

Programação Orientada a Objetos  
Lista de exercícios

# Lista de Exercícios

## Classes e objetos

**Exercício 1.1.** Descreva em no máximo 200 palavras o que é um automóvel e o que ele faz. Liste os substantivos e verbos separadamente. Cada substantivo corresponde a um objeto que precisará ser construído para implementar um sistema, nesse caso, um carro. Selecione 5 dos objetos que você listou e, para cada um, liste vários atributos e comportamentos. Descreva brevemente como esses objetos interagem entre si e com outros objetos na sua descrição. Estes passos que você seguiu são típicos do projeto orientado a objetos.

**Exercício 1.2.** [deite] Crie uma classe chamada `Invoice` que possa ser utilizado por uma loja de suprimentos de informática para representar uma fatura de um item vendido na loja. Uma fatura deve incluir as seguintes informações como atributos:

- o número do item faturado,
- a descrição do item,
- a quantidade comprada do item e
- o preço unitário do item.

Sua classe deve ter um construtor que inicialize os quatro atributos. Se a quantidade não for positiva, ela deve ser configurada como 0. Se o preço por item não for positivo ele deve ser configurado como 0.0. Forneça um método `set` e um método `get` para cada variável de instância. Além disso, forneça um método chamado `getInvoiceAmount` que calcula o valor da fatura (isso é, multiplica a quantidade pelo preço por item) e depois retorna o valor como um `double`. Escreva um aplicativo de teste que demonstra as capacidades da classe `Invoice`.

**Exercício 1.3.** A fim de representar empregados em uma firma, crie uma classe chamada `Empregado` que inclui as três informações a seguir como atributos:

- um primeiro nome,
- um sobrenome, e
- um salário mensal.

Sua classe deve ter um construtor que inicializa os três atributos. Forneça um método `set` e `get` para cada atributo. Se o salário mensal não for positivo, configure-o como 0.0. Escreva um aplicativo de teste que demonstra as capacidades da classe. Crie duas instâncias da classe e exiba o salário anual de cada instância. Então dê a cada empregado um aumento de 10% e exiba novamente o salário anual de cada empregado.

**Exercício 1.4.** Cria uma classe chamada `Complex` para representar números complexos e escreva um programa para testá-la.

1. Escolha uma representação para os números complexos, usando a forma retangular ou a forma polar.
2. Forneça três construtores que permitam que objetos dessa classe sejam inicializados ao serem alocados na memória:
  - um construtor sem parâmetros que inicializa o objeto como zero

- um construtor com um parâmetro representando a parte real; a parte imaginária será zero
  - um construtor com dois parâmetros representando as partes real e imaginária
3. Defina operações para obter a parte real, a parte imaginária, o módulo (valor absoluto) e o ângulo de um número complexo.
  4. Forneça a operação para determinar o inverso aditivo de um número complexo.
  5. Forneça as operações aritméticas básicas com números complexos: adição, subtração, multiplicação e divisão.
  6. Forneça as operações relacionais que permitem comparar dois números complexos.
  7. Defina a operação `toString` para converter um número complexo em string. Utilize o formato  $(a,b)$ , onde  $a$  é a parte real e  $b$  é a parte imaginária.
  8. Escreva um aplicativo de teste que demonstra as capacidades da classe `Complex`.

**Exercício 1.5.** Crie uma classe para representar datas.

1. Represente uma data usando três atributos: o dia, o mês, e o ano.
2. Sua classe deve ter um construtor que inicializa os três atributos e verifica a validade dos valores fornecidos.
3. Forneça um construtor sem parâmetros que inicializa a data com a data atual fornecida pelo sistema operacional.
4. Forneça um método `set` um `get` para cada atributo.
5. Forneça o método `toString` para retornar uma representação da data como string. Considere que a data deve ser formatada mostrando o dia, o mês e o ano separados por barra (/).
6. Forneça uma operação para avançar uma data para o dia seguinte.
7. Escreva um aplicativo de teste que demonstra as capacidades da classe.

Garanta que uma instância desta classe sempre esteja em um estado consistente.

**Exercício 1.6.** Escreva um programa completo para jogar o jogo da velha. Para tanto crie uma classe `JogoDaVelha`:

- a classe deve conter como dados privados um array bidimensional 3x3 para representar a grade do jogo
- crie uma enumeração para representar as possibilidades de ocupação de uma casa na grade (vazia, jogador 1 ou jogador 2)
- o construtor deve inicializar a grade como vazia
- forneça um método para exibir a grade
- permita dois jogadores humanos
- forneça um método para jogar o jogo; todo movimento deve ocorrer em uma casa vazia; depois de cada movimento, determine se houve uma derrota ou um empate.

**Exercício 1.7.** Crie uma classe `IntegerSet` para representar um conjunto de números inteiros. Cada objeto da classe `IntegerSet` pode armazenar inteiros no intervalo de 0 até um valor máximo específico para cada objeto. O conjunto deve ser representado por um array de booleanos. O elemento do array na posição  $i$  é verdadeiro se e somente se o inteiro  $i$  pertencer ao conjunto. O construtor inicializa o objeto como um conjunto vazio (isto é, um conjunto cuja representação de array contém todos os valores falso). Forneça métodos para implementar as operações de união, interseção e diferença de conjuntos. Forneça um método para inserir um novo elemento no conjunto e outro método para excluir um elemento do conjunto. Forneça ainda um método para converter um conjunto para string. Faça uma aplicação para testar a classe.

**Exercício 1.8.** Grafo é uma estrutura de dados muito comum em computação, e os algoritmos sobre grafos são fundamentais para a área.

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  $w$ .

Um grafo pode ser representado por listas de adjacência ou por uma matriz de adjacência, como é ilustrado na figura 1.1.

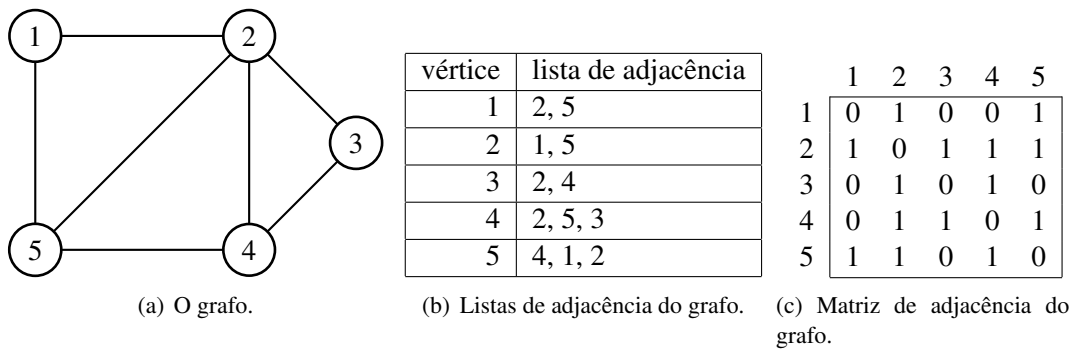


Figura 1.1: Um exemplo de grafo.

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.