



## **WesterosFit**

Licenciatura em Gestão de Sistemas de Informação

Desenvolvimento de Aplicações I

2ºAno| 1º Semestre | Ano lectivo 2018 /2019

### **Discentes:**

Pedro Duarte, nº170323017

João Pereira, nº 170323032

## Índice

Índice de Ilustrações	3
Introdução	4
Diagrama de Classes	5
Classe Membro	6
Classe Individual: Membro	7
Classe Grupo: Membro	7
Classe Bran: Individual	8
Classe Jaime: Individual	8
Classe Brienne: Grupo	9
Classe Ned: Grupo	9
Classe BDMembros	9
listaDeMembros	10
Método ObterMembroPorID()	10
Design da Interface	11
Manual de Utilização	17
Criação de um novo membro	17
Consultar/Editar dados dos membros criados	18
Ativar/Desativar Personal Trainer	18
Ativar/Desativar SPA	18
Ativar/Desativar Membro	19
Reativar Membro Punido	19
Adicionar Presenças	19
Atribuir Desconto	19
Adicionar Exercícios	20
Adicionar Calorias Por Sessão	20
Adicionar Minutos de Meditação	20
Adicionar Peso Máximo Levantado	21
Adicionar Duração da Aula	21
Adicionar Dias Desde Da Última Presença	21
Calcular Mensalidade	22
Verificar Dedicação	22
Propriedade <i>static</i>	23
Criação de Listas	24

<i>Enumerator</i>	24
DataGridView	25
Datasource	25
Conclusão	26
Bibliografia	27

## Índice de Ilustrações

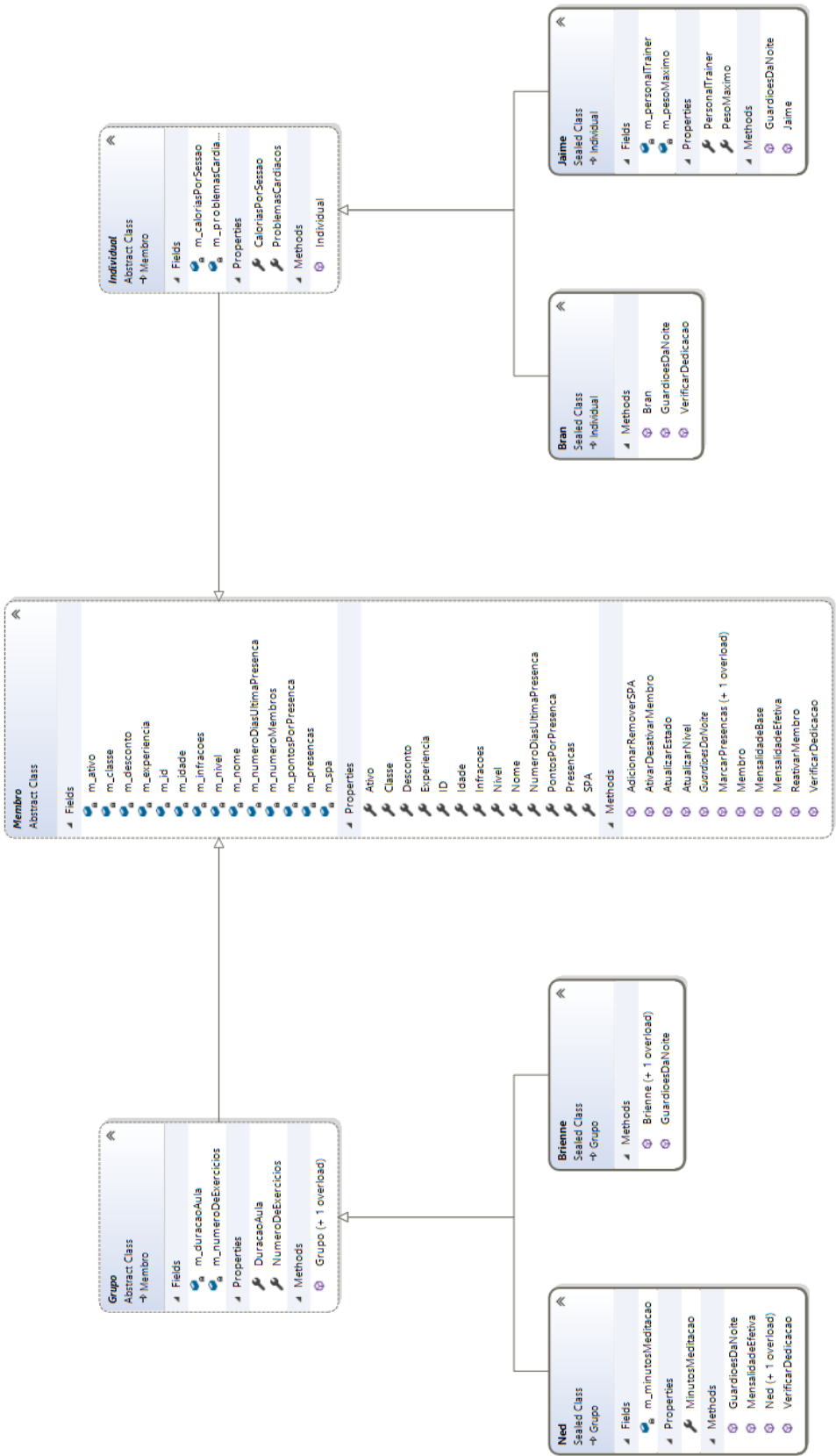
Figura 1 Login	11
Figura 2 Janela Inicial	11
Figura 3 Escolha do tipo de membro	12
Figura 4 Adicionar novo Bran	12
Figura 5 Adicionar novo Brienne	13
Figura 6 Adicionar novo Ned	13
Figura 7 Adicionar novo Jaime	14
Figura 8 Consultar/Editar Bran	14
Figura 9 Consultar/Editar Brienne	15
Figura 10 Consultar/Editar Ned	15

## Introdução

No âmbito do regime de avaliação continua da unidade curricular de Desenvolvimento de Aplicações I, foram realizados 9 exercícios que visavam a aplicação de um capítulo por exercício lecionado nas aulas teóricas. Por fim, foi nos proposto a realização deste trabalho que tem como objetivo a aplicação de toda a matéria lecionada e dando ainda a oportunidade de adicionar conteúdos não lecionados que o grupo achasse pertinente.

Este trabalho consiste na elaboração completa de um programa para a gestão de informação sobre os membros de um ginásio com um tema inovador, o “WesterosFit”. Para a sua realização foi utilizado o programa “Microsoft Visual Studio” com a linguagem C#.

# Diagrama de Classes



## Classe Membro

Membro foi o nome escolhido para a classe mãe de todo o projeto, esta classe contém todas as propriedades e métodos que são obrigatoriamente comuns a todos os membros que sejam criados na aplicação. Não deverá ser permitido a criação de membros com esta classe, assim sendo esta está definida como *abstract*.

Para as propriedades que não necessitam de ser alteradas pelo código cliente foi apenas atribuída a propriedade *get*. Quando é necessário fazer essa alteração a propriedade *set* é também incluída, como foi o caso dos *PontosPorPresenca* para que sejam atribuídos diferentes pontos por presença às diferentes subclasses, *Desconto* para que possa ser atribuído um desconto em euros e *Infracoes* para que possam ser atribuídas as respetivas infrações nos métodos *VerificarDedicacao()*.

Todos os métodos são *public* uma vez que são necessários utilizar pelas suas subclasses.

Para o método *VerificarDedicacao()* foi decidido que seria *virtual* para que seja possível às subclasses fazer *override* uma vez que em função da classe a dedicação pode ter objetivos adicionais a cumprir.

No caso do método *GuardioesDaNoite()* chegamos a conclusão que deveria ser *abstract* para que seja obrigatório as subclasses fornecerem um código específico para este método.

O método *MarcarPresencas()* pode ser invocado de duas maneiras, caso não seja fornecido o nº de presenças este método adiciona uma presença ao membro, se for fornecido um nº de presenças (Ex.: *MarcarPresencas(12)*), o método adiciona o nº de presenças que foi indiciado, a isto dá-se o nome de *overloading*.

O construtor da classe Membro, exige que sejam fornecidos alguns argumentos, caso contrário não é possível proceder à sua criação. São eles: nome e idade. Além da atribuição dos valores guardados nos argumentos aos respetivos campos, o construtor inicializa os campos *m\_nivel*, *m\_experiencia*, *m\_numeroDiasUltimaPresenca*, *m\_presencas* e *m\_desconto* a 0, e por fim, *m\_ativo* com o valor *true*.

Ainda no construtor, o campo *m\_ID* recebe o valor que está no campo *static m\_numeroMembros* que neste caso permitiu que fosse atribuído um número diferente (ID) a cada membro, garantindo assim que em situação alguma existam dois membros com as mesmas características. Caso o campo *m\_numeroMembros* não fosse *static* para cada novo membro criado, o campo seria novamente inicializado resultando num ID = 0 para todos.

## Classe Individual: Membro

A classe Individual é uma subclasse do Membro, assim sendo herda todas as características da sua classe mãe. Tal como a sua classe mãe esta é também *abstract*. O seu nome, Individual, faz referência ao facto de os membros a que a esta classe pertencem praticarem atividade física individualmente.

Além das que já possui por herança esta classe possui duas novas propriedades são elas, *CaloriasPorSessao* e *ProblemasCardiacos*, que por sua vez serão herdadas pelas suas subclasses Jaime e Bran.

Esta classe não tem métodos adicionais ao que já lhe são fornecidos por herança.

O construtor desta classe além dos argumentos já impostos pela sua classe mãe exige ainda que seja fornecida a informação sobre a saúde cardíaca do membro. É inicializado o campo *m\_caloriasPorSessao* a 200 e no campo *m\_problemasCardiacos* regista se o membro tem ou não histórico de problemas cardíacos, que por predefinição se encontra a false no formulário.

## Classe Grupo: Membro

A classe Grupo é uma subclasse do Membro, assim sendo herda todas as características da sua classe mãe. Tal como a sua classe mãe esta é também *abstract*. Grupo foi o nome escolhido uma vez que os seus pertencentes praticam atividade física em conjunto com outros membros.

Por herança esta classe já possui as propriedades da classe Membro além dessas possui outras adicionais que serão por sua vez herdadas pelas suas subclasses *Brienne* e *Ned* são elas *DuracaoAula* e *NumeroExercicios*.

Esta classe não tem métodos adicionais ao que já lhe são fornecidos por herança.

O construtor desta classe inicializa o campo *m\_duracaoDaAula* a 0. Foi usado overloading neste construtor para permitir que no caso de não ser fornecido o nº de exercícios como argumento, este seja inicializado a 1 e no caso deste valor ser fornecido o campo *m\_numeroDeExercicios* receberá esse mesmo valor.

## Classe Bran: Individual

A classe Bran é uma subclasse da classe Individual, assim sendo herda todas as características da sua classe mãe e as características que por esta já foram herdadas. Esta classe é *public sealed* para que não seja permitido criar subclasses a partir desta.

Tanto o método *GuardioesDaNoite()* como o *VerificarDedicacao()* são *override* nesta classe. No primeiro método é apenas atribuído *true*, no segundo é adicionado a condição de superar as 400 calorias por sessão como referido no enunciado.

O seu construtor atribui às propriedades herdadas, *PontosPorPresenca* e *Classe*, 3 e “Bran” respetivamente a propriedade *ProblemasCardiacos* que como as anteriores foi também herdada da classe Individual, recebe por sua vez a informação da checkbox presente no formulário que por predefinição se encontra em branco.

## Classe Jaime: Individual

A classe Jaime é uma subclasse da classe Individual para os especializados em levantamento de pesos e halteres, assim sendo herda todas as características da sua classe mãe e as características que por esta já foram herdadas. Esta classe é também *public sealed*.

Além das que já possui por herança esta classe acresce 2 novos campos, são eles: *m\_pesoMaximo* e *m\_personalTrainer*.

O método *GuardioesDaNoite()* é *override*, adicionando a condição do nível dos membros desta classe necessitar de ser superior ou igual a 5 e ainda o peso máximo ser superior a 100 para que o membro tenha acesso noturno ao ginásio .

O seu construtor atribui 5 à propriedade *PontosPorPresenca* que foi herdada bem como a propriedade *Classe* atribuindo-lhe “Jaime”. O campo *m\_personalTrainer* recebe *true* por predefinição para todos os membros, para desativar o personal trainer deve ser feito na janela “Consultar/Editar membro”. Por fim o campo *m\_pesoMaximo* é inicializado a 0.



## Classe Brienne: Grupo

A classe Brienne é uma subclasse do Grupo para os interessados em dança e aulas de grupo, assim sendo herda todas as características da sua classe mãe e as características que por esta já foram herdadas. Esta classe é também *public sealed*.

Sendo obrigatório o método *GuardioesDaNoite()* é *override* atribuindo a condição obrigatória do nível do membro ser superior ou igual a 3 caso contrário não é considerado um Guardião da noite.

No construtor desta classe bem como no método *MarcarPresencas()* é utilizada a propriedade *overloading* para que no caso de ser fornecido o nº de exercícios este seja registado e caso não seja, ainda assim o membro seja criado. Além de registar o nº de exercícios, atribui 2 à variável *PontosPorPresenca* e “Brienne” à classe.

## Classe Ned: Grupo

A classe Ned é uma subclasse do Grupo para quem a cabeça está primeiro (mindfulness & meditation), assim sendo herda todas as características da sua classe mãe e as características que por esta já foram herdadas. Esta classe é também *public sealed*.

Os membros Ned não podem ser Guardiões da noite logo o respetivo método é *override* retornando em qualquer circunstância, *false*. Não só este, mas também o método *VerificarDedicacao()* é *override* adicionando duas condições : minutos de meditação superior ou igual a 200 e nº de exercícios superior ou igual a 5.

Uma vez mais foi utilizado o *overloading* no construtor desta classe pela mesma lógica da classe Brienne. Para os membros Ned o desconto na mensalidade é igual à idade assim sendo a propriedade *desconto* recebe o valor que foi recebido para a idade, à propriedade *Classe* é atribuído “Ned” e por último à *PontosPorPresenca*, 1.

## Classe BDMembros

BDMembros foi o nome escolhido para esta classe, esta classe contém a *ListaDeMembros* onde irão ser guardados todos os membros criados e o método *ObterMembroPorID*. É também necessário que esta classe seja *static*, uma vez que é pretendido que todos os membros acedam à mesma instância da classe e não a uma diferente para cada membro.

## listaDeMembros

Foi criada uma *List<Membro>*, com o nome *listaDeMembros*. Nesta aplicação irá servir como base de dados para armazenar os membros criados. Esta funcionalidade é descrita num tópico próprio mais a frente.

Cada membro criado na *listaDeMembros*, terá associado um *Enumerator*, um género de como um índice (como utilizávamos nos *arrays*).

## Método ObterMembroPorID()

Este método tem como pressuposto de procurar na *listaDeMembros*, o membro criado anteriormente com o id de criação. O id que será obtido pelo número da linha selecionada na tabela, irá entrar como parâmetro no método de forma a encontrar o membro criado. Quando o *enumerator* for igual ao id, o membro com esse id é devolvido. Se por ventura o id não corresponder a nenhum *Enumerator*, não será devolvido nenhum membro.

## Design da Interface

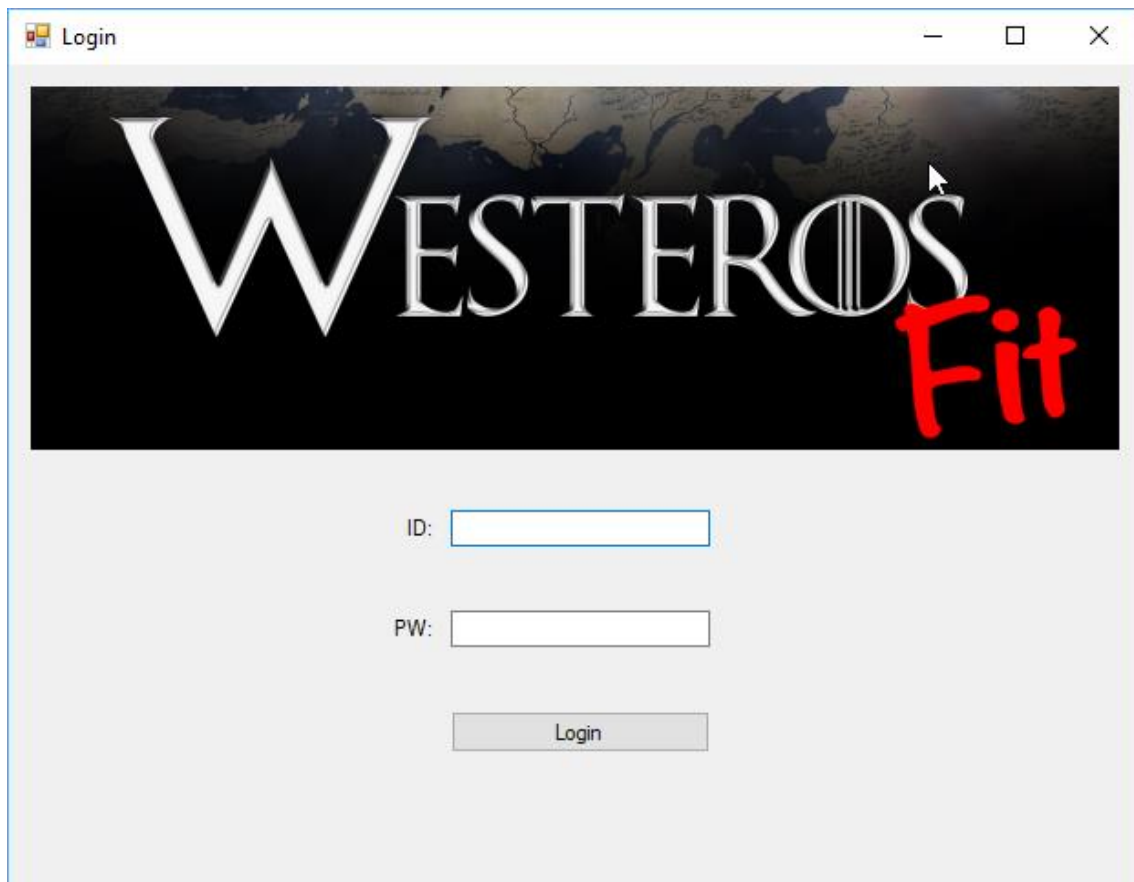



Figura 1 Login



Figura 2 Janela Inicial

Escolha do tipo de membro









<p><b>BRAN</b></p> 	<p><b>Cardiofitness</b></p> <p>Pontos por Presença: 3</p> <p><u>Requisitos de permanência:</u> 10 presenças 400 calorias por sessão</p>
<p><b>BRIENNE</b></p> 	<p><b>Dança e Aulas de Grupo</b></p> <p>Pontos por Presença: 2</p> <p><u>Requisitos de permanência:</u> 10 presenças</p>
<p><b>NED</b></p> 	<p><b>Mindfulness e Meditation</b></p> <p>Pontos por Presença: 1</p> <p><u>Requisitos de permanência:</u> 10 presenças 200 minutos de meditação ou 5 exercícios por aula</p>
<p><b>JAIME</b></p> 	<p><b>Levantamento de pesos e halteres</b></p> <p>Pontos por Presença: 5</p> <p><u>Requisitos de permanência:</u> 10 presenças</p>

Figura 3 Escolha do tipo de membro

Adicionar novo Bran





**Dados Pessoais**

\*Nome:

\*Idade:

\* Campos Obrigatórios

**Dados Adicionais**



☐ Problemas Cardíacos

Criar Utilizador

Voltar

Figura 4 Adicionar novo Bran

Adicionar novo Brienne



Dados Pessoais

\*Nome:

\*Idade:

\* Campos Obrigatórios



Dados Adicionais

Número de Exercícios

Criar Utilizador Voltar

Figura 5 Adicionar novo Brienne

Adicionar novo Ned



Dados Pessoais

\*Nome:

\*Idade:

\* Campos Obrigatórios



Dados Adicionais

Número de Exercícios

Criar Utilizador Voltar

Figura 6 Adicionar novo Ned

Adicionar Novo Membro

Dados Pessoais

\*Nome:

\*Idade:

\* Campos Obrigatórios

Dados Adicionais

☐ Problemas Cardíacos

Criar Membro

Voltar

Figura 7 Adicionar novo Jaime

Propriedades Extras:

Calorias Por Sessão:

☐ Problemas Cardíacos

☒ Guardião da Noite

Ativar SPA

Desativar Membro

Reativar Membro Punido

Dados referentes ao ultimo mês:

Adicionar Presenças

Atribuir Desconto

Adicionar Calorias Por Sessão

Dias Desde Da Ultima Presença

Calcular Mensalidade

Verificar Dedicação

Figura 8 Consultar/Editar Bran

Propriedades Extras:

Número de Exercícios:

Duração da Aula:

☐ Guardiã da Noite

Dados referentes ao ultimo mês:

Adicionar Presenças

Atribuir Desconto

Adicionar Exercícios

Adicionar Duração da Aula

Dias Desde Da Ultima Presença

Calcular Mensalidade

Verificar Dedicção

Ativar SPA

Desativar Membro

Reativar Membro Punido

Figura 9 Consultar/Editar Brienne

Propriedades Extras:

Minutos de Meditação:

Número de Exercícios:

Duração da Aula:

☐ Guardiã da Noite

Dados referentes ao ultimo mês:

Adicionar Presenças

Atribuir Desconto

Adicionar Exercícios

Adicionar Minutos de Meditação

Adicionar Duração da Aula

Dias Desde Da Ultima Presença

Calcular Mensalidade

Verificar Dedicção

Ativar SPA

Desativar Membro

Reativar Membro Punido

Figura 10 Consultar/Editar Ned

Propriedades Extras:

Calorias Por Sessão:

Peso Máximo:

☐ Problemas Cardíacos  
☒ Personal Trainer  
☐ Guardião da Noite

Desativar Personal Trainer

Ativar SPA

Desativar Membro

Reativar Membro Punido

Dados referentes ao ultimo mês:

Adicionar Presenças

Atribuir Desconto

Adicionar Calorias Por Sessão

Adicionar Peso Maximo

Dias Desde Da Ultima Presença

Calcular Mensalidade

Verificar Dedicção

Figura 11 Consultar/Editar Jaime

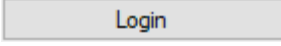


## Manual de Utilização

Todas as funcionalidades representadas em seguida requerem o login na plataforma (figura 1), sendo este um exercício académico esta fase é meramente ilustrativa e não se encontra ligada a uma base de dados.

Portanto, para entrar na plataforma foram definidos os seguintes dados:

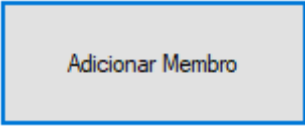
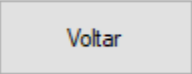
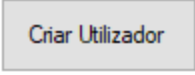
**ID:** bom ; **PW:** natal .

Insira os dados e pressione o botão . Será aberta uma nova janela “WesterosFit”.

Em todo o programa existe a possibilidade de voltar a esta página pressionando

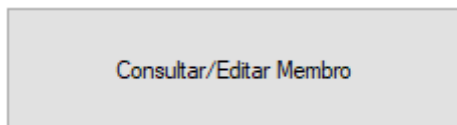
o botão .

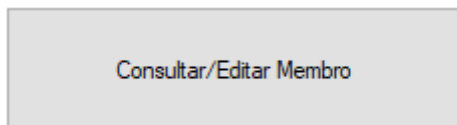
### Criação de um novo membro

1. Pressione o botão , será aberta uma nova janela “Escolha do tipo de membro” (figura 3);
2. Neste momento deve escolher o tipo de membro que pretende criar. São disponibilizadas algumas informações sobre os mesmo para que seja escolhido o que mais se adequa ao pretendido. Ao carregar num dos tipos de membros esta ação é reversível e pode retroceder à página anterior caso deseje trocar o tipo de membro que pretende criar pressionando o botão ;
3. Consoante o tipo de membro escolhido, “Bran”, Brienne”, “Ned” ou “Jaime”, será aberta uma nova janela (figura 4, 5, 6 ou 7) para que sejam introduzidos os dados necessários para a criação de um novo membro do tipo escolhido. Insira os dados e pressione o botão . No caso de os dados serem válidos será apresentada uma mensagem a confirmar que o membro foi criado, caso contrário será solicitado que corrija os dados inválidos inseridos;

## Consultar/Editar dados dos membros criados

Para que seja possível consultar/editar informação sobre os membros, é necessário que tenha sido criado no mínimo um membro (página 12).



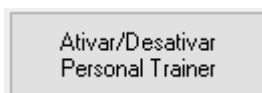
1. Pressione o botão , será aberta uma nova janela "Consultar/Editar membros" (figura 8);
2. Selecione o membro que pretende editar pressionando a linha do mesmo:

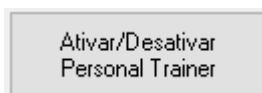
	ID	Classe	Nome
▶	0	Bran	Manual de Utiliza...

3. Consoante a classe do membro selecionado as opções disponíveis para a edição do mesmo serão apresentadas;

## Ativar/Desativar Personal Trainer

Apos ter selecionado um membro Jaime que deseja ativar ou desativar o Personal Trainer:




1. Pressione o botão , será ativado a opção de ter um Personal Trainer ao membro que não o tenha ativo ou será desativado para o membro que o tenha ativo.

## Ativar/Desativar SPA

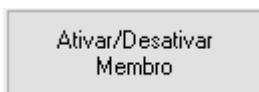
Após ter selecionado o membro que deseja ativar ou desativar o SPA:

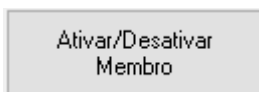


1. Pressione o botão , será ativado a opção do SPA ao membro que não o tenha ativo ou será desativado para o membro que o tenha ativo.

## Ativar/Desativar Membro

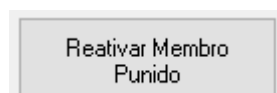
Após ter selecionado o membro que deseja Ativa/Desativar:

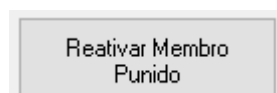


1. Pressione o botão , irá reativar um membro que tenha sido desativado (expeto ter sido desativado por razões de inatividade/dedicação), ou será desativado.

## Reativar Membro Punido



Após ter selecionado o membro que deseja reativar após ser punido:



1. Pressione o botão , o membro punido que se deseja reativar, terá que ter cumprido a sua punição, para poder ser reativo.



## Adicionar Presenças

Após ter selecionado o membro que deseja adicionar presenças:

1. Selecione o número de presenças que pretende adicionar na combobox  (terá de selecionar um número entre 1 a 31);
2. Pressione o botão , as presenças selecionadas na combobox, serão adicionadas.

## Atribuir Desconto

Após ter selecionado o membro que deseja atribuir um desconto:

1. Introduza um valor de desconto na caixa de texto respetiva  (o valor introduzido deverá ser maior do que 0 e caso ser decimal deverá ser representado por “,”);
2. Pressione o botão , o valor introduzido na caixa de texto será deduzido na mensalidade.

## Adicionar Exercícios

Após ter selecionado um membro Brienne ou Ned que deseja adicionar exercícios:

1. Introduza um número de exercícios que pretende adicionara na caixa de texto respetiva  (o valor introduzido deverá ser maior que 0 e deverá ser inteiro);

Adicionar Exercícios

2. Pressione o botão , o valor introduzido na caixa de texto, será guardado na respetiva propriedade.

## Adicionar Calorias Por Sessão

Após ter selecionado um membro Bran ou Jaime que deseja adicionar calorias por sessão:

1. Introduza um valor de Calorias Por Sessão que pretende adicionar na respetiva caixa de texto  (o valor introduzido deverá ser maior do que 0 e caso ser decimal deverá ser representado por “,”);

Adicionar Calorias Por Sessão

2. Pressione o botão , o valor introduzido na caixa de texto, será guardado na respetiva propriedade.

## Adicionar Minutos de Meditação

Após ter selecionado um membro Ned que deseja adicionar os minutos de meditação:

1. Introduza um número de minutos de Meditação que pretende adicionar na respetiva caixa de texto  (o valor introduzido deverá ser maior do que 0 e deverá ser um número inteiro);

Adicionar Minutos de Meditação

2. Pressione o botão , o valor introduzido na caixa de texto, será guardado na respetiva propriedade.

## Adicionar Peso Máximo Levantado

Após ter selecionado um membro Jaime que deseja editar:

1. Introduza um valor do Peso Máximo levantado que pretende adicionar na respetiva caixa de texto  (o valor introduzido deverá ser maior do que 0 e caso ser decimal deverá ser representado por “,”);

Adicionar Peso Maximo

2. Pressione o botão , o valor introduzido na caixa de texto, será guardado na respetiva propriedade.

## Adicionar Duração da Aula

Apos ter selecionado um membro Brienne ou Ned que deseja editar:

1. Introduza um valor da Duração da Aula que pretende adicionar na respetiva caixa de texto  (o valor introduzido deverá ser maior do que 0 e caso ser decimal deverá ser representado por “,”);

Adicionar Duração da  
Aula

2. Pressione o botão , o valor introduzido na caixa de texto, será guardado na respetiva propriedade.

## Adicionar Dias Desde Da Última Presença

Apos ter selecionado um membro que deseja editar:


1. Selecione o número de presenças que pretende adicionar na combobox  (terá de selecionar um número entre 1 a 31);

Dias Desde Da Ultima  
Presença

2. Pressione o botão , os dias selecionadas na combobox, serão adicionados à propriedade.

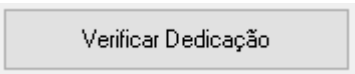
## Calcular Mensalidade

Após ter selecionado o membro que deseja consultar a Mensalidade:

1. Pressione o botão  , a mensalidade é calculada e é devolvido o valor da mensalidade numa mensagem.

## Verificar Dedicção

Após ter selecionado o membro que deseja verificar a dedicação:

1. Pressione o botão  , a dedicação será verificada e será mostrada uma mensagem a dizer se o membro é dedicado ou não.

## Propriedade *static*

Ainda que o enunciado do trabalho proposto não exigisse a utilização desta propriedade o grupo decidiu tentar tornar a aplicação um pouco mais complexa, ainda que tenha sido uma tarefa desafiante devido a esta propriedade não ter sido lecionada nas aulas, o resultado final acabou por compensar.

*Static* é utilizado quando apenas uma cópia é necessária permitindo que seja utilizada por todas as instâncias do projeto. Esta funcionalidade permitiu-nos que fosse atribuído um ID diferente para cada membro criado, não podendo assim haver dois membros com os mesmos dados, em último caso pelo menos o ID seria diferente.

Esta propriedade foi também usada quando criámos a lista “listaDeMembros” na classe “BDMembros” permitindo que esta fosse única e acessível para todos os membros registando sempre que um novo membro é criado. Uma vez mais, não era estritamente necessário para o funcionamento da aplicação como pedida no enunciado, mas fruto de decisões anteriores, esta foi uma funcionalidade que o grupo decidiu investigar e aprender uma vez que teria grande utilização no projeto.

## Criação de Listas

Ainda que no enunciado não referisse nenhuma forma obrigatória para guardar os dados dos membros criados, em grupo chegamos em consenso que poderíamos utilizar uma *List* em vez dos *Arrays* lecionados em aula.

Decidi-mos usar uma *List* em vez de um *Array*, visto que não necessitamos de implementar um limite de membros guardados, enquanto com um *array* teríamos que limitar. Não necessitamos também de criar um índice, será necessário um *Enumerator*. A *List* permitiu ainda que fosse usada como *DataSource* para a *DataGridView*, ambos serão descritos mais à frente. A lista é também *static* pelo motivo explicado anteriormente.

### *Enumerator*

O *enumerator* é do género de um índice utilizado nos *arrays* lecionados nas aulas. O *Enumerator* permite percorrer a *ListaDeMembros* devolvendo os dados de cada membro. O *Enumerator* irá enumerar todos os membros guardados por ordem de criação, na *ListaDeMembros*. Iremos utilizar o *Enumerator* para encontrar um membro na *ListaDeMembros*, com a propriedade *ID*, igual ao *id* que será seleccionado na *DataGridView* (*dgvMembro*).



## DataGridView

DataGridView é uma tabela que permite uma visualização bastante intuitiva de um conjunto de dados identificados pelas colunas. Esta funcionalidade precisa de uma fonte de informação, a datasource que será explicada mais a frente. Neste caso a tabela é *ReadOnly* uma vez que a alteração da informação é feita através de botões disponibilizados consoante o tipo de membro, á direita da tabela. A tabela criada recebeu o nome *dgvMembro* uma vez que contém a informação referente à classe Membro.

## Datasource

Como Datasource para a DataGridView é comum utilizar uma base de dados, no entanto até ao momento da realização deste trabalho os membros do grupo ainda não possuíam o conhecimento suficiente para o fazer e sairia também do âmbito da unidade curricular em questão. Desta forma outra opção que encontrámos para servir de datasource para a tabela foi a lista *ListaDeMembros*.

Nesta tabela irão aparecer todos os membros criados com as propriedades em comum. As propriedades que não sejam comuns a todos os tipos de membro, irão aparecer na groupBox as restantes propriedades. Para que as colunas necessárias fossem apresentadas na tabela foi utilizada como fonte a classe Membro tendo sido posteriormente editado a forma como estas colunas eram apresentadas, removidas as colunas que não seriam necessárias e até a alteração dos nomes das mesmas.

## Conclusão

A decisão de ir além do que foi lecionado nas aulas tanto práticas como teóricas da unidade curricular foi tomada pelo grupo. Esta decisão exigiu que fossem pesquisados novos conteúdos da linguagem de programação C# para a realização de ideias que o grupo teve para melhorar o trabalho. No entanto, foi sempre tido em atenção atingir aqueles que achávamos ser os objetivos impostos pelo enunciado e só depois, a partir desse ponto ter uma certa liberdade para pensar que outras funcionalidades seriam interessantes de implementar para tornar a aplicação mais desafiante.

A repetição de código foi talvez um dos problemas encontrados mais difíceis de resolver. Quando se começa a criar uma funcionalidade sem primeiro pensar na melhor maneira de o fazer evitando a repetição de código acaba por mais tarde ser uma tarefa bastante mais difícil de resolver. Ainda assim foi tido em conta esse aspeto e tentou-se que este problema fosse o menos presente possível ainda que não tenha sido eliminado por completo.

A atribuição de experiência através das presenças gerou alguma confusão devido à má interpretação nas primeiras leituras do enunciado. Tendo apenas o problema sido identificado já na fase final do trabalho, obrigando à remodelação não só de código, mas também da interface da aplicação. Ainda assim o problema foi apenas possível de encontrar devido ao trabalho ser realizado por mais que um elemento resultando de duas interpretações diferentes que precisam de ser discutidas para chegar a um entendimento.

Em suma, a realização deste trabalho foi uma experiência não só desafiante, mas também bastante agradável devido a ambos os membros terem tido interesse nesta nova unidade curricular de Desenvolvimento de Aplicações.

## Bibliografia

C# - How To Displaying Data From Selected Rows In DataGridView into TextBox Using C#. Disponível em: <http://1bestcsharp.blogspot.com/2015/02/c-how-to-get-selected-row-values-from-DataGridView-Into-TextBox-In-Csharp.html>

C# - How To Update A DataGridView Row With TextBoxes In C#. Disponível em: <http://1bestcsharp.blogspot.com/2015/02/c-how-to-update-datagridview-row-with-TextBox-In-Csharp.html>

C# Class Auto increment ID. Disponível em: <https://stackoverflow.com/questions/9262221/c-sharp-class-auto-increment-id>

VB.NET - How To Display DataGridView Selected Row Values On Another Form Using Visual Basic.NET. Disponível em: <http://1bestcsharp.blogspot.com/2016/09/vbnet-datagridview-show-selected-row-data-in-another-form.html>

How do I create a message box with “Yes”, “No” choices and a DialogResult?. Disponível em: <https://stackoverflow.com/questions/3036829/how-do-i-create-a-message-box-with-yes-no-choices-and-a-dialogresult>

C# | List Class. Disponível em: <https://www.geeksforgeeks.org/c-list-class/>