PRÁTICA LABORATORIAL 09 – PROGRAMAÇÃO ORIENTADA A OBJETOS

Objetivos:

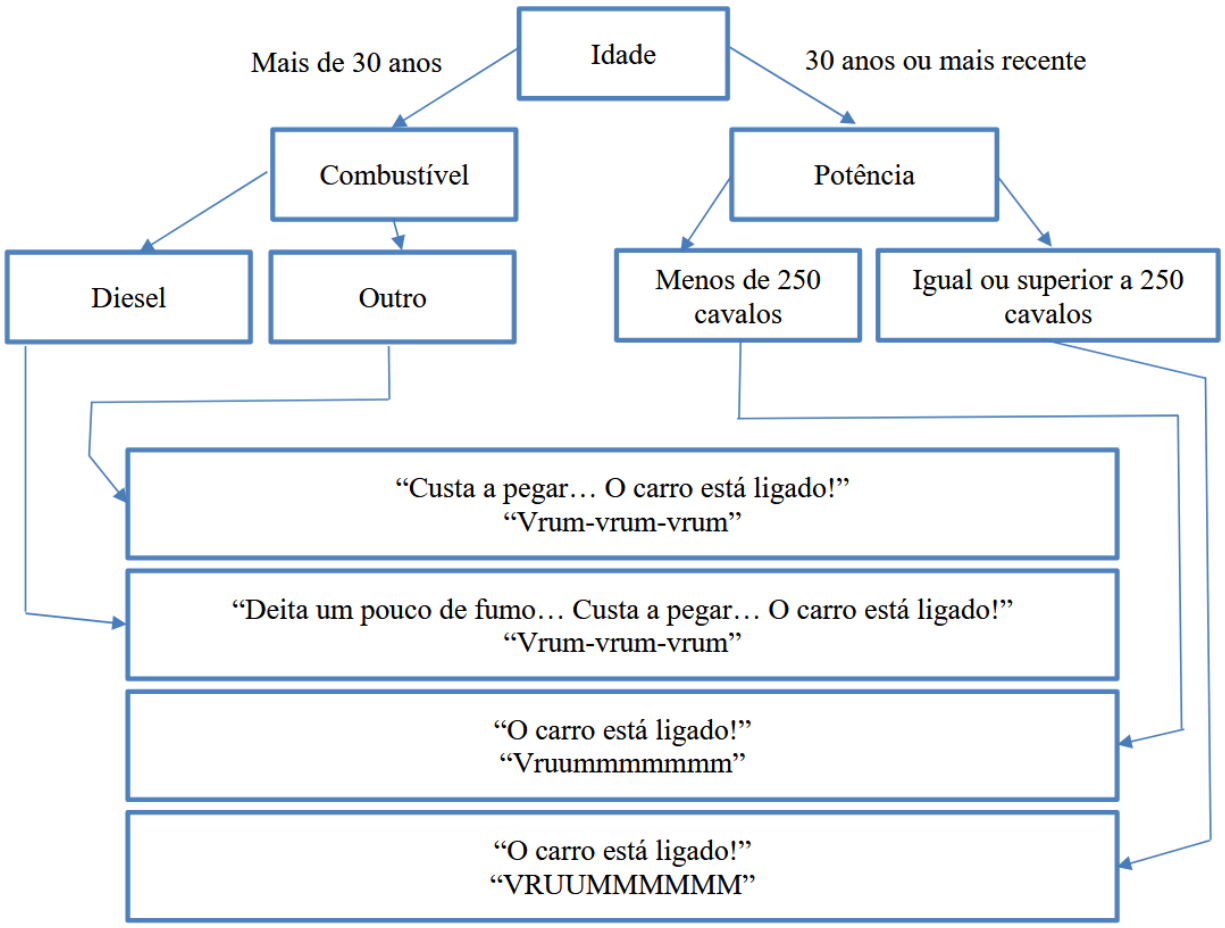
- Classes
- Objetos
- Métodos
- Enumerações

EXERCÍCIOS

1. Atualize a classe **Carro** (pode copiar a classe da Ficha Prática 08) e acrescente os seeguintes atributos: potência, cilindrada, tipoCombustivel, consumoLitros100Km.

   TipoCombustivel deve ser uma enumeração com os seguintes valores: GASOLINA, DIESEL, GPL.

   Altere o método **ligar** para que imprima uma mensagem de acordo com a idade, potência e tipo de combustível do carro. Tenha em consideração o seguintes esquema:



Crie um método **corrida** que recebe um adversário por parâmetro e retorna o Carro vencedor. O vencedor é o carro com mais potência (caso seja a mesma o fator de desempate é a cilindrada (o maior ganha), caso contrário o fator de desempate é a idade (o mais recente ganha), caso contrário temos empate).

Crie um método **calcularConsumo** que recebe um número real por parâmetro, que representa uma distância em quilómetros. Deve retornar o consumo que o carro fará, em litros, nessa viagem.

No Main:

a) Instancie 4 carros A B C D (nomes exemplificativos), definindo todos os seus atributos.
b) Invoque o método ligar para os quatro carros.
c) Faça uma corrida entre A e B
d) Faça uma corrida entre C e D
e) Faça uma corrida entre o vencedor da primeira e o vencedor da segunda corrida.
f) Calcule quanto é que o vencedor da corrida final consumiria numa viagem de 65 Km.

2. Atualize a classe **ContaBancaria** e acrescente os seguintes atributos: ano de abertura, margem de emprestimo e valor em divida. O ano de abertura deve ser definido, por predefinição, a 2025. A margem de emprestimo deve ser definida, por predefinição, a 0.5 (que representa 50%). O valor em dívida deve ser definido, por predefinição a 0. Sendo assim, o construtor não muda a nível de parâmetros.

Crie o método **pedirEmprestimo**, deve receber como parâmetro, o valor a pedir (em €). Seguidamente, avalia se a conta pode pedir um emprestimo com esse valor. Para tal, teremos de multiplicar o saldo pela margem de emprestimo para perceber o limite de valor a emprestar. A seguir, sabemos que este banco não faz multiplo emprestimo, ou seja, se a Conta tiver um valor em divida, o novo pedido é automáticamente desconsiderado. Caso o emprestimo seja concedido, aumento o valor pedido ao saldo da conta e aumenta o valor pedido ao valor em dívida, apresentando no fim uma mensagem de confirmação. Caso não seja concedido, apresenta uma mensagem de erro com a razão de recusa (valor muito alto ou divida ativa).

Crie o método **amortizarEmprestimo**, que recebe um valor (em €) por parâmetro e diminui esse valor ao valor em dívida. Tenha em atenção que não pode amortizar mais do que é possível (se deve 100€ não pode amortizar 500€). Seguidamente verifique se a Conta tem saldo para fazer essa operação. Caso seja possível, dminiui o saldo e diminui o valor em dívida, apresentando uma mensagem de confirmação. Caso não seja possível apresenta uma mensagem de erro com a razão (valor em excesso ou saldo insuficiente).

No Main:

a) Instancie 3 Contas, definindo todos os seus atributos.
b) Deposite algum valor nas contas.
c) Tente efetuar um empréstimo recusado.
d) Tente efetuar um empréstimo válido. (Teste a sua classe para todos os casos cenário).

3. Crie uma classe **Animal** com os seguintes atributos: nome, espécie, país de origem, peso (em Kg), alimentação (array de Strings) e classe do reino animal (MAMÍFERO, ANFÍBIO, AVE, PEIXE, RÉPTIL). O construtor recebe todos os atributos nos parâmetros.

   Crie o método **fazerBarulho** que, de acordo com o reino animal, faz um barulho diferente:

   - MAMÍFERO: Tinoninoni
   - ANFÍBIO: Brrrrrr
   - AVE: Kwak Kwak
   - PEIXE: Blub Blub Splash
   - RÉPTIL: Psssssss

   Crie o método **comer** que recebe uma String por parâmetro que representa o alimento, e um double que é o peso do alimento em gramas. Se o alimento constar no array de alimentos do Animal (porque ele come esse alimento), então o animal come, aumenta o seu peso, e apresenta uma mensagem de confirmação "O {espécie} {nome} comeu {alimento}". Para além disso, como o animal está contente, faz barulho por ter comido.
   Caso o alimento não conste no array de alimentação, então o animal recusa a comida, o peso não muda e apresenta uma mensagem de erro "O {espécie} {nome} não comeu {alimento}". Como está triste, não faz barulho.
   No Main:

   a) Instancie um Animal, definindo todos os seus atributos.
   b) O animal faz barulho.
   c) Tente alimentar o animal com algo que ele não gosta (verifique se o peso se mantém igual).
   d) Alimento o animal com algo que ele gosta.

4. Crie uma classe **Imovel** com os seguintes atributos: rua, número da porta, cidade, tipo (APARTAMENTO, CASA, MANSAO), acabamento (PARA_RESTAURO, USADA, NOVA, NOVA_COM_ALTO_ACABAMENTO), área total, número de quartos, número de casas de banho, área da piscina.

   Crie o método **calcularValor**, que retorna o valor do imóvel com base nas seguintes regras:

   - Apartamento: cada m² vale 1000€
   - Casa: cada m² vale 3000€
   - Mansão: cada m² vale 5000€
   - Para restauro: preço total tem 50% de desconto
   - Usada: preço total tem 10% de desconto
   - Nova: sem desconto
   - Nova com alto acabamento: valoriza 25%
   - Cada quarto adiciona 7500€ ao preço
   - Cada casa de banho adiciona 10500€ ao preço
   - A piscina custa 1000€/m²

Crie um método de acesso que altera o estado do imóvel (por exemplo, de usada para nova com alto acabamento).

Crie um método **compararImoveis**, que recebe outro imóvel como parâmetro e retorna o mais caro (com base em calcularValor).
No Main:

a) Instancie 2 Imóveis, definindo todos os seus atributos.
b) Apresente as específicações do Imóvel mais caro.

5. Crie uma classe **Pessoa** com os seguintes atributos: nome, idade, cidade, email, telemóvel.
Crie um método exibirDetalhes, que mostra na consola os detalhes da Pessoa.

Crie uma classe **Agenda** com os seguintes atributos: ano de criação e um array de Pessoas chamado listaContactos. O ano de criação deve começar, por defeito, a 2025. O construtor recebe o tamanho da lista de contactos e inicializa a lista de contactos (vazia).

Crie um método **adicionarPessoa**, que recebe uma pessoa por parâmetro e adiciona à lista.

Crie um método **listarContactos**, que imprime na consola todas as pessoas da lista.

Crie um método **pesquisarContactos**, que recebe um parametro String cidadePesquisa, e imprime na consola todas as pessoas da lista que moram naquela cidade.

No Main:

a) Instancie 4 Pessoas, definindo todos os seus atributos.
b) Instancie uma Agenda.
c) Adicione 2 pessoas à Agenda.
d) Liste os contactos
e) Adicione mais 2 pessoas à Agenda.
f) Lista novamente os contactos.
g) Faça uma pesquisa por cidade.

6. Crie uma classe **Atleta** com os seguintes atributos: nome, modalidade desportiva, altura, peso, país de origem, ranking mundial (inteiro).

   Crie uma classe **Competicao** com os seguintes atributos: nome da competição, país e a lista de Atletas participantes. O construtor recebe o nome, país e número máximo de inscritos e inicializa a lista de contactos (vazia).

   Crie um método **inscreverAtleta**, que recebe um Atleta por parâmetro e adiciona à lista.

   Crie um método **listarParticipantes**, que imprime na consola todos os atletas da lista.

   Crie um método **atletasDaCasa**, que imprime na consola todos os atletascujo país é o mesmo da competição.

   No Main:

   a) Instancie 6 Atletas, definindo todos os seus atributos.
   b) Instancie duas Competições.
   c) Adicione 2 pessoas à Agenda.
   d) Adicione e liste os participantes de cada uma das Competições.
   e) Apresente os atletas da casa das competições.

Bom trabalho! ☺