

Instituto Superior de Engenharia de Coimbra
Engenharia Informática e de Sistemas
Programação Orientada a Objectos (2020/21)
Exercícios

Ficha 2

Encapsulamento / Classes / Construtores / Destrutores

1. Defina uma estrutura chamada *Tabela*, que contenha uma matriz de inteiros. No decorrer das alíneas seguintes teste a funcionalidade pedida através de uma função *main* cujo código fica em aberto.
 - a) Escreva uma função que preencher toda a matriz da estrutura com um valor especificado.
 - b) Escreva uma função que permita listar o conteúdo da estrutura.
 - c) Escreva uma função que permita obter o valor num elemento da matriz, membro da estrutura, dado a posição do elemento pretendido. Este acesso deve ser protegido de tentativas de utilização de índices inválidos. Estas funções não interagem directamente com o ecrã.
 - d) Escreva funções que permitam actualizar o valor num elemento da matriz, membro da estrutura, dado a posição do elemento pretendido e o novo valor. Este acesso deve ser protegido de tentativas de utilização de índices inválidos. Esta função não interage directamente com o utilizador.
 - e) Unifique a funcionalidade das funções das duas alíneas anteriores numa só, cuja utilização se exemplifica a seguir. O acesso aos elementos da matriz da estrutura deve ser protegido de tentativas de utilização de índices não válidos.

```
int main(){  
    Tabela a;  
    cout << elementoEm(a,10);    // aparece um determinado valor  
    elementoEm(a,10) = 15;       // notar que a chamada à função fica  
                                // do lado esquerdo da atribuição  
    cout << elementoEm(a,10);    // aparece 15  
}
```

Objectivos do exercício

- Treinar a capacidade de abstracção e representação de informação.
 - Entender o uso de referências como retorno de funções, quais as restrições associadas ao seu uso e formas de resolver essas restrições.
 - Consciencializar que as funções que são responsáveis por armazenar e manipular dados não devem interagir directamente com os periféricos nem com o utilizador, e vice-versa.
 - Preparação para o exercício seguinte.
-

2. Pretende-se agora passar a estrutura *Tabela* do exercício anterior a classe de C++, com os mesmos membros variáveis da estrutura do exercício anterior e funções membros com funcionalidade análoga às funções consideradas para a referida estrutura.

- a)** Defina a classe *Tabela* com as características enunciadas. Se mantiver a estrutura do exercício anterior tenha em atenção que deve ter um nome diferente. Inclua uma função *main* que permita testar a classe *Tabela*.
- b)** Acrescente a seguinte requisito à classe: cada objecto de *Tabela*, quando é criado deve ter todos os seus valores inicializados a 0. Acrescente o código necessário à classe para cumprir este requisito.
- c)** Identifique as vantagens desta nova versão de *Tabela* e anote-as no seu caderno.

Objectivos do exercício

(na continuação do anterior)

- Consolidar os objectivos atingidos no exercício anterior.
 - Entender e treinar o conceito de encapsulamento e compreender as suas vantagens, incluindo: compreender a diferença entre funções globais e funções membro, as diferenças entre dados globalmente acessíveis e dados privados, e as vantagens de *data-hiding*.
 - Utilizar os modificadores de visibilidade de C++ para definir o que é público e o que é privado.
 - Entender o conceito de função membro que é chamada sobre um objecto e tomar conhecimento do ponteiro *this*.
 - Entender as diferenças e as semelhanças entre estrutura e classe.
 - Utilização de construtor.
-

3. Defina uma estrutura *Caderno* contendo os seguintes dados: marca, cor, número de folhas, tamanho. Na resolução deste exercício (e genericamente, em linguagens orientadas a objectos) evite funções membro que interajam directamente com o utilizador a não ser que a classe tenha por objectivo específico essa interacção. Teste o código do exercício com uma função *main* adequada.

a) Escreva uma função que preencha o conteúdo de uma variável do tipo *Caderno*.

b) Escreva uma função para obter cada dado individual da estrutura.

c) Escreva uma função para modificar cada dado individual da estrutura.

Objectivos do exercício

- Treinar a capacidade de abstracção e representação de informação.
 - Consolidar a noção de *separação de representação e manipulação de dados para um lado, interacção com o utilizador para outro*.
 - Preparação para o exercício seguinte.
-

4. Pretende-se agora *Caderno* como classe, com os mesmos membros variáveis da estrutura do exercício anterior e funções membros com funcionalidade análoga às funções consideradas para a referida estrutura.

a) Construa a classe *Caderno* e construa ou modifique a função *main* para testar a classe.

b) Identifique as alterações principais quando *Caderno* de estrutura para classe.

c) Identifique e explique as vantagens que se obtêm ao passar *Caderno* para classe. Anote as suas conclusões.

Objectivos do exercício

(na continuação do anterior)

- Consolidar os objectivos atingidos no exercício anterior.
 - Consolidar os conceitos de encapsulamento e compreender as suas vantagens, incluindo: compreender a diferença entre funções globais e funções membro, as diferenças entre dados globalmente acessíveis e dados privados, e as vantagens de *data-hiding*.
 - Utilizar os modificadores de visibilidade de C++ para definir o que é público e o que é privado.
 - Consolidar o conceito de função membro que é chamada sobre um objecto e tomar conhecimento do ponteiro *this*.
 - Entender e consolidar o conceito de funções para aceder e para actualizar dados da classe.
 - Consolidar o conhecimento acerca das diferenças e semelhanças entre estrutura e classe.
 - Utilização de construtores.
-

5. Defina uma estrutura *Automovel* que contenha os dados necessários à representação de um automóvel (matrícula, combustível, marca, modelo, ou outros). No decorrer das alíneas seguintes deve usar uma função *main* que lhe permita testar o código que for fazendo. A função *main* não faz directamente parte do exercício.

a) Analise o problema e determine que dados devem pertencer à estrutura e quais os mecanismos e regras que devem ser materializados em funções para lidar com estas estruturas.

b) Defina a estrutura e escreva as funções que identificou.

c) Escreva uma função que preencha o conteúdo de uma variável do tipo *Automovel* com valores fornecidos. Evite interagir com o utilizador directamente dentro das funções que manipulam os dados da estrutura.

d) Escreva funções que permitam obter e actualizar cada dado desta estrutura.

Objectivos do exercício

- Treinar e consolidar a capacidade de abstracção e representação de informação.
 - Treinar a capacidade de determinação autónoma de dados e funcionalidade a incluir numa estrutura ou classe.
 - Entender as desvantagens associadas a ter funções que lidam com dados e sua representação interna a interagirem com o utilizador, por oposição a ter para um lado, funções que lidam apenas com a representação e manipulação de dados, e para outro lado, funções que interagem com o utilizador.
 - Preparação para o exercício seguinte.
-

6. Pretende-se passar a estrutura *Automovel* para classe de C++, mantendo os dados e funcionalidade. Acompanhe o código deste exercício com uma função *main* que lhe permita testar a sua classe. Este exercício também pode ser feito sem passar primeiro pelo anterior, caso em que bastará ler o enunciado do exercício anterior para saber as características de *Automovel*.

a) Construa a classe *Automovel* com as características enunciadas e inclua na sua função *main* código que permita testar a classe.

b) Analise as regras mais óbvias do mundo real acerca de automóveis, nomeadamente no que diz respeito à sua construção e inicialização. Transporte para o seu programa e classe tanto quanto possível essas regras de forma que a construção de objectos da classe *Automovel* respeite sempre essas regras.

- c) Escreva uma função que permita obter a representação textual do conteúdo da estrutura. Não se pretende necessariamente imprimir essa informação no ecrã e evite fazê-lo directamente em código da função (“obter” ≠ “imprimir”).

Objectivos do exercício

(na continuação do anterior)

- Consolidar os conceitos de encapsulamento e compreender as suas vantagens.
- Utilizar os modificadores de visibilidade de C++ para definir o que é público e o que é privado.
- Consolidar o conceito de função membro que é chamada sobre um objecto e tomar conhecimento do ponteiro *this*.
- Entender e consolidar o conceito de funções para aceder e para actualizar dados da classe.
- Consolidar o conhecimento acerca das diferenças e semelhanças entre estrutura e classe.
- Utilização de construtores.
- Consolidar o entendimento acerca das desvantagens associadas às funções membro que interagem directamente com o utilizador (a evitar) e as vantagens de delegar essa interacção para outras partes do programa mantendo as funções membro apenas a receber ou devolver dados.
- Conhecer a classe *ostream* e as suas vantagens na construção de *strings* com aplicação prática na obtenção da representação textual agregada de dados.

7. Pretende-se uma classe chamada *MSG* cujos objectos respeitem as seguintes propriedades:

- Armazenam uma letra e um número inteiro. A letra pode ser fornecida na construção dos objectos. Se não for, será assumida a letra ‘x’. O número é um valor gerado automaticamente: cada novo objecto tem o número seguinte ao do objecto anterior. O primeiro objecto tem o número 1.
 - Deve existir um mecanismo para obter os dados dos objectos sob a forma de uma *string* com o formato “letra: ... número ...”.
 - Sempre que um objecto é criado deve ser automaticamente apresentada a sua informação (formato “criado: letra: ... número ...”).
 - Sempre que um objecto deixa de existir deve ser automaticamente apresentada a sua informação no ecrã (formato “terminado: letra: ... número ...”).
- a) Construa a classe com as características enunciadas e teste a sua funcionalidade criando dois objectos, *a* e *b*, indicando ao primeiro a letra ‘a’, e ao seguinte nenhuma letra. Execute o programa de forma a que este, ao terminar, aguarde pelo pressionar de uma tecla. Observe as mensagens no ecrã e relacione a ordem pela qual aparecem com a ordem pela qual os objectos são criados e destruídos.
- b) Acrescente à função *main*, a seguir às variáveis que já existem, uma referência com o nome *c* para o objecto que tem a letra ‘x’. Execute o programa, analise e justifique as mensagens e a ordem pela qual aparecem. Explique as diferenças nas mensagens relativamente à alínea anterior.

- c) Acrescente à função *main*, a seguir às variáveis que já existem, a seguinte declaração: *MSG d=b;* Execute o programa, analise e justifique as mensagens e a ordem pela qual aparecem. Explique as diferenças nas mensagens relativamente à alínea anterior.
- d) Acrescente à função *main* a atribuição *main: a = b;* . Execute o programa e confirme que o número e ordem de mensagens não se altera (apenas a o conteúdo relativo ao objecto *a*). Comparando com a alínea anterior, explique porquê e remova a atribuição que tinha acrescentado.
- e) Acrescente uma matriz de objectos de *MSG* de dimensão dois. É possível indicar a letra inicial aos objectos dessa matriz? Execute o programa, analise e justifique as mensagens e a ordem pela qual aparecem. Explique as diferenças nas mensagens relativamente à alínea anterior.
- f) Remova da classe *MSG* a característica que permite que os seus objectos fiquem com a letra 'x' se nenhuma letra for especificada. Explique porque é que o programa deixa de compilar. Volte a por a classe como estava.
- g) Acrescente uma função *teste* do tipo *void* e sem parâmetro nenhum. Declare nessa função um objecto classe *MSG* com o nome *aux* e com a letra inicial 'y'. Invoque a função teste na função *main*. Execute o programa, analise e justifique as mensagens e a ordem pela qual aparecem. Explique as diferenças nas mensagens relativamente à alínea anterior.
- h) Modifique a função *teste* de forma a que receba como parâmetro um objecto da classe deste exercício. O objecto é passado por valor. Na função *main* adapte a chamada à função teste passando-lhe o objecto que tem a letra 'x'. Execute o programa, analise e justifique as mensagens e a ordem pela qual aparecem. Explique as diferenças nas mensagens relativamente à alínea anterior.
- i) Acrescente um construtor privado à classe *MSG* que recebe como parâmetro o seguinte: *const MSG & z* e dê um corpo vazio a esse construtor. Compile o programa e confirme que não é possível. Explique porquê.
- j) Passe o construtor a publico e coloque no seu corpo apenas a seguinte linha *cout << "construído por cópia";* . Verifique que já consegue compilar o programa e explique porque é que já é possível.
- k) Execute o programa e repare que os valores relativos ao objecto parâmetro da função *teste* não são inicializados. Explique porquê. Coloque o código adequado ao construtor por cópia de forma a que os objectos inicializados por cópia fiquem correctamente inicializados, mas o valor numérico fique o simétrico do original (para se distinguirem). Execute o programa e confirme e explique as mensagens que aparecem.

- l) Modifique função *teste* de forma a que o seu parâmetro seja passado por referência. Execute o programa, analise e justifique as mensagens e a ordem pela qual aparecem. Explique as diferenças nas mensagens relativamente à alínea anterior.
- m) Modifique a função *teste* de forma a que retorne por valor um objecto do tipo *MSG* e remova o parâmetro. No corpo da função retorne o objecto *aux*. Execute o programa, analise e justifique as mensagens e a ordem pela qual aparecem. Explique as diferenças nas mensagens relativamente à alínea anterior.
- n) Modifique função *teste* de forma a que o retorno passe a ser por referência. Execute o programa, analise e justifique as mensagens e a ordem pela qual aparecem. Explique as diferenças nas mensagens relativamente à alínea anterior.
- o) Escreva no seu caderno todas as conclusões acerca do comportamento da criação e destruição de objectos que observou ao longo deste exercício.

Objectivos do exercício

- Treinar a representação de dados.
- Treinar a identificação de funcionalidade e dados necessários mediante uma especificação em que se indicam as características necessárias e não os dados e funções directamente.
- Uso de construtores.
- Uso de dados membros estáticos.

Este exercício tem um objectivo muito forte a nível de ordem de criação e destruição de objectos e quais as diversas situações que levam a que um objecto seja explícita ou implicitamente criado. A maior parte das alíneas destina-se a cobrir:

- Criação e posterior destruição de objectos explicitamente por declaração, na passagem por valor e no retorno por valor de funções.
- Criação e posterior destruição de objectos em matrizes.
- Criação e posterior destruição de objectos em declarações cuja inicialização envolve uma aparente atribuição.
- Utilização de referências na declaração de objectos e na passagem e retorno de funções.
- Ordem de criação e destruição de objectos
- Primeiro contacto com o construtor por cópia

-
8. Transporte para o computador em C++ o conceito de *pessoa* com as características (dados e funcionalidade) principais habituais associadas a esta entidade no mundo real. Esta classe deve respeitar os princípios de encapsulamento. Tente incluir o conceito de *constante* na funcionalidade que não se destina a alterar os objectos desta classe. Uma parte importante deste exercício é a modelização sensata e não existem soluções únicas. Este exercício foi planeado para haver interacção entre o docente e os alunos. Isto não significa que os alunos devam ficar à espera da

solução, pois uma parte importante deste exercício tem a ver com a proposta autónoma de dados e funções. Não existe uma solução única ou oficial. O percurso para atingir a solução é tão importante como a solução em si.

Objectivos do exercício

- Treinar a representação de entidades do mundo real em classes de C++.
 - Treinar a identificação de funcionalidade e dados necessários mediante uma especificação em que se indicam as características necessárias e não os dados e funções directamente. Isto inclui deduzir dados e funções mediante o conhecimento das entidades do mundo real e mediante o bom senso e que podem não ser directamente mencionados no enunciado.
 - Uso de construtores.
 - Uso de *const*.
-

9. Pretende-se modelizar televisões em classes de C++. As televisões têm:

- Um conjunto predefinido de canais que conseguem sintonizar. Cada canal corresponde a um nome (o nome desse canal).

As televisões suportam a seguinte funcionalidade:

- Está ligada ou desligada. Pode-se passar de um estado para o outro (ligar / desligar). Note-se que “ligar” ≠ “construir” e “desligar” ≠ “destruir”.
- Está num determinado nível de volume. O nível menor é 0 (sem som), e o máximo é 10. Pode-se aumentar e diminuir o som, sempre uma unidade de cada vez. Só se consegue mudar o volume se a televisão estiver ligada.
- Está num determinado canal. Muda-se de canal invocando uma determinada funcionalidade à qual se indica o número de canal pretendido. Se o canal for válido, a televisão envia para o ecrã o nome do canal para o qual passou. Caso contrário, mantém-se no mesmo canal e nada acontece. Só se consegue mudar de canal se a televisão estiver ligada.
- Deve ser possível obter a representação textual do estado interno da televisão. Se estiver desligada, essa informação corresponde a “desligada”. Se estiver ligada, a informação é o nome do canal, o número do canal (o primeiro nome corresponde ao canal 1, o segundo ao 2, etc.) e o volume de som. Note-se que “obter” ≠ “imprimir”.

As televisões, quando são construídas, obrigam a que seja definido logo um conjunto de canais e esse conjunto nunca mais se pode alterar.

a) Construa a classe pretendida assumindo que existe uma capacidade máxima de 10 canais. Construa uma função *main* para a testar. Construa várias televisões com diferentes conjuntos de canais. Ligue-as e desligue-as. Mude o volume e de canal. Não fique demasiado preso aos programas desportivos e escreva no caderno as conclusões deste exercício.

b) Modifique a classe assumindo que afinal não existe limite específico quanto ao número de canais. Não se pretende para já uma implementação com memória dinâmica.

Objectivos do exercício

- Treinar a representação de entidades do mundo real em classes de C++.
 - Treinar a identificação de funcionalidade e dados necessários mediante uma especificação em que se indicam as características necessárias e não os dados e funções directamente. Isto inclui deduzir dados e funções que não são sequer mencionados no enunciado.
 - Uso de construtores.
 - Uso de *const*.
 - Uso da classe *ostringstream*.
 - Primeiro contacto com a classe *vector* da *STL*.
-