Remover Consultas do Dictionay

/// </summary>

/// <param name="pessoa"></param>

/// <param name="chave"></param>

public static void RemoverConsulta(Dictionary<int, List<Consulta>> consulta, int chave)

{

// Verifica se a chave existe no dicionário antes de remover

if (consulta.ContainsKey(chave))

{

// Remove a chave do dicionário

consulta.Remove(chave);

}

else

{

Console.WriteLine("A chave '{chave}' não existe no dicionário. Não é possível remover.", chave);

}

}

}

}

## UtilizadorClinica

/\*

\*Gestão dos utilizadores da clínica

\*Autor: Eduardo Queirós, nº 23005

\*

\*/

using ObjetosNegocio;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.Serialization.Formatters.Binary;

using System.Text;

using System.Threading.Tasks;

namespace Dados

{

public class UtilizadorClinica

{

//dictionary de Pessoas que foi criada em ObjetosNegocio

static Dictionary<int, List<Pessoa>> pessoa = new Dictionary<int, List<Pessoa>>();

/// <summary>

/// Inicializar dictionary de utilizadores

/// </summary>

public UtilizadorClinica()

{

pessoa = new Dictionary<int, List<Pessoa>>();

}

/// <summary>

/// Funcao para inserir pessoas

/// </summary>

/// <param name="p"></param>

/// <param name="dia"></param>

/// <returns></returns>

public static bool InserePessoa(Pessoa p, int dia)

{

if (pessoa.ContainsKey(dia))

{

return true;

}

else

{

pessoa.Add(dia, new List<Pessoa>());

}

return false;

}

/// <summary>

/// Funcao para verificar se existem essas pessoas

/// para verificar vai comparar pelo nome

/// </summary>

/// <param name="id"></param>

/// <returns></returns>

public static bool ExistePessoa(string nome)

{

//Ver se o dictionary foi inicializado

if (pessoa == null)

{

return false;

}

foreach (var pair in pessoa)

{

List<Pessoa> listPessoas = pair.Value;

foreach (Pessoa p in listPessoas)

{

if (p.Nome.Equals(nome, StringComparison.OrdinalIgnoreCase))

{

return true;

}

}

}

return false;

}

//FAZER AS FUNÇÕES DE Remover, ler e guardar(e carregar)

/// <summary>

/// Guardar Pessoas num ficheiro de texto em binário

/// </summary>

/// <param name="pessoa"></param>

/// <param name="caminhoFicheiro"></param>

public static void GuardaPessoaBinario(Dictionary<int, List<Pessoa>> pessoa, string caminhoFicheiro)

{

using (FileStream stream = new FileStream(caminhoFicheiro, FileMode.Create))

{

//Cria um BinaryFormatter para serialização do dictionary

BinaryFormatter formatter = new BinaryFormatter();

//Serizalização do dictionary no arquivo

formatter.Serialize(stream, pessoa);

}

}

/// <summary>

/// Carregar ficheiro binário das Pessoas

/// </summary>

/// <param name="caminhoFicheiro"></param>

/// <returns></returns>

public static Dictionary<int, List<Pessoa>> CarregarFicheiroB(string caminhoFicheiro)

{

// Verifica se o arquivo existe antes de tentar carregar

if (File.Exists(caminhoFicheiro))

{

// Cria um FileStream para ler o arquivo binário

using (FileStream stream = new FileStream(caminhoFicheiro, FileMode.Open))

{

// Cria um BinaryFormatter para desserializar o dicionário

BinaryFormatter formatter = new BinaryFormatter();

// Desserializa o dicionário do arquivo

return (Dictionary<int, List<Pessoa>>)formatter.Deserialize(stream);

}

}

else

{

return new Dictionary<int, List<Pessoa>>();

}

}

/// <summary>

/// Função para remover num dictionary

/// </summary>

/// <param name="dicionario"></param>

/// <param name="chave"></param>

public static void RemoverPessoa(Dictionary<int, List<Pessoa>> pessoa, int chave)

{

// Verifica se a chave existe no dicionário antes de remover

if (pessoa.ContainsKey(chave))

{

// Remove a chave do dicionário

pessoa.Remove(chave);

}

else

{

Console.WriteLine("A chave '{chave}' não existe no dicionário. Não é possível remover.", chave);

}

}

}

}

# ObjetosNegocio

## Consulta

\*

\*"Parâmetros" de cada consulta

\*Autor: Eduardo Queirós, nº 23005

\*

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

/// <summary>

/// enumerado tipo de tratamanento que pode ser realizado durante uma consulta

/// </summary>

public enum TipoTratamento

{

ORTODONTIA,

BRANQUEAMENTO, //branqueamento do dente do utente

RESTAURACAO, //restauraçao do dente do utente

DESVITALIZACAO //desvitalização do dente do utente

}

/// <summary>

/// Tipo de pagamento disponivel para pagar a consulta

/// </summary>

public enum TipoPagamento

{

DINHEIRO,

CARTAO,

CHEQUE

}

namespace ObjetosNegocio

{

public class Consulta: IComparable

{

#region Atributos

//para ja tenho estas variaveis, dps ver se é preciso colocar mais ou n

private int id;

private TipoTratamento tipoTrat;

private TipoPagamento tipoPag;

private DateTime dataConsulta;

private int duracao; //em minutos

#endregion

#region Métodos

#region Construtores

/// <summary>

/// Construtor parametros default

/// </summary>

public Consulta()

{

id = 0;

tipoTrat = TipoTratamento.ORTODONTIA;

//tipoTrat = TipoTratamento.BRANQUEAMENTO;

//tipoTrat = TipoTratamento.RESTAURACAO;

//tipoTrat = TipoTratamento.DESVITALIZACAO;

tipoPag = TipoPagamento.DINHEIRO;

//tipoPag = TipoPagamento.CARTAO;

//tipoPag = TipoPagamento.CHEQUE;

dataConsulta = DateTime.Now;

duracao = 30;

}

/// <summary>

/// Construtor Consulta

/// </summary>

/// <param name="idConsulta"></param>

/// <param name="tipoTratamento"></param>

/// <param name="tipoPagamento"></param>

/// <param name="duracaoConsulta"></param>

public Consulta(int idConsulta, TipoTratamento tipoTratamento, TipoPagamento tipoPagamento, int duracaoConsulta)

{

id = idConsulta;

tipoTrat = tipoTratamento;

tipoPag = tipoPagamento;

dataConsulta = DateTime.Today;

duracao = duracaoConsulta;

}

/// <summary>

///Construtor Consulta

/// </summary>

/// <param name="idConsulta"></param>

/// <param name="tipoTratamento"></param>

/// <param name="tipoPagamento"></param>

/// <param name="dataCons"></param>

/// <param name="duracaoConsulta"></param>

public Consulta(int idConsulta, TipoTratamento tipoTratamento, TipoPagamento tipoPagamento, DateTime dataCons, int duracaoConsulta)

{

id = idConsulta;

tipoTrat = tipoTratamento;

tipoPag = tipoPagamento;

dataConsulta = dataCons;

duracao = duracaoConsulta;

}

#endregion

#region Propriedades

/// <summary>

/// Propriedade para o id da consulta

/// </summary>

public int idConsulta

{

get { return idConsulta; }

}

/// <summary>

/// Propriedade para o tipo de tratamento

/// </summary>

public TipoTratamento tipoTratamento

{

get { return tipoTratamento; }

set { tipoTratamento = value; }

}

/// <summary>

/// Propriedade para o tipo de pagamento

/// </summary>

public TipoPagamento tipoPagamento

{

get { return tipoPagamento; }

set { tipoPagamento = value; }

}

//Voltar a ver esta propriedade

/// <summary>

/// Propriedade para a data da consulta

/// </summary>

public DateTime DataConsulta

{

get { return dataConsulta;}

set { dataConsulta = value; }

}

#endregion

#region Override

/// <summary>

/// Formato dos dados da consulta ao ser "chamado"

/// </summary>

/// <returns></returns>

public override string ToString()

{

return string.Format("Consulta: Id {0}\n Tipo tratamento {1}\n Duracao {2}\n Data realizacao {3}\n Tipo de pagamento {4}\n", id, duracao, dataConsulta, tipoPagamento);

}

/// <summary>

/// Comparar se dois objetos consulta sao iguais

/// </summary>

/// <param name="obj"></param>

/// <returns></returns>

public override bool Equals(object? obj)

{

if (obj == null || !(obj is Consulta))

{

return false;

}

Consulta aux =(Consulta)obj;

if(id == aux.id && tipoTratamento == aux.tipoTratamento && duracao == aux.duracao && tipoPagamento == aux.tipoPagamento)

{

return true;

}

return false;

}

/// <summary>

/// Calcula o código Hash

/// </summary>

/// <returns></returns>

public override int GetHashCode()

{

return id.GetHashCode(); // Ou qualquer outra lógica para gerar um código hash único para a instância

}

#endregion

#region Outros Metodos

/// <summary>

/// Verificar se o objeto que está é consulta

/// </summary>

/// <param name="obj"></param>

/// <returns></returns>

/// <exception cref="Exception"></exception>

public int CompareTo(object? obj)

{

if (obj.GetType() == typeof(Consulta))

{

Consulta aux = (Consulta)obj;

//if (aux.nome > this.nome)

//if(aux.nome.CompareTo(this.nome))

//if(String.Compare(aux.nome,this.nome))

if (this.id.CompareTo(aux.id) > 0) return 1;

else

if (this.id.CompareTo(aux.id) < 0) return -1;

return 0;

}

else

throw new Exception("Impossivel");

}

#endregion

#endregion

}

}

## Pessoa

/\*

\*"Parâmetros" de pessoa(pode ser Utente, Funcionario, Medico)

\*Autor: Eduardo Queirós, nº 23005

\*

\*/

using System;

using System.Collections.Generic;

using System.Diagnostics.Contracts;

using System.Linq;

using System.Security.Cryptography.X509Certificates;

using System.Text;

using System.Threading.Tasks;

using static System.Net.Mime.MediaTypeNames;

/// <summary>

/// Que tipo de pessoa é

/// </summary>

public enum DescricaoPessoa

{

UTENTE,// em default funcionario tem codigo 10

FUNCIONARIO, // em default funcionario tem codigo 20

MEDICO// em default funcionario tem codigo 30

}

/// <summary>

/// Quando a pessoa é funcionário, escolhe o tipo de funcionário que é

/// </summary>

public enum CargoFuncionario

{

ASSISTENTE,

RECECIONISTA

}

/// <summary>

/// Especialidade de um médico

/// </summary>

public enum Especialidade

{

ORTODONTIA,

BRANQUEAMENTO, //branqueamento do dente do utente

RESTAURACAO, //restauraçao do dente do utente

DESVITALIZACAO //desvitalização do dente do utente

}

namespace ObjetosNegocio

{

public class Pessoa

{

#region Atributos

private string nome;

//para poder utilizar esta descrição noutras classes foi necessário colocar protected

protected DescricaoPessoa descricao;

/\*esta variavel vai ser a que vai identificar cada pessoa na clinica, e como esse codigo vai sempre ser o mesmo, coloquei static pois ao ser static,

\* ao fechar o programa, a memória vai continuar la \*/

private static int cod;

protected int codGlobal;

protected int idade;

#endregion

#region Métodos

#region Construtores

/// <summary>

/// Construtor de classe: Inicializa variaveis

/// </summary>

static Pessoa()

{

cod = 1;

}

/// <summary>

/// Construtor de pessoa com parâmetros default

/// </summary>

public Pessoa()

{

nome = "";

descricao = tipo;

codGlobal = cod;

cod++;

idade = 20;

descricao = DescricaoPessoa.UTENTE;

//descricao = DescricaoPessoa.MEDICO;

//descricao = DescricaoPessoa.FUNCIONARIO;

}

/// <summary>

/// Construtor Pessoa

/// </summary>

/// <param name="Nome"></param>

/// <param name="descricaoP"></param>

/// <param name="codGlob"></param>

/// <param name="Idade"></param>

public Pessoa(string Nome, DescricaoPessoa descricaoP, int codGlob, int Idade)

{

nome = Nome;

descricao = descricaoP;

codGlobal = codGlob;

idade = Idade;

}

#endregion

#region Propriedades

/// <summary>

/// Propriedade para o nome da pessoa

/// </summary>

public string Nome

{

set { nome = value; }

get { return nome; }

}

/// <summary>

/// Propriedade para o tipo de pessoa que é

/// </summary>

public DescricaoPessoa tipo

{

get { return descricao; }

set { descricao = value; }

}

//ACHO QUE FALTA UMA PARA O CODGLOBAL!!!!!!

/// <summary>

/// Propriedade para a idade da pessoa

/// </summary>

public int Idade

{

set { idade = value; }

get { return idade; }

}

#endregion

#region override

/// <summary>

/// Formato dos dados da pessoa ao ser "chamado"

/// </summary>

/// <returns></returns>

public override string ToString()

{

return String.Format("Ficha da Pessoa-> Nome: {0}; Funcao: {1} ; codigo {2}\n; idade{3}\n", nome, descricao, codGlobal, idade);

}

#endregion

#region Outros Metodos

/// <summary>

/// Comparar se dois objetos pessoa são iguais

/// </summary>

/// <param name="obj"></param>

/// <returns></returns>

public override bool Equals(object? obj)

{

if (obj == null || !(obj is Pessoa))

{

return false;

}

Pessoa aux = (Pessoa)obj;

//ver se neste if a primeira condição está correta...

if (cod == aux.codGlobal && System.String.Equals(nome, aux.nome) && descricao == aux.descricao)

{

return true;

}

return false;

}

//ver se nesta esta correta

/// <summary>

/// Calcula o código Hash

/// </summary>

/// <returns></returns>

public override int GetHashCode()

{

return cod.GetHashCode(); // Ou qualquer outra lógica para gerar um código hash único para a instância

}

#endregion

#endregion

}

public class Utente : Pessoa

{

#region Atributos

//verificar se estes atributos dps vao ser public ou n... n tenho a certeza se vou ligar esta classe à classe Consulta...

private string mail;

private int numTelef;

/\*meti static pq cada utente vai ter um codigo cartao de saude que vai ser necessário para a identificação na clínica

\* (nota: além de terem este codigo tambem têm o codigo que herdam de pessoa)\*/

static int numCartaoSaude;

#endregion

#region Métodos

#region Construtores

/// <summary>

/// Inicializar o numero de cartao de saude a 0

/// </summary>

static Utente()

{

numCartaoSaude = 0;

}

/// <summary>

/// Construtor para parametros default

/// </summary>

public Utente()

{

Nome = "Antonio";

codGlobal = 10;

Idade = 17;

mail = " ";

numTelef = 987654321;

numCartaoSaude++;

}

/// <summary>

/// Construtor de Utente

/// </summary>

/// <param name="nome"></param>

/// <param name="codUtente"></param>

/// <param name="Mail"></param>

/// <param name="NumTelefUtente"></param>

/// <param name="idadeUtente"></param>

public Utente(string nome, int codUtente, string Mail, int NumTelefUtente, int idadeUtente)

{

Nome = nome;

codGlobal = codUtente;

Idade = idadeUtente;

mail = Mail;

numTelef = NumTelefUtente;

numCartaoSaude++;

}

#endregion

#region Propriedades

/\*

\* nota: nao vou fazer nd para o num de cartao de saude porque quero que n se mexa nisso e que seja o programa a "atibuir" pois tenho o numCartaoSaude++;

\*/

/// <summary>

/// Propriedade para o email do utente

/// </summary>

public string Mail

{

set { mail = value; }

get { return mail; }

}

/// <summary>

/// Propriedade para o num de telefone do utente

/// </summary>

public int NumTelefUtente

{

set { NumTelefUtente = value; }

get { return NumTelefUtente; }

}

#endregion

#region Outros Metodos

#endregion

#endregion

}

public class Funcionario : Pessoa

{

#region Atributos

private CargoFuncionario cargo;

private int numTelef;

private int horario; // vai dizer quantos dias por semana trabalha(0 a 7)

private int salario;

#endregion

#region Métodos

#region Construtores

/// <summary>

/// Construtor parametros default para funcionário

/// </summary>

public Funcionario()

{

Nome = "Joaquim ";

codGlobal = 20;

idade = 23;

//descricao = DescricaoPessoa.FUNCIONARIO; -> penso que n faz sentido por isso retirei

cargo = CargoFuncionario.ASSISTENTE;

numTelef = 912345678;

horario = 2;

salario = 900;

}

/// <summary>

/// Construtor para funcionário

/// </summary>

/// <param name="nome"></param>

/// <param name="codigoFunc"></param>

/// <param name="CargoFunc"></param>

/// <param name="NumTeleFuncionario"></param>

/// <param name="HorarioFunc"></param>

/// <param name="SalarioFunc"></param>

/// <param name="idadeFunc"></param>

public Funcionario(string nome, int codigoFunc,CargoFuncionario CargoFunc, int NumTeleFuncionario, int HorarioFunc, int SalarioFunc, int idadeFunc)

{

Nome = nome;

codGlobal = codigoFunc;

Idade = idadeFunc;

cargo = CargoFunc;

numTelef = NumTeleFuncionario;

horario = HorarioFunc;

salario = SalarioFunc;

}

#endregion

#region Propriedades

/// <summary>

/// Propriedade para o cargo do funcionario

/// </summary>

public CargoFuncionario cargoFunc

{

get { return cargoFunc; }

set { cargoFunc = value; }

}

/// <summary>

/// Propriedade para o numero de telefone do funcionario

/// </summary>

public int NumTelef

{

get { return numTelef; }

set { numTelef = value; }

}

/// <summary>

/// Propriedade para ver qual o horario de trabalho do funcionario

/// </summary>

public int Horario

{

get { return horario; }

set { horario = value; }

}

/// <summary>

/// Propriedade para o salário do funcionario

/// </summary>

public int Salario

{

get { return salario; }

set { salario = value; }

}

#endregion

#region Outros Metodos

#endregion

#endregion

}

public class Medico : Pessoa

{

#region Atributos

private Especialidade tipoEspecialidade;

private int salario;

private int horarioTrabalho; //variavel para dizer quantos dias por semana o medico irá trabalhar(0 a 7)

#endregion

#region Métodos

#region Construtores

/// <summary>

/// Construtor com parâmetros default

/// </summary>

public Medico()

{

Nome = "Maria";

codGlobal = 30;

idade = 27;

salario = 1000;

horarioTrabalho = 5;

tipoEspecialidade = Especialidade.ORTODONTIA;

}

/// <summary>

/// Construtor para Medico

/// </summary>

/// <param name="nome"></param>

/// <param name="codMedico"></param>

/// <param name="Salario"></param>

/// <param name="Horario"></param>

/// <param name="TipoEsp"></param>

/// <param name="idadeMedico"></param>

public Medico(string nome, int codMedico, int Salario,int Horario,Especialidade TipoEsp, int idadeMedico)

{

Nome = nome;

codGlobal = codMedico;

Idade = idadeMedico;

salario = Salario;

horarioTrabalho = Horario;

tipoEspecialidade = TipoEsp;

}

#endregion

#region Propriedades

/// <summary>

/// Propriedade para o salário do medico

/// </summary>

public int Salario

{

get { return salario; }

set { salario = value; }

}

/// <summary>

/// Propriedade para verificar o horario de trabalho do medico

/// </summary>

public int HorarioTrabalho

{

get { return horarioTrabalho; }

set { horarioTrabalho = value; }

}

/// <summary>

/// Propriedade para a especialidade do medico

/// </summary>

public Especialidade TipoEspecialidade

{

get { return tipoEspecialidade; }

set { tipoEspecialidade = value; }

}

#endregion

#region Outros Metodos

#endregion

#endregion

}

}

# RegrasNegocio

## GereClinica

/\*

\* Regras da clínica

\*Autor: Eduardo Queirós, nº 23005

\*

\*/

using System;

using ObjetosNegocio;

using Dados;

using Validacoes;

using static System.Net.Mime.MediaTypeNames;

namespace RegrasNegocio

{

public class GereClinica

{

public GereClinica()

{

}

/// <summary>

/// Regra de negócio ao inserir pessoa

/// </summary>

/// <param name="p"></param>

/// <returns></returns>

/\*

public static bool InserePessoa(Pessoa p)

{

if (p.tipo != DescricaoPessoa.UTENTE) // Regra de negocio

{

return UtilizadorClinica.InserePessoa(p);

}

return false;

}

\*/

/// <summary>

/// Idade válida para ser utente da clínica

/// </summary>

/// <param name="idade"></param>

/// <returns></returns>

public static bool IdadeValidaUtente(int idade)

{

return (idade > 5 && idade < 100);

}

/// <summary>

/// Idade valida para poder trabalhar na clínica

/// </summary>

/// <param name="idade"></param>

/// <returns></returns>

public static bool IdadeValidaTrab(int idade)

{

return (idade > 18 && idade < 65);

}

}

}

# Validacoes

## Validacoes

/\*

\*Validações

\*Autor: Eduardo Queirós, nº 23005

\*

\*/

namespace Validacoes

{

public class Validacoes

{

public Validacoes()

{

}

/// <summary>

/// Validacao para o nome

/// </summary>

/// <param name="nome"></param>

/// <returns></returns>

public static bool NomePossivel(string nome)

{

// Verifica se o comprimento do nome está entre 1 e 120 caracteres

return !string.IsNullOrEmpty(nome) && nome.Length <= 120;

}

}

}