# Programação Orientada a Objectos 2023/2024

## Exercícios

## Ficha Nº 8

**Herança e polimorfismo**
Inclui assuntos de classes com recursos dinâmicos / memória dinâmica

**1. Imóveis / Imobiliária** – Exercício preparatório, feito/apresentado de forma interativa com o professor

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 objeto 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 faz normalmente parte de um enunciado (e muito menos num exame)**.*
> *Na sequência do exercício anterior deve ter ficado em condições de identificar os aspetos seguintes.*
>
> 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 correta 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 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. Atenção: no enunciado existem dois tipos de imóveis, mas facilmente poderiam ser 200, e pretende-se que o programa seja expansível para novos tipos de imóvel. Ajuda: vão ser necessárias três classes.
As classes devem garantir que os seus objetos só são construídos se lhe forem passados todos os dados relevantes.

**b)** Defina a classe Imobiliaria que representa 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 200 tipos de imóveis) - podem mesmo existir muitos tipos diferentes – deve unificar o armazenamento de todos os bens imobiliários numa única estrutura de dados.
Neste exercício os imóveis não representam artefactos internos à classe imobiliária, mas sim coisas com existência própria:

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

**c)** Debata com o professor: o que seria diferente se a imobiliária fosse dona objetos Imóvel? Nomeadamente, quanto a

(1) Inserção de novos objetos – já existem antes? É a imobiliária que os constrói? E se forem muitos tipos?

(2) O que acontece se a imobiliária for destruída, copiada, etc.?

Nesta alínea não se pretende código nenhum – apenas um breve debate de 3 ou 4 minutos.

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

No final do exercício deverá ter

- Compreendido os casos de uso onde se aplica herança
- Compreender o conceito de classe base e como se identifica, e o conceito de classe derivadas e como se relaciona com a classe base
- Compreendido a sintaxe envolvida na herança incluindo, sem exclusão de outros aspetos: a definição de construtores em classes derivadas, a e a forma correta de aceder a dados e funcionalidades da classe base
- Entendido e aplicado o conceito de polimorfismo, e percebido o conceito de função virtual.

## 2. CD de música / Coleção de música

Pretende-se construir um programa para organização de uma coleção de **CDs** de música. Existem vários tipos de **CD** e as alíneas seguintes conduzem à 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**.
- Deve existir uma função "play" para tocar (simbolicamente) o CD. Esta função apresenta apenas no ecrã o título do CD.

**b)** Sabendo que existem vários tipos de **CDs** (clássica, banda sonora, etc.), pretende-se agora que acrescente ao seu programa a capacidade de representar estes dois tipos específicos de 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**. Tome nota:

A função que retorna uma *string* com a descrição de toda a informação sobre o CD deve incluir **todos** os dados de cada classe.

A função para tocar o CD deve indicar:

- **Classica**: Indica: "a tocar música boa", seguido do nome do maestro e depois o título.
- **BandaSonora**: indica: o título, depois o nome do filme seguido de "já não se fazem filmes como dantes".

Para o caso de estar a pensar em usar uma estratégia "C-based" com, um "int tipo" + "int getTipo()" + "if-else" algures no programa, então o enunciado passa a ser: existem 500 tipos diferentes de CD e boa sorte com esse if-else. Posteriormente, ainda terá que acrescentar mais 1 tipo de CD.

**c)** Construa a classe **ColecaoCD** que armazena vários **CDs**. Tenha em atenção que nesta coleção 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 (recorde a chamada de atenção acerca de possíveis 500 tipos diferentes de CD mencionada na alínea anterior).

A classe deve cumprir também o seguinte:

- Devem existir funções membros que permitam acrescentar CDs dos diversos tipos a esta coleção.

  **Importante**: Neste aspeto a solução mais adequada neste momento poderá não ser a mais elegante (exige *templates*) ou a mais intuitiva face à estratégia seguida em exercícios de composição já feitos. É importante que perceba o que está em jogo na funcionalidade de acrescentar CD quando existem vários tipos de CD possíveis.

- Deve ser possível exibir toda a informação sobre os CDs da coleção a partir do operador << e do objeto *cout*.

No caso de estar a perguntar-se a si próprio como é que a classe vai armazenar CD, foque-se nestes três aspetos: 1) são vários CD, 2) são de tipos diversos, 3) Pode haver muitos tipos mesmo que no enunciado só sejam mencionados dois, 4) Já que o enunciado não o proíbe, pode usar coleções, mas pode usar outra estratégia qualquer desde a sua implementação seja completa e coerente.

Não se esqueça que os CD pertencem à coleção onde estão inseridos. Deitar a coleção fora significa deitar todos os CD fora.

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

No final do exercício deverá ter

- Compreendido os casos de uso onde se aplica herança
- Compreender o conceito de classe base e como se identifica, e o conceito de classe derivadas e como se relaciona com a classe base
- Compreendido a sintaxe envolvida na herança incluindo, sem exclusão de outros aspetos: a definição de construtores em classes derivadas, a e a forma correta de aceder a dados e funcionalidades da classe base
- Entendido e aplicado o conceito de polimorfismo, e percebido o conceito de função virtual.
- Entendido as ramificações e consequências de ter recursos polimórficos por composição numa classe e a relação deste cenário com objetos dinâmicos.
- Compreendido e aplicado o conceito de duplicação polimórfica, e ter entendido onde surge a necessidade de duplicar objetos sem saber que tipo de objetos se está a duplicar.

**3. Livros / Biblioteca** – Para consolidação / estudo autónomo em casa (TPC)

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

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

- Um título;
- Um autor;
- Um ISBN.

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

- Comparação de dois livros com o operador ==. Dois livros são iguais se tiverem o mesmo ISBN;
- Apresentação da informação através de *cout* << *objeto-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 detetive (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 ação de passa;
- Ano em que a ação 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 objetos.

**c)** Defina a classe Biblioteca que seja capaz de armazenar livros (note-se: qualquer tipo de livro) de uma forma eficiente e uniformizada. Cada objeto 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 objetos de Biblioteca. Os pormenores exatos 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.

No final do exercício deverá ter
(atingido os mesmos objetivos do exercício anterior)

- Compreendido os casos de uso onde se aplica herança
- Compreender o conceito de classe base e como se identifica, e o conceito de classe derivadas e como se relaciona com a classe base
- Compreendido a sintaxe envolvida na herança incluindo, sem exclusão de outros aspetos: a definição de construtores em classes derivadas, a e a forma correta de aceder a dados e funcionalidades da classe base
- Entendido e aplicado o conceito de polimorfismo, e percebido o conceito de função virtual.
- Entendido as ramificações e consequências de ter recursos polimórficos por composição numa classe e a relação deste cenário com objetos dinâmicos.
- Compreendido e aplicado o conceito de duplicação polimórfica, e ter entendido onde surge a necessidade de duplicar objetos sem saber que tipo de objetos se está a duplicar.

## 4. *Aquário com peixes diversos*

Pretende-se um programa que simule a existência de **peixes** de **diversas espécies** num **aquário**.

**Acerca dos peixes**: Há muitas espécies. Todas elas exibem as seguintes características comuns:

- Aparentam ter um nome da espécie (texto), uma cor (texto), um peso (gramas), ID (número de série que é um valor automaticamente atribuído e sempre crescente começando em 1000).

Permitem:

- Apresentar os seus dados com operações cout << peixe1 << peixe2 << etc.;
- Os peixes têm a possibilidade de serem alimentados com uma certa quantidade em gramas de comida. Existem diversas espécies de peixes e cada espécie exibe um comportamento diferente quando é alimentado.

Há muitas espécies de peixes. O enunciado aborda apenas duas, sem que isso constitua alguma simplificação:

- **Carpa**: acrescente ao seu peso os gramas de comida que lhe é dada. Se o seu peso exceder 50, divide-se em dois, ficando o original com 20 gramas, e passando a existir no aquário um novo peixinho com as mesmas características (mesma cor etc. mas com um número de série diferente). As carpas têm sempre o nome de espécie "carpa" e têm inicialmente o peso 5.
- **Tubarão**: ignora a comida que lhe é dada. Se tiver mais do que 20 gramas, diminui um grama de peso. Se tiver menos de 20 gramas procura um outro peixe qualquer e come-o, acrescentando ao seu peso o peso do peixe ingerido. Se não houver nenhum peixe para comer e já tiver menos de 20 gramas, diminui o seu peso em 2 gramas. Se tiver menos de 5 gramas morre. Tem sempre o nome de espécie "Tubarão" e o peso inicial de 15.

**Acerca do aquário**: tem uma quantidade indeterminada de peixes. Possui mecanismos para:

- Acrescentar um novo peixe. O peixe é "apresentado" ao aquário, o qual faz duplicado dele, guardando-o e controlando-o em exclusivo.

  **Importante**: Este comportamento (guardar uma cópia) não é coerente com o que se passa na vida real, mas é coerente com o que tem que acontecer face ao que já foi discutido no exercício da coleção de CD. Em caso de dúvida, pergunte ao professor as razões e quais as alternativas.
- Eliminar um peixe dado o seu ID.
- Dado o ID de um peixe, obter o ID de um outro peixe qualquer (um qualquer menos esse) (-1 se não houver).
- Obter o ponteiro para um peixe dado o seu ID. Deve proteger esse peixe contra modificações.
- Alimentar os peixes com uma dada quantidade de comida.
- Produzir uma cadeia de caracteres com a descrição de todos os peixes.

O aquário pode ser copiado. Os dois aquários ficam com conteúdo equivalente (apenas os números de série dos peixes serão diferentes) mas totalmente independentes um do outro.

**Ajuda1**: O processo de alimentação de um peixe pode modificar a estrutura de peixes no aquário: pode haver peixes a morrer ou peixes novos acrescentados. O ciclo que percorre essa estrutura, alimentando um peixe de cada vez, deve ter este aspeto em atenção.

**Ajuda2**: planeie primeiro as classes e a interação entre elas antes de fazer código.

No final do exercício deverá ter

- Aprendido a lidar com situações de inclusão (.h) circular e de como resolver essas situações.
- Aprendido a lidar com situações em que uma coleção pode mudar inesperadamente enquanto é percorrida.
- Consolidado os conceitos de herança e polimorfismo em cenários de composição. Esta descrição genérica abrange tudo o que está para trás em exercícios anteriores a este nesta ficha e, naturalmente, muito do que está para trás em fichas anteriores.

## 5. Simulador de Aquário com peixes diversos – *definitivamente TPC*

**Pegue no exercício anterior e complete os seguintes 3 desafios**

**Desfio 1**: atribua coordenadas (dentro do aquário) aos peixes e inclua uma interface que permita ao utilizador ver o aquário (considere um aquário "plano" - bidimensional). Use ncurses ou outra biblioteca qualquer que permita ver peixes nas suas coordenadas.

**Desafio 2**: acrescente comportamento aos peixes: "a cada instante cada peixe faz algo, consoante a sua espécie". A Carpa pode fugir se pressentir um tubarão perto e o tubarão poderá perseguir carpas ou até tubarões menores. Dotar o peixe da capacidade de se aperceber o que o rodeia é interessante e fácil de fazer.

**Desafio 3**: Adicione um mecanismo de "passagem de tempo" ao aquário de forma a que o utilizador possa ordenar a passagem para o instante seguinte e ver os peixes efetuarem as respetivas ações.

**Sugestão 1**: acrescente outras espécies de peixes. Acrescente a sua interface de utilizador para permitir adicionar peixes, alimentar, etc.

No final das sugestões e desafios: parabéns – construiu um simulador de aquário.

No final do exercício deverá ter

- Consolidado os conceitos de herança e polimorfismo em cenários de composição
- Percebido que C++ e programação orientada a objetos facilitam bastante a construção de programas que se fossem feitos em estilo-C seriam muito mais complexos de fazer.

## 6. Ginásio com diversos tipos de clientes – Consolidação e inclusão de matriz dinâmica

Um ginásio decidiu informatizar toda a parte da gestão de clientes. Esta gestão envolve clientes, tarifários e outros aspetos. A descrição do problema é feita a partir dos aspetos 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 efetuados. 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 coleção.
- *apagaTreinos*. Apaga as durações de todos os treinos. Esta função deve ser 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 objetos da classe Apressado são, naturalmente, também objetos 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 objeto Tarifário que o cliente tem.
- ***paga***. Este método pergunta ao objeto 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 efetuados 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 objetos deste tipo obriga à indicação (por parâmetro) do nome, BI e do objeto 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 reflete este aspeto.
- 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 objetos.
- Por alguma razão (na verdade, é para simplificar o problema) é proibido copiar e atribuir objetos da classe Cliente. O código da classe deve impedir a cópia e atribuição dos seus objetos.

**d)** Afinal parece que se podem copiar e atribuir objetos de Cliente. Analise o código já feito e veja se é preciso alguma coisa adicional para que essas operações sejam suportadas de forma correta. Escreva as alterações que entender serem necessárias 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 efetivamente objetos 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 aspetos relacionados com atribuição e cópia de objetos 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.

No final do exercício deverá ter

- Consolidado os conceitos de herança e polimorfismo em cenários de composição. Esta descrição genérica abrange tudo o que está para trás em exercícios anteriores a este nesta ficha e, naturalmente, muito do que está para trás em fichas anteriores.
- Focado situações de herança com gestão de recursos dinâmicos.
- Lidado com exercícios mais complexos e de maior dimensão, envolvendo combinações de coleção-de-biblioteca com *arrays* dinâmicos.

## 7. **Operadora de telemóveis** - Consolidação

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 efetuadas pelos cartões que têm o tarifário. Podem envolver dados que descrevem as chamadas efetuadas.
- **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 coleção 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.

No final do exercício deverá ter

- Consolidado os conceitos de herança e polimorfismo em cenários de composição. Esta descrição genérica abrange tudo o que está para trás em exercícios anteriores a este nesta ficha e, naturalmente, muito do que está para trás em fichas anteriores.
- Focado situações de herança com gestão de recursos dinâmicos.
- Lidado com exercícios mais complexos e de maior dimensão, envolvendo combinações de coleção-de-biblioteca com *arrays* dinâmicos.