



INSTITUTO POLITÉCNICO
DO CÁVADO E DO AVE
ESCOLA SUPERIOR DE TECNOLOGIA

RELATÓRIO DE TRABALHO PRÁTICO 1

Sistema de Registo de Auditorias V.3

CARLOS SANTOS

ALUNO Nº 19432

RÚBEN SILVA

ALUNO Nº 19433

Trabalho realizado sob a orientação de:
Luís Ferreira

Linguagens de Programação II

Licenciatura em Engenharia de Sistemas Informáticos

Barcelos, maio de 2020

Índice

1	IMPLEMENTAÇÃO	1
1.1	Descrição do Problema	1
1.2	Solução e atualização	1
2	ESTADO DE ARTE	2
3	BIBLIOTECAS	3
3.1	a1_TrabalhoPratico	3
3.2	a2_RegrasNegocio	5
3.3	a3_DadosClasses	5
3.4	c1_ObjetoNegocio	10
4	CONCLUSÃO	14
	BIBLIOGRAFIA	15

Lista de Figuras

Figura 1: a1_TrabalhoPratico <i>variáveis</i>	3
Figura 2: a1_TrabalhoPratico <i>saves</i>	4
Figura 3: a1_TrabalhoPratico <i>gravar</i>	4
Figura 4: a2_RegrasNegocio Função com Verificação	5
Figura 5: a2_DadosClasses <i>ExisteAuditoria(...)</i>	5
Figura 6: a2_DadosClasses <i>QuantidadeAuditorias(...)</i>	6
Figura 7: a2_DadosClasses <i>AdicionaVulnerabilidade(...)</i>	6
Figura 8: a2_DadosClasses <i>EditaAuditoria(...)</i>	7
Figura 9: a2_DadosClasses <i>ApresentaAuditoriasColaborador(...)</i>	8
Figura 10: <i>Utilização da função ApresentaAuditoriasColaborador</i>	8
Figura 11: a3_DadosClasses <i>ApresentaAuditoriasOrdenadasVuln(...)</i>	8
Figura 12: a3_DadosClasses <i>GuardarAuditorias(...)</i>	9
Figura 13: a2_DadosClasses <i>VerificarAtividade(...)</i>	9
Figura 14: a2_DadosClasses <i>TornarColaboradorInativo(...)</i>	9
Figura 15: a2_DadosClasses <i>EquipamentosCompleto</i>	10
Figura 16: a2_DadosClasses <i>AuditoriaCompleta</i>	10
Figura 17: c1_ObjetoNegocio <i>Auditoria</i>	11
Figura 18: c1_ObjetoNegocio <i>Colaborador</i>	11
Figura 19: c1_ObjetoNegocio <i>Equipamento</i>	12
Figura 20: c1_ObjetoNegocio <i>Vulnerabilidades</i>	13
Figura 21: c1_ObjetoNegocio <i>Pessoa</i>	13

1 Implementação

1.1 Descrição do Problema

Necessidade de criar um programa que auxilie colaboradores de uma empresa na criação de auditorias internas. Sendo que este deve oferecer uma vasta quantidade de funcionalidades para o utilizador, tais como, a inserção e edição de colaboradores e vulnerabilidades, a remoção de colaboradores, a edição das vulnerabilidades de uma auditoria, entre outras. Para mostrar resultados e dados, o programa deve também dar a possibilidade de apresentar diferentes listas, como uma lista onde as auditorias estão ordenadas pela quantidade de vulnerabilidades ou então numa auditoria, estas vulnerabilidades estão agrupadas por nível de impacto. Também devem ser apresentados os detalhes de um colaborador mostrando as auditorias realizadas por este, e também mostrar as vulnerabilidades identificadas num equipamento.

Por fim, deve também ser apresentado uma tabela geral onde devem ser comparadas todas as auditorias mostrando a que tem mais e a que tem menos vulnerabilidades, a quantidade de auditorias e a média de vulnerabilidade em todas as auditorias registadas.

1.2 Solução e atualização

Para solucionar o problema, na terceira e última fase da realização do programa, foi-lhe aprimorada a estrutura final ao remover uma camada auxiliar não necessária, de modo a corresponder ao estilo de programação por camadas Ntier. Foram também adicionadas classes auxiliares e listas dessas classes para permitir o output exclusivo na camada mais superficial. Todas as exceptions foram revistas para que funcionem corretamente e que o output de erro seja realizado apenas no main da camada mais superficial. Todas as funções que permitem a edição de objetos foram corrigidas de modo a receber apenas os atributos a serem alterados e os códigos respetivos. Foram implementadas algumas funções LINQ para simplificar operações.

O programa conta com opções de adição, edição, remoção e apresentação de auditorias, colaboradores, equipamentos e vulnerabilidades.

2 Estado de Arte

O Regulamento Geral sobre a Proteção de Dados entrou em vigor em 25 de maio de 2018 e é um regulamento do direito europeu sobre privacidade e proteção de dados pessoais, aplicável a todos os indivíduos na União Europeia, regulamenta também os dados pessoais exportados para fora da União Europeia.

Este regulamento obriga a informar acerca da base legal para o tratamento de dados, prazo de conservação dos mesmos e a sua transferência.

O regulamento obriga a controlar as circunstâncias em que foi obtido o consentimento dos titulares quando isso for base legal do tratamento dos dados pessoais. Existem um conjunto de exigências para obtenção desse consentimento e o seu não cumprimento obriga à obtenção de um novo consentimento.

O regulamento obriga a um grande controlo do risco associado ao possível roubo de informação. Este controlo de risco deverá passar a ser garantido por medidas de segurança efetivas que garantam a confidencialidade, a integridade dos dados e que previnam a destruição, ou a divulgação/acesso não autorizado de dados.

Existem DPO's – Data Protection Officer – (Delegado de Proteção de Dados) que são um canal de comunicação entre uma determinada empresa e uma agência de Proteção de Dados e os titulares dos dados pessoais e têm como função auxiliar a empresa no processo de adaptação e estruturação de um programa com foco na proteção de dados, que tem de se manter a par de todas as obrigações do RGPD. Estes podem usar várias aplicações que os ajudam nesta tarefa, como por exemplo, a GDPR App.

3 Bibliotecas

As bibliotecas deste trabalho foram implementadas de forma a respeitar a programação por camadas Ntier, e por isso existem 3 camadas que foram apelidadas de “ax_” como prefixo para facilitar a sua identificação. Assim existe “a1_TrabalhoPratico” que se trata da camada mais superficial, *Presentation Layer*, de seguida, “a2_RegrasNegocio” que tem como objetivo impor regras, segurança e validações, *Business Rules Layer*. Por fim, existe a camada mais profunda “a3_DadosClasses” que tem como objetivo manusear os dados das classes, *Data Layer*.

Existem também uma camada auxiliar necessária que tem como prefixo “c1_” que é “c1_ObjetosNegocio” que contém todas as classes que servem de objeto de negócio, classes de auxílio a transporte de dados entre camadas e a interface de cada classe de objeto de negócio.

3.1 a1_TrabalhoPratico

Nesta parte final é onde se encontra o *main* e onde são implementados todos os métodos que o programa oferece. A função principal *main* não se encontra propriamente organizada visto que o seu único propósito é demonstrar e aplicar as possibilidades que o programa oferece a partir dos seus métodos.

Aqui serão criadas as algumas variáveis auxiliares para obter informação:

```
#region VARIÁVEIS
int quant, opcao = 0;

AuditoriaAux amaior;
AuditoriaAux amenor;
Auditoria a = new Auditoria();
Colaborador c = new Colaborador();
Equipamento e = new Equipamento();
Vulnerabilidade v = new Vulnerabilidade();
EquipamentoAux e2;

List<int> vulnsAux;
List<ColaboradorAux> colAux;
List<AuditoriaAux> audAux;
List<EquipamentoAux> equAux;
List<VulnerabilidadeAux> vulAux = new List<VulnerabilidadeAux>();
#endregion
```

Figura 1: a1_TrabalhoPratico variáveis

Ao arranque da aplicação são carregados os **ficheiros binários (saves)**:

```
#region CARREGAMENTOS
try
{
    EquRegras.CarregarEquipamentos(@"..\..\Ficheiros\Equipamentos");
}
catch (IOException x)
{
    Console.WriteLine(x.Message);
}
catch (Exception x)
{
    Console.WriteLine(x.Message);
}
```

Figura 2: a1_TrabalhoPratico *Exemplo de save*

Ao encerrar a aplicação dá a opção de serem guardadas todas as modificações.

```
#region CARREGAMENTOS
try
{
    EquRegras.CarregarEquipamentos(@"..\..\Ficheiros\Equipamentos");
}
catch (IOException x)
{
    Console.WriteLine(x.Message);
}
catch (Exception x)
{
    Console.WriteLine(x.Message);
}
```

Figura 3: a1_TrabalhoPratico *gravar*

3.2 a2_RegrasNegocio

```
public static bool AdicionaVulnerabilidadeAuditoria(int cod, int codv)
{
    try
    {
        return Auditorias.AdicionaVulnerabilidade(cod, codv);
    }
    catch (IndexOutOfRangeException x)
    {
        throw new FormatException("ERRO: " + x.Message);
    }
    catch (Exception x)
    {
        throw new Exception("ERRO: " + x.Message);
    }
}
```

Figura 4: a2_RegrasNegocio Função com Verificação

Quase todas as funções presentes nesta biblioteca são responsáveis pela verificação das regras e exceções antes das variáveis prosseguirem para a camada dos dados. Estas atuam nas funções de a2_RegrasNegocio: das auditorias, dos colaboradores, dos equipamentos e das vulnerabilidades.

3.3 a3_DadosClasses

Nesta biblioteca, todos os dados são manipulados e não acessíveis pelo utilizador.

Estes dados estão todos guardados em estruturas de dados *Collections Generic* do tipo *List<Type>*.

Assim, nesta biblioteca, estão presentes funções focadas em:

Verificar a existência de algum objeto:

```
2 references
public static bool ExisteAuditoria(int cod)
{
    return aud.Exists(a => a.a.Codigo == cod);
}
```

Figura 5: a3_DadosClasses *ExisteAuditoria(...)*

Neste caso, a função é um exemplo da utilização de LINQ, a função verifica se existe uma auditoria com o código recebido como parâmetro, se existir, retornará true, se não, retornará false.

Obter quantidade de objetos numa lista:

```
public static int QuantidadeAuditorias()  
{  
    return aud.Count;  
}
```

Figura 6: a3_DadosClasses *QuantidadeAuditorias(...)*

Esta função simplesmente retorna a quantidade de objetos que existe na lista “aud”.

Adicionar/Remover algum objeto:

```
public static bool AdicionaVulnerabilidade(int cod, int codv)  
{  
    try  
    {  
        foreach (AuditoriaCompleta ac in aud)  
            if (cod == ac.a.Codigo)  
            {  
                ac.codVulns.Add(codv);  
                return true;  
            }  
        return false;  
    }  
    catch (IndexOutOfRangeException x)  
    {  
        throw new FormatException(x.Message);  
    }  
    catch (Exception x)  
    {  
        throw new Exception(x.Message);  
    }  
}
```

Figura 7: a3_DadosClasses Auditorias.*AdicionaVulnerabilidade(...)*

Esta função recebe como parâmetro um código de uma auditoria e também um código de uma vulnerabilidade. Depois percorre toda a lista de objetos e adiciona a vulnerabilidade a uma auditoria quando esta for encontrada na lista. Esta função tem a verificação de exceptions devido à sua importância.

Editar algum dado:

```
public static bool EditaAuditoria(int cod, DateTime dataRegisto, int duracao)
{
    try
    {
        if (ExisteAuditoria(cod) == false) return false;
        for (int i = 0; i < aud.Count; i++)
            if (aud[i].a.Codigo == cod)
            {
                aud[i].a.DataRegisto = dataRegisto;
                aud[i].a.Duracao = duracao;
            }
        return true;
    }
    catch (IndexOutOfRangeException x)
    {
        throw new FormatException(x.Message);
    }
    catch (Exception x)
    {
        throw new Exception(x.Message);
    }
}
```

Figura 8: a3_DadosClasses *EditaAuditoria(...)*

Neste tipo de função, a lista é percorrida até encontrar alguma com o mesmo código do objeto que recebeu como parâmetro, se encontrar alguma com o mesmo código, os dados serão editados, senão, retornará false;

Apresentar algum dado:

```
public static List<AuditoriaAux> ApresentaAuditoriasColaborador(int codc)
{
    var audc = aud.Where(aud => aud.a.CodColab == codc);
    foreach (AuditoriaCompleta ac in audc)
    {
        AuditoriaAux aAux = new AuditoriaAux(ac.a.Codigo, ac.a.Duracao, ac.a.DataRegisto, ac.a.CodColab, ac.a.CodEqui);
        audAux.Add(aAux);
    }
    if (audc.Count() == 0)
        return null;
    return audAux;
}
```

Figura 9: a3_DadosClasses *ApresentaAuditoriasColaborador(...)*

```
Console.WriteLine(" Auditorias por Colaborador:");
audAux = AudRegras.ApresentaAuditoriasColaborador(1);
Console.WriteLine(" Código Colaborador: {0}", 1.ToString());
Console.WriteLine(" Quantidade auditorias: {0}", audAux.Count.ToString());
if (audAux == null)
    Console.WriteLine("- Este Colaborador não tem auditorias registadas.");
foreach (AuditoriaAux aAux in audAux)
{
    Console.WriteLine("\nCódigo da Auditoria: {0}", aAux.Codigo.ToString());
    Console.WriteLine("Data: {0}", aAux.DataRegisto.ToShortDateString());
    Console.WriteLine("Equipamento Associado: {0}", aAux.CodEqui.ToString());
    Console.WriteLine("Quantidade Vulnerabilidades: {0}", AudRegras.ObterQuantidadeVulnerabilidadesAuditoria(aAux.Codigo).ToString());
}
```

Figura 10: a1_TrabalhoPratico *Utilização da função ApresentaAuditoriasColaborador*

A função *ApresentaAuditoriasColaborador* cria uma lista auxiliar que contém um conjunto de auditorias associadas ao código do colaborador recebido como parâmetro, depois no main a função é executada e a lista criada é percorrida e são apresentadas as auditorias.

Existe também a variação deste código que apresentará os dados ordenados e tabelados, neste caso por ordem decrescente de vulnerabilidades

```
public static List<AuditoriaAux> ApresentaAuditoriasOrdenadasVuln()
{
    var auditoriasOrdenadas = from a in aud
                              orderby a.codVulns.Count descending
                              select a;
    foreach (AuditoriaCompleta ac in auditoriasOrdenadas)
    {
        AuditoriaAux aAux = new AuditoriaAux(ac.a.Codigo, ac.a.Duracao, ac.a.DataRegisto, ac.a.CodColab, ac.a.CodEqui);
        audAux.Add(aAux);
    }
    if (auditoriasOrdenadas.Count() == 0)
        return null;
    return audAux;
}
```

Figura 11: a3_DadosClasses *ApresentaAuditoriasOrdenadasVuln(...)*

Esta função foi melhorada desde a segunda fase passando a utilizar uma função LINQ para criar uma lista auxiliar com as auditorias ordenadas por ordem decrescente do parâmetro "codVulns" e dar *return* a esta mesma lista.

Guardar/Carregar algum dado em ficheiro binário:

```
public static bool GuardarAuditorias(string fileName)
{
    try
    {
        Stream stream = File.Open(fileName, FileMode.Create);
        BinaryFormatter bin = new BinaryFormatter();
        bin.Serialize(stream, aud);
        stream.Close();
        return true;
    }
    catch (IOException x)
    {
        throw new IOException(x.Message);
    }
    catch (Exception x)
    {
        throw new Exception(x.Message);
    }
}
```

Figura 12: a3_DadosClasses *GuardarAuditorias(...)*

Esta função, neste caso, guarda toda a informação que se encontra armazenada na lista “aud”, recebendo como argumento o diretório de onde o ficheiro deve ser guardado.

Verificar/alterar a atividade de um colaborador:

Um colaborador, em momento algum, deve ser removido dos registos da aplicação, para isso foram implementadas duas funções, uma para verificar atividade, e a outra para alterar a sua atividade:

```
public static bool VerificaAtividade(int cod)
{
    foreach (Colaborador c in col)
        if (c.Codigo == cod)
            if (c.Atividade == Atividade.ATIVO) return true;
    return false;
}
```

Figura 13: a3_DadosClasses *VerificaAtividade(...)*

```
public static bool TornarColaboradorInativo(int cod)
{
    foreach (Colaborador c in col)
        if (c.Codigo == cod && c.Atividade == Atividade.ATIVO)
        {
            c.Atividade = Atividade.INATIVO;
            return true;
        }
    return false;
}
```

Figura 14: a3_DadosClasses *TornarColaboradorInativo(...)*

Estas duas funções manipulam o *enum* relativo à classe Colaborador. Recebem como parâmetro o código do colaborador e verificam o estado do *enum*, no caso da segunda função altera o estado do *enum* para Inativo.

Nesta biblioteca nomeadamente nos ficheiros *Equipamentos.cs* e *Auditorias.cs* existem também duas classes auxiliares “AuditoriaCompleta” e “EquipamentoCompleto”, que servem para complementar as classes singulares “Auditoria” e “Equipamento” com uma lista de inteiros com códigos de vulnerabilidades.

```
class EquipamentoCompleto
{
    #region ATRIBUTOS
    public Equipamento e;
    public List<int> codVulns;
    #endregion

    #region CONSTRUTORES
    /// <summary> Construtores
    1 reference
    public EquipamentoCompleto(Equipamento e)
    {
        this.e = e;
        codVulns = new List<int>();
    }

    0 references
    public EquipamentoCompleto(Equipamento e, int codV)
    {
        this.e = e;
        codVulns = new List<int>();
        codVulns.Add(codV);
    }
    #endregion
}
```

```
class AuditoriaCompleta
{
    #region ATRIBUTOS
    public Auditoria a;
    public List<int> codVulns;
    #endregion

    #region CONSTRUTORES
    /// <summary> Construtores
    1 reference
    public AuditoriaCompleta(Auditoria a)
    {
        this.a = a;
        codVulns = new List<int>();
    }

    0 references
    public AuditoriaCompleta(Auditoria a, int codV)
    {
        this.a = a;
        codVulns = new List<int>();
        codVulns.Add(codV);
    }
    #endregion
}
```

Figura15: a3_DadosClasses *EquipamentosCompleto*

Figura 16: a3_DadosClasses *AuditoriaCompleta*

Estas listas são usadas para armazenar a informação quando existe uma auditoria ou um equipamento com todos os dados, isto acontece para que ao serem criados estes objetos não existam atributos criados, mas sem informação e para que não sejam manipuladas listas a partir da camada mais superficial.

3.4 c1_ObjetoNegocio

Nesta biblioteca do programa são criadas todas as classes designadas por objetos de negócio. Cada classe tem o seu construtor por defeito e construtor com valores vindos do exterior criados, e também cada atributo destas classes tem a sua propriedade criada.

São criadas também as interfaces relativas a cada classe. A classe “Colaborador” tem como herança a classe Pessoa.

Auditoria:

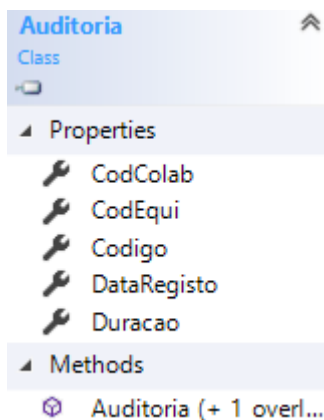


Figura 17: c1_ObjetoNegocio *Auditoria*

codigo-> Código da auditoria;

duracao-> Duração da auditoria;

codColab-> Código do colaborador responsável;

codEqui-> Código do Equipamento em questão;

dataRegisto-> Data da realização da auditoria.

Colaborador:

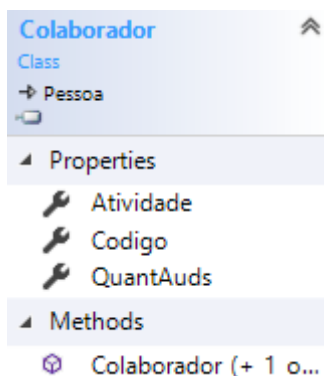


Figura 18: c1_ObjetoNegocio *Colaborador*

codigo-> Código do colaborador;

quantAuds-> Quantidade de Auditorias;

atividade-> Estado de atividade do colaborador (Ativo/Inativo);

Equipamento:

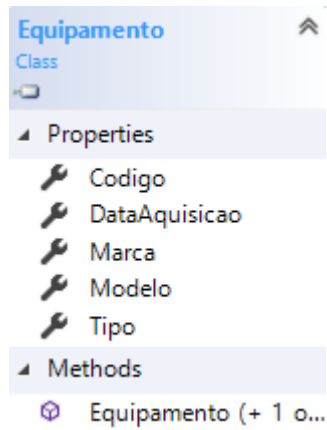


Figura 19: c1_ObjetoNegocio *Equipamento*

codigo-> Código do Equipamento;

tipo-> Tipo de Equipamento;

marca-> Marca do Equipamento;

modelo-> Modelo do Equipamento;

dataAquisicao-> Data de aquisição do Equipamento;

Vulnerabilidades:

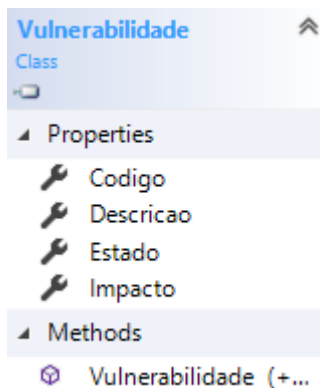


Figura 20: c1_ObjetoNegocio *Vulnerabilidades*

codigo-> Código da Vulnerabilidade;

descricao-> Descrição da Vulnerabilidade;

impacto-> Nível de impacto da Vulnerabilidade;

estado-> Estado de resolução da Vulnerabilidade;

Pessoa:

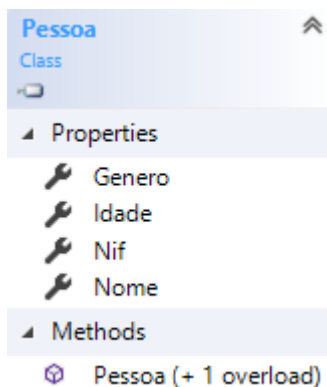


Figura 21: c1_ObjetoNegocio *Pessoa*

Esta classe serve apenas de herança para a classe Colaborador.

idade-> Idade do colaborador;

nif-> Nif do colaborador;

nome-> Nome do colaborador;

genero-> Género do colaborador;

4 Conclusão

A construção do programa nesta terceira fase encontra-se concluído dentro dos objetivos solicitados no enunciado do trabalho, encontra-se visualmente agradável e de forma organizada. O código tem os comentários necessários para que qualquer pessoa consiga facilmente perceber do que cada parte se trata.

Com esta parte do trabalho prático, as nossas capacidades relativas ao paradigma POO, ao LINQ, às *Collections Generics*, ao Ntier foram aprimoradas.

O programa encontra-se alojado no seguinte repositório GitHub:
https://github.com/carlosantos2103/19432_19433_LP2

Bibliografia

<https://mydataprivacy.eu/o-regulamento/>

<https://eur-lex.europa.eu/legal-content/PT/TXT/HTML/?uri=OJ:L:2016:119:FULL>

<https://gdprapp.com/>