Instituto Superior de Engenharia de Coimbra

Engenharia Informática e de Sistemas

Programação Orientada a Objectos (2020/21)

Exercícios

Ficha 7

Herança e polimorfismo
Exercícios englobando toda a matéria

1. Uma agência imobiliária tem, para vender, apartamentos e lojas comerciais. Os apartamentos são caracterizados pelo preço, área, andar e número de assoalhadas. As lojas são caracterizadas pelo preço e área, situando-se sempre no rés-do-chão. Qualquer destes bens imobiliários é caracterizado também por um código, gerado automaticamente, sendo formado pela palavra "apartamento" ou "loja", seguida por um inteiro que corresponde à ordem pela qual o objecto foi gerado (a sequência é a mesma para todos os bens). OS apartamentos têm um preço que é sempre 10 x área. As lojas têm um preço que é 15 x área. Tanto os apartamentos como as lojas devem permitir:

- Obter o código
- Obter o preço
- Obter uma *string* com a descrição de todos os dados
- O mesmo que que o ponto anterior mas através de *cout* <<

> ***O texto seguinte não faria parte do exercício se este aparecesse num exame****. No contexto de exercício para treino, são acrescentados algumas indicações que mais tarde é suposto nem ser preciso dizer.*
>
> Repare nos seguintes aspectos:
> Existem aqui dois conceitos muito relacionados entre si, mas ao mesmo tempo claramente distintos: apartamentos e lojas. Aquilo que partilham é relativamente óbvio. As diferenças que existem são tanto a nível de dados como e a nível de comportamento/restrições:
> - Dados: um apartamento tem assoalhadas; uma loja não.
> - Comportamento: uma loja está sempre no primeiro andar; um apartamento pode estar em qualquer andar.
>
> A forma correcta de modelizar esta situação em C++ será a de fazer corresponder uma classe distinta a cada entidade Apartamento e Loja, e usar herança para fazer ambas as classes herdarem aquilo que partilham a partir de uma classe base comum: ambas as classes derivam de uma classe que encapsula os dados e comportamentos partilhados.

a) Defina as classes que representam estes bens dois bens imobiliários usando adequadamente os mecanismos de C++ vocacionados para a reutilização de código e extensão de conceitos da forma conceito base – conceito especializado. Ajuda: vão ser necessárias três classes. Todas as classes garantir que os seus objectos só são construídos se lhe forem passados todos os dados relevantes.

b) Defina a classe Imobiliaria que representa (que armazena) todo o conjunto de bens imobiliários que a agência tem para vender neste edifício. Deve ser possível acrescentar bens imobiliários do edifício, fazer listagens de bens imobiliários por andar e pesquisa por código, obter os preços e descrição, e remover por código. Deve também ter em atenção que não deve utilizar uma estrutura de dados para cada tipo diferente de bem imobiliário (caso contrário o professor vai propor a existência de 50 tipos de imóveis) - podem mesmo existir muitos tipos diferentes – deve unificar o armazenamento de todos os bens imobiliários numa única estrutura de dados.
A imobiliária deste exercício partilha algumas semelhanças com as imobiliárias da vida real:

- Não é a imobiliária que constrói os edifícios
- Se ocorrer um curto-circuito no sistema de informação da imobiliária, os edifícios continuam a existir.

c) Teste a funcionalidade da alínea anterior criando alguns imóveis, registando-os na imobiliária, e depois procurando-os/listando-os/etc.

Objectivos do exercício

- Treinar a identificação de situações em que é apropriado o uso de herança, a identificação e a definição das classes base e derivadas envolvidas.
- Treino das questões sintácticas envolvidas na herança e em particular, nos construtores. Acesso a dados privados da classe base a partir da classe derivada, e à funcionalidade em funções homónimas da classe derivada para a classe base (alínea a).
- Uso de polimorfismo (alínea b).

2. Pretende-se construir um programa para organização de uma colecção de **CDs** de música. Existem vários tipos de **CD** e as alíneas seguintes conduzem a elaboração do programa um passo de cada vez.

a) Construa uma classe **CD**. Esta classe deve ter a informação sobre o título do **CD** e de cada uma das suas faixas. A classe **CD** deve cumprir o seguinte:

- Deve ter uma função membro *acrescentaFaixa* que permite acrescentar o título de uma nova faixa (*string*) ao **CD**.
- Não deve ser possível a criação de um **CD** sem dados acerca do seu título.
- Deve existir uma função que retorna uma string com a descrição de toda a informação sobre o **CD**.

b) Sabendo que existem vários tipos de **CDs** (clássica, banda sonora, etc.), pretende-se agora que defina as seguintes classes derivadas da classe CD:

- **Classica**: Deve incluir o nome do maestro e do compositor.
- **BandaSonora**: Deve incluir o nome do filme.

O comportamento das classes derivadas deve ser semelhante ao da classe **CD**. A função que retorna uma *string* com a descrição de toda a informação sobre o CD deve incluir todos os atributos de cada classe.

c) Construa a classe **ColeccaoCD** que armazena vários **CDs**. Tenha em atenção que nesta colecção os **CDs** podem ser dos vários tipos de **CDs** já indicados e não deve ter uma estrutura de dados para cada tipo de CD diferente (imagine que existem muitos tipos possíveis de CD).

A classe deve cumprir também o seguinte:

- Devem existir funções membros que permitem acrescentar CDs dos diversos tipos a esta colecção.
- Deve ser possível exibir toda a informação sobre os CDs da colecção a partir do operador << e do objecto cout.

No caso de estar a perguntar-se a si próprio como é que a classe vai armazenar CD, foque-se nestes três aspectos: 1) são vários CD, 2) são de tipos diversos, 3) ninguém obrigou/proibiu nenhuma solução em particular.

d) Provavelmente não pensou no caso da atribuição e da construção por cópia de colecções de CD. Apesar de ser contra as leis de direitos de autor, a classe deve suportar estes mecanismos, e sabe-se que os CD pertencem à colecção em que estão inseridos. Não se esqueça também que existem vários tipos de CD na colecção.

Objectivos do exercício

- Treinar a identificação de situações em que é apropriado o uso de herança, a identificação e a definição das classes base e derivadas envolvidas.
- Treino das questões sintácticas envolvidas na herança, incluindo a invocação de construtores, o acesso a dados privados da classe base a partir da classe derivada, e a invocação de funcionalidade em funções homónimas da classe derivada para a classe base.
- Uso de polimorfismo.
- Duplicação polimórfica (alínea d).

---

3. Pretende-se um conjunto de classes para lidar com os conceitos de livros e biblioteca.

a) Defina a classe Livro, que tem por objectivo encapsular o conceito de Livro, o qual é composto por:

- Um título;
- Um autor;
- Um ISBN.

Pretende-se que os objectos da classe apenas possam ser inicializados com a informação relativa ao título, autor e ISBN. Devem ser suportadas as seguintes operações sobre os objectos de Livro:

- Comparação de dois livros com o operador ==. Dois livros são iguais se tiverem o mesmo ISBN;
- Apresentação da informação através de *cout* << *objecto-de-livro*.

b) Defina as classes **LivroPolicial** e **FiccaoCientifica** com as características a seguir definidas.

A classe **LivroPolicial** representa um livro como foi considerado na pergunta anterior mas com as seguintes diferenças/acrescentos:

- Nome do detective (cadeia de caracteres);
- Número de tiros disparados na história;
- Apresentação dos dados no ecrã: se o número de tiros for superior a 10, em vez do número, é apresentada a recomendação "Não aconselhado a crianças".

A classe **FiccaoCientifica** também representa um livro como foi considerado na pergunta anterior mas com as seguintes diferenças/acrescentos:

- Nome do planeta em que a acção de passa;
- Ano em que a acção de passa;
- Realista (Sim/Não) - indica se a ficção relatada é realista ou puramente fantasiosa.

Ambas as classes devem exigir os dados aqui listados na inicialização dos seus objectos.

c) Defina a classe Biblioteca que seja capaz de armazenar livros (note-se: qualquer tipo de livro) de uma forma eficiente e uniformizada. Cada objecto desta nova classe deve armazenar:

- Um número indeterminado de livros;
- A morada da biblioteca.

Em qualquer altura deve ser possível:

- Acrescentar um livro; o livro apenas será adicionado se ainda não existir na biblioteca. A biblioteca toma a posse exclusiva do livro (passa a ser da biblioteca).
- Remover um livro dado o seu ISBN;
- Listar o título de todos os livros.

d) Provavelmente esqueceu-se de pensar no que acontecerá se atribuir e copiar objectos de Biblioteca. Os pormenores exactos dependem da forma como armazenou os livros, mas uma coisa é certa: um livro não pertence a duas bibliotecas diferentes e existem livros de muitos tipos diferentes na biblioteca. Trate do assunto da cópia e atribuição de bibliotecas sem rasgar livros ao meio nem baralhar as capas dos livros.

Esta situação é semelhante à da última alínea do exercício anterior. É um assunto importante. Se tiver questões deve mesmo perguntar ao professor do laboratório.

Objectivos do exercício

- Treinar a identificação de situações em que é apropriado o uso de herança, a identificação e a definição das classes base e derivadas envolvidas.
- Treino das questões sintácticas envolvidas na herança, incluindo a invocação de construtores, o acesso a dados privados da classe base a partir da classe derivada, e a invocação de funcionalidade em funções homónimas da classe derivada para a classe base.
- Uso de polimorfismo.
- Duplicação polimórfica (alínea d).

---

4. Um ginásio decidiu informatizar toda a parte da gestão de clientes. Esta gestão envolve clientes, tarifários e outros aspectos. A descrição do problema é feita a partir dos aspectos mais pormenorizados para os mais gerais. É mesmo necessário ler o enunciado todo antes de começar a responder.

a) O montante a pagar por cada cliente do ginásio é determinado por um tarifário. Os tarifários são representados pela classe **Tarifario** que tem as seguintes características:

- Armazena um conjunto de inteiros que representam as durações dos treinos efectuados. Este armazenamento é feito sem usar os contentores da STL (vector, etc.).

Tem os métodos:

- *acrescentaTreino*. Recebe uma duração de treino feito e acrescenta à colecção.
- *apagaTreinos*. Apaga as durações de todos os treinos. Esta função é obrigatoriamente protected.
- *calculaPagamento*. Calcula o montante em dívida correspondente aos treinos armazenados, apaga os treinos e devolve o valor a pagar. O algoritmo que determina o valor a pagar não é conhecido nesta classe por isso este método não pode ser implementado.

Implemente a classe **Tarifário**.

b) Existe um tarifário especial destinado a promover treinos curtos. O cálculo do custo neste tarifário é o seguinte: Cada treino até 10 minutos custa 10; um treino entre 11 e 20 minutos custa 15; um treino de 21 minutos ou mais custa 25. Este tarifário é representado pela classe Apressado. Os objectos da classe Apressado são, naturalmente, também objectos de Tarifário. Existem outros tipos de tarifário, mas para este exercício apenas é necessário considerar este.

Implemente a classe **Apressado**.

c) Implemente a classe **Cliente** que representa um cliente genérico do ginásio. Os clientes deste ginásio são de variados tipos, mas todos eles têm as seguintes características em comum:

- Têm nome e BI.
- Têm um tarifário.

Têm os seguintes métodos:

- ***iniciaTreino*** que recebe um inteiro que representa a hora em que começou um treino no ginásio. A hora é representada como o número de minutos a partir de um certo instante (mais pormenores na explicação acerca do ginásio). A hora é memorizada.
- ***terminaTreino*** que recebe o inteiro que representa a hora de saída do treino. A duração do treino é calculada e passada para um objecto Tarifário que o cliente tem.

- ***paga***. Este método pergunta ao objecto Tarifário quanto é que é devido e esse valor é por sua vez devolvido por este método. A devolução do valor simboliza o pagamento pelos treinos efectuados desde o último pagamento.
- ***reageEntrada*** que reage à entrada de um cliente qualquer (outro cliente que não ele) no ginásio. Este método é genérico e não pode ser implementado já.
- ***reageSaida*** que reage à saída de um cliente qualquer (outro cliente que não ele) do ginásio. Este método é genérico e não pode ser implementado já.
- A criação de objectos deste tipo obriga à indicação (por parâmetro) do nome, BI e do objecto Tarifário associado ao cliente.

Ainda acerca da classe **Cliente** há a dizer o seguinte:

- O cliente aqui descrito é um cliente genérico e estes comportamentos podem ser modificados em clientes mais específicos. A classe Cliente reflecte este aspecto
- Este cliente é tão genérico que tem métodos que nem se sabe por agora o que fazem (ex., ***reageEntrada***, ***reageSaida***). O melhor é que o código da classe seja feito de tal forma que impeça a criação de objectos.
- Por alguma razão (na verdade, é para simplificar o problema) é proibido copiar e atribuir objectos da classe Cliente. O código da classe deve impedir a cópia e atribuição dos seus objectos.

d) Afinal parece que se podem copiar e atribuir objectos de Cliente. Analise o código já feito e veja se é preciso alguma coisa adicional para que essas operações sejam suportadas de forma correcta. Escreva as alterações que entender serem necessários às classes que estão para trás.

e) Existem diversos tipos de cliente. Um deles é um tipo muito peculiar, com as seguintes características.

- Quando um (outro) cliente entra, não faz nada.
- Quando um (outro) cliente sai, verifica se está alguém (para além dele) no ginásio. Se não estiver, sai também, pois não gosta de estar sozinho.

Por razões óbvias, este cliente é representado pela classe **Sociável**. Implemente a classe **Sociável** tendo em atenção que se pretende que haja efectivamente objectos desta classe.

f) Por fim, pretende-se implementar o ginásio através da classe com o nome de **Ginasio**. As características do ginásio são as seguintes:

- Armazena um conjunto de clientes que podem ser de diversos tipos. Os “clientes” pertencem mesmo ao ginásio. Não se trata de armazenar pessoas dentro do ginásio, mas sim informação que representa uma pessoa que é cliente do ginásio. Imagine que o cliente

é um pedaço de papel num arquivo no escritório do ginásio (por outras palavras: os clientes pertencem ao ginásio – este texto parêntesis não apareceria num exame).

- Possui um relógio (para controlar os minutos a cobrar aos clientes). O relógio segue uma novíssima norma mundial simplificada: começa em zero, tem incrementos de 1 e não tem segundos nem horas. É apenas um inteiro que começa em zero e aumenta de 1 em 1 (se achar que este modelo de relógio não é realista, pode implementar um relógio mais complexo, mas o tempo para responder à questão é o mesmo).

O ginásio tem mecanismos (métodos) para:

- Fazer passar um certo número de minutos.
- Acrescentar um cliente ao ginásio. O cliente é previamente construído e passado ao método. Este método é compatível com qualquer tipo de cliente.
- Remover um cliente, dado o BI.
- Entrada de um cliente no ginásio – o cliente indicado pelo BI passa a treinar (entra na sala de treino). Todos os clientes a treinar no ginásio (subconjunto dos que existem ao todo no ginásio) são notificados que este cliente entrou. O cliente é informado da hora a que entrou. O cliente é especificado por BI e terá que existir previamente no ginásio.
- Saída de um cliente – o cliente indicado pelo BI termina o treino e sai do ginásio (da sala de treino). Todos os clientes restantes a treinar no ginásio são notificados que este cliente saiu. O cliente é informado da hora a que saiu. O cliente é especificado por BI e terá que existir previamente no ginásio.
- Pagamento da conta de um cliente, dado o BI.

Importante: Nesta alínea não precisa de se preocupar com aspectos relacionados com atribuição e cópia de objectos desta classe.

g) Reveja o código da classe Ginasio e verifique se as operações de atribuição e cópia funcionaram bem. Se não for o caso, faça as alterações que forem necessárias

Eventualmente, na resolução das alíneas deste exercício, pode ser necessário acrescentar mais campos ou métodos às classes. Se achar que isso acontece, acrescente e assinale a razão.

Nota: Este exercício é, fundamentalmente, igual a um que saiu em exame. Pode ser considerado com um bom treino para os próximos exames. Deve ser tido em consideração que existiam algumas simplificações no enunciado do exame que aqui foram levantadas. Esta versão é maior do que aquilo que saiu no exame. Outro aspecto a ter em consideração é o seguinte: este exercício corresponde aproximadamente ao limite de complexidade que sai no exame.

> Objectivos do exercício
>
> - Treinar a identificação de situações em que é apropriado o uso de herança, a identificação e a definição das classes base e derivadas envolvidas.

- Treino das questões sintácticas envolvidas na herança, incluindo a invocação de construtores, o acesso a dados privados da classe base a partir da classe derivada, e a invocação de funcionalidade em funções homónimas da classe derivada para a classe base.
- Uso de polimorfismo
- Treinar resolução de exercícios que envolvem situações menos-que-triviais envolvendo objectos de classes diferentes que interagem entre si de forma bidireccional (consolidação de exercícios da ficha 6).
- Ver exemplos de exercícios muito semelhantes ao que pode sair no exame, parte prática.

---

5. Pretende-se um programa para uma operadora de telemóveis. Esta operadora permite aos seus clientes a utilização de diversos tarifários com cartões recarregáveis. Os conceitos principais envolvidos no programa são:

- **Cartão** – representa essencialmente um número de telemóvel (que pertence a alguém) e ao qual está associado um tarifário.
- **Tarifário** – representa essencialmente um mecanismo que determina quanto custam as chamadas efectuadas pelos cartões que têm o tarifário. Podem envolver dados que descrevem as chamadas efectuadas.
- **Rede** – Representa o conjunto de cartões, tarifários e funcionalidade para lidar com estes.

Vão existir diversos tipos de tarifários e o cartão tem que ser compatível com todos.

a) Construa a classe **Cartao**, a qual tem as seguintes características:

- A cada cartão corresponde o número de telemóvel e o saldo, e está associado um tarifário.

O cartão tem a seguinte funcionalidade:

- Autorizar uma chamada;
- Registar uma chamada dada a duração em segundos;
- Fazer um carregamento, dada a quantia.

b) Pretende-se construir as várias classes para os vários tipos de tarifários que se descrevem mais abaixo. Uma vez que existem diversos tipos tarifários e o cartão tem que ser compatível com todos, descreva como é que se deve proceder para unificar todos os tarifários. Proponha a solução para esta questão e implemente aquilo que for preciso. É necessário ler o enunciado todo, inclusive as alíneas que se seguem, para conseguir responder a esta alínea. Os tarifários concretos propriamente ditos não são para fazer nesta alínea mas sim nas seguintes.

c) Implemente o tarifário **Tagarela**.

O tarifário **Tagarela** tem o seguinte comportamento:

- Uma chamada é autorizada se o saldo for positivo ou, se o saldo for negativo, não ultrapassar o preço do primeiro minuto.
- O primeiro minuto é sempre pago, custa 0.5E, os minutos seguintes pagam-se a 0.02E
- Só admite carregamentos superiores ou iguais a 25E. Em carregamentos maiores ou iguais a 50E o cartão recebe um bónus de 5E.

d) Implemente o tarifário **FalaPouco**

O tarifário **FalaPouco** tem o seguinte comportamento:

- Uma chamada é autorizada se o saldo for positivo ou, se o saldo for negativo, não ultrapassar 10 vezes o preço do minuto.
- Todos os minutos são pagos a 0.25E.
- Só admite carregamentos superiores ou iguais a 10E. Em carregamentos maiores ou iguais a 10E existe 20% de probabilidade de o cartão duplicar o carregamento que foi feito.

e) Implemente a classe **Rede**

A classe rede representa a operadora de telemóveis e tem uma colecção que abrange os diversos tarifários e os cartões. Permite as seguintes operações:

- Acrescentar cartões dos diversos tipos;
- Listar os cartões existentes;
- Remover cartões;
- Verificar se é autorizada uma chamada a partir dum cartão com um certo número;
- Registar uma chamada, dado o número do cartão e a duração da chamada em segundos;
- Fazer um carregamento, dado o número do cartão e a quantia.

Eventualmente, na resolução das alíneas deste exercício, pode ser necessário acrescentar mais algum campo ou método a uma ou outra classe. Se achar que isso acontece, acrescente e assinale a razão.

Nota: Este exercício também é, fundamentalmente, igual a um que saiu em exame. Pode ser considerado com um bom treino para os próximos exames

Objectivos do exercício

- Treinar a identificação de situações em que é apropriado o uso de herança, a identificação e a definição das classes base e derivadas envolvidas.
- Treino das questões sintácticas envolvidas na herança, incluindo a invocação de construtores, o acesso a dados privados da classe base a partir da classe derivada, e a invocação de funcionalidade em funções homónimas da classe derivada para a classe base.
- Uso de polimorfismo.
- Treinar resolução de exercícios que envolvem situações menos-que-triviais envolvendo objectos de classes diferentes que interagem entre si de forma bidireccional (consolidação de exercícios da ficha 6).
- Ver exemplos de exercícios muito semelhantes ao que pode sair no exame, parte prática.
