

## Trabalho Prático - Parte 2 - Simulador de Corrida de Veículos

**Data de entrega:** 24/06/2022 (até 23h55), via moodle.

# O trabalho deverá ser individual.

Um simulador de corrida de veículos irá controlar veículos do tipo bicicleta, motocicleta, carro passeio e esportivo através de seu centro de comandos. Os veículos estarão competindo no estilo corrida.

Cada veículo criado possuirá uma identificação única (que deverá ser um número inteiro, gerado automaticamente) e uma quantidade de rodas (cujos pneus estarão calibrados ou não de acordo com um sorteio<sup>1</sup>). Além disso, os veículos motorizados terão uma quantidade inicial de combustível, 3,5 litros, o valor comercial e se o IPVA (Imposto sobre a Propriedade de Veículos Automotores) do veículo está pago ou não.<sup>2</sup>

O IPVA de um veículo deverá ser calculado utilizando-se o valor base de R\$ 500,00 multiplicado pelo valor da alíquota de cada tipo, ou seja, para motocicleta o valor corresponde a 0,75, para carro passeio 1,3 e para esportivo 3,15.

Os veículos motorizados podem ser abastecidos e consomem combustível à medida que se deslocam. Eles apenas se movimentam se há combustível suficiente para tal, se os pneus das rodas estiverem todos calibrados e se o IPVA estiver pago. Assume-se que para mover um “*bloco*” de espaço, a motocicleta gasta 0,25 litros de combustível, o carro passeio gasta 0,75 litros e o esportivo gasta 2,3 litros. Portanto, um veículo automotivo não deve se movimentar se:

- não possuir a quantidade de combustível suficiente,
- se um ou mais pneus não estiverem calibrados (também válido para bicicleta) e,
- se o IPVA não estiver pago.

Os veículos devem ser desenhados em modo texto<sup>3</sup>, e se movem sempre na horizontal da esquerda para direita de acordo com suas respectivas quantidades de “*blocos*” de espaço (unidade de movimento):

- bicicleta: de dois em dois “*blocos*” de espaço,
- motocicleta: de três em três “*blocos*” de espaço,
- carro passeio: de cinco em cinco “*blocos*” de espaço,
- esportivo: de dez em dez “*blocos*” de espaço.

---

<sup>1</sup>Por exemplo, para cada roda, sorteia-se um número de 0 a 100, se for par, calibra-se o iésimo pneu, caso contrário, não.

<sup>2</sup>Também neste caso, sorteia-se um número de 0 a 100, se for par, o IPVA já está pago, caso contrário, não.

<sup>3</sup>Devem ser utilizados, para cada veículo, os desenhos em código *ascii* fornecidos no arquivo [desenhaCarro.java](#).

Com base no detalhamento anterior, faça:

- I. Descreva o diagrama UML das classes do simulador tomando como modelo o esboço apresentado na Figura 1 (gerar o arquivo pdf do diagrama).

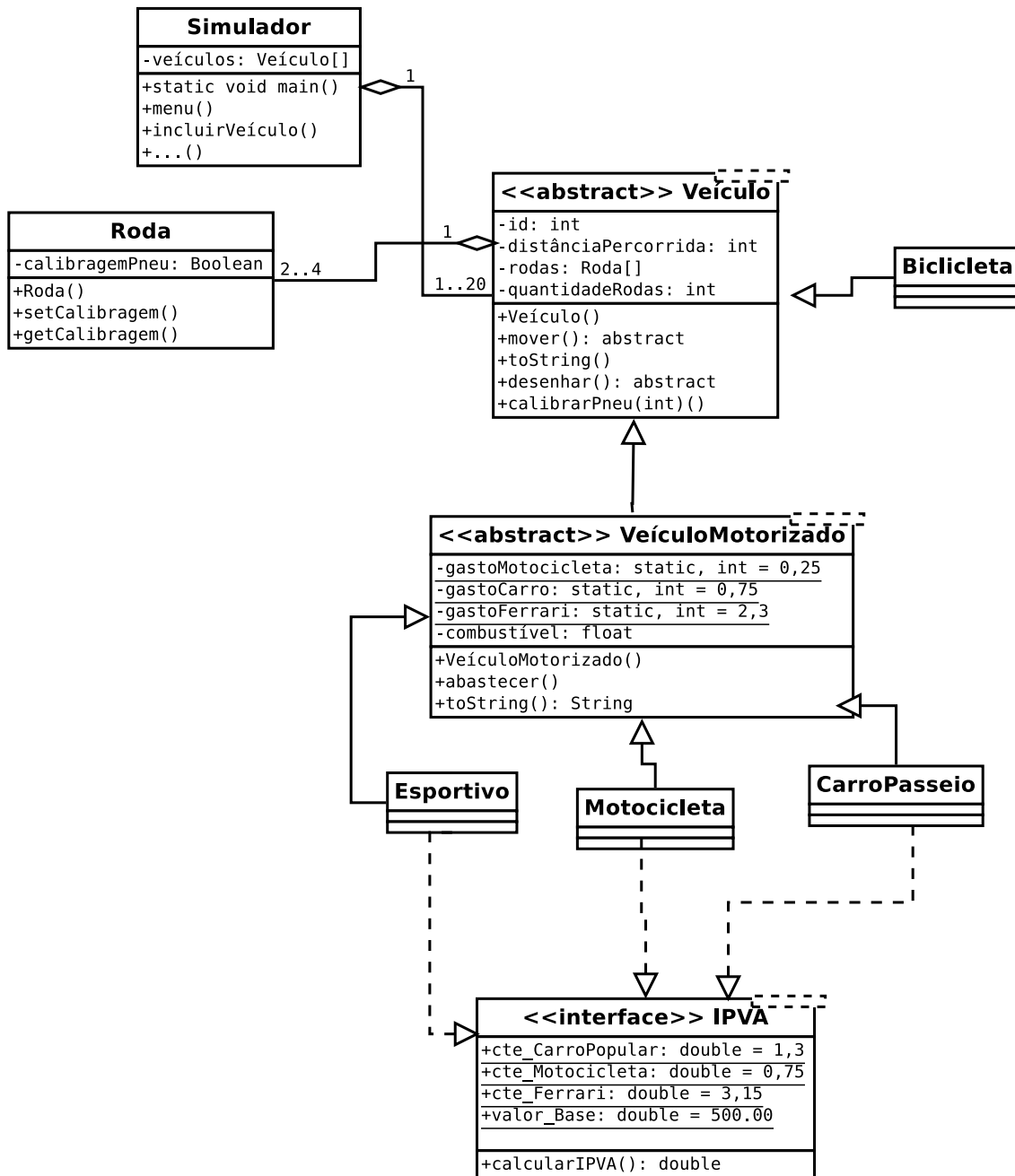


Figura 1: Esboço do diagrama UML a ser seguido.

- II. Com base do diagrama UML elaborado acima, desenvolva um aplicativo Java com um **menu interativo** que permita ao usuário executar o simulador de corrida de veículos com no máximo 20 veículos:

(1) Incluir veículo

[Solicitar o tipo do veículo (B, M, C, E). Gerar um *id* (inteiro) automático para o veículo e assumir que cada pneu está vazio ou não de acordo com um

sorteio. Para os automotivos sortear se o IPVA estará pago ou não e calcular o valor do IPVA.]

(2) Remover um veículo

[deve-se informar o *id* do veículo]

(3) Abastecer veículo

[deve-se informar o *id* do veículo e a quantidade de combustível em litros]

(4) Movimentar um veículo específico

(5) Movimentar veículos por tipo

(6) Movimentar todos os veículos

(7) Imprimir todos os dados de todos os veículos

(8) Imprimir dados de veículos por tipo

(9) Esvaziar/calibrar um pneu específico,

[Solicitar o *id* do veículo e qual pneu será esvaziado/calibrado (primeiro, segundo,...), caso não seja dada uma entrada correta, repetir a leitura.]

(10) Calibrar todos os pneus de veículos por tipo

(11) Esvaziar todos os pneus de veículos por tipo

(12) Imprimir pista de corrida

[imprime na ordem em que estão no *array*, os veículos com seus respectivos “*blocos*” de espaços percorridos. Para cada tipo de veículo, utilizar os desenhos *ascii* correspondentes (contidos no arquivo [desenhaCarro.java](#)), como mostra o exemplo abaixo, que correspondem a uma motocicleta, um carro passeio, uma bicicleta e um esportivo respectivamente:]

```

      ,_oo
     ./c-//::
    ( _ )'==( _ )

      ----
     --/   | \_
    |  _   _ ' _ .
    '-( _ )----( _ )--'

```

```

      __o
     _'\<,_
    (*)/  (*)

```

```

      --
     ~( @\ \
      -----] [_/_>-----
    /  _ _ \<>  |  _ _ \
   =\_/_\_\_\_\_ | \/_\_\_D
      ( _ )      ( _ )

```

(13) Gravar veículos em arquivo

(14) Ler veículos do arquivo

(15) Sair da aplicação

## Avaliação:

O trabalho será avaliado em função da:

- Correção (o aplicativo cumpre com as exigências?);
- Documentação (o aplicativo está devidamente comentado?);
- Paradigma orientado a objetos (o aplicativo está seguindo os princípios da programação OO: -encapsulamento, -associação de classes, - herança, - polimorfismo?);
- Modularidade (o aplicativo está bem estruturado onde necessário, com métodos (funções) parametrizados?);
- Robustez (o aplicativo trava em tempo de execução?).

Detalhamento de itens a serem avaliados:

Item	Atendeu?
Respeitar o princípio do encapsulamento de dados	
Usar modificadores de acesso adequados ( <b>private</b> e <b>public</b> )	
Criar getters e setters que forem necessários	
Criar métodos construtores parametrizados	
Fazer sobrecarga de pelo menos um método (qualquer um)	
Ter pelo menos um atributo <b>final</b>	
Fazer uso da palavra reservada <b>this</b>	
Fazer uso do operador <b>instanceof</b>	
Ter pelo menos um atributo <b>static</b>	
Criar relacionamento entre classes (Agregação ou Composição)	
Fazer uso de classe abstrata	
Fazer uso de interface	
Fazer uso do conceito de herança e polimorfismo	
Não utilizar modificador <b>protected</b>	
Não apresentar erro em tempo de execução	
Apresentar o diagrama UML	