Relatório de trabalho prático

Gestão de Auditorias

Yuri Serediuk Lemos

Nº 19256

João Pedro Marques Figueiredo

Nº 19337

Trabalho realizado sob a orientação de:

Luís Ferreira

Linguagens de Programação II

Licenciatura em Engenharia de Sistemas Informáticos

Barcelos, 25 de Abril de 2020

Índice

Índice de Imagens

1. Introdução

Este projeto foi realizado tendo como objetivo a avaliação da compreensão dos alunos para a linguagem de programação orientada a objetos (POO). Nesta primeira parte do projeto desenvolvido mostra a capacidade de compreensão dos alunos dos termos básicos do paradigma desta linguagem.

1.1. Contextualização

O advento da tecnologia fez com que fossem necessários novos arranjos para a sociedade e para o poder público. Não raro, a lei não consegue acompanhar as mudanças velozes do mundo conectado, mas há anos que a demanda por leis mais protetivas aos dados pessoais do cidadão ganha espaço entre especialistas e entusiastas da tecnologia. Isto porque, o cidadão comum, por vezes, não é capaz de reconhecer os riscos e oportunidades criadas para a exploração de seus dados pessoais. Assim, muitas vezes abrem mão de informações pessoais importantes em troca de simples acesso a sites e aplicativos de telemóveis. Não apenas isto, mas toda a troca de dados realizada em estabelecimentos comerciais físicos e em registos quaisquer transações do tipo também deveriam constar no escopo de uma lei geral de proteção de dados. A União Europeia há alguns anos já gozava de tal proteção com a Diretiva 95/46 CE, a qual buscava regular a coleta, uso e tratamento de dados no território europeu. Porém, acompanhando as tendências de uso global, transferência e necessidade de apagamento ou prestação de contas sobre dados pessoais, um novo Regulamento foi elaborado a fim de expandir a proteção ao indivíduo. Sendo assim, após um período de dois anos, entrou em vigor o Regulamento Geral de Proteção de Dados europeu — GDPR. Com texto extenso e uma preocupação notável em abranger as mais diversas possibilidades de transação envolvendo dados, o GDPR sagrou-se um marco na proteção de dados e da proteção à privacidade do utilizador. Instituiu princípios sólidos e claros, a fim de não abrir margem para interpretações diversas. Assim, o documento cria no indivíduo a possibilidade de domínio sobre os próprios dados, reclamando a propriedade destes como algo pessoal e não comercial, pertencente às empresas ou amplamente explorado pelo Poder Público.

1.2. Motivações e objetivos

O propósito do trabalho foi montar de forma criativa, consoante ao RGPD um programa de fácil acessibilidade utilizando a linguagem C#, com isso pode-se então ter uma melhor perceção das falhas ao RGPD numa determinada empresa.

1.3. Estado da arte

O Regulamento Geral sobre a Proteção de Dados (RGPD), aprovado no Parlamento Europeu em abril de 2016, visa regular a proteção das pessoas singulares no que diz respeito ao tratamento de dados pessoais e à livre circulação desses dados, revogando assim a Diretiva 95/46/CE. Este Regulamento introduziu alterações significativas às regras mantidas em matérias de proteção de dados, impondo novas e rigorosas obrigações e severas coimas como punição ao seu no seu não cumprimento.

O RGPD proporcionou um período de transição de 2 anos para que as organizações implementassem um plano de transformação que proporcione o cumprimento de todos os requisitos necessários à aplicabilidade do RGPD, em 25 de maio de 2018. Em Portugal, a 8 de agosto de 2019, foi publicada a nova Lei nº 58/2019 que assegura a execução, na ordem jurídica nacional, do RGPD, e que revoga a anterior Lei nº 67/98.

O mundo virtual está em crescimento todos os dias, e a proteção dos dados é essencial para que a confidencialidade de uma companhia não seja comprometida. E para que os dados sejam resguardados de forma segura judicialmente, surgiu o *Data Protection Officer* (DPO) ou, em português, diretor de Proteção de Dados.

De acordo com a lei LGPD (Lei Geral de Proteção de Dados – 13.709/2018), a profissão entrará em vigor em agosto de 2020. O especialista será responsável pelo aconselhamento sobre as normas vigentes relativas à proteção de dados pessoais, o monitoramento do cumprimento do GDPR (*General Data Protection Regulation*) pela entidade a que está vinculado e a cooperação com as autoridades públicas supervisoras da norma.

1.4. Descrição

As auditorias internas podem ser efetuadas por vários colaboradores. Cada auditoria deve ter em conta um código (ano/numero), quem foi o colaborador responsável, a data, quantidade de vulnerabilidade identificadas. Depois de inseridos os dados acerca das auditorias, quando terminar, a aplicação deverá mostrar:

- Uma relação das auditorias efetuadas, indicando o código e o colaborador responsável.
- Qual a auditoria onde foram detetadas mais e menos vulnerabilidades, indicando o seu código, data e quantidade de vulnerabilidades.

A aplicação deverá ter também uma estrutura de colaboradores, equipamentos, vulnerabilidades e um registo de ocorrência

1.5. Solução

1.5.1. Diagrama

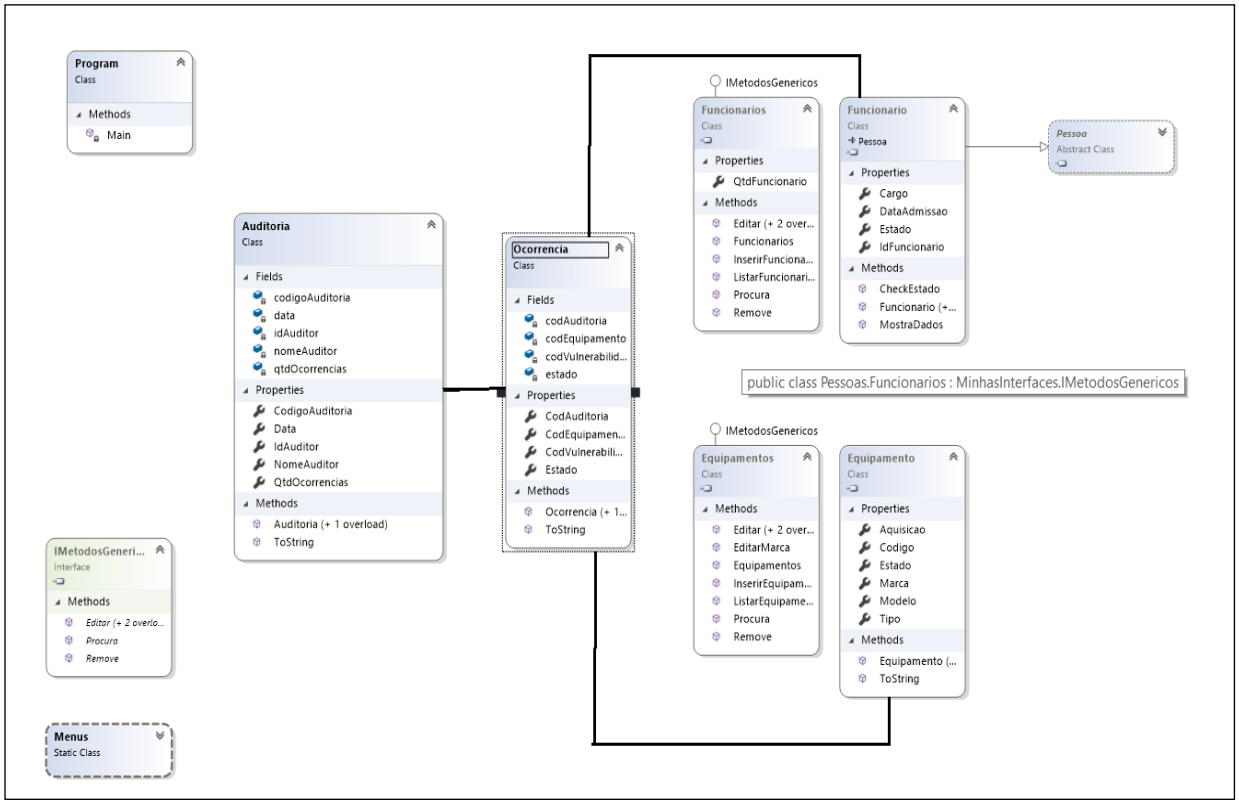


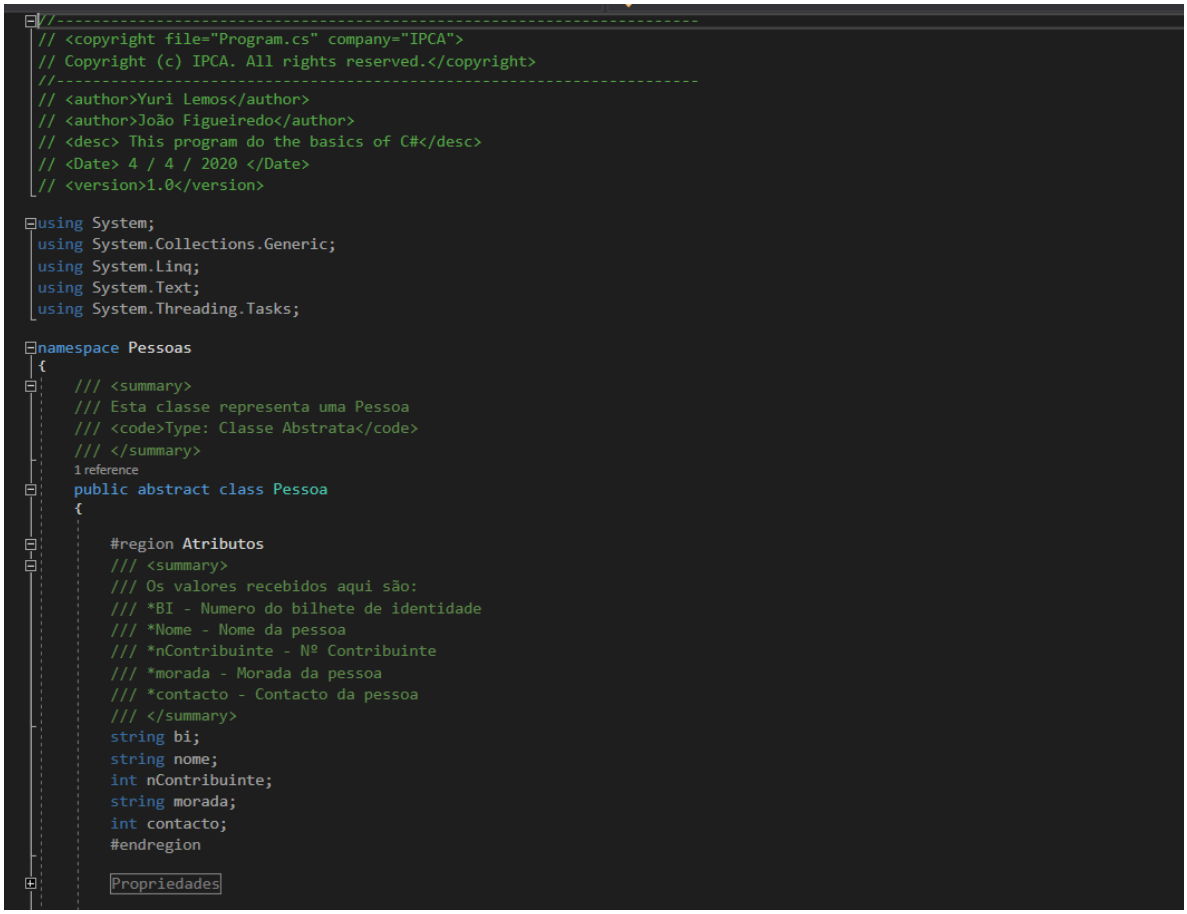
Figura 1 Diagrama de classes

1.5.2. Criação de Classes

1.5.2.1. Classe Pessoa

Esta classe representa uma Pessoa, nesta classe contém:

- Nome
- Bilhete de identidade
- Contacto
- Morada
- Número Contribuinte



```
//-----------------------------------------------------------------------
// <copyright file="Program.cs" company="IPCA">
// Copyright (c) IPCA. All rights reserved.</copyright>
//-----------------------------------------------------------------------
// <author>Yuri Lemos</author>
// <author>João Figueiredo</author>
// <desc> This program do the basics of C#</desc>
// <Date> 4 / 4 / 2020 </Date>
// <version>1.0</version>

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Pessoas
{
    /// <summary>
    /// Esta classe representa uma Pessoa
    /// <code>Type: Classe Abstrata</code>
    /// </summary>
    1 reference
    public abstract class Pessoa
    {

        #region Atributos
        /// <summary>
        /// Os valores recebidos aqui são:
        /// *BI - Numero do bilhete de identidade
        /// *Nome - Nome da pessoa
        /// *nContribuinte - Nº Contribuinte
        /// *morada - Morada da pessoa
        /// *contacto - Contacto da pessoa
        /// </summary>
        string bi;
        string nome;
        int nContribuinte;
        string morada;
        int contacto;
        #endregion

        Propriedades
```

Figura 2 Classe Pessoa

1.5.2.2. Classe Funcionário

Esta classe representa um Funcionário, ela é uma classe abstrata e por isso não poderá ser criada instâncias a partir desta classe. Nesta classe contém os seguintes atributos:

- Cargo
- Data Admissão
- Estado
- Identificação do funcionário

```
//-----------------------------------------------------------------------
// <copyright file="Program.cs" company="IPCA">
// Copyright (c) IPCA. All rights reserved.</copyright>
//-----------------------------------------------------------------------
// <author>Yuri Lemos</author>
// <author>João Figueiredo</author>
// <desc> This program do the basics of C#</desc>
// <Date> 4 / 4 / 2020 </Date>
// <version>1.0</version>

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Pessoas
{
    /// <summary>
    /// Esta Classe representa uma instancia de Funcionario
    /// <code>Herança: Pessoa</code>
    /// </summary>
    9 references
    public class Funcionario : Pessoa
    {

        #region Atributos
        /// <summary>
        /// Os valores recebidos aqui são:
        /// *idFuncionario - Numero de identificação do funcionario na empresa
        /// *cargo - Qual o cargo ou função que ocupa
        /// *dataAdmissao - A data em que foi admitido na empresa
        /// *estado - Estado da pessoa na empresa ( Se ainda trabalha para a empresa ou não)
        /// </summary>
        int idFuncionario;
        string cargo;
        DateTime dataAdmissao;
        bool estado;

        #endregion

        Construtor

        Propriedades
```

Figura 3 Classe Funcionário

1.5.2.3. Classe Funcionários

Esta classe representa vários funcionários, nesta classe contém:

- Lista de funcionários
- Quantidade de funcionários

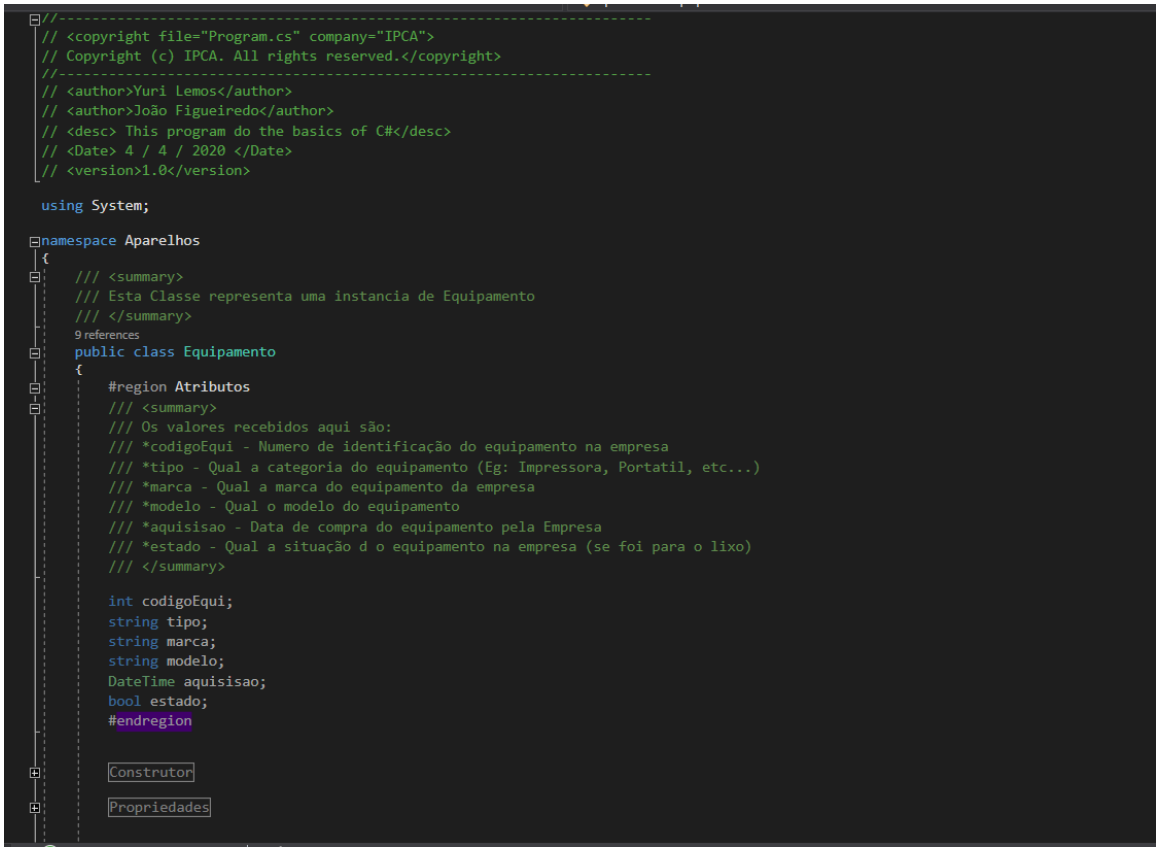
```
//-----------------------------------------------------------------------
// <copyright file="Program.cs" company="IPCA">
// Copyright (c) IPCA. All rights reserved.</copyright>
//-----------------------------------------------------------------------
// <author>Yuri Lemos</author>
// <author>João Figueiredo</author>
// <desc> This program do the basics of C#</desc>
// <Date> 4 / 4 / 2020 </Date>
// <version>1.0</version>

using MinhasInterfaces;
using System;
using System.Collections.Generic;

namespace Pessoas
{
    /// <summary>
    /// Esta Classe representa uma instancia de Funcionarios
    /// <code>Implementa: IMetodosGenericos</code>
    /// </summary>

    3 references
    public class Funcionarios : IMetodosGenericos
    {

        #region Atributos
        /// <summary>
        /// Os valores recebidos aqui são:
        /// *funcionarios - Criação de uma lista do tipo Funcionarios
        /// *qtdFuncionarios - Quantidade de funcionarios que existe na lista
        /// </summary>
        List<Funcionario> funcionarios;
        int qtdFuncionarios;
        #endregion

        Construtor

        Propriedades

        Metodos

    }
}
```

Figura 4 Classe Funcionários

1.5.2.4. Classe Equipamento

Esta classe representa um equipamento, nesta classe contém:

- Código
- Marca
- Modelo
- Data aquisição
- Estado
- Tipo de equipamento



```
//-----------------------------------------------------------------------
// <copyright file="Program.cs" company="IPCA">
// Copyright (c) IPCA. All rights reserved.</copyright>
//-----------------------------------------------------------------------
// <author>Yuri Lemos</author>
// <author>João Figueiredo</author>
// <desc> This program do the basics of C#</desc>
// <Date> 4 / 4 / 2020 </Date>
// <version>1.0</version>

using System;

namespace Aparelhos
{
    /// <summary>
    /// Esta Classe representa uma instancia de Equipamento
    /// </summary>
    9 references
    public class Equipamento
    {
        #region Atributos
        /// <summary>
        /// Os valores recebidos aqui são:
        /// *codigoEqui - Numero de identificação do equipamento na empresa
        /// *tipo - Qual a categoria do equipamento (Eg: Impressora, Portatil, etc...)
        /// *marca - Qual a marca do equipamento da empresa
        /// *modelo - Qual o modelo do equipamento
        /// *aquisisao - Data de compra do equipamento pela Empresa
        /// *estado - Qual a situação d o equipamento na empresa (se foi para o lixo)
        /// </summary>

        int codigoEqui;
        string tipo;
        string marca;
        string modelo;
        DateTime aquisisao;
        bool estado;
        #endregion


        Construtor

        Propriedades
```

Figura 5 Classe Equipamento

1.5.2.5. Classe Equipamentos

Esta classe representa vários equipamentos, nesta classe contém:

- Lista de equipamentos
- Quantidade de Equipamentos

```
//-----------------------------------------------------------------------
// <copyright file="Program.cs" company="IPCA">
// Copyright (c) IPCA. All rights reserved.</copyright>
//-----------------------------------------------------------------------
// <author>Yuri Lemos</author>
// <author>João Figueiredo</author>
// <desc> This program do the basics of C#</desc>
// <Date> 4 / 4 / 2020 </Date>
// <version>1.0</version>
using MinhasInterfaces;
using System;
using System.Collections.Generic;

namespace Aparelhos
{

    /// <summary>
    /// Esta Classe representa uma instancia de Eqipamentos
    /// <code>Implementa: IMetodosGenericos</code>
    /// </summary>

    3 references
    public class Equipamentos : IMetodosGenericos
    {

        #region Atributos
        /// <summary>
        /// Os valores recebidos aqui são:
        /// *equipamentos - Criação de uma lista do tipo Equipamento
        /// *qtdEquipamentos - Quantidade de equipamentos que existe na lista
        /// </summary>
        List<Equipamento> equipamentos;
        int qtdEquipamentos;
        #endregion


        Construtor

        Metodos


    }
}
```

Figura 6 Classe Equipamentos

Classe Vulnerabilidade

Esta classe representa uma vulnerabilidade, nesta classe contém:

- Código
- Descrição
- Nível de impacto

```
//-----------------------------------------------------------------------
// <copyright file="Program.cs" company="IPCA">
// Copyright (c) IPCA. All rights reserved.</copyright>
//-----------------------------------------------------------------------
// <author>Yuri Lemos</author>
// <author>João Figueiredo</author>
// <desc> This program do the basics of C#</desc>
// <Date> 4 / 4 / 2020 </Date>
// <version>1.0</version>

namespace Defeitos
{
    /// <summary>
    ///
    /// </summary>
    10 references
    public class Vulnerabilidade
    {
        /*
         * Vulnerabilidades Contém:
         *         - Código da vulnerabilidade
         *         - Descrição
         *         - Nivel de impacto que são eles:
         *                 + Elevado
         *                 + Médio
         *                 + Baixo
         */
        #region Atributos
        int codigo;
        string descricao;
        string nivelImpacto;

        #endregion

        Construtor

        Propriedades

        Metodos

    }
}
```

Figura 7 Classe Vulnerabilidade

1.5.2.6. Classe Vulnerabilidades

Esta classe representa uma lista de vulnerabilidades, nesta classe contém:

- Lista de vulnerabilidades
- Quantidade de vulnerabilidades

```
//-----------------------------------------------------------------------
// <copyright file="Program.cs" company="IPCA">
// Copyright (c) IPCA. All rights reserved.</copyright>
//-----------------------------------------------------------------------
// <author>Yuri Lemos</author>
// <author>João Figueiredo</author>
// <desc> This program do the basics of C#</desc>
// <Date> 4 / 4 / 2020 </Date>
// <version>1.0</version>

using System;
using System.Collections.Generic;
using MinhasInterfaces;
namespace Defeitos
{
    /// <summary>
    ///
    /// </summary>
    3 references
    public class Vulnerabilidades : IMetodosGenericos
    {

        /*
         * Vulnerabilidades Contém:
         *      - Uma lista de vulnerabilidades
         *      - Quantidade de vulnerabilidades
         */
        #region Atributos
        List<Vulnerabilidade> vulnerabilidades;
        int qtdVulnerabilidades;
        #endregion

        Construtor

        Propriedades

        Metodos
    }
}
```

Figura 8 Classe Vulnerabilidades

1.5.2.7. Classe Ocorrência

Esta classe representa uma ocorrência, nesta classe contém:

- Código da auditoria
- Código do equipamento
- Código da vulnerabilidade
- Estado

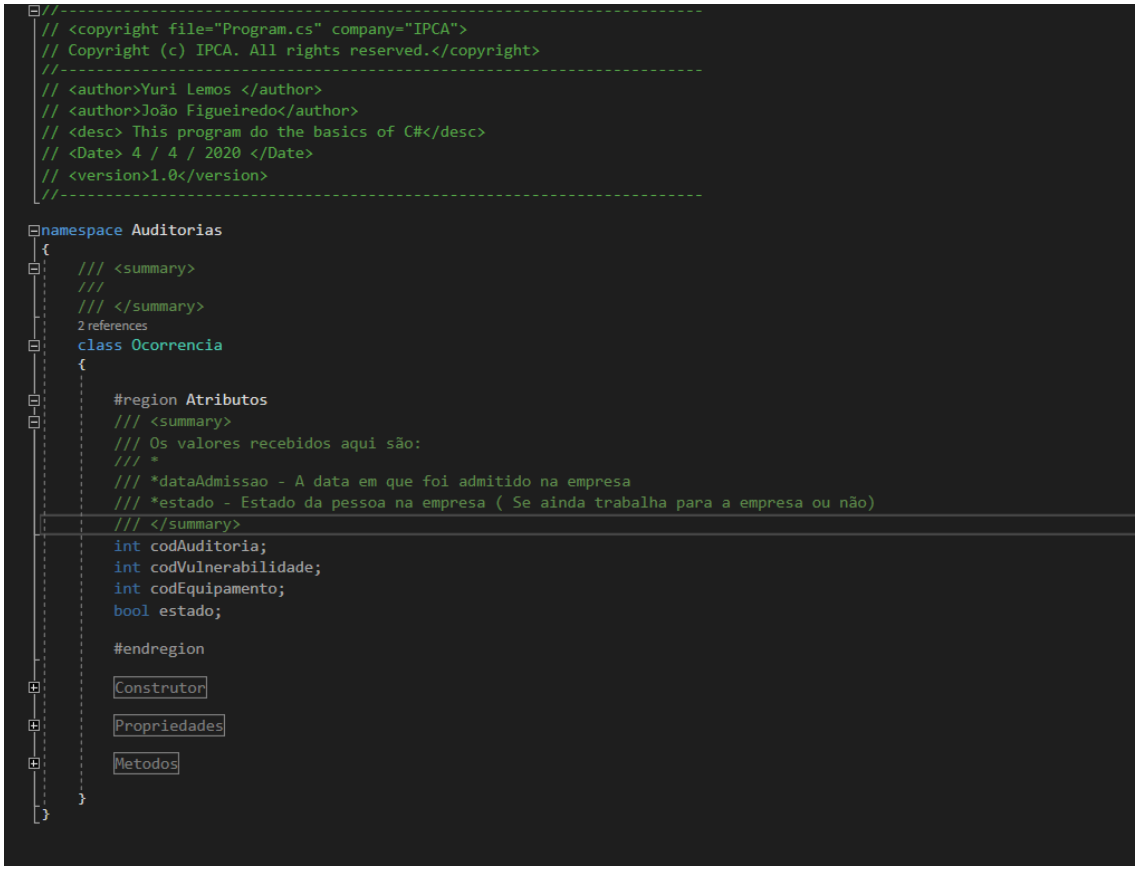


Figura 9 Classe Ocorrência

1.5.2.8. Classe Interface IMetodosGenericos

Esta classe é uma interface contendo vários métodos abstratos, estes métodos são:

- Editar
- Procura
- Remove

Estes métodos podem ser utilizados por qualquer classe desde que esta interface seja chamada e que todos seus métodos inseridos na classe que está a implementá-lo.

```
// <copyright file="Program.cs" company="IPCA">
// Copyright (c) IPCA. All rights reserved.</copyright>
//-----------------------------------------------------------------------
// <author>Yuri Lemos</author>
// <author>João Figueiredo</author>
// <desc> This program do the basics of C#</desc>
// <Date> 4 / 4 / 2020 </Date>
// <version>1.0</version>

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MinhasInterfaces
{
    /// <summary>
    /// Isso é uma interface com métodos genéricos
    /// </summary>
    3 references
    public interface IMetodosGenericos
    {

        15 references
        int Procura(int id);

        6 references
        bool Remove(int id);

        3 references
        bool Editar(int id, string nome);
        6 references
        bool Editar(int id, int numero);
        3 references
        bool Editar(int id, DateTime data);


    }
}
```

Figura 10 Classe Interface IMetodosGenericos

1.5.2.9. Classe Program

Esta classe é onde executa o método principal para inicialização do programa, aqui podem ser criados as instâncias das classes e executar os métodos das mesmas.

```
// <copyright file="Program.cs" company="IPCA">
// Copyright (c) IPCA. All rights reserved.</copyright>
//-----------------------------------------------------------------------
// <author>Yuri Lemos</author>
// <author>João Figueiredo</author>
// <desc> This program do the basics of C#</desc>
// <Date> 4 / 4 / 2020 </Date>
// <version>1.0</version>

using System;
using Pessoas; // Blibliotecas Externas
using Aparelhos;
using Defeitos;

namespace Auditorias
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            bool exit = true;
            int id; // Id para procura de qualquer objeto
            /**
             * Criação de instancias das classes Funcionario e Funcionarios
             */

            Funcionario f1 = new Funcionario("Abf2886392", "José António", 123423, "Eng.Informático", new DateTime(2001, 08, 18));
            Funcionario f2 = new Funcionario("QUSD7282", "Pedro Henriques", 1878924, "Técnico Informático", new DateTime(2005, 07, 28));

            Equipamento e1 = new Equipamento("Laptop", "HP", "Omen", new DateTime(2019, 02, 12));
            Equipamento e2 = new Equipamento("Laptop", "Lenovo", "IdeaPad", new DateTime(2019, 9, 20));

            Vulnerabilidade v1 = new Vulnerabilidade(1, "Virus", "Elevado");
            Vulnerabilidade v2 = new Vulnerabilidade(2, "Trojan", "Elevado");


            Funcionarios listaFuncionarios = new Funcionarios();

            Equipamentos listaEquipamnetos = new Equipamentos();
```

Figura 11 Classe Program

2. Exceções (Exceptions)

Uma exceção é qualquer condição de erro ou comportamento inesperado encontrado por um programa em execução. Exceções podem ser lançadas devido a uma falha no seu código ou no código que você chama (como uma biblioteca compartilhada), recursos indisponíveis do sistema operacional, condições inesperadas que o tempo de execução encontra (como código que não pode ser verificado) e, portanto, em. Seu aplicativo pode se recuperar de algumas dessas condições, mas não de outras. Embora você possa recuperar da maioria das exceções de aplicativos, não é possível recuperar da maioria das exceções de tempo de execução.
Exceções oferecem vantagens sobre outros métodos de notificação de erro, como códigos de retorno. As falhas não passam despercebidas porque, se uma exceção é lançada e você não a trata, o tempo de execução finaliza seu aplicativo. Valores inválidos não continuam a se propagar pelo sistema como resultado de um código que falha na verificação de um código de retorno de falha.
A instrução "try" permite definir um bloco de código a ser testado quanto a erros enquanto ele está sendo executado.
A instrução "catch" permite definir um bloco de código a ser executado, se ocorrer um erro no bloco "try".
As exceções devem ter uma hierarquia, estando no topo as mais especificas e descendo para as mais genéricas.

```
public bool SaveFuncionarios()
{
    string filename = Path.GetFullPath("./SavedFiles/funcio.bin");
    if (File.Exists(filename))
    {
        try
        {
            Stream stream = File.Open(filename, FileMode.Create);
            BinaryFormatter bin = new BinaryFormatter();
            bin.Serialize(stream, funcionarios);
            stream.Close();
            return true;
        }
        catch (IOException e)
        {
            throw e;
        }
    }
    return false;
}
```

Figura 12 Exemplo do uso de exceções

3. Coleções (Collections)

As coleções contêm interfaces e classes que permitem aos programadores criar coleções fortemente tipadas (são de um tipo, por exemplo: tipo pessoa ou tipo carro) que fornecem melhor segurança e desempenho do tipo do que coleções não genéricas fortemente tipadas.
Neste trabalho foi usado "List<T> Class" ("T" indica o tipo de elemento na lista), presente no ""namespace: System.Collections.Generic ". É uma lista fortemente ligada de objetos á qual se pode ter acesso pelo índice. Fornece métodos para pesquisar, classificar e manipular a lista.
A classe List <T> é muito semelhante a ArrayList, ambas possuem funcionalidades semelhantes, mas a classe List <T> apresenta melhor desempenho na maioria dos casos e é do tipo mais seguro.

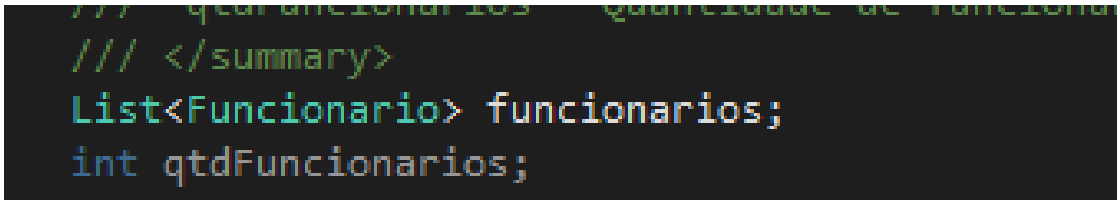


Figura 13 Exemplo de uso de uma List do tipo Funcionário

4. Ficheiros (Files)

Os ficheiros são um meio simples e pratico de guardar dados importantes numa aplicação. A "namespace: System.IO" fornece métodos estáticos para a criação, cópia, exclusão, deslocamento e abertura de um arquivo, além de ajudar na criação de objetos Muitos dos métodos de File retornam outros tipos de e/s quando você cria ou abre arquivos. Você pode usar esses outros tipos para manipular ainda mais um arquivo. Para obter mais informações, consulte membros específicos do File, como OpenText, CreateText ou Create.
Todos os métodos de File exigem o caminho para o arquivo que você está manipulando. Para usar ficheiros binários é preciso usar "using System.Runtime.Serialization.Formatters.Binary" para Serializar as classes e os atributos que devem ser guardados.

```
public bool SaveFuncionarios()
{
    string filename = Path.GetFullPath("./SavedFiles/funcio.bin");
    if (File.Exists(filename))
    {
        try
        {
            Stream stream = File.Open(filename, FileMode.Create);
            BinaryFormatter bin = new BinaryFormatter();
            bin.Serialize(stream, funcionarios);
            stream.Close();
            return true;
        }
        catch (IOException e)
        {
            throw e;
        }
    }
    return false;
}
```

Figura 14 Exemplo de uso de ficheiros binários

5. MVC (Model-View_Controller)

O padrão Model-View-Controller (MVC) separa um aplicativo em três grupos principais de componentes: Modelos, Vistas e Controladores. Esse padrão ajuda a alcançar a separação de preocupações. Usando esse padrão, as solicitações do usuário são roteadas para um Controller responsável por trabalhar com o Model para executar ações do utilizador ou recuperar resultados de consultas. O Controller escolhe a View para exibir ao usuário e fornece todos os dados do Model necessários.
O diagrama a seguir mostra os três componentes principais e quais referenciam os outros:

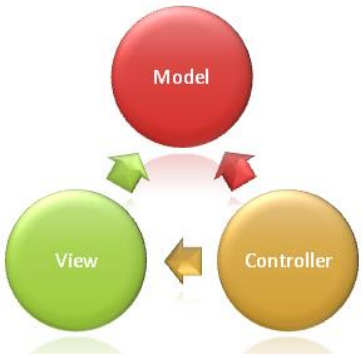


Figura 15 MVC

Essa definição de responsabilidades ajuda a dimensionar o aplicativo em termos de complexidade, porque é mais fácil codificar e testar algo (model, view ou controller) que possui uma única tarefa. É mais difícil atualizar e testar códigos com dependências espalhadas por duas ou mais dessas três áreas. Por exemplo, a lógica da interface do usuário tende a mudar com mais frequência do que a lógica comercial. Se o código de apresentação e a lógica de negócios forem combinados em um único objeto, um objeto contendo a lógica de negócios deverá ser modificado sempre que a interface com o usuário for alterada. Isso geralmente introduz erros e requer um novo teste da lógica de negócios após cada alteração mínima na interface do usuário.

- Responsabilidades do Model

O Model em um aplicativo MVC representa o estado do aplicativo e qualquer lógica ou operação comercial que deve ser executada por ele. A lógica de negócios deve ser encapsulada no modelo, juntamente com qualquer lógica de implementação para persistir o estado do aplicativo. As visualizações fortemente tipadas geralmente usam os tipos ViewModel projetados para conter os dados a serem exibidos nessa exibição. O controlador cria e preenche essas instâncias do ViewModel a partir do modelo.

- Responsabilidades da View

A view é responsável por apresentar o conteúdo por meio da interface ao utilizador. Deve haver lógica mínima nas visualizações, e qualquer lógica nelas deve estar relacionada à apresentação de conteúdo. Se for necessário executar uma grande quantidade de lógica nos arquivos de exibição para exibir dados de um model complexo, considere usar um View View Component, ViewModel ou view template para simplificar a exibição.

- Responsabilidades do Controller

Controllers são os componentes que lidam com a interação do utilizador, trabalham com o model e, finalmente, selecionam uma view para visualizar. Em um aplicativo MVC, a view exibe apenas informações; o controller lida e responde à entrada e interação do utilizador. No padrão MVC, o controller é o ponto de entrada inicial e é responsável por selecionar quais tipos de modelo trabalhar e qual visualização mostrar (daí o nome - ele controla como o aplicativo responde a uma determinada solicitação).

6. Conclusão

6.1. Lições aprendidas

Os conhecimentos mais técnicos e teóricos sobre RGPD como algumas normas, legislações e como deve ser implementado ajudou muito a compreender e adquirir mais conhecimento sobre esta área que é a privacidade e proteção de dados pessoais. Isso ajudou muito a ter um olhar mais voltado ao mercado e á área de tecnologia com suas revoluções.
Na esfera da prática, as habilidades e conhecimentos ao desenvolver o programa, revisando e implementando novos cálculos foram aperfeiçoadas.

6.2. Apreciação Final

A experiência propiciou a real prática das teorias vivenciadas em sala de aula, atribuindo mais gosto pelas disciplinas além de evoluir o raciocínio para adaptar propostas ao desenvolvimento. Na realização deparamos com alguns problemas na integração de conceitos mais complexos como programação por camadas. Os objetivos mais básicos foram concluídos com sucesso.