Instituto Superior de Engenharia de Coimbra
Departamento de Engenharia Informática e de Sistemas
Engenharia Informática

Programação Orientada a Objectos 2021/2022

Exercícios

Ficha Nº 3

Composição
Agregação
Vectores
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* tem um ou mais objetos da classe *B*. Significa que os objetos de *A* de alguma forma usam/vêm/controlam 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.

Aspetos de implementação: Normalmente será preciso: a) armazenar os objetos de *B* de uma forma que não dependa da duração e visibilidade do objeto de *A* e b) garantir que o objeto de *A* é informado quando os "seus" objetos de *B* desaparecem (de maneira a que não tente usar um objeto que já não existe). Numa situação de agregação em que objetos de *A* agregam objetos de *B*, normalmente, existirá uma função em *A* para registar ("acrescentar") os objetos de *B*. Esta função recebe um ponteiro ou referência para o objeto de *B*, o qual já existe (a função não deve receber por cópia/valor, caso contrário seria um novo objeto local à função e de existência temporária).

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 restrita de agregação, sendo que agora os objetos contidos também desaparecem se o objeto que os contém desaparecer. Ou seja, um objeto da classe *A* "tem" objetos da classe *B*. Se o objeto da classe *A* desaparecer, os objetos da classe *B* que lhe pertencem também desaparecem. Este "desaparecimento" não é automático – o mecanismo de *clean-up* (destrutor, se for C++) terá que tratar do assunto.

Aspetos de implementação: Uma forma simples de lidar com a composição é a de tornar o objeto de A o responsável tanto pela criação como pela destruição dos objetos de B, deixando o resto do programa de controlar a existência de objetos de B. Nesta situação, a típica função da classe *A* para "adicionar" um objeto de *B* recebe, não um objeto *B* que já existe, mas sim os dados necessários à sua criação, sendo o objeto criado apenas no código de *A*.

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. Leia os objetivos de cada exercício e verifique que o seu código bate certo com as alíneas e que no fim do exercício usou as características listadas nos objetivos. Peça a ajuda do professor caso haja divergências.
- Os exercícios têm inicialmente muitas chamadas de atenção quanto ao que se deve fazer ou não fazer. Progressivamente essas chamadas de atenção vão desaparecendo à medida que se vai assumindo um maior conhecimento por parte dos alunos.

---

1. 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*, com as coordenadas (1,3) , (2,4) e (5,7).

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).

Objetivos do exercício

- Consolidar o conceito de encapsulamento.
- Consolidar o uso de const, nomeadamente as funções membro constantes, e identificar as situações onde se aplicam e quais as restrições ao seu uso.
- Compreender a utilização de matrizes de objetos e a sintaxe de inicialização associada a esses casos.
- Preparação para o exercício seguinte.

2. 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 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 a 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.

Objetivos do exercício

- 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 agora 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.
- Preparação para o exercício seguinte.

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.

b) Construa a classe pretendida usando um vetor para armazenar os retângulos. Nas operações de manipulação do vetor, experimente usar a notação de matriz e também os iteradores. Verifique que algumas operações apenas podem ser executadas através de iteradores.

Objetivos do exercício

- Consolidar o conceito de composição de objetos.
- Consolidar o conhecimento acerca das regras sintáticas associadas à composição de objetos.
- Consolidação de competências no uso de vetores da STL, incluindo iteradores.
- Treino em algoritmos relacionados com classes que usam conjuntos de objetos de outra classe.
- Treino em 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.

Objetivos do exercício

- Consolidar 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.
- Consolidar o conhecimento acerca das regras sintáticas associadas à composição de objetos.
- Usar de referências como dados membro em classes e treino da sintaxe associada a esta situação.
- Treinar a implementação de agregação através de referências e através de ponteiros.

6. Caso não tenha ainda feito uma classe *Pessoa* no contexto dos exercícios anteriores, faça agora essa classe tendo como atributos nome, número de bilhete de identidade, e número de contribuinte. 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 o resto do exercício e também para o exercício seguinte.

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 muda a sua residência para a 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 tratar 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 indicada um objeto pessoa 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.
- Não existe limite para o número de sócios. Isto não significa que se aventure já na memória dinâmica.

a) Analise com cuidado a relação existente entre o clube e as pessoas, ou seja, entre objetos de *Clube* e objetos de *Pessoa*. Determine se se trata de um caso de **composição** ou de **agregação**. Confirme com o professor. Reveja os aspetos de implementação do caso apropriado e escreva a classe com as características pretendidas. Escreva uma função *main* e declare nelas algumas pessoas e alguns clubes. Teste a funcionalidade da classe que fez. Teste aspetos menos triviais, por exemplo, acrescente uma pessoa a mais do que um clube.

b) Reveja a função *main* que fez na alínea anterior. Se tiver feito tudo corretamente, deverá ter um conjunto de pessoas independentes dos clubes a que pertencem. Repare que a responsabilidade de armazenar e manter em existência as pessoas pertence ao exterior da classe clube. É como se algures no programa existisse um "armário" de pessoas, sendo que o clube apenas toma conhecimento da sua existência quando elas se tornam sócias. Este aspeto é coerente com o facto de as pessoas serem independentes dos clubes a que talvez pertençam. Repare também que há ainda questões por resolver, como por exemplo, se uma pessoa que é sócia de um é removida do programa, o clube não é automaticamente informado (neste exercício) e pode haver problemas se tentar interagir com essa pessoa. Debata estas questões todas na aula com o professor e anote o que aprendeu no caderno.

Objetivos do exercício

- Consolidar 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. Incide-se mais na agregação neste exercício.
- Treino da definição de arquitetura das classes.
- Uso de referências como dados membro em classes e treino da sintaxe associada a esta situação.
- Treino de implementação de agregação através de referências ou ponteiros.
- Consolidação de competências no uso de vetores da STL.
- Treino em algoritmos relacionados com classes que usam conjuntos de objetos de outra classe
- Preparação para o exercício seguinte.

7. Pretende-se representar a informação existente num banco. O banco armazena contas. Uma conta é um artefacto cuja existência só faz sentido no contexto do banco. Cada conta diz respeito a uma pessoa, a qual já existe, quer esteja associada a uma conta, quer não. Existem então três conceitos: Banco, Conta e Pessoa.

a) Repare que o tipo de relacionamento entre *Banco* e *Conta* não é bem o mesmo que entre *Conta* e *Pessoa* (e não existe um relacionamento direto entre *Banco* e *Pessoa*). Reveja os conceitos de agregação e composição e use-os apropriadamente neste exercício. Verifique se se está perante um caso de duas situações de composição, ou então duas situações de agregação, uma de cada. Confirme com o professor antes de avançar. Reveja os aspetos de implementação dados na introdução teórica para o ajudar no resto do exercício.

b) Usando a classe *Pessoa* do exercício anterior, escreva a classe *Conta* tendo em atenção o seguinte:

- Os atributos de Conta são: o saldo (inteiro) e uma pessoa. A pessoa não pertence à conta. Pelo contrário: o objeto de Pessoa é totalmente externo à conta e com existência independente. A forma como armazena a informação acerca da pessoa deve garantir que não se armazena uma mera cópia da pessoa original. Se está a pensar armazenar o nome da pessoa com o intuito de o usar para chegar ao objeto Pessoa titular dessa conta, pense novamente: C++ não é SQL – pretende-se conseguir chegar de Conta ao objeto Pessoa diretamente e não através de pesquisas. Se este aspeto lhe parece críptico, peça ajuda ao professor.
- Deve ser possível criar uma conta apenas mediante a indicação da pessoa titular. O saldo inicial é zero.
- Deve ser possível aumentar ou diminuir o saldo. O valor mínimo do saldo é zero.
- Deve ser possível obter a pessoa titular da Conta, mas essa informação é fornecida como constante para prevenir modificações indesejadas.

c) Defina uma classe *Banco*. Um banco é caracterizado por um nome e um conjunto de contas. Recorde que as contas são artefactos que apenas fazem sentido no contexto do banco que as têm. Ao criar um banco deve ser indicado o seu nome. Inicialmente não tem nenhum cliente, ou seja, nenhuma conta. Não existe limite quanto ao número de contas, mas evite ir já para memória dinâmica. O banco deve permitir:

- Acrescentar uma conta.
- Encontrar/obter uma conta dado o BI do seu titular.
- Eliminar uma conta dado o BI do seu titular.
- Depositar (Levantar) uma quantia na conta de um cliente dado o seu BI e o valor a depositar (levantar).
- Obter a soma dos saldos de todos os clientes do banco;
- Obter uma *string* com todos os nomes dos clientes do banco.

d) Acrescente uma função *main* para testar as classes que fez neste exercício. Crie na função *main* várias pessoas e dois bancos. Faça com que uma pessoa seja cliente dos dois bancos. Mude o nome dessa pessoa. Confirme que essa alteração é automaticamente visível na listagem dos clientes de ambos os bancos. Se assim não for, o seu código não está correto e peça ajuda ao professor). Anote no caderno as suas conclusões.

e) Acrescente à função *main* código para ler um conjunto de objetos *Pessoa* a partir de um ficheiro de texto. Esta alínea serve apenas para começar a treinar o uso de classes biblioteca para ler ficheiros de texto e aproveita-se o contexto deste exercício. O formato do ficheiro é o seguinte: cada linha diz respeito a uma pessoa. A primeira palavra em cada linha é o nome da pessoa (para simplificar, é só o primeiro nome), a segunda palavra é o BI, e a terceira é o número de contribuinte. Crie um ficheiro assim com o *notepad* ou algo igualmente simples e use-o. Devem ser lidas e criadas tantas pessoas até um máximo de 10 (menos se o ficheiro não tiver tantas). Use essas pessoas como clientes no banco tal e qual como na alínea anterior.

Objetivos do exercício

- Consolidar 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, inclusive a aplicação de ambos em simultâneo.
- Treino da definição de arquitetura das classes.
- Uso de referências e/ou ponteiros como membro em classes.
- Consolidação de competências no uso de vetores da STL.
- Treino em algoritmos relacionados com classes que usam conjuntos de objetos de outra classe
- Introdução à leitura de ficheiro de texto com a classe *ifstream*.

8. Construa em C++ o conceito de Turma de Alunos. Uma turma é um conjunto de pessoas que se encontram a estudar a mesma disciplina com o mesmo professor. Usando a classe Pessoa que aparece nos dois últimos exercícios, escreva a classe *Turma*, a qual deve cumprir os seguintes requisitos:

- A turma tem (aparentemente) um objeto Pessoa que é o professor dessa turma e um conjunto de objetos Pessoa que são os alunos. Tanto o professor como os alunos têm vida própria para além da turma e a suas existências não podem controladas pelo objeto Turma.
- A criação da turma obriga a indicação do seu professor. Os alunos são acrescentados mais tarde.
- Deve ser possível acrescentar alunos até ao máximo de 20.
- Deve ser possível remover um aluno dado o seu BI.
- Deve ser possível listar os nomes dos alunos na turma.

a) Construa a classe *Turma* comas características apresentadas. Deve ter em atenção o relacionamento existente entre o objeto *Turma* e os objetos *Pessoa* e a forma correta de o implementar. Para não sobrecarregar o texto do enunciado, não se vão repetir aqui as chamadas de atenção que já foram feitas nos exercícios anteriores – essas questões continuam presentes, mas assume-se que a matéria está as ser progressivamente adquirida e que começa a deixar de ser necessário estar sempre a bater nas mesmas teclas e pormenores.

b) Acrescente uma função *main* e teste a funcionalidade da classe. Declare várias pessoas e pelo menos duas turmas. Inscreva um aluno nas duas turmas e depois experimente mudar o seu nome para confirmar que essa alteração é automaticamente visível na listagem de alunos das duas turmas, provando assim que essas turmas não "têm" cópias das pessoas mas sim os originais.

c) Modifique a classe *Turma* de forma a que afinal pode existir um número arbitrariamente grande de alunos numa turma. Se a sua classe já tinha essa capacidade (não era necessário porque foi mencionado o limite de 20 alunos), então não precisa de alterar nada.

d) Acrescente à sua função *main* a capacidade de ler alunos diretamente de um ficheiro de texto com o mesmo formato que o mencionado na última alínea do exercício anterior. Construa um ficheiro com o *notepad* para testar esta nova funcionalidade. Registe essas pessoas como alunos em algumas turmas.

Objetivos do exercício

- Consolidar os conceitos de composição e agregação de objetos, neste caso incidindo-se mais na agregação.
- Uso de referências e/ou ponteiros como membro em classes, e os casos em que um é apropriado e o outro não.
- Consolidação de competências no uso de vectores da STL.
- Treino em algoritmos relacionados com classes que usam conjuntos de objetos de outra classe.
- Consolidação da competência de leitura de ficheiros de texto com a classe *ifstream*.