Programação Orientada a Objetos 2023/2024

Exercícios

Ficha Nº 4

Composição
Agregação
Ficheiros

Introdução teórica aos conceitos usados nesta ficha de exercícios

Esta ficha lida com a **associação** simples entre objetos. As associações podem tomar essencialmente duas formas: a **agregação** e a **composição**.

Associação

Refere-se a um relacionamento entre objetos de duas classes. Esta associação é genérica (não tem nenhum significado especial) e pode envolver um ou mais objetos de cada lado (um ou mais objetos da classe *A* está associado a um ou mais objetos da classe *B*).

Agregação

Trata-se de uma forma mais específica de associação na qual existe o significado de "tem" – um objeto da classe *A* "***vê***" um ou mais objetos da classe *B*. Significa que os objetos de *A* de alguma forma usam/vêm/atuam-sobre os objetos da classe *B*. No entanto, se o objeto da classe *A* desaparecer, os objetos de *B* que participavam na associação continuam a existir: Os objetos de A não controlam nem são responsáveis pela existência dos objetos de B

Aspetos de implementação.

- De forma a que os objetos de B tenham uma existência independente dos objetos de A, na classe A os objetos de B apenas podem ser referidos (por referência ou ponteiro), ou seja, não será possível ter os objetos de B fisicamente contidos dentro dos objetos de A, caso contrário, quando o objeto de A desaparece, os objetos de B nele contidos também desapareceriam, contrariando a lógica da agregação.
- Se os objetos de B referidos num objeto de A desaparecerem, o objeto de A deve ter a capacidade de ser avisado desse facto, de forma a não voltar a usar esses objetos de B referidos, que agora já não existem. Isto normalmente envolve mecanismos explicitamente criados pelo programador, dado que o compilador não faz nada relativamente a esta situação.

Exemplo: classe *Turma* e classe *Aluno*. Um objeto *Turma* tem (agrega) um conjunto de objetos de *Aluno*. Os objetos de *Aluno* envolvidos já existiam antes e continuam a existir mesmo que o objeto Turma que os agrega desapareça.

Composição

Trata-se de uma forma mais restritiva de associação, em que o significado da associação é normalmente "tem" e o objeto que *contém* controla a existência dos objetos *contidos* e os objetos contidos também desaparecem se o objeto que os contém desaparecer. Ou seja, um objeto de *A* "tem" objetos de *B*. Se o objeto de *A* desaparecer, os objetos de *B* que lhe pertencem também desaparecem. O desaparecimento não será automático se os objetos de *B* forem dinâmicos (esta variante aparece numa outra ficha mais tarde).

Aspetos de implementação.

Existem duas formas de implementação:

a) os objetos contidos estão fisicamente dentro do objeto que contém. O compilador trata da maior parte do trabalho de criação e destruição dos objetos contidos no momento em que o objeto que contém é criado/destruído. Esse automatismo implica que o compilador tem que saber como criar os objetos contidos, o que normalmente se traduz em algumas restrições sintáticas. Esta forma é, habitualmente mais simples, mas nem sempre pode ser usada. É abordada nesta ficha de exercícios
b) os objetos contidos são dinâmicos, residindo no exterior do objeto que contém, mais especificamente, no espaço de memória dinâmica do programa. Dentro do objeto que contém apenas reside um ponteiro para o objeto contido. A criação e destruição dos objetos contidos não é automática e tem que ser tratada explicitamente pelo programador. Esta forma pode ser sempre usada, é mais poderosa e flexível, mas mais trabalhosa e propensa a erros. É abordada noutra ficha mais tarde.

Quando se está perante uma situação de composição, independentemente se os objetos contidos são ou não dinâmicos, a classe que contém deve assumir a responsabilidade de gestão (criação e destruição) dos seus objetos contidos, e o resto do programa abdica dessa responsabilidade. A a típica função da classe que contém para "adicionar" um objeto contido recebe, não um objeto já existente, mas sim os dados necessários à sua criação, sendo o objeto criado apenas no código da classe que contém.

Exemplo: classe Desenho e classe Figura. Um objeto de classe Desenho tem vários objetos da classe Figura. Os objetos de Figura apenas existem no contexto de um determinado desenho. Se o desenho desaparecer, também desaparecem as figuras nele contidas. O resto do programa pode "ver" as figuras, mas não controla a sua existência.

Exercícios de aplicação - Observações

- Alguns destes exercícios têm muitas alíneas. Destinam-se a garantir que os vários aspetos da linguagem são devidamente explorados e nenhum objetivo do exercício fica por cumprir.
- Esclareça com o professor os aspetos de carácter mais teórico que estão envolvidos de forma a garantir que entende a matéria.
- Alguns exercícios são para fazer em casa. É mesmo importante que realmente os faça.
- Cada uma das classes envolvidas deve ser escrita num par de ficheiros .h e .cpp. A função *main()* é colocada num ficheiro .cpp à parte.

---

1. Ponto

Considere pontos de um plano representados pelas suas coordenadas cartesianas *x* e *y*. Pretende-se uma classe em C++ cujos objetos representem pontos. A classe deve cumprir o seguinte:

- Não deve ser possível construir objetos desta classe sem a indicação das suas coordenadas (nota: "indicação" ≠ "perguntar ao utilizador"). Qualquer valor inteiro é válido, tanto para *x* como para *y*.
- Deve ser possível obter e modificar cada uma das coordenadas, mas sem desrespeitar o conceito de encapsulamento. As funções que permitem obter os dados devem poder ser chamadas sobre objetos Ponto constantes, e as que modificam as coordenadas não.
- Obter o valor que corresponde à distância entre o ponto e um outro que é fornecido.
- Obter um objeto *string* com a descrição textual do seu conteúdo (formato "(x / y")).

a) Escreva a classe *Ponto* com as características enunciadas. Teste a classe através de uma função *main* que tenha dois pontos *a* e *b* nas coordenadas iniciais (1,2) e (3,4). Confirme que não é possível ter pontos sem especificar as suas coordenadas. Se pretendesse ter pontos sem especificar as suas coordenadas, qual seria a alteração precisava de fazer? (responda apenas e não modifique o código).

b) Obtenha e imprima a distância entre os pontos *a* e *b*. Repare que a função tem apenas um parâmetro que corresponde a um dos pontos. Como sabe o computador qual é o outro ponto?

c) Declare na função *main* anterior um novo ponto *c* constante de coordenadas iniciais (5,6). Confirme que é possível obter as suas coordenadas, e a sua descrição textual, apresentando-as no ecrã, mas que não é possível modificar as coordenadas. Identifique:

- Qual a característica que impede a invocação das funções que modificam as coordenadas sobre o ponto *c*.
- Qual a característica que permite invocar as funções que obtêm as coordenadas do ponto *c*.

d) Sem modificar nada na classe *Ponto*, acrescente à função *main()* uma matriz de três objetos *Ponto*.

i) Tente declarar a matriz sem qualquer indicação acerca dos pontos. É possível? Por quê?

ii) Indique as seguintes coordenadas dos pontos da matriz: (1,3), (2,4) e (5,7). Repare na sintaxe que utilizou. Relacione esta informação com a exigência que a classe Ponto estipulou para a criação e inicialização dos seus objetos.

e) Acrescente à sua classe um mecanismo que permita a criação de pontos sem especificação das suas coordenadas. Neste caso os novos pontos ficam na posição (0,0). Tente novamente criar uma matriz sem qualquer indicação das coordenadas dos seus pontos.

No final do exercício deverá ter

- Consolidado o conceito de encapsulamento.
- Consolidado o uso de *const*, nomeadamente as funções membro constantes, e identificar as situações onde se aplicam e quais as restrições ao seu uso.
- Compreendido a utilização de matrizes de objetos, as restrições aplicáveis, e a sintaxe de inicialização associada a esses casos.

2. Triângulos usando pontos

Considere os triângulos como figuras geométricas planas definidas por três pontos. Pretende-se construir em C++ uma classe chamada *Triangulo* cujos objetos representem triângulos. Os dados da classe representam os vértices dos triângulos e devem **usar obrigatoriamente a classe Ponto** do exercício anterior. Cada triângulo tem também um nome. A classe deve ter mecanismos que permitam obter e modificar cada um dos vértices e o nome, mas sem desrespeitar o conceito de encapsulamento. O ponto a obter/modificar é identificado pelo seu número de ordem (0 = primeiro, 1 = segundo, etc.). Deve também ser possível obter a descrição textual dos dados ("triangulo nome: (x1,y1) (x2,y2) (x3,y3)").

a) Escreva a classe *Triangulo* com as características enunciadas. Teste a classe através de uma função *main()* e declare um objeto Triângulo *q*. Quais são os dados iniciais desse triângulo? Explique de onde vêm esses valores.

b) Modifique a classe *Ponto* do exercício anterior e que está a usar neste exercício de forma a que volte a não ser possível a criação de pontos sem a especificação das suas coordenadas. Ou seja, elimine a alteração correspondente à última alínea do exercício anterior. Verifique que deixa de ser possível a criação do objeto *Triangulo q* que já tinha na alínea anterior deste exercício. Explique qual o problema (ou peça a ajuda do professor).

c) Acrescente à classe *Triangulo* um mecanismo que permita inicializar os seus objetos com um nome e três pontos. Cada ponto é indicado pelas suas coordenadas *x* e *y* (se no decorrer deste exercício pensar em colocar os dados da classe *Ponto* como públicos, pense novamente: é incorreto e além disso não resolvia o problema). Modifique a declaração do objeto *Triangulo q* que já tem na sua função *main()* e confirme que agora já é possível compilar novamente o programa. Confirme junto do professor as regras sintáticas relativas à inicialização de objetos compostos e anote no seu caderno essas regras.

d) Coloque mensagens nos construtores das classes *Ponto* e *Triangulo* nos quais é apresentada a mensagem "construindo" seguida da descrição textual do objeto que está a ser construído. Acrescente destrutores a ambas as classes nos quais é apresentada a mensagem "destruindo" também seguida da informação textual do objeto em questão. Execute o programa como "*execute without debugging*" de forma a ter tempo para ver as mensagens. Confirme pelas mensagens que os objetos *Ponto* são construídos antes do objeto *Triangulo* e destruídos depois (se as mensagens não confirmarem esta sequência, verifique o seu código e peça ajuda do professor). Analise a sequência de mensagens e debata com o professor a razão de ser dessa sequência e de não poder ser por outra.

No final do exercício deverá

- Compreender e aplicar o conceito de composição de objetos.
- Entender as regras sintáticas associadas à composição de objetos, nomeadamente as que estão relacionadas com a inicialização de objetos compostos.

3. Pretende-se uma classe chamada *Retangulo* cujos objetos representam retângulos com lados paralelos aos eixos coordenados. Cada retângulo é descrito pelo seu canto superior esquerdo, que é um objeto ponto da classe Ponto dos exercícios anteriores, e pelas suas dimensões largura (medida do lado paralelo ao eixo dos xx) e altura (medida do lado paralelo ao eixo dos yy) que são valores sempre positivos. A classe *Retangulo* deve respeitar as seguintes características:

- Só se pode criar um retângulo apenas mediante a indicação de todos os seus atributos.
- Deve ser possível obter cada um dos seus dados e estas funções devem funcionar mesmo sobre objetos que sejam constantes.
- Deve ser possível obter a área de um retângulo (esta função também pode ser chamada sobre objetos constantes).

a) Analise a relação existente entre *Retangulo* e *Ponto*. Será um caso de composição ou será um caso de agregação? Sabendo que é obrigatória a utilização da classe *Ponto* na classe *Retangulo*, identifique a melhor forma de armazenar esse ponto na classe *Retangulo*. Será uma cópia que pertence exclusivamente ao retângulo? Ou deverá ser um objeto ponto que existe fora do objeto retângulo e que pode ser partilhado entre vários retângulos distintos (sendo então armazenado por ponteiro ou referência)? Tem de ter uma noção clara quanto a estas questões antes de prosseguir. Confirme a sua escolha junto do professor. Este aspeto vai influenciar os dados da classe e o construtor.

b) Construa a classe e teste-a através de uma função *main* com um objeto ponto *p1* nas coordenadas (1,2) e dois retângulos *a* e *b* quaisquer que por acaso partilham o mesmo canto superior esquerdo. Esse canto é o objeto ponto *p1*. Apresente os dados de todos os objetos no ecrã.

c) Modifique as coordenadas do ponto canto-superior-esquerdo do retângulo *a* para as coordenadas (4,5). Obtenha e imprima todos os dados de todos os objetos. O que observa face às coordenadas do canto superior esquerdo do retângulo *b*? Confirme que o canto superior esquerdo do retângulo *b* se manteve na mesma. É este o cenário que se pretendia. Cada retângulo tem o seu próprio ponto e esse ponto só diz respeito a esse retângulo. Confirme que o comportamento do seu programa está de acordo com esta descrição e confirme que é este cenário que faz mais sentido. Se não concorda exponha as suas razões ao professor.

d) Acrescente construtores e destrutores às classes envolvidas neste exercício de forma a que seja possível ver no ecrã os dados dos objetos que estão a ser criados e destruídos. Execute o programa em modo "*execute without debugging*". Analise as mensagens de construindo/destruindo relativos a pontos e aproveite para rever a matéria relativa à criação e destruição de objetos, nomeadamente nas situações:

- Ordem de criação de objetos em função da ordem pela qual eles são declarados
- Passagem de objetos por valor/cópia em parâmetros
- Utilização de objetos "dentro" de objetos (composição de objetos).

Anote no seu caderno as suas conclusões.

Objetivos do exercício

- Consolidar o conceito de composição de objetos.
- Consolidar o conhecimento acerca das regras sintáticas associadas à composição de objetos.
- Consolidar o conhecimento acerca da ordem de criação e destruição dos objetos componentes de um objeto composto.

4. Pretende-se uma classe Desenho caracterizada por um nome e um conjunto de retângulos (objetos da classe do exercício anterior). Ao criar um desenho deve ser dado apenas o conhecimento do seu nome. A classe deve suportar a seguinte funcionalidade:

- Acrescentar um retângulo; esta operação só será possível se o novo retângulo não intersectar os que já existem no desenho. Esta operação deve retornar um código de sucesso ou insucesso da operação (bool); Nota: a verificação se dois retângulos se intersectam é uma operação que diz respeito a retângulos.
- Obter o conjunto de retângulos cujo canto superior esquerdo esteja num dado ponto.
- Eliminar todos os retângulos cuja área seja superior a um valor dado;
- Obter a soma das áreas de todos os retângulos do desenho;
- Obter uma string com a descrição de toda a informação relativa ao desenho.

a) Construa a classe pretendida usando uma matriz para armazenar os retângulos.

No final do exercício deverá ter

- Consolidado o conceito de composição de objetos.
- Consolidado o conhecimento acerca das regras sintáticas associadas à composição de objetos.
- Treinado algoritmos relacionados com classes que usam conjuntos de objetos de outra classe.
- Treinado classes com funções com código não-trivial.

5. Assuma a existência da classe *Prego* (passe o código para o seu computador). Os objetos desta classe representam pequenos objetos metálicos afiados que se afixam a paredes em coordenadas *x*,*y*. Os pregos podem mudar de sítio (removem-se e voltam-se a afixar).

```
class Prego{
   int x,y;
 public:
   Prego(int a, int b) {
      x = a; y = b;
      cout << "construindo prego em " << x << "," << y << "\n";
   }
   ~Prego() {
      cout << "construindo prego em " << x << "," << y << "\n";
   }
   void mudaDeSitio(int a, int b) {
      x = a; y = b;
   }
}
```

Pretende-se agora uma classe *Aviso* com as seguintes características:

- Tem um texto.
- Está preso a um prego. O prego não pertence ao aviso. O prego já existia antes e pode estar associado a mais do que um aviso.
- Quando é criado, o aviso tem sempre um texto e é sempre posto num dado prego que já se encontra afixado na parede. Não é preciso nem possível mudar o aviso de um prego para outro.
- Sempre que o objeto é criado ou destruído deve ser apresentada uma mensagem no ecrã que inclua o texto do aviso (para distinguir os objetos, se existirem vários).
- Deve ser possível saber as coordenadas x e y do prego em que o aviso está pendurado.

a) Determine a forma como a classe Aviso vai representar o prego no qual está pendurado. Tenha em atenção as seguintes características que devem ser respeitadas:

- Vários avisos podem estar pendurados no mesmo prego. Trata-se do mesmo objeto prego e não de cópias de pregos que por acaso estão no mesmo sítio na parede.
- Se se mudar um prego de sítio, o aviso que está pendurado muda implicitamente para o novo sítio. Se está a pensar acrescentar um mecanismo de “mudaDeSitio” à classe Aviso, pense novamente: quem muda de sítio é o prego e não o aviso. Este apenas vai atrás. Além disso o

aviso não sabe diretamente em que sítio está. Apenas sabe que está num determinado prego, e este é que sabe onde está. Mais, a mudança de sítio de um prego pode afetar mais do que um aviso (afeta todos os que estão lá pendurados).

Existem basicamente duas formas de tratar esta questão de representação do prego na classe Aviso. Identifique-as, confirme com o professor, escolha uma e prossiga

b) Faça a classe com as características pretendidas. Elabore uma função *main* e construa um objeto de *Aviso a*. Confirme que não consegue ter esse objeto sem ter também um preciso ter também um objeto *Prego* já existente. Construa então um prego *p* primeiro e pendure nele o aviso *a*.

c) Acrescente um novo *Aviso b* ao *Prego p*. De seguida efetue a seguinte sequência de instruções: obtenha e imprima a localização dos avisos *a* e *b*. Mude o *Prego* para uma nova posição. Obtenha novamente e imprima a posição dos avisos *a* e *b*. Se os avisos não tiverem mudado de local, o seu código não está bom e deve chamar o professor.

d) Compare a situação presente neste exercício com a dos *Pontos*, *Triângulos,* e *Retângulos* dos exercícios anteriores. Repare numa diferença importante:

- Os triângulos têm pontos. Cada ponto pertence a apenas um triângulo. Quando o triângulo é destruído os seus pontos também são destruídos.
- Os retângulos têm um ponto. Cada ponto pertence apenas a um retângulo. Quando o retângulo é destruído o ponto também é.
- Os avisos usam pregos. Cada prego pode ser usado por mais do que um aviso. Os pregos não pertencem aos avisos. Se um aviso desaparecer, os pregos mantêm-se em existência.

Reveja a introdução teórica desta ficha e identifique para cada situação (Triângulos, Retângulos, Avisos) se se trata de composição ou de agregação. Confirme junto do professor que identificou corretamente cada situação.

e) Reveja a forma como representou na classe *Aviso* o conhecimento acerca do *Prego* onde o aviso está. Há duas formas de representar esse conhecimento de forma adequada a este exercício. Escolha agora a outra forma e refaça o exercício. Compare as duas versões.

No final do exercício deverá ter

- Consolidado os conceitos de composição e agregação de objetos, incluindo as diferenças entre os dois conceitos e os cenários em que se deve aplicar um ou o outro.
- Consolidado o conhecimento acerca das regras sintáticas associadas à composição de objetos.
- Usado de referências como dados membro em classes e treino da sintaxe associada a esta situação.
- Treinado a implementação de agregação através de referências e através de ponteiros.

6. Pessoa

Escreva uma classe *Pessoa* tendo como atributos nome, número de bilhete de identidade (BI), e número de contribuinte (NIF). Um objeto de *Pessoa* só pode ser criado se lhe forem indicados todos os seus dados. A classe *Pessoa* deve ter funções membros que permitam obter cada um dos seus dados, atualizar o nome (apenas o nome), e obter uma *string* com a descrição do seu conteúdo. Esta classe vai ser necessária para os exercícios seguintes.

7. Registo Civil de pessoas

Escreva uma classe **RegistoCivil** que representa a o conjunto de pessoas existentes num determinado país. Um objeto desta classe tem as seguintes características

- Quando construído, exige o nome do país a que diz respeito. Inicialmente não tem pessoa nenhuma.
- É possível obter, mas não mudar o nome do país (isto não significa necessariamente o uso de *const*).
- Suporta a existência de um número finito, conhecido em compile-time de pessoas (sugestão: 50).
- Permite adicionar uma nova pessoa, a qual é construída no contexto da classe e não fora dela. Para esta operação são indicados os dados da nova pessoa. A operação falha se já existir nesse país uma pessoa com o BI indicado, ou se já tiver atingido o número máximo de pessoas.
- Permite adicionar pessoas lidas de um ficheiro de texto (nome do ficheiro é indicado). O ficheiro tem uma pessoa por linha, no formato nome BI NIF. O nome tem apenas uma palavra.
- Permite apagar uma pessoa dado o seu BI.
- Permite obter o nome de uma pessoa dado o seu BI.
- Permite obter a listagem de todas as pessoas numa string, uma pessoa por linha.
- Permite atualizar o nome de uma pessoa dado o seu BI e o novo nome.
- Permite obter o número de pessoas atualmente existentes no país.

Dados da classe

Não são dadas nenhumas indicações, obviamente. O que interessa ao programa é o comportamento dos objetos e não o seu interior, tal como já sabe por esta altura. Os dados são os que forem precisos para que o comportamento seja o esperado. Na implementação da sua classe comece pelas funções e não pelos dados.

Restrições na implementação:

Não pode usar memória dinâmica (caso saiba usar memória dinâmica em C++), nem pode usar *coleções standard* (como por exemplo, *vector*, quem nem sequer foram ainda abordados).

Se tiver dificuldades de implementação, pode (*) modificar a classe pessoa de forma a que seja possível construir objetos pessoa sem informação nenhuma, sendo que o BI fica inicializado a -1, que significa que este objeto pessoa não representa pessoa nenhuma. (*) -> Se recorrer a esta alteração tem que a justificar ao professor.

a) Identifique o **tipo de associação** existente entre a classe **RegistoCivil** e a classe **Pessoa**, e a **forma de implementação** adequada face ao necessário e à matéria já dada.

b) Implemente a classe (.h e .cpp + função *main*). Teste-a com algumas operações sintéticas de uso na função *main*.

No final do exercício deverá ter

- Consolidado os conceitos de composição e agregação de objetos, incluindo as diferenças entre os dois conceitos e os cenários em que se deve aplicar um ou o outro.
- Consolidado o conhecimento acerca das regras sintáticas associadas à composição de objetos.

8. Clube de bairro de pessoas

Pretende-se representar o conceito de *clubes de bairro* em C++. Um clube de bairro é constituído por um conjunto de vizinhos que partilham o interesse por uma determinada atividade. A classe pretendida chama-se *Clube* e deve cumprir a seguinte funcionalidade:

- Tem um nome, uma descrição de atividade (texto) e, aparentemente, um conjunto de pessoas. Repare na palavra "aparentemente". As pessoas talvez estejam dentro do clube, ou talvez tenham vida e existência própria para além do clube e alojadas em qualquer outra parte do programa. Vá pensando neste aspeto e interaja com o professor se tiver dúvidas.
- Um objeto de Clube só pode ser criado mediante a indicação do seu nome e da atividade. Inicialmente não existem pessoas nesse clube.
- Deve ser possível acrescentar uma pessoa a um clube. Acrescentar uma pessoa significa fazer essa pessoa sócia do clube. A pessoa não passa a "viver" na sede do clube - simplesmente o clube passa a saber que a pessoa existe e deixa-a entrar nas instalações e usar o equipamento, máquina de café, etc. Esta descrição é importante para que possa determinar exatamente qual é a forma como os objetos de Clube vão ver os objetos de Pessoa.
- **Nota**: muito cuidado na forma como passa a pessoa a adicionar por parâmetro à função que trata desta questão. Das três formas que conhece de passar parâmetros em funções, uma não é compatível com este exercício. Uma escolha errada pode levar ao crash do programa mais tarde. Confirme com o professor que fez uma escolha adequada.
- Deve ser possível saber se uma pessoa é sócia do clube. Para tal é indicado um BI e se o seu bilhete de identidade for igual ao de alguém no clube, a resposta será positiva ("resposta" = retorno da função que trata dessa questão).
- Deve ser possível remover um sócio do clube dado o seu bilhete de identidade (provavelmente não pagou a quota).
- Deve ser possível listar os dados de todos os sócios de um clube.
- Assuma um limite para o número de sócios conhecido em *compile-time*.

Implementação: as pessoas a usar no **Clube** são geridas pelo **RegistoCivil** do exercício anterior. Tem duas alternativas:

A. O clube sabe que o RegistoCivil existe e dirige-se a este para obter dados sobre os sócios. Precisa apenas de armazenar os BI dos sócios e o conhecimento que o RegistoCivil existe. É muito fácil saber se um BI deixou de ser válido (a pessoa em questão emigrou) porque o RegistoCivil intermedeia todos os acessos à Pessoa por parte do Clube, bastando analisar aquilo que o RegistoCivil retorna

B. O clube armazena "referências" (referencias/ponteiros) para as pessoas. Essas referências são-lhe passadas como parâmetros de funções. Não sabe que o RegistoCivil existe. O Clube fica fragilizado na medida em que a pessoa pode deixar de existir sem que o clube se aperceba e as referências /ponteiros para as pessoas (sócios) podem estar já inválidas sem que o Clube o saiba.

a) Escreva a classe **Clube** através da estratégia **A**. Identifique o **tipo de associação** existente entre a classe **Clube** e a classe **RegistoCivil**, e a **forma de implementação** adequada face ao necessário e à matéria já dada. Poderá nem precisar de usar a classe Pessoa de todo.

b) Escreva a classe **Clube** através da estratégia **B**. Identifique o **tipo de associação** existente entre a classe **Clube** e a classe **Pessoa**, e a **forma de implementação** adequada face ao necessário e à matéria já dada.

A classe clube não usará a classe RegistoCivil. No entanto, as pessoas a acrescentar ao clube continuam a ser geridas por esta a partir da função main. Vai ser preciso acrescentar ao RegistoCivil um método que permita obter uma referência ou ponteiro para uma determinada pessoa dado o seu BI.

- Identifique qual dos dois é o mais apropriado neste caso: referência ou ponteiro
- O método deve devolver a informação de forma a que a pessoa em questão não possa ser modificada a partir de fora da classe RegistoCivil.

No final deste exercício é suposto

- Compreender os conceitos de composição e agregação de objetos, incluindo as diferenças entre os dois conceitos e os cenários em que se deve aplicar um ou o outro.
- Compreender a forma de escolher referências *vs.* ponteiros.
- Compreender a implementação de agregação através de referências ou ponteiros.