UAlg ISE
UNIVERSIDADE DO ALGARVE
INSTITUTO SUPERIOR DE ENGENHARIA

Ficha de Exercícios

Universidade do Algarve - Instituto Superior de Engenharia

Departamento de Engenharia Eletrotécnica

Tecnologias de Informáticas

Programação Orientada a Objetos

2023-24

Pedro Cardoso, Pedro Vaz Martins


Última Atualização: 7 de novembro de 2023

Conteúdo

Lista de Exercícios para avaliação.

- Os Exercícios serão classificados 0 - 5 de acordo com ponderação: 0 não defendeu **ou não soube explicar a sua resolução/copiou/usou ferramentas online**, 1 - mau, 2 - insuficiente, 3 - suficiente, 4 - bom, 5 - muito bom.
- **Cópia / plágio / uso de ferramentas online / não saber explicar implementação são classificadas com 0 e não tem direito a melhoria. O aluno fica reprovado à componente prática, e correspondentemente à UC, se tiver 5 exercícios classificados com 0 devidos ao motivos apresentados neste item.**
- **Os exercícios devem ser apresentados até 14 dias após a data da aula a que correspondem – preferencialmente na aula.**
- **Os exercícios terão o desconto de 5% por semana de atraso à data limite de apresentação (a contar do 1º dia de atraso). Neste contexto, a defesa deve ser feita até à data do exame de recurso.**
- A lista de exercícios para avaliação poderá sofrer alterações pontuais pelo que se aconselha validar com alguma frequência os exercícios que são para avaliação.

Aula 1. 1.f, h, i, j, 2.c, e, 3.f, i, j, k, 4.g, h, 6.a, f [2 valores]

Aula 2. 9 [2 valor]

Aula 3. 18.a, b, c, d, f, i, j [1,5 valores]

Aula 4. 23 [2 valores]

Aula 5. (–)

Aula 6. 25 [1 valor]

Aula 7. 30.i, iv, x, xii, xxi, xxvi, xxx, xxxiv [1,5 valores]

Aula 8. 31.e [1 valor]

Aula 9. 32.a [1 valor]

Aula 10. 34[1 valor]

Aula 11. 35.b, c, e, i, k [1 valor]

Aula 12. 38 [1,5 valores]

Aula 13. 41 [4,5 valores]

1 Revisões de Python

Exercícios de revisão de Python.

Ex 1. (a) Faça um programa que mostre a mensagem "Olá mundo"na ecrã.

(b) Faça um programa que peça um número e mostre a mensagem "`O número inserido foi o: [número]`".

(c) Faça um programa que peça dois números e imprima a soma.

(d) Faça um programa que converta metros para centímetros.

(e) Faça um programa para calcular a área de um circulo (πr^2) pedido o raio.

(f) Faça um programa que calcule a área e o perímetro de um quadrado, pedido o lado.

(g) Faça um programa que pergunte quanto ganha por hora e o número de horas trabalhadas por mês. Calcule e mostre o total do salário no referido mês.

(h) Faça um programa que peça a temperatura em graus Farenheit, transforme e mostre a temperatura em graus Celsius.

$$C = (5 \times (F - 32)/9)$$

(i) Tendo como dado de entrada a altura (h) de uma pessoa, construa um algoritmo que calcule seu peso ideal, utilizando a fórmula para homens $(72.7 \times h) - 58$ e para mulheres: $(62.1 \times h) - 44.7$.

(j) Faça um programa para uma loja de tintas. O programa deverá pedir o tamanho em metros quadrados da área a ser pintada. Considere que a cobertura da tinta é de 1 litro para cada 3 metros quadrados e que a tinta é vendida em latas de 18 litros, que custam 80 euros. Informe ao utilizador da quantidades de latas de tinta a serem compradas e o preço total.

Ex 2. (a) Faça um programa que peça dois números e imprima o maior deles.

(b) Faça um programa que peça um valor e mostre se o valor é positivo ou negativo.

(c) Faça um programa que verifique se uma letra digitada é "F"ou "M". Conforme a letra escrever: F - Feminino, M - Masculino, Sexo Inválido.

(d) Faça um programa que verifique se uma letra digitada é vogal ou consoante.

(e) Faça um programa para a leitura de duas notas parciais de um aluno. O programa deve calcular a média alcançada por aluno e apresentar:

- A mensagem "Aprovado", se a média alcançada for maior ou igual a 9.5 e menor que 19;
- A mensagem "Reprovado", se a média for menor do que 9.5;
- A mensagem "Aprovado com Distinção", se a média for pelo menos 19.

(f) Faça um programa que leia três números e mostre o maior deles.

Ex 3. (a) Faça um programa que peça uma nota, entre zero e dez (continue a pedir até que o utilizador insira um valor válido).

(b) Faça um programa que leia um nome de utilizador e a sua senha e não aceite a senha igual ao nome do utilizador, mostrando uma mensagem de erro e voltando a pedir as informações se necessário.

(c) Supondo que a população de um país A seja da ordem de 80000 habitantes com uma taxa anual de crescimento de 3% e que a população de B seja 200000 habitantes com uma taxa de crescimento de 1.5%. Faça um programa que calcule e escreva o número de anos necessários para que a população do país A ultrapasse ou iguale a população do país B, mantidas as taxas de crescimento.

(d) Altere o programa anterior permitindo ao utilizador informar as populações e as taxas de crescimento iniciais. Valide as entradas e permita repetir a operação.

(e) Faça um programa que leia n números e informe o maior número.

(f) Faça um programa que leia n números e informe a soma e a média destes.

(g) Faça um programa que imprima na tela apenas os números ímpares entre 1 e 50.

(h) Faça um programa que receba dois números inteiros e gere os números inteiros que estão no intervalo compreendido por eles.

(i) Desenvolva um gerador de tabuada, capaz de gerar a tabuada de qualquer número inteiro entre 1 a 10. O utilizador deve informar qual número de que deseja ver a tabuada.

(j) Faça um programa que peça 10 números inteiros, calcule e mostre a quantidade de números pares e a quantidade de números impares.

(k) A série de Fibonacci é formada pela sequência 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ... Faça um programa capaz de gerar a série até ao n-ésimo termo.

(l) A série de Fibonacci é formada pela sequência 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ... Faça um programa que gere a série até que o valor seja maior que 500.

(m) Faça um programa que calcule o fatorial de um número inteiro fornecido pelo utilizador. Ex.: 5!=5.4.3.2.1=120

Ex 4. (a) Faça um programa que leia um vetor de 5 números inteiros e mostre-os.

(b) Faça um programa que leia um vetor de 10 números reais e mostre-os na ordem inversa.

(c) Faça um programa que leia um vetor de 10 caracteres, e diga quantas consoantes foram lidas. Imprima as consoantes.adicione

(d) Faça um programa que leia 20 números inteiros e armazene-os num vetor. Armazene os números pares no vetor PAR e os números IMPARES no vetor ímpar. Imprima os três vetores.

(e) Faça um programa que peça as quatro notas de 10 alunos, calcule e armazene num vetor a média de cada aluno, imprima o número de alunos com média maior ou igual a 9.5

(f) Faça um programa que leia um vetor de 5 números inteiros, mostre a soma, a multiplicação dos números.

(g) Faça um programa que peça a idade e a altura de 5 pessoas, armazene cada informação no seu respetivo vetor. Imprima a idade e a altura na ordem inversa a ordem lida.

(h) Considere o vetor

```
v = list(range(50))     # v = [0, 1, 2, 3, ... 49]
```

Sem utilizar ciclos[1], imprima:

(a) os primeiros 10 elementos

(b) os últimos 10 elementos

(c) os valores entre a posição 10 e 20 (inclusive)

(d) apague o número na posição 5

(e) apague o número 20

(f) imprima os números por ordem inversa

(g) faça a união com o conjunto $['a','b','c']$

Ex 5. Implemente cada uma das alíneas do exercício 4 como uma função Python de modo a que em vez de imprimir no ecrã os resultados, estes são devolvidos pela função.

Ex 6. (a) Faça uma função que recebe 2 strings e informe se possuem o mesmo comprimento e são iguais ou diferentes no conteúdo.

Compara duas strings

```
String 1: Portugal é campeão da Europa
String 2: Portugal! é campeão da Europa!
Tamanho de "Portugal é campeão da Europa": 28 caracteres
Tamanho de "Portugal! é campeão da Europa!": 30 caracteres
As duas strings são de tamanhos diferentes.
As duas strings possuem conteúdo diferente.
```

(b) Faça uma função para pedir o o seu nome. Faça outra função que devolve o o nome introduzido de trás para frente utilizando somente letras maiúsculas.

(c) Faça uma função que escreve uma string passada como argumento na vertical.

```
S
T
R
I
N
G
```

(d) Modifique o programa anterior de forma a mostrar o nome em formato de escada.

```
S
ST
STR
STRI
STRIN
STRING
```

[1] use "slicing"

(e) Faça um programa que solicite a data de nascimento (dd/mm/aaaa) ao utilizador e imprima a data com o nome do mês por extenso.

```
Data de Nascimento: 29/10/1973
Você nasceu em  29 de Outubro de 1973.
```

(f) Dada uma string, conte:

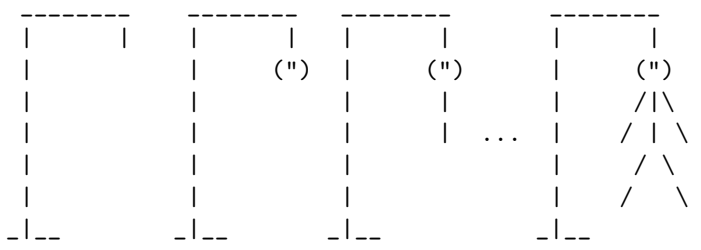
  (a) quantos espaços em branco existem na frase.

  (b) quantas vezes aparecem as vogais a, e, i, o, u.

(g) Um palíndromo é uma sequência de caracteres cuja leitura é idêntica se feita da direita para esquerda ou vice-versa. Por exemplo: OSSO e OVO são palíndromos. Em textos mais complexos os espaços e pontuação são ignorados. A frase SUBI NO ONIBUS é o exemplo de uma frase palíndroma onde os espaços foram ignorados. Faça uma função que recebe uma sequência de caracteres, e devolve se é um palíndromo ou não.

(h) Escreva uma função que recebe um número e devolve o mesmo mas por extenso. Ex.: `extenso(53)` devolve "cinquenta e três"

Ex 7. Desenvolva o jogo da forca. O programa terá uma lista de palavras lidas de um ficheiro de texto e escolherá uma aleatoriamente. O jogador poderá errar 7 vezes antes de ser enforcado.



2 Programação Orientada a Objetos

Ex 8. Identifique as classes e implemente um programa para a seguinte especificação: "O supermercado vende diferentes tipos de produtos. Cada produto tem um nome, preço e uma quantidade em stock. Um pedido de um cliente é composto de itens, onde cada item especifica o produto que o cliente deseja e a respetiva quantidade. Esse pedido pode ser pago em dinheiro, cheque ou cartão. Ao efetuar um pedido o stock é atualizado. Caso o stock não permita o pedido (insuficiência de produto) é emitida uma mensagem de aviso e a operação sobre esse produto não se realiza. Tudo isto é controlado a partir de um programa principal que tem um menu com todas as operações possíveis."

Ex 9. Implemente as classes apresentadas no diagrama da Fig. 1 (de preferência, use o pycharm ou outro IDE avançado). **Use decoradores para implementar o encapsulamento**. Crie um programa que permita de modo interativo listar, inserir, remover, e editar carros de uma lista. Crie ainda uma opção para gravar essa lista num ficheiro (veja o pacote pickle)[2].

[2]Template para as classes: `https://bitbucket.org/pcardoso/complementos-de-programa-o-sti-ise-ualg/src/master/Python/templates_exercicios/car`

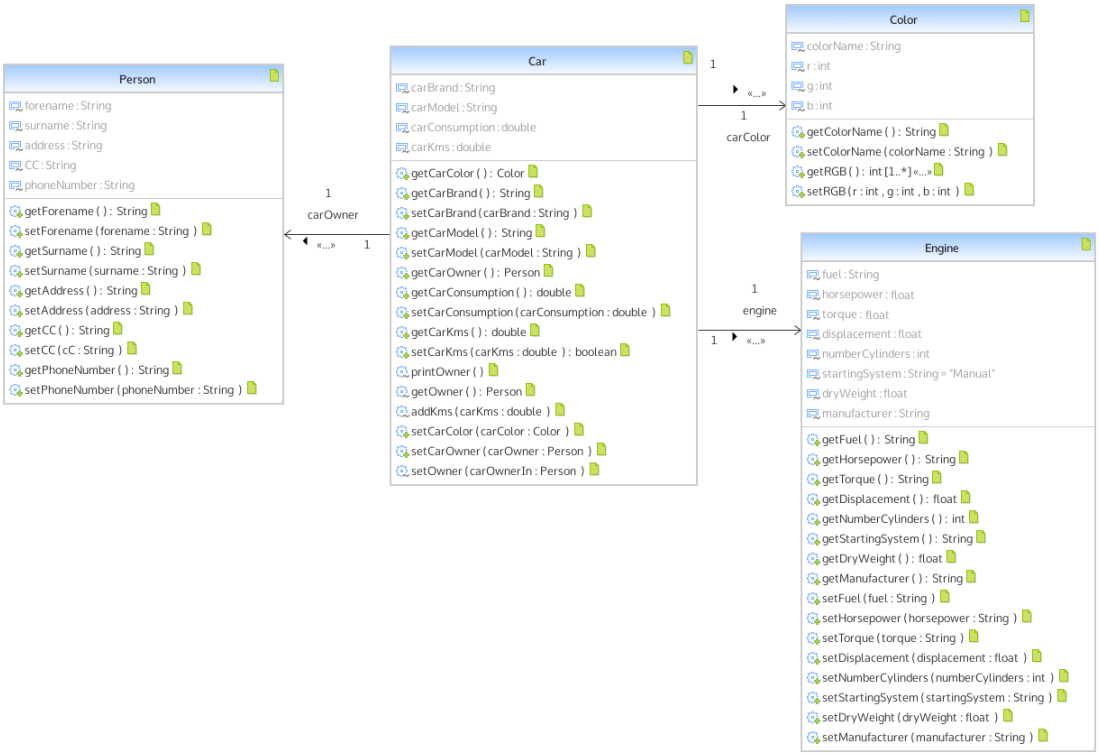


Figura 1: UML carro

Exemplo de menu:

```
11 - lista carros       21 - lista pessoas      ...
12 - novo carro         22 - nova pessoa        ...
13 - apaga carro        23 - apaga pessoa       ...
14 - edita carro        24 - edita pessoa       ...
```

Ex 10. Crie uma classe para guardar o nome e 3 notas de um aluno. Quando pedido, as instâncias da classe informam se ele foi aprovado, se tem de fazer teste de recuperação ou se foi reprovado. A média de aprovação é superior ou igual 10.0; A média para recuperação é entre 8 e 10; A média do reprovado é inferior 8.0.

Ex 11. O objetivo neste exercício é criar um sistema para gerir os funcionários de um banco usando programação orientada a objetos.

(a) Modele (diagrama de classe) um funcionário de modo a gerir (entre outras coisas que possa achar conveniente):

- o nome do funcionário;
- o departamento onde trabalha;
- o salário;
- a data de entrada no banco;

- o número de CC; e
- o número de telefone.

(b) Crie métodos adequados para definir e aceder aos atributos da classe.

(c) Crie o método `recebe_aumento` que aumenta o salário do funcionário de acordo com um parâmetro de entrada.

(d) Crie o método `calcula_ganho_anual` que devolve o valor do salário multiplicado por 12.

(e) Crie o método `mostra`, que imprime todos os atributos do funcionário ("formatados").

Ex 12. Relativamente ao exercício 11

(a) Crie duas variáveis a referenciar dois objetos do tipo funcionário com atributos diferentes e compare-os usando "==". O que aconteceu?

(b) Crie duas variáveis a referenciar dois objetos do tipo funcionário com atributos iguais e compare-os usando "==". O que aconteceu?

(c) Em que situação é que duas variáveis que referenciam objetos do tipo funcionário devolvem verdade ao serem comparadas (com "==")?

Ex 13. Escreva o código para implementar o seguinte:

(a) Implemente a classe `Quadrado`, com o atributo `lado` (encapsulado) e os métodos `calcular_area`, `calcular_perimetro` e `imprimir`. O método `imprimir` deve mostrar os valores do atributos, a área e o perímetro.

(b) Implemente a classe `Retangulo`, com os atributos `comprimento` e `largura` (encapsulados) e os métodos `calcular_area`, `calcular_perimetro` e `imprimir`. O método `imprimir` deve mostrar os valores do atributos, a área e o perímetro.

(c) Implemente a classe `Circulo` com atributos `raio` (encapsulado) e os métodos `calcular_area`, `calcular_perimetro` e `imprimir`. O método `imprimir` deve mostrar os valores do atributos, a área e o perímetro.

(d) Para cada uma das classes anteriores implemente o método `__repr__`.

(e) Para cada uma das classes anteriores implemente um contador de instâncias (atributo "privado"). Para saber o número de instâncias deverá usar uma **método de classe**.

(f) Para cada uma das classes anteriores implemente **métodos estáticos** para a área e para o perímetro que devolvem os valores pedidos, dados os argumentos necessários (p.e., para calcular a área de um circulo faz-se `Circulo.area(raio)`).

(g) Crie um programa com um menu que use as classes anteriores para tratar áreas e perímetros das figuras descritas.

Ex 14. Escreva uma classe `Moto`, com atributos

- `marca`
- `modelo`
- `cor`

- `marcha` – em que a marcha a moto se encontra no momento, sendo representado de forma inteira, onde 0 - neutro, 1 – primeira, 2 – segunda, etc.)
- `menor_marcha` e `maior_marcha` – a menor marcha possível para a moto e o atributo `maior_marcha` indica qual será a maior marcha possível. Por exemplo, `menor_marcha` = -1 poderá significar que a moto tem marcha atrás.
- `ligada` – indicar se a moto está ligada ou não.

e métodos

- `imprimir` – mostra os valores de todos os atributos.
- `marcha_acima` e `marcha_abaixo` – efetuam a troca de marchas, onde o método `marcha_acima` deverá subir uma marcha, ou seja, se a moto estiver em primeira marcha, deverá ser trocada para segunda marcha e assim por diante. O método `marcha_abaixo` deverá realizar o oposto, ou seja, descer a marcha.
- `ligar`
- `desligar`
- *getters* e *setters* para os atributos

Ex 15. Escreva uma classe `Televisor`, com atributos para

- `ligado`
- `canal` – indica o canal atual em que o televisor está sintonizado
- `numero de canais` – numero máximo de canais que o televisor admite sintonizar
- `grelha de canais` – lista com nomes de canais e programação (use uma classe ou mais...)
- `volume`
- `volume máximo` – volume máximo
- `taxa de contraste` – escala 0-100
- `taxa de brilho` – escala 0-100
- `taxa de saturação` – escala 0-100

e métodos

- `imprimir` – mostra os valores de todos os atributos.
- `ligar`
- `desligar`
- `sintonizar` um canal dando a posição na grelha e frequência (inteiro)
- aumentar/diminuir o volume
- aumentar/diminuir o canal

- aumentar/diminuir a taxa de contraste
- aumentar/diminuir a taxa de brilho
- aumentar/diminuir a taxa de saturação
- definir a programação de um canal
- ...
- *getters* e *setters* para os atributos

Crie um programa com um menu adequado para operar o televisor.

Ex 16. Crie uma classe para representar datas.

(a) Represente uma data usando três atributos: o `dia`, o `mês`, e o `ano`.

(b) A classe deve ter um construtor que inicializa os três atributos e verifica a validade dos valores fornecidos. Pode verificar que um ano é bissexto usando a seguinte expressão lógica `eBissexto = (ano % 4 == 0) && ( (ano % 100 != 0) || (ano % 400 == 0))`.

(c) Encapsule os elementos da classe

(d) Re-implemente o método `__repr__` para devolve uma representação da data como string. Considere que a data deve ser formatada mostrando o dia, o mês e o ano separados por barra (/).

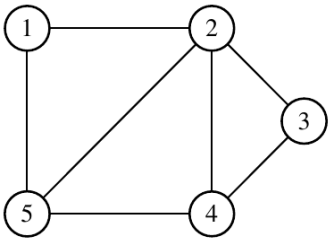
(e) Forneça uma operação para avançar uma data para o dia seguinte.

(f) Escreva um programa de teste que demonstra as capacidades da classe. Garanta que uma instância desta classe esteja sempre num estado consistente.

Ex 17. Um Grafo é uma estrutura de dados muito comum em computação, e os algoritmos sobre grafos são fundamentais para muitas áreas. Um grafo $G = (V, A)$ consiste em:

- um conjunto finito de pontos V . Os elementos de V são chamados de vértices de G.
- um conjunto finito A de pares não ordenados de V, que são chamados de arestas de G. Uma aresta a em A é um par não ordenado (v, w) de vértices v, w em V, que são chamados de extremidades de a.

Uma aresta a em A é chamada de incidente com um vértice v em V , se v for uma extremidade de a. Um vértice v em V diz-se vizinho de outro vértice w em V se existir uma aresta a em A incidente com v e w. Dado um grafo como o ilustrado na figura



este pode ser representado por

listas de adjacência

vértices	lista de adjacência
1	2, 5
2	1,5
3	2,4
4	2, 5, 3
5	4, 1, 2

matriz de adjacência

	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Escreva uma classe para representar grafos. Escolha entre a representação por listas de adjacência ou por matriz de adjacência. A classe deve oferecer uma operação para determinar se dois vértices são vizinhos, e outra operação para determinar a lista de todos os vértices que são vizinhos de um dado vértice. Considere que cada vértice é representado por um número inteiro.

Escreva um aplicativo para testar a classe.

Ex 18. Implementando cada classe num módulo (ficheiro) separado, implemente o seguinte:

(a) Cria uma Classe `Pessoa`, contendo os atributos encapsulados, com seus respetivos seletores (*getters*) e modificadores (*setters*), e ainda o construtor padrão. Atributos: `nome; endereço; telefone;`

(b) Implemente o método de classe (`cria_anonimo`) para a classe Pessoa que gera uma instância com o nome="John Doe", endereco="Unknown", telefone="Unknown".

(c) Considere, como subclasse da classe `Pessoa` (desenvolvida no exercício anterior) a classe `Fornecedor`. Considere que cada instância da classe `Fornecedor` tem, para além dos atributos que caracterizam a classe `Pessoa`, os atributos `valor_credito` (correspondente ao crédito máximo atribuído/admitido pelo fornecedor) e `valor_divida` (montante da dívida para com o fornecedor). Implemente na classe `Fornecedor`, para além dos usuais métodos seletores e modificadores, um método `obter_saldo` que devolve a diferença entre os valores dos atributos `valor_credito` e `valor_divida`. Depois de implementada a classe Fornecedor, crie um programa de teste adequado que lhe permita verificar o funcionamento dos métodos implementados na classe `Fornecedor` e os herdados da classe `Pessoa`.

(d) Considere, como subclasse da classe Pessoa, a classe `Empregado`. Considere que cada instância da classe `Empregado` tem, para além dos atributos que caracterizam a classe `Pessoa`, os atributos `codigo_setor`, `salario_base` (vencimento base) e `imposto` (percentagem retida dos impostos). Implemente a classe `Empregado` com métodos seletores e modificadores e um método `calcular_salario`. Escreva um programa de teste adequado para a classe `Empregado`.

(e) Implemente a classe `Administrador` como subclasse da classe `Empregado`. Um determinado administrador tem como atributos, para além dos atributos da classe Pessoa e da classe Empregado, o atributo `ajuda_de_custo` (ajudas referentes a viagens, estadias, etc.). Note que deverá redefinir na classe Administrador o método herdado calcular_salario (o

salário de um administrador é equivalente ao salário de um empregado usual acrescido das ajuda de custo). Escreva um programa de teste adequado para esta classe.

(f) Implemente a classe `Operario` como subclasse da classe `Empregado`. Um determinado operário tem como atributos, para além dos atributos da classe `Pessoa` e da classe `Empregado`, o atributo `valor_producao` (que corresponde ao valor monetário dos artigos efetivamente produzidos pelo operário) e `comissao` (que corresponde à percentagem do `valor_producao` que será adicionado ao vencimento base do operário). Note que deverá redefinir nesta subclasse o método herdado `calcular_salario` (o salário de um operário é equivalente ao salário de um empregado usual acrescido da referida comissão). Escreva um programa de teste adequado para esta classe.

(g) Implemente a classe `Vendedor` como subclasse da classe `Empregado`. Um determinado vendedor tem como atributos, para além dos atributos da classe `Pessoa` e da classe `Empregado`, o atributo `valor_vendas` (correspondente ao valor monetário dos artigos vendidos) e o atributo `comissao` (percentagem do `valor_vendas` que será adicionado ao vencimento base do Vendedor). Note que deverá redefinir nesta subclasse o método herdado `calcular_salario` (o salário de um vendedor é equivalente ao salário de um empregado usual acrescido da referida comissão). Escreva um programa de teste adequado para esta classe.

(h)

(i) Faça o *override* ao método `__repr__` em cada uma das classes.

(j) Teste a sua implementação.

Ex 19. Implemente as classes para o diagrama da Figura 2.

Ex 20. O BibTex é uma ferramenta de formatação usada em documentos LaTeX, para facilitar a separação da bibliografia com a apresentação do texto. Para os tipos de entradas dados a seguir, e tendo em conta os campos obrigatórios, crie uma estrutura de classes adequada. Pode ainda ter como referência a aplicação JabRef. Crie um *parser* capaz de ler/escrever os dados de/para um ficheiro.

- article
  - Um artigo de um periódico ou revista.
  - Campos obrigatórios: author, title, journal, year
  - Campos opcionais: volume, number, pages, month, note, key
- book
  - Um livro com editora explícita.
  - Campos obrigatórios: author/editor, title, publisher, year
  - Campos opcionais: volume/number, series, address, edition, month, note, key
- booklet
  - Um trabalho que foi impresso e encadernado, mas que não é distribuído sob o nome de alguma editora ou instituição.
  - Campos obrigatórios: title

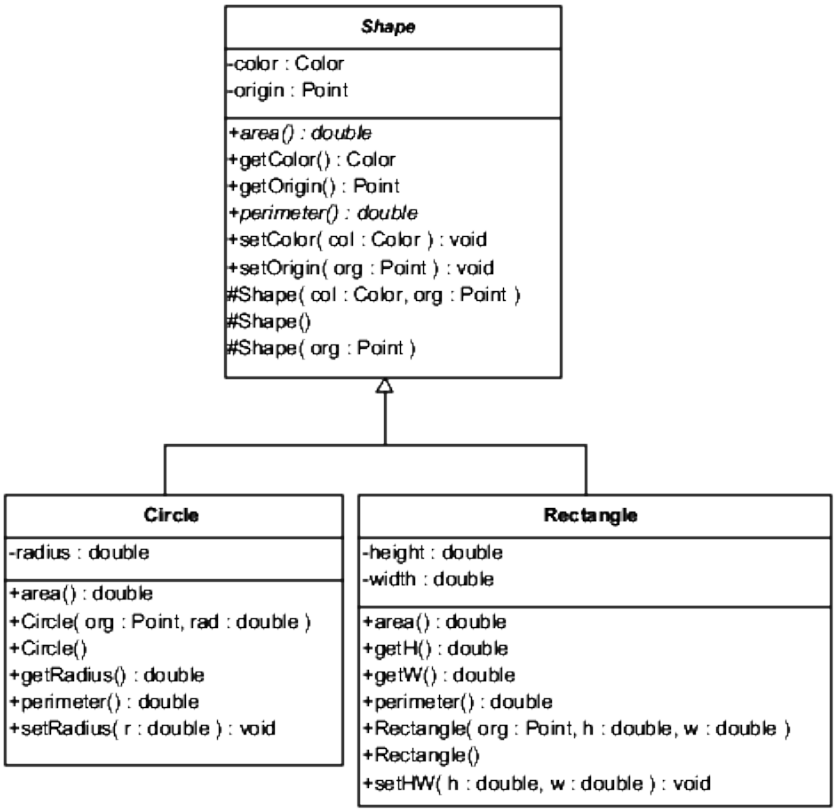


Figura 2: Diagrama de UML

  - Campos opcionais: author, howpublished, address, month, year, note, key

- inbook
  - Uma parte de um livro, geralmente sem título. Pode ser um capítulo (ou seção, etc.) e/ou um intervalo de páginas.
  - Campos obrigatórios: author/editor, title, chapter/pages, publisher, year
  - Campos opcionais: volume/number, series, type, address, edition, month, note, key

- incollection
  - Uma parte de um livro que possui título próprio.
  - Campos obrigatórios: author, title, booktitle, publisher, year
  - Campos opcionais: editor, volume/number, series, type, chapter, pages, address, edition, month, note, key

- inproceedings

  - Um artigo nos anais de uma conferência.
  - Campos obrigatórios: author, title, booktitle, year
  - Campos opcionais: editor, volume/number, series, pages, address, month, organization, publisher, note, key

- manual
  - Documentação técnica.
  - Campos obrigatórios: title
  - Campos opcionais: author, organization, address, edition, month, year, note, key

- mastersthesis
  - Uma dissertação de Mestrado.
  - Campos obrigatórios: author, title, school, year
  - Campos opcionais: type, address, month, note, key

- misc
  - Usado quando nada mais se adequar à referência.
  - Campos obrigatórios: none
  - Campos opcionais: author, title, howpublished, month, year, note, key

- phdthesis
  - Uma tese de doutorado.
  - Campos obrigatórios: author, title, school, year
  - Campos opcionais: type, address, month, note, key

- proceedings
  - Anais de uma conferência.
  - Campos obrigatórios: title, year
  - Campos opcionais: editor, volume/number, series, address, month, publisher, organization, note, key

- techreport
  - Um relatório publicado por uma escola ou outra instituição, geralmente numerado em série.
  - Campos obrigatórios: author, title, institution, year
  - Campos opcionais: type, number, address, month, note, key

- unpublished
  - Um documento que possui um autor e um título, mas que não foi formalmente publicado.
  - Campos obrigatórios: author, title, note
  - Campos opcionais: month, year, key

Um exemplo de um ficheiro de BibTex pode conter o seguinte texto

```
@article{article,
    author  = {Peter Adams},
    title   = {The title of the work},
    journal = {The name of the journal},
    year    = 1993,
    number  = 2,
    pages   = {201--213},
    month   = 7,
    note    = {An optional note},
    volume  = 4
}

@book{book,
    author    = {Peter Babington},
    title     = {The title of the work},
    publisher = {The name of the publisher},
    year      = 1993,
    volume    = 4,
    series    = 10,
    address   = {The address},
    edition   = 3,
    month     = 7,
    note      = {An optional note},
    isbn      = {3257227892}
}

@booklet{booklet,
    title        = {The title of the work},
    author       = {Peter Caxton},
    howpublished = {How it was published},
    address      = {The address of the publisher},
    month        = 7,
    year         = 1993,
    note         = {An optional note}
}

@conference{conference,
    author       = {Peter Draper},
    title        = {The title of the work},
    booktitle    = {The title of the book},
    year         = 1993,
    editor       = {The editor},
    volume       = 4,
    series       = 5,
    pages        = 213,
    address      = {The address of the publisher},
    month        = 7,
    organization = {The organization},
    publisher    = {The publisher},
    note         = {An optional note}
}

@inbook{inbook,
    author       = {Peter Eston},
    title        = {The title of the work},
    chapter      = 8,
    pages        = {201--213},
    publisher    = {The name of the publisher},
    year         = 1993,
    volume       = 4,
    series       = 5,
    address      = {The address of the publisher},
    edition      = 3,
    month        = 7,
    note         = {An optional note}
}

@incollection{incollection,
    author       = {Peter Farindon},
    title        = {The title of the work},
    booktitle    = {The title of the book},
    publisher    = {The name of the publisher},
```

```
    year         = 1993,
    editor       = {The editor},
    volume       = 4,
    series       = 5,
    chapter      = 8,
    pages        = {201--213},
    address      = {The address of the publisher},
    edition      = 3,
    month        = 7,
    note         = {An optional note}
}

@manual{manual,
    title        = {The title of the work},
    author       = {Peter Gainsford},
    organization = {The organization},
    address      = {The address of the publisher},
    edition      = 3,
    month        = 7,
    year         = 1993,
    note         = {An optional note}
}

@mastersthesis{mastersthesis,
    author       = {Peter Harwood},
    title        = {The title of the work},
    school       = {The school where the thesis was written},
    year         = 1993,
    address      = {The address of the publisher},
    month        = 7,
    note         = {An optional note}
}

@misc{misc,
    author       = {Peter Isley},
    title        = {The title of the work},
    howpublished = {How it was published},
    month        = 7,
    year         = 1993,
    note         = {An optional note}
}

@phdthesis{phdthesis,
    author       = {Peter Joslin},
    title        = {The title of the work},
    school       = {The school where the thesis was written},
    year         = 1993,
    address      = {The address of the publisher},
    month        = 7,
    note         = {An optional note}
}

@proceedings{proceedings,
    title        = {The title of the work},
    year         = 1993,
    editor       = {Peter Kidwelly},
    volume       = 4,
    series       = 5,
    address      = {The address of the publisher},
    month        = 7,
    organization = {The organization},
    publisher    = {The name of the publisher},
    note         = {An optional note}
}

@techreport{techreport,
    author       = {Peter Lambert},
    title        = {The title of the work},
    institution  = {The institution that published},
    year         = 1993,
    number       = 2,
    address      = {The address of the publisher},
    month        = 7,
```

```
    note         = {An optional note}
}

@unpublished{unpublished,
    author       = {Peter Marcheford},
    title        = {The title of the work},
    note         = {An optional note},
    month        = 7,
    year         = 1993
}
```

Ex 21. Crie uma classe o jogo da forca (Ex. 7). A classe instancia-se recebendo um número de jogadores. Depois vai perguntando aos jogadores quais as letras até que acerte na palavra ou falhe uma letra. No final mostra uma mensagem de vitória.

Ex 22. (a) Crie uma base de dados que permita armazenar os dados das classes apresentadas no diagrama UML da Figura 2.

(b) Altere as classes do exercício 19 de modo a poder realizar operações CRUD sobre os dados presentes na base de dados.

(c) Crie um menu que permita:

i listar os objetos por categoria (*Circle* e *Rectangle*).

ii editar um objeto, i.e., depois de listados os objetos de uma dada categoria, o utilizador escolhe um "id"(correspondente ao que está na base de dados) e pode editar os dados do objeto, (p.e., para um *Circle* pode alterar o raio ou o centro).

iii use a a biblioteca graphics.py[3] para representar graficamente os seus objetos[4].

Ex 23. Tendo como base o exemplo das classes abstrata (`Jogo`) e concreta (`Galo`), apresentadas no código abaixo, implemente o jogo dos 4 em linha. Neste contexto, implemente (apenas) os métodos abstratos (i.e., `inicializa_tabuleiro`, `joga_humano`, `terminou`, `ha_jogadas_possiveis` e `mostra_tabuleiro`) não devendo alterar o código dos outros métodos. Pode descarregar o exemplo daqui bitbucket.org:exemplo jogo do galo

```python
from abc import ABC, abstractmethod
import random


class Jogo(ABC):
    """ implementa uma classe para um jogo com 2 humanos"""

    def __init__(self):
        print('bom jogo...')
        self.inicializa_tabuleiro()

    @abstractmethod
    def joga_humano(self, jogador):
        """ metodo que solicita ao humano :jogador: a proxima jogada e coloca-a no tabuleiro
        :param jogador: numero do jogador (0 ou 1)
        """
        pass

    @abstractmethod
    def terminou(self):
        """ devolve `True` se foi verificada a condicao de paragem, i.e., um jogador ganhou.
        devolve `False` caso contrario """
        pass
```

[3] https://mcsp.wartburg.edu/zelle/python/graphics.py
[4] https://mcsp.wartburg.edu/zelle/python/

```
    @abstractmethod
    def mostra_tabuleiro(self):
        """desenha o tabuleiro"""
        pass

    @abstractmethod
    def inicializa_tabuleiro(self):
        """ inicializa o tabuleiro de jogo"""
        pass

    @abstractmethod
    def ha_jogadas_possiveis(self):
        """ verifica se ainda ha jogadas possiveis ou se o jogo esta empatado"""
        pass

    def jogar(self):
        """ corre o jogo..."""
        jogador = random.randint(0, 1)
        while True:
            self.mostra_tabuleiro()
            self.joga_humano(jogador)
            if self.terminou():
                self.mostra_tabuleiro()
                print(f'o jogador {jogador} ganhou!!')
                return
            elif not self.ha_jogadas_possiveis():
                print(f'Empataram!!!')
                return
            jogador = (jogador+1) % 2


class Galo(Jogo):

    def inicializa_tabuleiro(self):
        self.numero_de_jogadas_realizadas = 0  # conta as jogadas, serve para saber se ainda
            ha jogadas validas
        self.tabuleiro = {(l, c): ' ' for l in range(3) for c in range(3)}  # o tabuleiro e
            um dicionario!

    def _le_linha_coluna_valida(self, s):
        """ metodo auxiliar para ler uma posicao que seja 0, 1 ou 2"""
        while True:
            x = input(s)
            if x in ['0', '1', '2']:
                return int(x)

    def joga_humano(self, jogador):
        print(f'jogador {jogador} insira a sua jogada')
        while True:
            linha = self._le_linha_coluna_valida('Linha?')
            coluna = self._le_linha_coluna_valida('Coluna?')
            if self.tabuleiro[(linha, coluna)] == ' ':  # verifica se a posicao nao esta
                preenchida, i.e., e valida
                self.tabuleiro[(linha, coluna)] = ['X', 'O'][jogador]
                self.numero_de_jogadas_realizadas += 1
                return
            else:
                print('Jogada invalida. Tente de novo')

    def terminou(self):
        lista_posicaoes_ganhadoras = (
            ((0, 0), (0, 1), (0, 2)),  # linha 0
            ((1, 0), (1, 1), (1, 2)),  # linha 1
            ((2, 0), (2, 1), (2, 2)),  # linha 2
            ((0, 0), (1, 0), (2, 0)),  # coluna 0
            ((0, 1), (1, 1), (2, 1)),  # coluna 1
            ((0, 2), (1, 2), (2, 2)),  # coluna 2
            ((0, 0), (1, 1), (2, 2)),  # diagonal
            ((0, 2), (1, 1), (2, 0)),  # anti-diagonal
        )

        for teste in lista_posicaoes_ganhadoras:
```

```
            if self.tabuleiro[teste[0]] == self.tabuleiro[teste[1]] == self.tabuleiro[teste
                [2]] \
                    and self.tabuleiro[teste[0]] != ' ':
                return True  # encontrou posicao ganhadora
        return False

    def mostra_tabuleiro(self):
        print(13 * '-')
        for l in range(3):
            for c in range(3):
                print(f'| {self.tabuleiro[(l, c)]} ', end='')
            print('|\n' + 13*'-')

    def ha_jogadas_possiveis(self):
        return self.numero_de_jogadas_realizadas < 9


galo = Galo()
galo.jogar()
```

Ex 24. Dado o diagrama UML da Figura 3, construir um programa capaz de simular o funcionamento de folha de pagamento com quatro classes de trabalhadores: `Empregado`, `PorHora`, `PorComissao` e `PorHoraeComissao`. A classe `Empregado` deve ser abstrata, pois o método `getPay()`, que devolve o quanto cada tipo de empregado deve ganhar, só poderá ser definido nas subclasses. Para todas as classes cujo ganho dos trabalhadores está relacionado com a comissão relativa ao montante de vendas (`PorComissao` e `PorHoraeComissao`), deve-se empregar o método `setVendas` e a informação contida no campo `COMMISSION_RATE`. Por último, a classe `FolhadePagamento` emprega objetos de todas as classes.

3 Exceções

Ex 25. Programe a seguinte situação garantindo que todas as possíveis exceções são tratadas adequadamente. Por exemplo, se for pedido um nº e o utilizador falhar essa introdução o programa deve continuar a trabalhar adequadamente. Assim, faça um programa que:

(a) Pede ao utilizador um número e devolve a sua raiz quadrada. (Exemplo de situações a tratar: não foi introduzido um número real; o número real é negativo; ...)

(b) Solicite ao utilizador 2 números inteiros. A seguir, calcule e mostre a divisão do primeiro pelo segundo.

(c) Solicita ao utilizador o nome completo e guarda somente o 4º nome numa lista chamada quarto_nome.

(d) Pede ao utilizador um número e devolve o carater que na frase ''`complementos de programacao`'' está nessa posição.

Ex 26. Escreva o código de uma classe Data que permite guardar datas. As datas devem ser validadas e caso não o sejam uma exceção dever ser levantada (os dados de entrada devem ser inteiros, os dias são validos no mês/ano introduzido etc.) [5]

[5] um ano é bissexto se `((year%400 == 0) or ((year%4 == 0) and (year%100 != 0)))`. Ex:

- 2000 foi bissexto: `(2000 % 400 == 0) or ((2000 % 4 == 0) and (2000 % 100 != 0))` avalia a verdade.
- 1900 não foi bissexto: `(1900 % 400 == 0) or ((1900 % 4 == 0) and (1900 % 100 != 0))` avalia a falso.

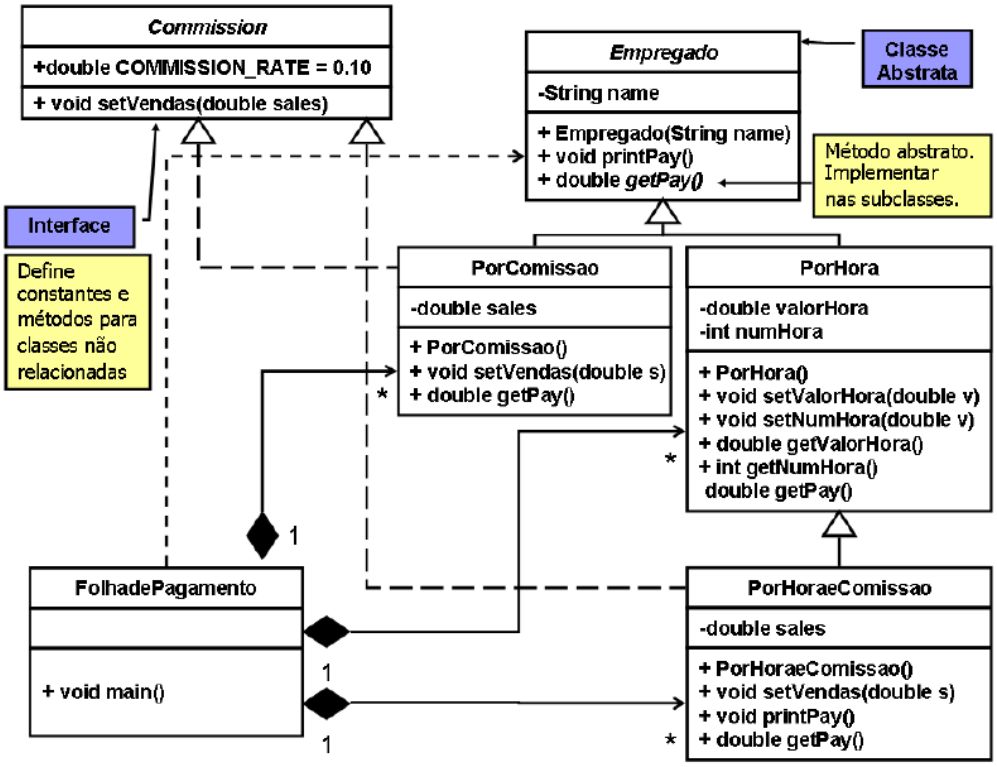


Figura 3:

```
class Data:
    """ classe que implementa o armazenamento de uma data.
    Se for instanciada com valores invalidos levanta uma excecao"""


    def __init__(self, a, m, d):
        Data.valida(a, m, d) # levanta excecao em caso de erro
        self.a, self.m, self.d = a, m, d

    def __repr__(self):
        return f'{self.a}/{self.m}/{self.d}'

    @staticmethod
    def valida(a, m, d):
        ''' devolve True se a data e a/m/d valida. Caso contrario levante uma excecao
            adequada
        :raises:
            TypeError: se os valores a, m ou d nao forem inteiros, indicando qual
            AnoInvalido: Nao existe ano 0 (xxx excecao a ser implementada pelo aluno
                xxxx)
            MesInvalido: se m < 1 ou m > 12
            DiaInvalido: o mes m nao tem dia d
        '''
        raise Exception("To-do")
```

4 Bases de Dados

Ex 27. Construa um DER para a seguinte situação: Uma clínica médica pretende informatizar os seus serviços. Para já querem começar por informatizar os dados referentes a médicos, a clientes (ou pacientes) e a consultas.

- Na clínica trabalham vários médicos, de várias especialidades diferentes. Cada médico tem uma única especialidade.
- De cada médico, identificado internamente por um número de empregado, a clínica pretende armazenar o nome, especialidade, morada e telefone.
- Como é usual, os médicos dão consultas a clientes.
- A clínica pretende ter sempre disponível a informação dos nomes e moradas dos seus clientes.
- A cada consulta está associado um só médico e um só cliente.
- Actualmente as consultas são numeradas para cada um dos médicos, ou seja para cada médico há uma consulta 1, uma consulta 2, etc. Para a clínica é importante manter este sistema de identificação das consultas, por forma a poderem ser introduzidos dados relativos ao tempo em que a clínica ainda não estava informatizada.
- Da consulta pretende armazenar-se a data em que ocorreu, bem como os vários fármacos que foram receitados pelo médico na consulta.

Ex 28. Construa um DER para a seguinte situação: Considere a seguinte informação sobre a base de dados de uma universidade:

- Os professores têm um número de contribuinte, um nome, uma idade, um posto, e uma especialidade de investigação.
- Existem projectos que têm um número, um organismo financiador, uma data de início, uma data de final, e um orçamento.
- Os estudantes de pós-graduação têm um número de contribuinte, um nome, uma idade, e um plano de curso (ex. mestrado, doutoramento).
- Cada projecto é gerido por um professor (o investigador principal do projecto).
- Cada projecto tem a participação de um ou mais professores.
- Os professores podem gerir e/ou trabalhar em vários projectos.
- Cada projecto tem um ou mais estudantes de pós-graduação (conhecidos como os assistentes de investigação).
- Sempre que um estudante de pós-graduação trabalha num projecto, terá que existir um professor a supervisionar esse trabalho. Os estudantes podem trabalhar em vários projectos com supervisores eventualmente diferentes.
- Os departamentos têm um número, um nome, e um escritório principal.
- Os departamentos são liderados por um professor.

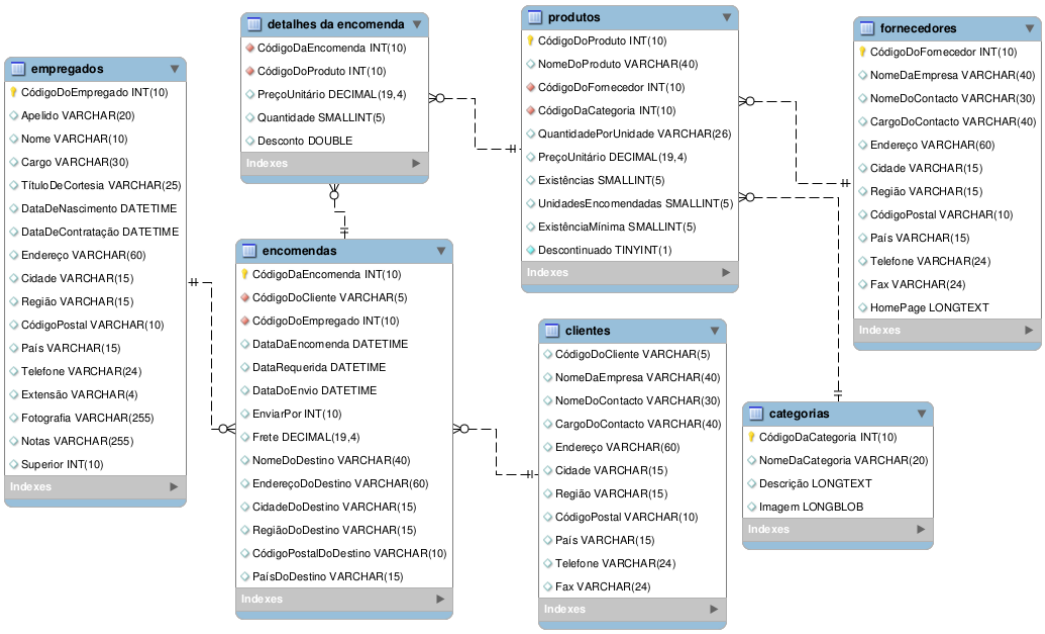


Figura 4: DER Adamastor

- Os professores podem trabalhar num ou mais departamentos. Associada a cada uma destas funções está uma percentagem do seu tempo.
- Os estudantes de pós-graduação estão associados a um departamento no qual fazem o seu curso.
- Cada estudante de pós-graduação tem um outro estudante mais velho que é o seu aconselhador.

Ex 29. Para os exercícios 27 e 28 construa no MySQL Workbench o modelo relacional normalizado até pelo menos à 3ª forma normal.

Ex 30. Comece por restaurar a base de dados Adamastor num servidor MySQL. Observe a Figura 4 que apresenta o diagrama de entidade relacionamento da referida base de dados. Usando uma conexão Python crie uma classe com um método que responde/faz o necessário para responder às seguintes questões/pedidos (veja exemplo na Figura 5 – ver bitbucket para ter código fonte):

(i) apresente uma listagem de produtos ordenados pelo preço unitário.

(ii) apresente uma listagem de produtos com stock inferior a 10 unidades.

(iii) apresente uma listagem de produtos ordenados pelo preço e "sub-ordenados" pelo nome caso tenham igual preço.

(iv) recebe como argumento o id de uma categoria a apresenta um listagem de produtos dessa categoria.

```
import mysql.connector
from pprint import pprint

config = {
    'host' : 'localhost',
    'user' : 'adam',
    'password' : 'adam',
    'db' : 'adamastor' # notem que no phpmyadmin dei permissoes ao utilizador "adam"
        para ter acesso a base de dados adamastor
}


class Adamastor:
    def __init__(self, config):
        self.cnx = mysql.connector.connect(**config)
        self.cursor = self.cnx.cursor(dictionary=True)

    def listar_clientes(self):
        sql = 'SELECT * FROM clientes'
        self.cursor.execute(sql)
        return self.cursor.fetchall()

    def listar_clientes_da_cidade(self, cidade):
        sql = 'SELECT * FROM clientes where cidade = %s'
        self.cursor.execute(sql, (cidade, ))
        return self.cursor.fetchall()


adam = Adamastor(config)

print(" listar_clientes ".center(80, "."))
pprint(adam.listar_clientes())

print(" listar_clientes_da_cidade ".center(80, "."))
pprint(adam.listar_clientes_da_cidade("Lisboa"))
```

Figura 5: Exemplo de implementação para a classe Adamastor.

(v) apresente uma listagem de produtos com menos de 10 unidades em stock e sem encomendas.

(vi) apresente uma listagem de produtos com stock inferior à existência mínima.

(vii) apresente uma listagem de produtos com stock inferior à existência mínima e sem encomendas.

(viii) apresente uma listagem de produtos com stock inferior à existência mínima e cujas encomendas não cobrem o stock mínimo.

(ix) apresente uma listagem de produtos com os respetivos fornecedores.

(x) apresente uma listagem das encomendas dos dias 10-7-1996 a 17-7-1996 ordenadas por id do cliente.

(xi) apresente uma listagem das encomendas cujo país de destino era França.

(xii) apresente uma listagem das datas das encomendas efetuadas por um cliente passado como argumento do método.

(xiii) apresente uma listagem do contato do fornecedor de "Outback Lager".

(xiv) apresente uma listagem dos empregados que trataram de encomendas feitas por "Hanari Carnes".

(xv) apresente uma listagem dos produtos da encomenda feita por "Hanari Carnes" no dia 08-07-1996.

(xvi) apresente uma listagem dos fornecedores dos produtos da encomenda feita por "Hanari Carnes" no dia 08-07-1996.

(xvii) quais os produtos com stock inferior à existência mínima e cujas encomendas não cobrem o stock mínimo?

(xviii) quantos fornecedores distintos existem?

(xix) Altere o número de unidades encomendadas para produto com "id" igual a 1 para 30 unidade.

(xx) Alterar o número de unidades em stock para as existentes mais as encomendadas nos produtos do fornecedor com id 1 e colocar as encomendadas feitas a este a 0.

(xxi) Inserir um novo produto cujos dados são passados como argumentos do método.

(xxii) Altere o novo registo atribuindo valores aos restantes campos.

(xxiii) Inserir um novo produto chamado "Alfarroba"e simultâneamente atribua valores aos restantes campos.

(xxiv) Inserir um novo funcionário (invente os dados necessários).

(xxv) O que é que seria necessário para criar uma nova encomenda?

(xxvi) Qual o preço mínimo entre todos os produto?

(xxvii) Calcule o preço médio dos produtos?

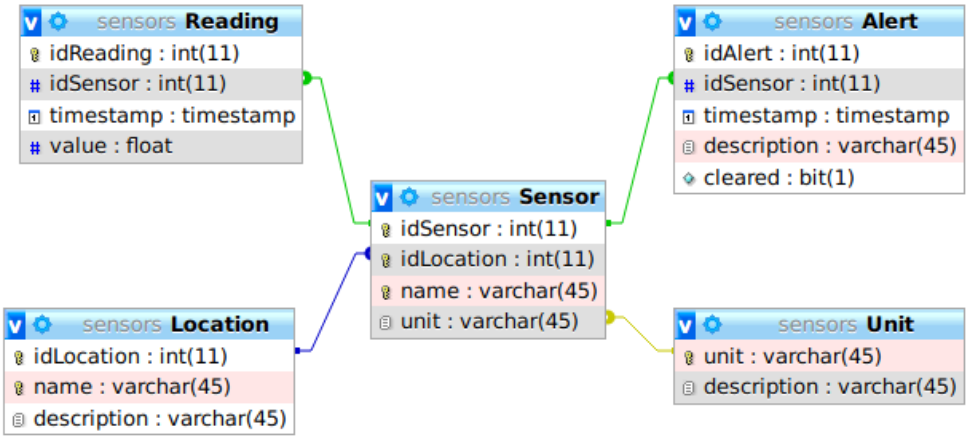


Figura 6: Base de dados Sensores

(xxviii) Liste o produtos acima do preço médio.

(xxix) Qual o numero de telefone do contacto de fornecedores dado pelo utilizador?

(xxx) Quais os fornecedores de uma dada cidade passada como argumento do método?

(xxxi) Quais as cidades que têm fornecedores (devolver o resultado todos em maiúsculas)?

(xxxii) Quais os nomes dos contactos dos fornecedores que têm no nome as letras C e D?

(xxxiii) Quais os empregados com bacharelato?

(xxxiv) Qual a maior encomenda em termos de valor?

(xxxv) Qual a maior encomenda em termos de número de itens?

5 Aplicações web

Ex 31. Considere a base de dados Sensores esquematizada na Figura 6. Implemente uma API/*web service RESTful* com os métodos GET, POST, PUT e DELETE, para as operações CRUD, sobre a tabela

(a) `Alert`.

(b) `Unit`.

(c) `Location`.

(d) `Sensor`.

(e) `Reading`.

Crie scripts para testar aplicação.

Ex 32. Considere a base de dados Sensores esquematizada na Figura 6. Implemente uma pequena aplicação para mostrar os dados presentes na tabela

(a) `Alert`.

(b) `Unit`.

(c) `Location`.

(d) `Sensor`.

(e) `Reading`.

6 Testes unitários

Ex 33. Implemente o metodo `sauda` da Figura 7 de modo a satisfazer os `doctest` exibido e o `unittest` apresentado na Figura 8.

Ex 34. Implemente a classe `Conta` da Figura 9 de modo a satisfazer o `unittest` apresentado na Figura 10. [Código no bitbucket: conta.py e test_conta.py]

7 Aplicações com GUI

Ex 35. Criar um interface com as seguintes características:

(a) **Interface** Uma caixa de texto.

Comportamento Quando o texto é alterado o texto é copiado para a consola.

(b) **Interface** Duas caixas de texto.

Comportamento O texto da 1ª caixa aparece escrito de forma inversa na 2ª caixa.

(c) **Interface** Duas caixas de texto e um label.

Comportamento Quando os textos 1ª e 2ª caixa são inversos então o *label* muda para "textos inversos" com cor verde, caso contrário o *label* diz "textos não inversos" com cor vermelha.

(d) **Interface** Uma caixa de texto e um botão.

Comportamento Quando o botão é clicado o texto da caixa de texto é copiado para a consola e apagado da caixa de texto.

(e) **Interface** Duas caixas de texto (esquerda/direita) e um botão.

Comportamento Quando o botão é clicado o texto nas caixas de texto é trocado, i.e., o texto da caixa da direita passa para a da esquerda e vice-versa.

(f) **Interface** Uma caixa de texto e 2 botões (esquerdo/direito).

Comportamento Quando clicado o botão esquerdo é copiado o texto da caixa de texto desde o início até ao cursor; Quando clicado o botão direito é copiado o texto da caixa de texto desde o cursor até ao fim (Sugestão ver as propriedades selection_to/ selection_from).

(g) **Interface** Um label ("Contador"), uma caixa de texto e um botão.

Comportamento O valor na caixa de texto começa a zero e cada vez que o botão é clicado o valor na caixa de texto é incrementado.

```
def sauda(nomes=None):
    """ Funcao que recebe um nome e devolve uma string com uma saudacao
    :param nomes: o nome
    :ensures: a saudacao

    Requisito 1
    Escreva um metodo sauda(name) que interpola o nome em uma saudacao simples. Por
        exemplo, quando o nome e 'Bob', o metodo deve devolver uma string 'Hello, Bob
        .'.
    >>> sauda('Bob')
    'Hello, Bob.'

    Requisito 2
    Trate os nulos introduzindo um substituto. Por exemplo, quando o nome e nulo, o
        metodo deve devolver a string 'Hello, my friend.'
    >>> sauda()
    'Hello, my friend.'

    Requisito 3
    Lidar com gritos. Quando o nome esta em maiusculas, o metodo deve gritar de volta
        para o utilizador. Por exemplo, quando o nome e 'JERRY', o metodo deve
        devolver a string 'HELLO, JERRY!'
    >>> sauda('JERRY')
    'HELLO, JERRY!'

    Requisito 4
    Manipule dois nomes de entrada. Quando nome e um array de dois nomes, entao ambos
        os nomes devem ser impressos. Por exemplo, quando o nome e ['Jill', 'Jane'], o
         metodo deve devolver a string 'Hello, Jill e Jane'.
    >>> sauda(['Jill', 'Jane'])
    'Hello, Jill and Jane.'

    Requisito 5
    Manipule um numero arbitrario de nomes como entrada. Quando o nome representa mais
         de dois nomes, separe-os com virgulas e feche com uma virgula Oxford e 'and'.
         Por exemplo, quando o nome e ['Amy', 'Brian', 'Charlotte'], o metodo deve
        devolver a string 'Hello, Amy, Brian, and Charlotte.'
    >>> sauda(['Amy', 'Brian', 'Charlotte'])
    'Hello, Amy, Brian, and Charlotte.'

    Requisito 6
    Permitir a mistura de nomes normais e gritados, separando a resposta em duas
        saudacoes. Por exemplo, quando o nome e ['Amy', 'BRIAN', 'Charlotte'], o
        metodo deve devolver a string 'Hello, Amy and Charlotte. AND HELLO BRIAN!'
    >>> sauda(['Amy', 'BRIAN', 'Charlotte'])
    'Hello, Amy and Charlotte. AND HELLO, BRIAN!'

    Requisito 7
    Se alguma entrada em nome for uma string contendo uma virgula, divida-a como sua
        propria entrada. Por exemplo, quando o nome e ['Bob', 'Charlie, Dianne'], o
        metodo deve devolver a string 'Hello, Bob, Charlie, and Dianne.'.
    >>> sauda(['Bob', 'Charlie, Dianne'])
    'Hello, Bob, Charlie, and Dianne.'

    Requisito 8
    Permita que a entrada escape virgulas intencionais introduzidas pelo Requisito 7.
    Elas podem ter escape da mesma maneira que o CSV, com aspas duplas ao redor da
        entrada.
    Por exemplo, quando o nome e ['Bob', '"Charlie, Dianne"'], o metodo deve devolver
        a
    string 'Hello, Bob and Charlie, Dianne.'.
    >>> sauda(['Bob', '"Charlie, Dianne"'])
    'Hello, Bob and Charlie, Dianne.'
    """

    pass
```

Figura 7: sauda.py

```
import unittest
from sauda import sauda


class TestaSaudacao(unittest.TestCase):
    def testa_1_nome(self):
        assert sauda('Bob') == 'Hello, Bob.'

    def testa_0_nomes(self):
        assert sauda() == 'Hello, my friend.'

    def testa_1_gritado(self):
        assert sauda('JERRY') == 'HELLO, JERRY!'

    def testa_2_nomes(self):
        assert sauda(['Jill', 'Jane']) == 'Hello, Jill and Jane.'

    def testa_3_nomes(self):
        assert sauda(['Amy', 'Brian', 'Charlotte']) == 'Hello, Amy, Brian, and
            Charlotte.'

    def testa_com_normais_e_gritos(self):
        assert sauda(['Amy', 'BRIAN', 'Charlotte']) == 'Hello, Amy and Charlotte. AND
            HELLO, BRIAN!'

    def testa_com_necessidade_de_expansao(self):
        assert sauda(['Bob', 'Charlie, Dianne']) == 'Hello, Bob, Charlie, and Dianne.'

    def testa_com_nomes_com_aspas(self):
        assert sauda(['Bob', '"Charlie, Dianne"']) == 'Hello, Bob and Charlie, Dianne.
            '


if __name__ == '__main__':
    unittest.main()
```

Figura 8: test_sauda.py

```
class Conta:

    def __init__(self, dono, taxa_de_juro=0, saldo=0):
        pass

    @property
    def dono(self):
        """ devolve o dono"""
        pass

    @dono.setter
    def dono(self, value):
        """ Guarda uma string formatada em "title" (e.g., 'luigi vercotti' -> 'Luigi
            Vercotti)"""
        pass

    @property
    def taxa_de_juro(self):
        """ devolve a taxa de juro """
        pass

    @taxa_de_juro.setter
    def taxa_de_juro(self, value):
        """ Guarda a taxa de juro. Deve ser float ou int em percentagem (0-100%).
        A taxa_de_juro e nao negativa, sendo que se for fornecido um valor negativo a
            taxa_de_juro e colocada a 0.
        """
        pass

    @property
    def saldo(self):
        """ Devolve o saldo"""
        pass

    @saldo.setter
    def saldo(self, value):
        """Guarda ao saldo. Deve ser float ou int. O Saldo e nao negativa, sendo que
            se for fornecido um valor negativo o saldo e colocada a 0.
        """
        pass

    def capitaliza(self):
        """ Acresencenta os juros ao saldo.
        E.g., se saldo = 1000 e taxa_juro = 2 entao saldo passa a 1020
        """
        pass

    def cobra_comissao(self, comissao):
        """ o valor da comissao e retirado ao saldo.
        Se o saldo for maior do que a comissao entao cobra tudo, senao cobra o
            equivalente ao existente em saldo. E.g.:
        saldo = 10 e comissao = 5 -> saldo = 5 e cobrado = 5
        saldo = 10 e comissao = 15 -> saldo = 0 e cobrado = 10

        :ensures: valor descontado ao saldo
        """
        pass

    def faz_levantamento(self, valor):
        """Subtrai ao saldo o valor desde que o saldo se mantenha positivo.
        :ensures: True se o levantamento foi possivel, False caso contrario
        """
        pass

    def faz_deposito(self, valor):
        """ Acrescenta ao saldo o valor """
        pass
```

Figura 9: conta.py

```python
import unittest
from conta import Conta

class TestConta(unittest.TestCase):

    def setUp(self):
        self.conta = Conta("mr. teabag", 2, 1000)

    def test_dono(self):
        assert self.conta.dono == "Mr. Teabag"

    def test_taxa_de_juro(self):
        self.assertIsInstance(self.conta.taxa_de_juro, (int, float))
        self.assertGreaterEqual(self.conta.taxa_de_juro, 0)


    def test_saldo(self):
        self.assertIsInstance(self.conta.saldo, (int, float))
        self.assertGreaterEqual(self.conta.saldo, 0)
        self.conta.saldo = -10
        self.assertGreaterEqual(self.conta.saldo, 0)

    def test_capitaliza(self):
        self.conta.capitaliza()
        self.assertAlmostEqual(self.conta.saldo, 1020)

    def test_cobra_comissao(self):
        # tem valor que chegue
        self.conta.saldo = 1000
        valor = 10
        cobrado = self.conta.cobra_comissao(valor)
        self.assertAlmostEqual(self.conta.saldo, 990)
        self.assertEqual(cobrado, 10)

        # nao tem valor que chegue
        self.conta.saldo = 100
        valor = 200
        cobrado = self.conta.cobra_comissao(valor)
        self.assertAlmostEqual(self.conta.saldo, 0)
        self.assertEqual(100, cobrado)

    def test_faz_levantamento(self):
        self.conta.saldo = 1000
        valor = 100
        resp = self.conta.faz_levantamento(valor)
        self.assertTrue(resp)
        self.assertAlmostEqual(self.conta.saldo, 900)

        self.conta.saldo = 100
        valor = 200
        resp = self.conta.faz_levantamento(valor)
        self.assertFalse(resp)
        self.assertAlmostEqual(self.conta.saldo, 100)

    def test_faz_deposito(self):
        self.conta.saldo = 1000
        valor = 100
        self.conta.faz_deposito(valor)
        self.assertAlmostEqual(self.conta.saldo, 1100)

if __name__ == '__main__':
    unittest.main()
```

Figura 10: test_conta.py

(h) **Interface** Um label ("n") e uma caixa de texto, mais outro label ("n!") e outra caixa de texto, e um botão.

Comportamento Quando o botão é clicado na caixa de texto associada ao "n!" apresenta o valor de n fatorial (e.g., 5! = 5 * 4 * 3 * 2 * 1 = 120).

(i) **Interface** Um label ("Contador"), uma caixa de texto, uma *checkbox* e um botão.

Comportamento O valor na caixa de texto começa a zero. Cada vez que o botão é clicado o valor na caixa de texto é incrementado uma unidade se a *checkbox* estiver ativada, sendo decrementada uma unidade caso contrário.

(j) **Interface** Um label ("Celsius") e caixa de texto, outro label ("Fahrenheit") e outra caixa de texto.

Comportamento Quando o valor de uma caixa de texto é alterado na outra aparece o valor correspondente na outra unidade.

(k) **Interface** Duas caixas de texto, um *label* (ou mais se preferir) e um botão.

Comportamento Escreva um jogo de adivinhação de números. O programa deve gerar um número aleatório entre 1 e 100, mostrado numa caixa de texto mascarado (como uma password). Na outra caixa de texto o utilizador tenta acertar no número seguindo as pistas que são mostradas no label ("tenta um valor maior"/"tenta um valor menor"). Quando acertar aparece "Acertaste" no *label* e o número deixa de estar mascarado.

Ex 36. Considere a API que "https://exchangeratesapi.io/" que lhe dá o cambio entre Euros e outras moedas. I.e., um

```
GET https://api.exchangeratesapi.io/latest HTTP/1.1
```

devolve

```
{
"base": "EUR",
"date": "2018-04-08",
"rates": {
    "CAD": 1.565,
    "CHF": 1.1798,
    "GBP": 0.87295,
    "SEK": 10.2983,
    "EUR": 1.092,
    "USD": 1.2234,
    ...
    }
}
```

Construa uma aplicação que faz o cambio automática entre Euros e outra das moedas disponíveis (escolhido de um *spinner*)

Ex 37. Construa uma calculadora com a 4 operações básicas

Ex 38. Implemente o jogo do galo (2 humanos) tendo em conta o seguinte:

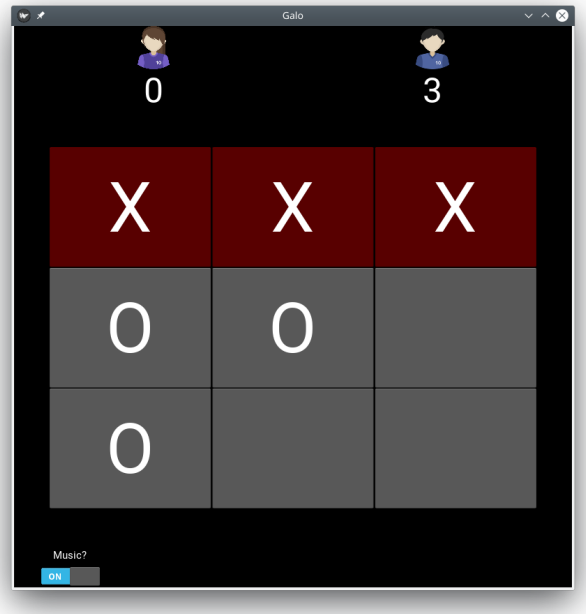


Figura 11: Exemplo de interface para o jogo do galo

- use a linguagem kivy para definir o interface. Como exemplo base de interface pode usar o exemplo na Figura 11.
- o jogo é composto por um conjunto de partidas terminando quando um dos jogadores tiver ganho 3 partidas.

Ex 39. Implemente o jogo *Mine Sweeper*.

Ex 40. Implemente o jogo da forca.

8 Exercícios globais

Ex 41. Considere uma aplicação para encomendar pizzas que deverá ter as seguintes componente:

- Uma base de dados que guarda (descarregue-a do bitbucket):
  - Clientes: idCliente, nome, morada e telefone.
  - Pizzas: Nome (será a chave primária) e ingredientes.
  - Encomendas: idCliente, Nome (da pizza), quantidade, tamanho (pequena, média, familiar) e data/hora
- *Web service* que intermedia todas as comunicações entre a base de dados e o interface GUI. Comece por pensar nos *endpoints* que serão necessários. P.e., para obter a lista de clientes, para obter a lista de pizzas, para fazer uma encomenda, etc.

- Um interface GUI (para o *call center*) para fazer as encomendas. O interface deve ser implementado usando kivy. Comece por esboçar o interface (pode usar papel e lápis) e depois implemente-o.

Algumas ideias: (a) no sistema deve ser fácil encontrar os clientes por nome; (b) sistema fácil para filtrar a lista de Pizzas; (c) As pizzas apresentarem uma imagem ilustrativa; etc.

Observação final: as classificações finais são de 0 a 5... Não tem de fazer tudo o que é enunciado para poderem apresentar o trabalho! O fundamental é ter um sistema "funcional"!

Ex 42. Este exercício consiste na implementação de uma aplicação para um stand de motorizadas clássicas, com as seguintes características:

- Pretende-se guardar numa **base de dados** as entidades:
  - MOTORIZADAS com os atributos (id_motorizada, marca, modelo, matrícula, cilindrada).
  - CLIENTES com os atributos (id_cliente, nome, número de telefone, NIF, email)
  - COMPRA com os atributos (id_compra, id_motorizada, id_cliente, data, preço)[6].
- Para gerir motorizadas, clientes e compras deve ter um **interface GUI implementado usando kivy**.
- Um **web service** intermedia as comunicações entre a base de dados e o interface GUI, i.e., o interface não comunica diretamente com a base de dados mas sim com este serviço.
- Algumas funcionalidades extra serão (até 20% da classificação): (a) no sistema deve ser fácil encontrar os clientes por nome; (b) a aplicação deve ter um sistema de autenticação para o utilizador do *stand*; (c) no sistema será fácil filtrar a lista de motorizadas (p.e., por marca); (d) As motorizadas apresentarem uma imagem; (e) o sistema tem um módulo estatístico que apresente as vendas resumidas de um determinado dia, para um cliente, etc.

Observação final: as classificações finais são de 0 a 5... Não tem de fazer tudo o que é enunciado para poderem apresentar o trabalho! O fundamental é ter um sistema "funcional"!

Ex 43. Este exercício consiste na implementação de uma aplicação para gestão de torneios. Comece por definir um jogo de equipas que lhe interesse (e.g., futebol, andebol, CS Go, padel, etc.) e que se possa encaixar nas seguintes características[7]:

- Pretende-se guardar numa **base de dados** as entidades:
  - Torneios – cada torneio tem os atributos id_torneio, nome, datas (início-fim) e local.
  - Equipa – cada equipa é constituída para um torneio e tem os atributos id_equipa, sigla, nome e id_torneio (chave estrangeira).
  - Jogador – cada jogador tem id_jogador, nome, data_de_nascimento (dia-mês-ano).

[6] id_cliente permite saber se a motorizada faz parte do stock (NULL) ou se já foi vendida (não NULL)

[7] Pode fazer alterações que ache conveniente à base de dados desde que não a simplifique

  - Jogo: cada jogo tem 2 equipas, uma data de realização e é guardado o resultado final (e.g., números de golos das 2 equipas).
  - Ao longo do tempo cada jogador pode participar em várias equipas e cada equipa tem vários jogadores.

- Para gerir as entidades deve ter um **interface GUI implementado usando kivy**.
- Um **web service** intermedia as comunicações entre a base de dados e o interface GUI, i.e., o interface não comunica diretamente com a base de dados mas sim com este serviço.
- Algumas funcionalidades extra serão (até 30% da classificação): (a) no sistema deve ser fácil encontrar os jogadores por nome; (b) no sistema será fácil filtrar a lista de torneios (p.e., por nome ou por datas); (c) o sistema permite a gestão de um torneio, incluindo a definição manual ou aleatória das partidas num sistema de eliminação; (d) As equipas têm um logo que é apresentado no interface; (e) o sistema tem um módulo estatístico que apresente número de vitórias por jogador, jogador com mais vitórias num determinado torneio, tornei com mais "golos", jogo com mais "golos", etc.

Observação final: as classificações finais são de 0 a 5... Não tem de fazer tudo o que é enunciado para poderem apresentar o trabalho! O fundamental é ter um sistema "funcional"!